Physical Attacks and Countermeasures on Real-World Cryptographic Implementations

Niels Samwel

ISBN: 978–94–6419–631–3
Typeset using LaTeX

Cover design by Promotie In Zicht (www.promotie-inzicht.nl)
Printed by Gildeprint (www.gildeprint.nl)

Radboud University

This work was partially supported by the European Commission through the ERC Advanced Grant 788980 (ESCADA).

Physical Attacks and Countermeasures on Real-World Cryptographic Implementations

Proefschrift
ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op **maandag 5 december** 2022
om **14.30** uur precies

door

Niels Samwel
geboren op 22 juli 1991
te Wateringen

Promotoren

Prof. dr. L. Batina
Prof. dr. J.J.C. Daemen

Copromotor

Dr. Y. Yarom (University of Adelaide, Australië)

Manuscriptcommissie

Prof. dr. T.M. Heskes
Prof. dr. S. Mangard (TU Graz, Oostenrijk)
Dr. E. De Mulder (Rambus, Verenigde Staten)
Dr. D.F. Aranha (Aarhus Universitet, Denemarken)
Prof. dr. P.A. Fouque (Université Rennes, Frankrijk)

Acknowledgements

I would like to take this opportunity to thank the amazing people who were by my side and made this PhD journey possible.

First of all, I would like to thank my promoters and supervisors, Lejla Batina and Joan Daemen. For the feedback, encouragement and the papers we worked on together. You were not just my supervisors, but you have opened your home to me. Thank you for providing an office dog. It was nice cuddling with Rocky.

Furthermore, I also thank Yuval Yarom, my co-promoter. I am grateful for the invitation to visit you at the University of Adelaide in the summer of 2018. It was great to work together and it was an amazing experience to visit Australia and to pet koalas.

I thank the manuscript committee Tom Heskes, Stefan Mangard, Elke De Mulder, Diego Aranha and Pierre-Alain Fouque for taking time to read my thesis.

Special thanks to my paranymphs and friends Łukasz Chmielewski and Kostas Papagiannopoulos:

Łukasz, we have met at Radboud University and worked together on several papers. After work we sometimes had drinks or dinner together. I am sorry for all the days you had to bike through the rain just to drink a beer at my place. It seems like every time you came over, it was raining in the Nijmegen area. But the rain never stops you! Łukasz, I want to thank you for all your help, support and for all the good times at the university and after work.

Kostas, your support began during my master's thesis project. Later on, we became co-workers and we worked together in the lab. I was honored to be one of your "bridesmaids" during your PhD defense. Thank you for your support during my own PhD and today as my paranymph at my defense.

I want to thank Paul Scheidt my internship supervisor at Google, for giving me the opportunity to see what it is like to work for one of the most dynamic tech firms in the world. I also want to thank Paul and his family, for making me feel at home in California during my 6-month internship.

Daniel Genkin, I thank you for all the invitations to come over to the United States and work at two different universities together. We have been working long hours in the different labs, but we had a lot of fun too.

Furthermore, I thank the co-workers from the Radboud University, the Digital Security Group. A special thanks goes out to my office-mates, Pedro, Joost & Joost, Ko, Benoît, Leo, Omid, Parisa and Konstantina for the nice times and all the interesting discussions.

Moreover, I am grateful to all my friends, family and my in-laws for showing interest in my work and supporting me along the way.

I am deeply grateful to my wife Anne-Simone, whom I met just before I started my PhD. I am sorry for all the days I left you to work abroad and attend conferences. I am also sorry for all the days I was stressed out and grumpy while writing my manuscript. But most of all, I am thanking you for the endless support and encouragement during my PhD. I could not have done it without you, and of course our cat Wibi.

Last but not least, a big thank you to my parents, Yvonne and Erik, and my sister Marit. Ik wil jullie bedanken voor alle steun en het bieden van een luisterend oor als ik vast zat tijdens mijn werk. Bedankt Mama, Papa & Marit!

Niels Samwel
Beek, October 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the current age where much of our life is digital, we use electronic items such as ID cards, passports and bankcards. These items carry our information in a physical way, such as names, pictures and other information that identifies us or in case of a bankcard, the card number. In addition to this physical representation of information they also carry the information in a digital form that can be interacted with using different (contactless) interfaces. To prevent misuse such as forgeries or clones that cause identity fraud or theft of money, the information must be protected. Protecting the information has different requirements, in case of ID cards or passports the information is typically not secret but the authenticity should be ensured. The information that is stored on a bankcard allows the owner to authorize payments. Thus this information is secret and must be kept confidential. Protecting information with these requirements is accomplished by using cryptography.

Data is an important and valuable commodity. People say data is the new gold. This makes it important to keep data secure and private. When data that should be kept private gets in the wrong hands, it can have serious consequences. For instance, a data breach at a government agency that leads to leaking secret information to an enemy state. When data from a company is leaked it can put secret information in the hands of a competitor which could give them an advantage. A data breach of personal data from for instance a government institution or a web store can lead to identity theft. Data can be kept secure and private with the use of cryptography.

Keeping data secret so that an unrelated party cannot learn its contents is what is referred to as confidentiality of data. It is also important that data remains unchanged, and that it is possible to verify the content. This is called the integrity of data. Cryptography can be used to ensure both the confidentiality and the integrity of data. To keep data confidential, it is modified in a way such that an outsider is not able to read it. This is accomplished using symmetric cryptography, where each party holds the same secret key that is used to encrypt and decrypt the data. To determine the integrity of data, a hash function is used. A hash function typically compresses an arbitrary amount of data into a fixed size output or digest that can be compared to determine that the input is the same. When encrypted data is transmitted it is possible to add a hash digest before the data is encrypted so that after decryption, the integrity of the data can be verified. Sometimes it is required to know if the

source of the data is authentic. To accomplish this, authentication is used where a Message Authentication Code (MAC) is computed for some data or a message with a secret key. The key is only known to the sender and the receiver, so that the receiver can also computed the MAC for the received message and verify the result. This way the receiver can be sure of the source of the data. A MAC does not only verify the authenticity but also the integrity of a message.

Cryptographic primitives are generic building blocks used in cryptography, such as block ciphers, stream ciphers, hash functions or permutations. When they are used on their own they typically do not provide any meaningful cryptographic security properties. To achieve this cryptographic primitives must be used in a mode of operation. Different modes of operation allow a cryptographic primitive to be used in multiple ways. A cryptographic permutation can, for instance, be used for encryption and decryption, (keyed) hashing or message authentication codes.

In the early days of cryptography it was almost exclusively used by the military. This started already as early as the Roman empire where Julius Caesar used a basic substitution cipher for his correspondence with his officers, now known as Caesar's Cipher. This approach of performing cryptography where both parties use the same secret key is called symmetric cryptography. Modern cryptography started in the seventies with the publication of the Data Encryption Standard (DES) [278] for symmetric cryptography. Another major development in cryptography came with the introduction of the Diffie-Hellman [100] key exchange algorithm which was the start of asymmetric cryptography. In secret this algorithm was also developed by the GCHQ prior to the publication for government and military use.

As technology advanced it became clear that the key length of DES which is 56 bits resulting in 2^{56} possible keys would not provide adequate security. This demanded a more secure symmetric cryptographic algorithm and led to its replacement by the Advanced Encryption Standard (AES) [96] in 2001. After two decades, AES is still the de facto standard for symmetric cryptography. During this time a lot of research went into this area. In cryptography data that is encrypted is typically also checked for integrity and authenticity using for instance a MAC. The actions of encryption and validating the integrity and authenticity can be combined, this is called authenticated encryption. In 2013, the Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAESAR) was announced. This lead to a final portfolio of submissions with different advantages which was announced in 2019. In 2015, a competition for lightweight cryptography was announced by NIST. The competition was announced in search of a lightweight cipher that works well on highly-constrained devices. In March 2021, ten finalists were announced, it is expected that the final round of the standardization process ends at the end of 2022.

A cryptographic key is needed to encrypt data. There are two types of keys, secret keys and, public and private key pairs. Secret keys used for symmetric cryptography are mainly used to encrypt data, they have the advantage that symmetric ciphers are much faster. The most commonly used symmetric cipher is AES [96]. To decrypt data, both the sender and the receiver must use the same secret key. So to securely transmit data, a secret key must be shared using a secure channel. If a sender and a receiver already have a secure channel they could just use that channel to transmit the data, so they need a way to share a key over a public channel. This is where public keys are used, public keys are derived from their private counterparts and

always belong together in a key pair. To setup a secure channel, the sending party needs the public key of the receiving party. Next, the sender uses the private key with the receivers public key to compute a shared secret. The receiver does a similar computation with the senders public key and its own private key. From this shared secret a secret key can be derived that is used for symmetric cryptography. The sender and the receiver were able to share a secret key without disclosing any secret information. Well known key exchange schemes are Diffie-Hellman (DH) [100] and Elliptic Curve Diffie-Hellman [170, 204]. The public key scheme RSA [249] is used for public key encryption and signatures.

Cryptographic operations such as encryption, decryption and signature generation are computed on hardware. This hardware consumes power during operation. When this power consumption is measured, information about the operation can be deducted that in some cases can lead to the recovery of secrets. The leakage of the power consumption is an unwanted side-effect of the operation, it is called a side-channel. When information from a side-channel is used to recover secret information it is called a side-channel attack. In addition to the power consumption there are other side-channels, they include the electromagnetic emanations from the hardware that computes cryptographic operations and the time it takes to compute such operations. Using side-channel information to recover secret information does not require active involvement with the operations and thus is called a passive attack. Hardware that is used for cryptographic operations can also be manipulated from the outside to affect the data that it operates on, this is called an active attack or a fault injection attack. An active attack requires an attacker to have the ability to modify the target. If the data is manipulated in a specific way and at the correct time, this can also lead to the full recovery of secret information. Countermeasures must be implemented to thwart side-channel attacks. To counter passive attacks, a technique called masking is often implemented. With masking the information is typically shared using random data such that each share appears random. When a share is processed there is only side-channel leakage of this random data. Passive attacks are typically countered using redundancy or verifying integrity of data.

Measuring the leaking side-channel information can be a costly exercise and time consuming exercise. It often requires expensive hardware and technical skills. This high cost can be overcome by simulating the side-channel leakage. The simulated leakage is analyzed in the same way as real leakage. This allows countermeasures against side-channel leakage to be tested in a short amount of time and without expensive equipment. Once the implementation is finished real side-channel data should still be collected to confirm that there is no information leaking. Even though the expensive equipment is still needed, the process is now much faster and thus cheaper.

With the current developments of quantum computers, it is still expected to take several years before a large functioning quantum computer exists. Current public key cryptosystems based on following problem "What is a given that $g^a \mod p$, where p is a prime?", which is called the discrete logarithm problem. Or are based on the problem of factoring large numbers. Both problems can be solved in polynomial time with a quantum computer using Shor's algorithm [270]. Symmetric key algorithms only need to increase their key length to withstand the computational power of quantum computers. So encrypted messages that are currently transmitted could

be stored until then to be decrypted with the help of a quantum computer. As a result a lot of research is currently done on post-quantum cryptography. The National Institute of Standard and Technology (NIST), has a competition running to standardize a portfolio of post-quantum cryptographic algorithms. The competition consists of several rounds, out of the 69 submissions in round one, 15 submissions are left in the current third round. The result of the competition will be a portfolio of algorithms for public key encryption and key-establishment, and for digital signatures. The submitted algorithms use different mathematical problems as a base for the cryptographic security with each their own advantages and disadvantages, such a (public) key length, cipher text length, computation time, etc. While post-quantum cryptography is out of scope for this thesis, but because there is a lot of work done on quantum computers and as a result also on post-quantum cryptography, we briefly mention it here.

1.1 Organization of this Thesis and Contributions

Background (Chapter 2) This chapter provides the background for the rest of this thesis. First we introduce cryptographic algorithms and implementations. In the second part, we introduce elliptic curve cryptography. There is a description of the different building blocks used in elliptic curve cryptography, such as point representations, scalar multiplication and different elliptic curves. Primarily, we discuss how different digital signature schemes that are based on elliptic curve cryptography work. The chapter also introduces symmetric key cryptography, where we describe different primitives, in particular permutations. Finally, we provide an overview of different implementation aspects. This section focuses on side-channel analysis, leakage assessment and countermeasures. Additionally, we also introduce different fault-injection attacks.

Physical Attacks on Symmetric Cryptography (Chapter 3) This chapter describes a side-channel attack on Keccak and Ascon. Ascon is an authenticated encryption scheme that was a candidate in the CAESAR competition for authenticated encryption schemes and used in the final portfolio for lightweight applications. The Keccak algorithm is the winner of the SHA-3 competition and is now standardized. The side-channel attack on Keccak was a practical execution of a theoretical description of an attack on a hardware implementation in an earlier paper. The side-channel attack on Ascon was also a practical execution of the attack on a hardware implementation. At the time of writing, the side-channel attack was a new application to Ascon. For both attacks the cipher was running on an FPGA and the power was measured using an oscilloscope. In this work, I implemented both ciphers for the side-channel attacks and ported them to the FPGA platform that was used to collect the traces. Additionally, I collected the traces and executed the practical experiments and helped writing the paper.

Niels Samwel and Joan Daemen. "DPA on hardware implementations of Ascon and Keyak". In: *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*. ACM, 2017, pp. 415–424. DOI: 10.1145/3075564.3079067

The second publication presents a symmetric cryptographic primitive FRIET that has fault detection built-in the design of the permutation. The work describes the design of the primitive and the security of the primitive with modes of operation. In this chapter, we provide a practical evaluation of the fault resistance by executing an electromagnetic fault injection attack. In this work, I executed the electromagnetic fault analysis on FRIET that was implemented on a Cortex-M4. Additionally, I was involved with discussions on the countermeasure and wrote part of the paper.

Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. "Friet: An Authenticated Encryption Scheme with Built-in Fault Detection". In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I.* ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 581–611. DOI: 10.1007/978-3-030-45721-1_21

Physical attacks on Real-World Implementations (Chapter 4) This chapter presents a side-channel attack and a fault injection attack on EdDSA implemented on a Cortex-M4. First, the chapter explores the design of EdDSA and determines what part is vulnerable to a side-channel attack. Next, the different components of the side-channel attack are explained. Finally, for the side-channel attack, the experimental results are presented. In this work, I implemented and executed the side-channel attack on Ed25519, an instance of EdDSA that was running on an ARM Cortex-M4 board. Additionally, I actively participated in the discussion regarding the suggested countermeasure and wrote part of the paper.

Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. "Breaking Ed25519 in WolfSSL". in: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings.* Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 1–20. DOI: 10.1007/978-3-319-76953-0_1

The second publication in this chapter presents a fault injection attack on the same signature scheme EdDSA. The second part of the chapter starts with explaining the attack and continues with the experimental results. Finally, this chapter present a countermeasure that thwarts both the side-channel attack and the fault-injection attack. In this work, I implemented the fault injection attack and did the experimental analysis by applying both voltage fault injection and electromagnetic fault injection techniques. I also wrote part of the paper.

Niels Samwel and Lejla Batina. "Practical Fault Injection on Deterministic Signatures: The Case of EdDSA". in: *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May*

7-9, 2018, Proceedings. Ed. by Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 10831. Lecture Notes in Computer Science. Springer, 2018, pp. 306–321. DOI: 10.1007/978-3-319-89339-6_17

Systematic Evaluation of Countermeasures for Curve25519 Cryptographic Implementations (Chapter 5) This chapter presents an optimized implementation of X25519 for the ARM Cortex-M4 that covers three levels of protection against side-channel and fault injection attacks. The first level offers no protection. The second level of protection includes countermeasures against side-channel and fault injection attack in case an ephemeral key is used. The third level uses a static key and thus requires the highest level of protection. The chapter introduces all the countermeasures that are implemented and explains why they are necessary. The implementations are evaluated using TVLA to determine the effectiveness of the implemented countermeasures. Additionally, their performance is compared to show the trade-offs that must be made for a protected implementation. In this work, I was involved with the discussion on the selection of the different countermeasures. I was also involved with the implementation and evaluation of the different levels of side-channel protection.

Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. *SCA-secure ECC in software – mission impossible?* Cryptology ePrint Archive, Report 2021/1003. https://ia.cr/2021/1003. 2021

Faults and Genetic Algorithms (Chapter 6) This chapter presents a technique to increase the number of successful with a limited number of injected glitches using a genetic algorithm. First, the chapter provides an introduction to genetic algorithms and how they are used for optimization problems. Next, the chapter discusses the experimental setup, which uses electromagnetic fault injection and lists the different parameters that result in a massive search space that must be optimized. Given the search space, the genetic algorithm that is used for this optimization problem is explained. Finally, the results are presented with a practical example for an attack on SHA-3. In this work, I was involved with the implementation of the target and the fault-injection attack. I was also involved with creating the setup, collecting and analyzing the data, and wrote part of the paper.

Antun Maldini, Niels Samwel, Stjepan Picek, and Lejla Batina. "Genetic Algorithm-Based Electromagnetic Fault Injection". In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018*. IEEE Computer Society, 2018, pp. 35–42. DOI: 10.1109/FDTC.2018.00014

Automatically Fixing Side-Channel Leakage With the Use of Simulators (Chapter 7) This chapter presents a tool that automatically eliminates side-channel leakage of masked implementations. Additionally, the chapter presents an extension

of that tool that also works with second order masked implementations. The chapter introduces the simulator that used to emulate the side-channel leakage and the methods that are used to assess the leakage. The tool rewrites assembly code that is generated by a compiler. The chapter explains how the assembly is rewritten. To evaluate the effectiveness of the tool, real traces are acquired and the same leakage assessment methods are applied. Finally, the chapter discusses some of the limitations that are the result of using simulators to detect leakage. In both publications, I was involved with the creating the masked implementations that were used. Additionally, I was involved with the setup and the analysis of the real traces.

Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. "Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021.* The Internet Society, 2021

Madura A. Shelton, Lukasz Chmielewski, Niels Samwel, Markus Wagner, Lejla Batina, and Yuval Yarom. "Rosita++: Automatic Higher-Order Leakage Elimination from Cryptographic Code". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021.* Ed. by Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi. ACM, 2021, pp. 685–699. DOI: 10.1145/3460120.3485380

Conclusion and Discussion This chapter concludes the thesis by providing a summary of the most important results. Additionally, the chapter provides future research directions that follow from work in this thesis. Finally, there is a discussion on the results of this thesis in relation to related work.

Chapter 2

Background

In this chapter we introduce some basic concepts of cryptography that we use throughout the rest of this thesis. First, we describe different cryptographic algorithms and their uses. Next, we introduce elliptic curve cryptography (ECC) and symmetric key cryptography. Hash functions from symmetric key cryptography together with ECC are used in digital signatures, we describe several signature generation and verification algorithms. Finally, we discuss implementation aspects where we describe different side-channel attacks and countermeasures.

2.1 Cryptographic Algorithms and Implementations

Cryptographic algorithms are used in numerous applications and services, such as encrypted communication over the internet with other people using Skype or Zoom but also with organizations like banks or the government. Recently, also personal identification methods, such as passports, ID cards and drivers licenses are equipped with RFID chips that use cryptography to prove their authenticity. A common distinction in cryptography is symmetric cryptography and asymmetric cryptography. Typically, a combination of the two is used in different applications such as encryption or authentication, for example in protocols such as Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

Authentication is the process of recognizing a user's identity. It can be accomplished using both symmetric or asymmetric cryptographic signatures to suit different security requirements. In symmetric cryptography, a secret key is used and only parties that know the secret key are able to verify the signature. A frequently used symmetric signature scheme is the HMAC algorithm. However this creates the problem that both parties need to obtain the same key. This can be done using a secure channel, but this creates another problem of setting up a secure channel, which requires another secret key known to both parties. To counter this, a solution must be used that allows communication about a secret key using a public, insecure channel. The solution is to use asymmetric cryptography. With asymmetric cryptography, a key pair is involved that consists of a private and a public key. Since the public key is assumed to be known by all parties, each party that can read the signature is able to verify the signature. Common asymmetric signature schemes are the Digital Signa-

ture Algorithm (DSA) and the Elliptic Curve Digital Signature Algorithm (ECDSA), see Section 2.4 for more details.

Typically digital signatures are used to sign a message to prove its authenticity and integrity. Since messages often have any length so using a digital signature algorithm with fixed-length parameters is an issue. A solution is to compress the message to a fixed length bit-string and this can be accomplished with a cryptographic hash function. A hash function takes an arbitrary length input and maps it to a fixed length output, typically called the hash value or the digest. The bits are scrambled and distributed evenly over the space. A good cryptographic hash function is efficient to compute, different input values should not map to the same output value and it should be infeasible to determine the input value given a hash value. Well-known hash functions are MD5 [247] and SHA-1 [151], but they are both compromised [297, 279] and should no longer be used. Recommended hash functions for use in cryptography protocols are SHA-2 [238] and SHA-3 [105] which is based on the SHA-3 competition winner Keccak. MD5, SHA-1 and SHA-2 are all based on the Merkle-Damgard construction and SHA-3 is a permutation function.

The building blocks for ciphers exist in different forms and types, the three most common are: block ciphers, stream ciphers and permutations. A more detailed discussion of symmetric key cryptography can be found in Section 2.3. Block ciphers take a plain text and a key as input and produce a cipher text as output. The inputs and outputs have a fixed length. The most famous and widely used block cipher is AES [96]. Other notable block ciphers are DES [278], 3DES and RC5 [248], although these are not recommended to be used any longer as they are not considered secure anymore. In a stream cipher, the plain text is combined with a pseudorandom keystream which is derived from the key. Each plain text byte is encrypted one at a time with the corresponding digit from the keystream to form the resulting cipher text stream. The key is typically of fixed length and the plain text is of arbitrary length. Commonly used stream ciphers are Salsa20 [44] and Chacha20 [40]. A more recent building block is the permutation, not to be confused with the permutation layer in block ciphers. A permutation has an internal state of fixed size that is permuted in a number of rounds to shuffle and alter the state. The most famous one is the winner of the SHA-3 competition, Keccak [51]. Other notable permutations are Xoodoo [94] and Ascon [102].

A cryptographic primitive by itself is only suitable for the cryptographic transformation of one fixed-length group of bits called a block (in case of a block cipher). A mode of operation describes how to apply a cryptographic primitive repeatedly to securely transform larger amounts of data. Well known modes for block ciphers to encrypt data are, electronic code book (ECB), cipher block chaining (CBC), and counter mode (CTR). Another mode is authenticated encryption with associated data (AEAD), which verifies the integrity of the encrypted message with additional public data. For permutations there also exist a number of modes that are based on two constructions, the sponge [60] and the duplex [57]. Typically, these are used in hashing and encryption mode, respectively. Sponge construction consists of two distinct phases, first an arbitrary amount of data can be input to the state. Next there is the output phase where an arbitrary amount of data can be output. In the duplex construction an arbitrary amount of data can be input and output simultaneously. Both constructions can also be combined for modes, such as AEAD. Including the

previously mentioned modes there are numerous other modes of operation with each their own unique advantages and disadvantages.

2.2 Elliptic Curve Cryptography

In 1978 an asymmetric cryptosystem was introduced by Rivest, Shamir and Adleman (RSA) [249]. It is still used but because of the increasing parameter sizes there is a shift towards a different cryptosystem that is more suitable for embedded devices. Elliptic curve cryptography (ECC) was independently introduced by Koblitz [169] and Miller [204] in 1985. ECC widely gained popularity for asymmetric cryptographic protocols, implemented on embedded devices. Compared to RSA [249], the smaller parameters such as the key length and the lower memory requirements make ECC especially more suitable in this context. The security of elliptic curve cryptography relies on the hardness of the discrete log problem in the prime-order subgroup of points where the operation takes hold.

The main building block of elliptic curve cryptography schemes is the scalar multiplication. There are different scalar multiplication algorithms with different advantages such as constant run-time and speed. A naive left-to-right double and add algorithm is outlined in Algorithm 1. To counter timing attacks [171], the Montgomery ladder [205] or a double and add always [90, 159] algorithms can be used which all execute the scalar multiplication in a constant amount of steps allowing it to run in constant time. There multiple algorithms to optimize for speed, such as a windowed method or changing the scalar to its Non-Adjacent Form (NAF) [142]. Both methods reduce the number of point additions and with that the total number of steps.

Algorithm 1 Left-to-right double and add scalar multiplication

Input: $k = (k_{n-1}, .., k_1, k_0)_2, P \in E(\mathbb{F}_q)$
Output: kP

```
1: Q ← O
2: for i from n − 1 to 0 do
3:     Q ← 2Q
4:     if k[i] == 1 then
5:         Q ← Q + P
6: return Q
```

The arithmetic of elliptic curve cryptography is performed in terms of the underlying field operations that are defined over binary fields or prime fields. In this thesis we only consider prime fields. A prime field $\mathbb{F}_p$ where p is a prime, consists of all integers $\{0, ..., p-1\}$, where the addition, multiplication and the inversion of field elements are performed modulo p.

An elliptic curve $\mathcal{E}_{\mathbb{F}_p}$ over $\mathbb{F}_p$ is defined as the set solutions (x, y) of a curve equation, such as the short Weierstrass equation. For $\mathbb{F}_p$, if $p > 3$ the equation is defined as follows:

$$\mathcal{E}_{\mathbb{F}_p} : y^2 = x^3 + ax + b \tag{2.1}$$

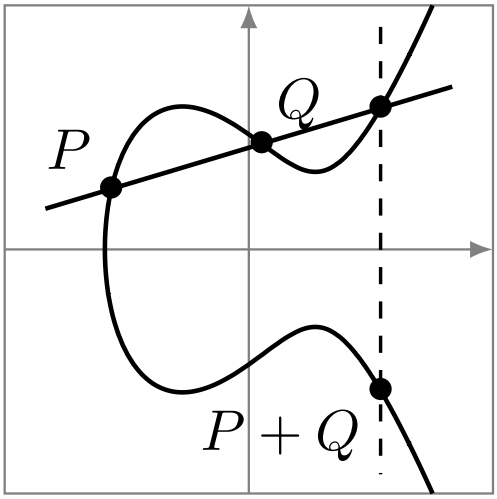


Figure 2.1: Point addition on an elliptic curve.

A solution $\mathbf{P} = (x, y)$ is called a point, represented in affine coordinates. Together with the point at infinity $\mathcal{O}$, the set $(\mathcal{E}_{\mathbb{F}_p} \cup \mathcal{O}, +)$ forms an Abelian group with neutral element $\mathcal{O}$.

Points $\mathbf{P} = (x_0, y_0)$ and $\mathbf{Q} = (x_1, y_1)$ can be added to obtain another point of the curve $\mathbf{P} + \mathbf{Q} = (x_2, y_2)$ using the following formulas

$$\begin{aligned} x_2 &= \lambda^2 - x_0 - x_1, \\ y_2 &= \lambda(x_0 - x_2) - y_0, \end{aligned} \tag{2.2}$$

with $\lambda = \frac{y_0 - y_1}{x_0 - x_1}$ if $\mathbf{P} \neq \pm\mathbf{Q}$ and $\lambda = \frac{3x_0^2 + a}{2y_0}$ if $\mathbf{P} = \pm\mathbf{Q}$. See Figure 2.1 for a visualization of point addition over the real numbers.

Affine point addition only takes a few finite field additions and multiplications but also a costly inversion. To compute point additions without the inversion, several different types of coordinate systems are used, here we list the most common ones:

- Projective coordinates $\mathbf{P} = (X, Y, Z)$, with $x = \frac{X}{Z}, y = \frac{Y}{Z}, Z \neq 0$,
- Jacobian coordinates $\mathbf{P} = (X, Y, Z)$, with $x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}, Z \neq 0$.

These formulas use a small number of additional field multiplications and additions but postpone the costly inversion to the conversion to affine coordinates that usually happens only once at the end of a scalar multiplication. This significantly improves the performance of the group operations.

There are several types of elliptic curves defined by their curve equation. We have already seen the elliptic curve that is defined by the short Weierstrass equation. These elliptic curves are widely used in cryptography but had a major downside because the addition formulas were not complete. Adding a point to itself or doubling a point is not defined by the adding formula. So different formulas had to be used in case

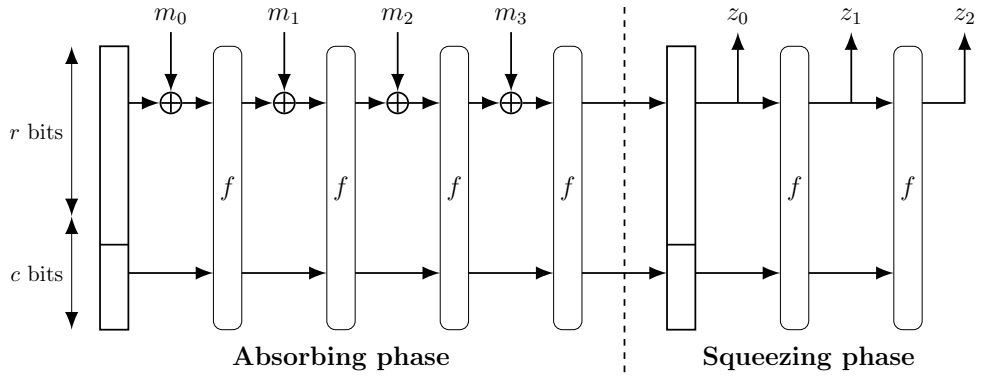


Figure 2.2: Cryptographic sponge with permutation f.

of point doubling, this lead to side-channel leakage such as differences in timing of a scalar multiplication or patterns could be observed in the power consumption. In 2016 complete formulas were introduced [242] to counter these effects at the cost of a slight increase in field operations.

Other commonly used elliptic curves in cryptography are Edwards curves [107] and its generalization, Twisted Edwards curves [45] with the following curve equation:

$$\mathbb{E}_{a,d} : y^2 + ax^2 = 1 + dx^2y^2. \tag{2.3}$$

Additional commonly used curves in cryptography are Montgomery [205] curves and Hessian curves.

Different protocols are based on elliptic curve cryptography such as the Elliptic Curve Diffie-Hellman (ECDH) protocol which is a variation of the Diffie-Hellman key exchange protocol [100], Elliptic Curve Digital Signature Algorithm (ECDSA) [158] to compute digital signatures, and there is Elliptic Curve ElGamal [123] as an encryption/decryption scheme.

This thesis is considering defenses against side-channel attacks for the Elliptic Curve Diffie-Hellman protocol for Curve25519 implemented on an ARM Cortex-M4 in Chapter 5.

2.3 Symmetric Key Cryptography

In this thesis we mostly focus on algorithms and implementations that are based on permutations. Therefore, this section about symmetric key cryptography focuses mostly on permutation based cryptography.

Different use cases, such as hashing, authenticated encryption, pseudorandom number generation or key derivation often require different cryptographic primitives. A single cryptographic permutation is capable of all of these by applying different modes. This is especially useful for resource constrained devices. In Figure 2.2, we see an example of a permutation used as a sponge to compute a cryptographic hash function. In the figure, we have a b-bit permutation f, with $b = r + c$, where c is the

capacity and r is the rate. The rate denotes the number of bits that can be absorbed into the state or squeezed out of the state for each call to the permutation. The capacity is related to the security of the permutation. If f is random it is proven that generic attacks up to $2^{c/2}$ are resisted [60]. The size of the rate and the capacity is flexible as long as they add up to b and allow for a trade-off between efficiency and security. There are two distinct phases, first there is the absorption phase where an arbitrary number of message blocks of size r are absorbed into the state. After each block is absorbed a number of rounds of the permutation is computed. The second phase is the squeezing phase, in this phase an arbitrary number of blocks of length r are squeezed out, after each block a number of rounds of the permutation is computed. The hashed messaged should be padded such that the length is a multiple of r. The internal state is typically initialized with zeros.

Some cryptographic primitives can also be used without a key, in this case it is an unkeyed mode. The sponge is such an unkeyed mode for the permutation that is used for hashing. When a key is absorbed into the state before the padded message blocks, a message authentication code (MAC) can be computed using the output that is squeezed out. This is a keyed mode. The length of the key must be padded to a multiple of r such that it fits in a number of blocks. The sponge can be used to generate a key stream, first a key and optionally a nonce are absorbed, next the key stream is squeezed out that can be used to encrypt a plain text. With a permutation, the generation of a key stream and the computation of a MAC can be done simultaneously. First the padded key and nonce are absorbed, and second a function call to the permutation f is made. The output is used as a key stream to encrypt a block of plain text and at the same time the block of plain text is absorbed into the state after which another call to permutation f is made. This process can repeat itself an arbitrary number of times, to encrypt an arbitrary number of plain text blocks. Once all blocks are encrypted and absorbed, and a final call to permutation f is made and a MAC is produced. This is called authenticated encryption and it uses the duplex [57] construction. Finally, it is also possible to add additional associated data to verify the encrypted cipher text. This is accomplished by absorbing the additional data into the state after the key and nonce absorbed and before the first plain text block is absorbed. The additional data is padded to a multiple of r and the number of blocks can be of arbitrary length. This is authenticated encryption with associated data (AEAD).

In 2015 NIST announced a competition for Lightweight Cryptography. At that time, the majority of cryptographic algorithms were designed for desktop or server environments. With the recently emerging areas such as Internet of Things, cyber-physical systems, sensor networks, etc. Which are highly constrained interconnected devices, a new solution was needed. In 2019 NIST received 57 submissions but after an initial review, 56 were selected as round 1 candidates. Most submissions are based on block ciphers but there are also a number that are based on permutations. Out of the 56 round 1 candidates 32 were selected as round 2 candidates. NIST announced the finalists at the end of 2021. As part of the competition, the submissions have to be implemented, both in hardware and in software so that the performance of the algorithms can be compared and evaluated. The winner of the competition will be standardized and widely adopted into cryptographic libraries. In this thesis, we discuss two submissions to the competition, Ascon [102] and Xoodoo [94].

This thesis describes side-channel attacks and defenses on symmetric cryptography in Chapter 3.

2.4 Elliptic Curve Digital Signatures

In this section, we will discuss digital signatures. Digital signatures use both elliptic curve cryptography and parts of symmetric cryptography, as discussed in the previous sections, will come together in protocols such as the Elliptic Curve Digital Signature Algorithm (ECDSA) [158] and the Edwards-curve Digital Signature Algorithm (EdDSA) [46]. A commonly used instance of EdDSA is Ed25519. These schemes deliver signatures with comparable security to the Digital Signature Algorithm (DSA) [165], but allow for much smaller key pairs and signatures, and short computation times. This led to the wide adoption and standardization of the schemes.

The signature protocols use an elliptic curve $\mathcal{E}_{\mathbb{F}_p}$, a base point $B \in \mathcal{E}_{\mathbb{F}_p}$ with large prime order n.

2.4.1 ECDSA

Signature Generation

To generate a signature one must first generate a key pair as follows:

1. Randomly select a value $d \in [1, n-1]$.
2. Compute scalar multiplication $Q = [d]B$.
3. Return key pair (d, Q), with private key d and public key Q.

The key pair can be reused to generate multiple signatures. Next the signature for message m is generated.

1. Compute $h = H(m)$, where H is a secure cryptographic hash function.
2. Randomly select ephemeral key $k \in [1, n-1]$.
3. Compute scalar multiplication $(x_0, y_0) = [k]B$.
4. Let z be the leftmost $n-1$ bits of h.
5. Compute $r = x_0 \mod n$. If $r = 0$, go back to step 2.
6. Compute $s = k^{-1}(z + rd) \mod n$. If $s = 0$, go back to step 2.
7. Return signature pair (r, s).

Signature Verification

Given a signature pair (r, s), public key Q and message m, the signature is verified as follows:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is not valid.

2. Compute $h = H(m)$, where H is the same hash function that was used during signature generation.
3. Let z be the leftmost $n - 1$ bits of h.
4. Compute $u_0 = zs^{-1} \mod n$ and $u_1 = rs^{-1} \mod n$.
5. Compute $(x_0, y_0) = [u_0]B + [u_1]Q$. If $(x_0, y_0) = \mathcal{O}$, the signature is invalid.
6. The signature is valid if $r \equiv x_0 \mod n$.

2.4.2 EdDSA

Signature Generation

Edwards-curve Digital Signatures are based on Schnorr signatures [262] and Twisted-Edwards Curves [45] and were designed to be faster than other signature schemes, while keeping the same level of security. EdDSA uses an elliptic curve $\mathcal{E}_{\mathbb{F}_p}$, with order $2^c l$, where l is a large prime and 2^c is the cofactor. Additionally there is a base point $B \in \mathcal{E}_{\mathbb{F}_p}$ with prime order l.

To generate an EdDSA signature one must generate a key pair that can be reused for multiple signatures, given a private key k and base point B.

1. Compute $(h_0, h_1, ..., h_{2b-1}) = H(k)$, where H is a secure cryptographic hash function and b is the input length in bits.
2. Let $a = (h_0, ..., h_{b-1})$.
3. Let $b = (h_b, ..., h_{2b-1})$.
4. Compute public key $A = [a]B$.

Next the signature for message m is generated.

1. Compute ephemeral private key $r = H(b, m)$.
2. Compute ephemeral public key $R = [r]B$.
3. Compute $h = H(R, A, m)$ and convert to integer.
4. Compute $S = (r + ha) \mod l$.
5. Return signature pair (R, S).

Signature Verification

Given signature (R, S), public key A and message m, an EdDSA signature is verified as follows.

1. Compute $h = H(R, A, m)$ and convert to integer.
2. If $2^c SB = 2^c R + 2^c hA$, then accept the signature.

This thesis describes side-channel attacks on elliptic curve digital signatures in Chapter 4.

2.5 Implementation Aspects

In this section we describe a widely used side-channel attack called differential power analysis (DPA) and a protection against this class of attacks, called masking. Additionally, we describe a method how to attack implementations protected using masking.

Cryptographic implementations run on different platforms. Software implementations can be run on large servers or on small microcontrollers, where bottlenecks can be optimized by manually writing assembly code for the specific architecture. Sometimes commonly used operations are offloaded to specific hardware, to be used as a special instruction. When more optimizations are required, the complete cryptographic implementation is made into a hardware implementation and executed on an FPGA or even developed into an ASIC. Implementations that run on servers where resources are abundant, not many tradeoffs are made to achieve high speed or high bandwidth implementations. But with more constrained devices such as small microcontrollers typically found in IoT devices, different tradeoffs are often made. For instance on the memory usage, size of the implementation, speed, power consumption, etc. The constrained devices are also typically accessible by an adversary. Because the implementations cannot be fully focused on defending against attacks due to the constraints, they typically are easier to attack. In this thesis, we discuss several attacks on such constrained devices and present countermeasures that can thwart such attacks.

2.5.1 Passive Attacks

Differential Power Analysis

Differential Power Analysis (DPA) is the statistical analysis of the power consumption of a device. The name DPA is often interchangeably used with Correlation Power Analysis (CPA) [71] and Differential Electromagnetic Analysis (DEMA). We use the statistical analysis to exploit leakage of the power consumption caused by the switching activity of a register, we call this the Hamming Distance model. Additionally, there is also the Hamming Weight model that counts the number of bits that are equal to 1. We attack a value of a specific bit called the sensitive variable. Whether or not the value of the sensitive variable switches causes a difference in the power consumption. We exploit this leakage by correlating the actual power consumption with intermediate values. These values can be computed for each trace using known input and part of the key. We pick a sensitive variable that depends on a small number of key bits such that we can compute the intermediate values for all possible candidates for that part of the key. Instead of the correlation we could also compute the difference of means. These methods are explained in detail in [192]. Many power traces are often necessary to be able to distinguish between the correct and incorrect key candidates due to noise. Since many traces are required we use formulas from [103, 70] to compute the correlation in a memory efficient way. With these formulas we can split up the trace set in different smaller partitions and compute the correlation for each partition of traces separately. In the end, the resulting coefficients can be recombined. These methods also allow for parallel computation to speed up the

process.

Profiled vs. Non-Profiled attacks

An approach like DPA is limited to the assumed leakage model and a selection function that exploits a sensitive variable. They are also univariate, where each sample is being attacked independently. Depending on the leakage model and selection function, a univariate approach might not be sufficient to exploit leakage from a target. In case the assumptions on the leakage model or the sensitive variable are not correct, the attack may fail. In some cases it may prove difficult to determine the correct leakage model or select the leaking sensitive variable. So called profiled attacks or template attacks [80] make no assumptions on a leakage model or a selection function. They are also not limited to a univariate setting and allow multivariate attacks. Template attacks consist of two phases, in the first phase a device is profiled. An attacker typically requires full control over a device and uses it to acquire a number of traces to model the leakage, creating templates. A template is created for a class that is profiled, where the different classes can be the Hamming weight of a byte (9 classes), the value of a bit, certain operations, etc. The second phase is called template matching phase, where a trace is collected from a device under attack that is identical to the device that was used in the first phase. Only, this device is not under control of the attacker. The acquired trace is used to match against all the templates from the different classes and the best match is assumed to be the correct class. Knowledge about what operations occurred or what values are processed at specific moments in the execution of an cryptographic algorithm may lead to the recovery of the secret key. Different methods can be applied during the profiling and matching phase to create and match the templates. In a univariate setting, selecting the point of interest can prove to be difficult but a sufficient method can usually be found. When a multivariate approach is used, the number of points of interest can grow too large, a technique called Principle Component Analysis (PCA) [275] can be used to reduce the number of dimensions which reduces the complexity. More recently machine learning, especially deep learning [37] has seen a huge increase in interest for profiled attacks.

Leakage Assessment

Leakage exploited by DPA or template attacks of different cryptographic implementations can be evaluated using Test Vector Leakage Assessment (TVLA) [134]. TVLA uses statistical hypothesis testing to detect if a sensitive variable is leaking through the measured data. TVLA consists of two phases, the first is a measurement phase where side-channel traces are collected with specific inputs. The second is the analysis phase, where the Welch's t-test is used to test whether or not the null-hypothesis can be rejected. The measured trace set usually consists of two partitions, group A is collected with a fixed key and random input. Traces for group B are collected with the same fixed key, but input is fixed with some possible constraints like a sensitive variable with a low Hamming weight. The described grouping method is called non-specific TVLA, but there are other methods that target specific sensitive

variables [33]. The t-values are computed using the following formula:

$$t = \frac{|\mu_A - \mu_B|}{\sqrt{\frac{\sigma_A^2}{N_A} + \frac{\sigma_B^2}{N_B}}}, \tag{2.4}$$

where μ_x, σ_x^2, and N_x denote respectively the mean, the variance and the number of traces in group x. The equation needs to be computed for each time sample in the trace set. Typically, when the resulting t-values exceed a threshold of 4.5, the two groups are distinguishable from each other at that specific time sample meaning that the implementation leaks. The threshold of 4.5 assumes that only a single test is computed, and when TVLA is conducted on a large trace set, it becomes statistically likely that this threshold is exceeded even though the two groups are not distinguishable. Therefore, a method is described in [101] to compute a new threshold to overcome this issue. In Chapter 5 and Chapter 7, we use TVLA to evaluate the side-channel security of different cryptographic implementations.

Countermeasures

In applications such as IoT, designers of devices that perform cryptographic operations must protect their designs against such attacks. This can be accomplished on different levels in the architecture of a cipher, for instance in the protocol where the usage of a single key can be limited or in the actual design of the implementation of the algorithm. There are several techniques for protecting the design on implementation level. One is masking [236, 79] where the data is masked and split up into multiple shares. The algorithm is computed separately on both shares. At the end of the computation the shares are combined to obtain the correct output. The computations on the data and the mask can be done in parallel. A specific type of masking is a threshold implementation [65] where more than two shares are used. Masking techniques make it harder to correlate the intermediate value with the power consumption as the power consumption is no longer correlated to the data that is processed. A more detailed explanation about threshold implementations is given in section 3.4.3.

Higher-Order Attacks

To attack masked implementations, DPA such as described earlier in this thesis are likely not adequate. These attacks fall into the category of first-order attacks. To attack implementations with multiple shares that is implemented correctly, higher-order attacks need to be used [229, 277]. A second-order attack is similar to a difference of means attack except for the distinguisher. Instead of the mean, the variance is used which is the average of the squared distance to the mean of all the samples in a distribution. The variance of a distribution X is defined as follows:

$$Var(X) = E(X - \mu)^2,$$

where $E()$ is the expected value and μ is the mean of distribution X. Other names for the mean and the variance are respectively the first and second central moment.

When an implementation uses three shares then no first or second order leakage should be present. Then a third-order attack needs to be used. The attack is again similar to difference of means except we use the difference of the skewness [79] instead of the difference of means. The skewness determines if a distribution is symmetrical on the mean or leans more to the left or to the right. The skewness of distribution X is defined as follows:

$$Skew(X) = \frac{E(X-\mu)^3}{\sigma^3}.$$

From the previous formula the $E(X-\mu)^3$ is also called the third central moment and the skewness is the third standardized moment.

Higher-order attacks typically require a high amount of traces, because more noise is present. To compute the difference of skewness in a memory efficient way we use formulas from [228]. With these formulas we can compute the skewness for small partitions and combine them in the end. Doing this reduces the usage of RAM and allows for parallel computation of the results.

2.5.2 Active Attacks

Fault injection attacks are active attacks and aim at exploiting the leakage of sensitive information due to some irregular conditions i.e. faulty computation. This is distinctive to side-channel attacks that observe signals while the device under attack is working "normally". With fault attacks, an attacker attempts to alter environmental conditions so the device changes its behavior. One way to accomplish this is by "glitching" the device, i.e., forcing the changes in the values of relevant physical parameters outside the prescribed intervals. There are several approaches to accomplish this as follows:

- **Clock fault injection** [7]. In this case a glitch is caused by altering the clock signal. This is typically done with devices that allow the use of an external clock.
- **Voltage fault injection** [17]. The attacker can induce this kind of glitch by adding a short positive or negative spike in the power line.
- **Electromagnetic fault injection** [291]. A glitch is caused by emitting a short electromagnetic (EM) pulse towards the device resulting in similar effects as voltage glitching.
- **Optical fault injection** [273]. Optical fault injection uses a laser and is more invasive as the chip typically has to be decapsulated and it is possible to cause permanent damage to a device.

Fault injection requires a precise trigger, to clock frequency is usually high and the interval where to inject the fault is very short. In contrast to passive attacks where post processing can be applied to align the traces, this is not possible for active attacks. In this thesis we focus on glitches caused by voltage and electromagnetic fault injection. If a glitch has an effect that alters the behavior of the device such that it produces a faulty output that does prevent the device from producing an output, we call this "a successful fault".

Fault Model

A fault model describes the kind and the extent of faults an attacker is able to induce while the device is operating. In this thesis our target platform is a micro-controller with a 32-bit architecture so we assume a fault model where a glitch can create an error such that the value of a 32-bit word is modified. This alteration can happen at different stages, for instance when the value is processed by the CPU or when the value is on the memory bus. Typically, a glitch could also alter instructions that affect other sensitive values (e.g. loop counters etc.) or other behavior of the algorithm but our attacks do not rely on this particular assumption.

Differential Fault Analysis

Differential fault analysis (DFA) is a special attack based on faults produced during computation. Typically, the attacker uses the difference between the correct output and one (or more) faulty outputs to recover secret data. This can lead to the total key recovery like in the case of Bellcore attacks on the RSA cryptosystem [69] or merely to forging a signature.

Voltage Fault Injection

Fig. 2.3 shows a schematic overview of a voltage fault injection setup. The PC handles communication with the target, collects traces from the oscilloscope and controls the VC Glitcher. The target is powered by the VC Glitcher so it is able to inject glitches in the power line. The glitch amplifier increases the current and with the current probe we are able to measure the power consumption and see the effects of the glitch. The oscilloscope and the current probe are only used to collect an overview trace and determine an offset to induce the glitch. Once a suitable offset is located, the current probe and oscilloscope are disconnected and removed from the setup.

Once we identify this offset we try to improve the success rate and we continuously under-power the device by actively inducing faults. To do this we introduce a glitch after a trigger event occurs. We set a trigger at the start of the scalar multiplication in the signature generation.

Electromagnetic Fault Injection

Electromagnetic fault injection (EMFI) is an active attack where the attacker emits a short EM pulse as a glitch from a close distance. If the glitch is strong enough, it can cause a fault. The EM pulse is emitted using a small coil. Different coils could have different effects on the size and the polarity of the EM pulse, but this point is not relevant for our work and we use a single coil.

As with voltage fault injection, there exist several parameters to be optimized for EMFI. Those are also different parameters and the most distinctive ones are the x and y coordinates corresponding to a location on the chip where the coil that emits the EM pulse is positioned. Figure 2.4 shows an overview of the setup. We use the XY-table to precisely position the EMFI probe on the device. With the XY-table we are able to automate a systematic scan of the chip's surface to find a location with the highest success rate of the attack.

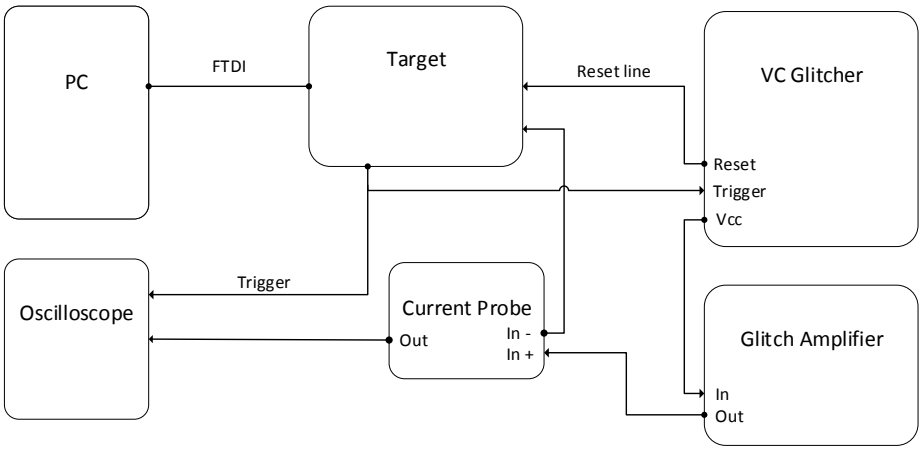


Figure 2.3: Voltage fault injection setup

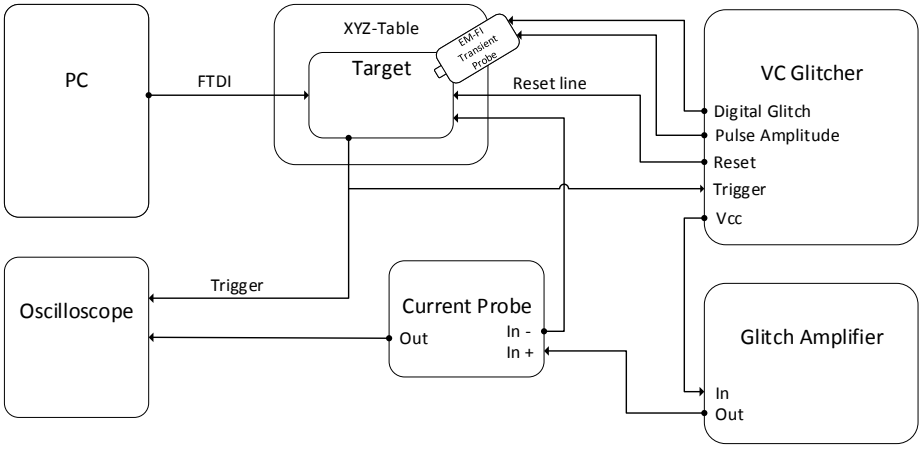


Figure 2.4: EMFI setup

This thesis is considering fault injection with memetic algorithms in Chapter 6 and differential fault analysis in Chapter 3.

Chapter 3

Physical Attacks on Symmetric Cryptography

3.1 Introduction

Side-channel analysis on cryptographic implementations have been known for quite a while [172]. Implementations can leak through many different channels, one of them is the amount of power that is consumed during cryptographic operations performed on a device. A technique to exploit this leakage is differential power analysis (DPA). DPA uses the difference of means to distinguish between correct and incorrect key candidates. Other distinguishers are also used, a typical one uses the Pearson correlation and is called correlation power analysis (CPA) [71]. In this chapter we apply DPA and CPA on two authenticated encryption schemes, namely Keyak and Ascon. The ciphers are candidates in the Competition for Authenticated Encryption: Security, Applicability, and Robustness or in short CAESAR [2], which is a competition for an authenticated encryption scheme. Both of these schemes are based on the duplex construction [59] and apply an iterated permutation on a state. The round functions of these permutations contain a non-linear step which is similar in both ciphers. We present attacks on straightforward hardware implementations and apply them. A countermeasure against DPA and CPA is a threshold implementation. We also present a higher-order DPA attack on a threshold implementation of Ascon.

Another physical attack is called fault analysis (FA). In contrast to DPA and CPA, FA is an active attack where the attacker alters the environment of a chip to affect the computation. A commonly used technique to inject faults is called: electro-magnetic fault injection. This technique emits electro-magnetic pulses from a short distance to the device at very specific times to cause an internal effect in the device that alters the result of the computation. When this technique is applied to cryptographic ciphers, this modified result could leak information about the cipher, such as the secret key. Friet is a permutation that has built-in resistance against such fault attacks. In this chapter, we apply electro-magnetic fault injection on Friet to evaluate the effectiveness of the countermeasure.

3.1.1 Related Work

Related work to the attack on Keyak is work on MAC-Keccak [282, 185]. In these papers the authors focus the attack on the linear part of Keccak, while in this work we focus on the non-linear part of the Keccak permutation.

In Bertoni et al. [54] the authors present an attack on the internal state of Keccak. They simulate traces to determine the effectiveness of the attack. They extend the attack for a threshold implementation that is secure against first order side-channel attacks and present a model to simulate traces. In this chapter, we apply this attack on traces collected using an FPGA.

Previous work related to Friet are proposing certain modifications to cryptographic algorithms to defend against side channel and fault attacks. Intra-Instruction Redundancy [227] and Internal Redundancy Countermeasure [177] are generic countermeasures that can be applied to any cipher and they imply interleaving k copies of the plaintext with some fixed data. While the method can detect up to k faults, it is also quite expensive.

Some other approaches aim at combining resistance against both fault and side channel attacks. Schneider et al. [261] introduce a countermeasure for cryptographic hardware implementations that combines the concept of threshold implementation with an error detecting approach. Similarly to this, Reparaz et al. [244] propose a countermeasure that claims security against higher-order SCA, multiple-shot DFA, and also combined attacks.

Craft [35] is a cipher designed to be used in conjunction with various linear codes which aims at implementations resistant against fault attacks. Craft differs from the approaches mentioned above because the technique is not applied to existing ciphers as an add-on, but takes into account fault attack resistance in the design phase.

3.1.2 Contributions

We summarize the contributions of this chapter as follows:

- We apply the attack from [54] on Keyak implemented on an FPGA.
- We craft an attack which uses a larger sensitive variable to increase the efficiency compared to the basic attack and apply it.
- We craft an attack for Ascon and apply it on an FPGA implementation.
- We simulate traces for a toy version of Ascon protected by a threshold implementation and attack it using a third-order distinguisher.
- We provide insight in the effect of the linear step of a permutation in DPA.
- We provide an experimental electro-magnetic fault analysis of the fault resistant cipher Friet-PC implemented on a Cortex-M4.

3.1.3 Organization of This Chapter

The rest of this chapter is organized as follows. First we describe the experimental setup used for side-channel analysis in Section 3.2. Next, in Section 3.3 we describe

the algorithm Keyak, then we describe the attack from [54] to recover single bits of the internal state and provide experimental results using traces collected from a hardware implementation that runs on an FPGA. Additionally in that section, we describe a more optimized attack that simultaneously recovers a complete row from the internal state and provide the experimental results.

In Section 3.4, we first provide an algorithm description of Ascon, next we introduce an attack on the internal state of Ascon. We apply this attack on traces from a hardware implementation that runs on an FPGA and provide the experimental results. Additionally, we provide a short introduction to threshold implementations and extend the attack to a threshold implementation of Ascon. Finally in this section, we simulate traces for Ascon and provide the results.

In Section 3.5, we evaluate the effectiveness of the fault resistant permutation Friet-P. We start with a description of both the Friet-PC and the Friet-P permutations. Then, we describe the experimental setup and we provide our results.

Finally in Section 3.6, we conclude this chapter.

3.2 Experimental Setup

The experiments are executed on an FPGA, a Spartan-6 XC6SLX75. The implementations for Keccak-f[1600] and Ascon-128 are written in VHDL. In both permutations one round is computed in a single clock cycle. The computations are done in the combinatorial logic of the FPGA. The registers are updated at the end of a round. Both implementations are adapted to run on the SAKURA-G evaluation board. This board contains an additional FPGA, a Spartan 6 XC6SLX9 which controls the communications between the main FPGA and a PC. The PC is connected to the FPGA through USB using the FTDI interface. To measure the power consumption of the main FPGA the SAKURA-G already has circuitry implemented. There is a connector to measure the raw signal and one where the signal is amplified. We use the amplified signal to measure the power consumption for the experiments.

The oscilloscope we use is a Lecroy Waverunner z610i. It is triggered at the start of each encryption run. The trigger is provided by an I/O pin on the SAKURA-G. The trigger is very stable, as a result the traces are perfectly aligned. The internal clock in both FPGA's of the SAKURA-G run at 48MHz. The internal amplifier has a bandwidth of 360MHz. We used a sampling rate of 500 million samples per second. Figure 3.1 shows an overview of the setup.

3.3 Keyak

Keyak [56] is a cipher which is a candidate in the CAESAR [2] competition.

Keyak is based on the Motorist authenticated encryption mode defined in [56]. A Motorist consists of several layers. The top layer is the Motorist itself. Below that is the Engine and in the bottom layer are the Pistons. A Piston contains a b-bit state and applies the permutation f to it. The Engine layer controls the pistons. Pistons work in parallel. An Engine contains at least one Piston. The Engine keeps the Pistons busy and ensures correct use of the input and output for the Pistons. The

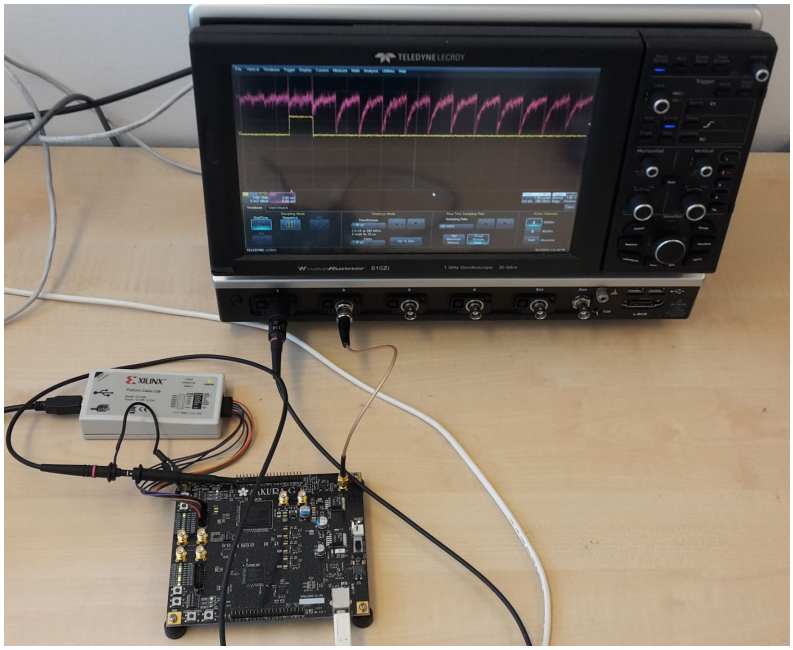


Figure 3.1: A photo of the setup for capturing traces with the SAKURA-G

Motorist layer implements the user interface. It starts a session to encrypt messages or decrypt cryptograms by using the Engine.

In this chapter we only look at Lake Keyak obtained from the recommended list of parameters. From this point on, when we refer to Keyak we mean Lake Keyak. This instance of Keyak has only a single piston which means there is only a single core of Keccak-p. The Keccak-p permutations are derived from the Keccak-f permutations [51]. Keccak-$p[b, n_r]$ is defined by its width b and its number of rounds n_r. Keccak-$p[b, n_r]$ applies the last n_r rounds of Keccak-$f[b]$. In the case of Lake Keyak $b = 1600$ and $n_r = 12$. The permutation Keccak-$p[b, n_r]$ is described as a sequence of operations on a state a that is a three-dimensional array of elements of GF(2), namely $a[5, 5, 64]$ in case of Lake Keyak. Coordinates x and y should be taken modulo 5 and coordinate z should be taken modulo 64. Sometimes an index is omitted implying the statement is valid for all values of the omitted indices. Keccak-$p[b, n_r]$ is an iterated permutation over n_r rounds of R.

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

$$\begin{aligned}
\theta: \quad & C_{(x)} = \sum_{y=0}^{4} A_{(x,y)} \\
& D_{(x)} = C_{(x-1)} + rot(C_{(x+1)}, 1) \\
& A_{(x,y)} = A_{(x,y)} + D_{(x)} \\
\pi \text{ and } \rho: \quad & B_{(y,2x+3y)} = \mathrm{rot}(A_{(x,y)}, r(x,y)) \\
\chi: \quad & A_{(x,y)} = B_{(x,y)} + (B_{(x+1,y)} \times B_{(x+2,y)} + 1) \\
\iota: \quad & A_{(0,0)} = A_{(0,0)} + RC
\end{aligned}$$

All operations are carried out in GF(2). Function $\text{rot}(W, i)$ is a bitwise cyclic shift operation. The constants $r(x, y)$ are rotation offsets. RC is the round constant. For more info see [51]. We refer to the linear steps as $\lambda = \pi \circ \rho \circ \theta$.

To start a session a fixed length key and arbitrary length nonce (in CAESAR also known as the public message number) is absorbed into the state by the Piston. The whole state is used except for one 64-bit lane. In our case we use a 40-byte keypack which contains a padded secret key and a 344-byte nonce. Together that is two complete blocks except for two lanes. The first block containing the 40-byte keypack together with the first 152 bytes of the nonce are absorbed. Following up 12 rounds of Keccak-p are applied. Next the remaining block which contains 192 bytes of the nonce are absorbed and again 12 rounds of Keccak-p are applied. Finally the plaintext can be absorbed, this phase is not used for the attack, so for more details we refer to the Keyak reference document [56]. In both attacks on the internal state of the Piston, where from this point we refer to as the Keyak state, the first block is kept at a constant value. The second block is distinct each time. In our experiments we use random values for the second block. We attack the first round after the second block is absorbed into the state.

3.3.1 Attack on a single bit of Keyak state

In this attack, the sensitive variable is a single bit of the 1600-bit Keyak state. The attack on this sensitive variable is equivalent to the attack on the simulation in [54] where the round function of Keccak-p is attacked. At the previously described point, it is possible to compute the sensitive variable which is a function of the bits of the secret state and bits of the second block of the nonce. Once we know the complete secret state, we can apply the inverse of λ on the bits of the state we found. We can reconstruct the key by applying the inverse of the permutation. Or we can use the state and apply the algorithm from this point. We call these bits, bits from the key state. The variable nonce is called M. Next we compute the sensitive variable using the selection function.

The non-linear step χ of Keccak-p is defined as follows,

$$\chi(a_{(x,y,z)}) \leftarrow a_{(x,y,z)} + (a_{(x+1,y,z)} + 1)a_{(x+2,y,z)}.$$

The linear part can be computed separately for different data like the input and the key state after the absorption of the key and combined later before χ,

$$\chi(\lambda(k + m)) = \chi(\lambda(k) + \lambda(m)).$$

This way the input of χ can be split up into bits from the key state and bits from the message. We compute the sensitive variable as follows,

$$\chi(a_0) \leftarrow k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2).$$

Where m_* are bits of $\lambda(M)$ and k_* are the bits of $\lambda(keystate)$. We are interested in the activity d of the register where the bit is stored. The values depend on the

previous value a_0 of the register which is unknown as it depends on the key,

$$\begin{aligned} d &= a_0 + k_0 + m_0 + (k_1 + m_1 + 1)(k_2 + m_2) \\ &= a_0 + k_0 + m_0 + k_1k_2 + m_1k_2 + k_2 + m_2k_1 + m_2m_1 + m_2 \\ &= a_0 + k_0 + (k_1 + 1)k_2 + m_0 + (m_1 + 1)m_2 + m_2k_1 + m_1k_2. \end{aligned}$$

The result of $a_0 + k_0 + (k_1 + 1)k_2$ is equal for each trace and contributes a constant amount to the activity so it can be removed. This results in the following selection function,

$$S(M, K^*) = m_0 + (m_1 + 1)m_2 + m_2k_1^* + m_1k_2^*.$$

The selection function contains two key bits resulting in four key candidates.

3.3.2 Theoretical success probability

We use a power consumption model where we assume we can exploit the difference in the power consumption of a register flipping or keeping its value. This leakage can be modeled using formulas derived in [54].

$$G_h(\sigma^2) = \int_0^\infty \left(erf\left(\frac{t}{\sqrt{2}\sigma}\right)\right)^{h-1} \left(\mathcal{N}_{(-1;\sigma^2)}(t) + \mathcal{N}_{(1;\sigma^2)}(t)\right) dt$$

$$P_{\text{success}} = G_h\left(\frac{b}{|M|}\right) \tag{3.1}$$

Where b is the width of the state, h is the number of key candidates, $|M|$ is the number of traces and $\mathcal{N}_{(\mu,\sigma^2)}(x)$ is a value of a normal distribution with a mean μ and a variance σ^2 at x. The equation uses that the number of traces required to distinguish between two distributions with a certain probability is inversely proportional to the Kullbach-Leibler Divergence. The left part of the equation with the error function presumes no correlation occurs for a hypothesis on an incorrect key candidate. The right part presumes only the switching of the b registers in a state contribute to the noise. Other algorithmic noise can be characterized as additional b'' bits. This means there is a total of $b' = b + b''$ bits that contribute to the algorithmic noise. Noise other than algorithmic noise can also be present but is not considered in this theoretical success probability of a first-order attack on the state.

3.3.3 Result Keyak single bit

We can compute the theoretical success probability using the formula's from Section 3.3.2. Figure 3.2 shows this probability on the y-axis with number of traces on the x-axis where $b = 1600$ and $h = 4$. The figure also shows the result of the attack on a single bit of the Keyak state. When sixty thousand or more traces are used the success probability approaches one. As we can see the result from the attack on the FPGA is not equal to the theory. We need twice as many traces compared to the theoretical success probability. Which means other noise then just noise caused by the switching of the registers is present in the traces. Averaging traces over equal

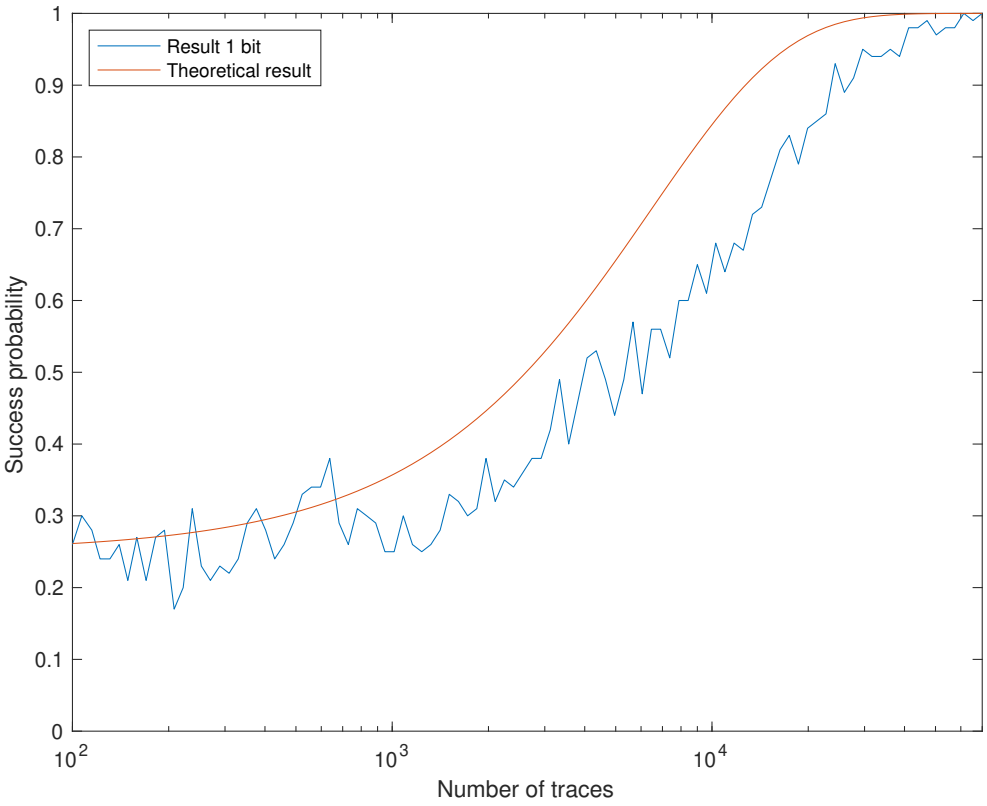


Figure 3.2: Success rate of attack on single bit of the Keyak state

inputs did not reduce the noise much. The greater part of the remaining noise is caused by the algorithmic noise of the combinatorial logic of the FPGA. We go into more detail in Section 3.4.2.

3.3.4 Attack on a row of Keyak state

Using the previous attack we can obtain two correct key bits with a certain probability. To find the key bits of one row using the previous attack we have to guess 5 times 2 key bits, in total this results in 5 key bits. We try to find an attack where we only need to guess 5 key bits which hopefully increases the success probability to make the attack more efficient. With the new attack, we find the value of 5 bits in a row of the key state. To accomplish this we try to correlate the traces with values of the sensitive variable which consists of more than a single bit. A row of the state seems to be an efficient choice as for those five bits we only need to guess five key bits. To create a selection function for a row of five bits of the state of Keyak, it seems like a good idea to extend the previous attack on one bit in the following way. One might think this can easily be accomplished by summing up the resulting bits from

the previous selection function for each bit in the targeted row,

$$\begin{aligned} S(M, K^*) = & \, m_0 \oplus (m_1 \oplus 1)m_2 \oplus m_2 k_1^* \oplus m_1 k_2^* \\ & + m_1 \oplus (m_2 \oplus 1)m_3 \oplus m_3 k_2^* \oplus m_2 k_3^* \\ & + m_2 \oplus (m_3 \oplus 1)m_4 \oplus m_4 k_3^* \oplus m_3 k_4^* \\ & + m_3 \oplus (m_4 \oplus 1)m_0 \oplus m_0 k_4^* \oplus m_4 k_0^* \\ & + m_4 \oplus (m_0 \oplus 1)m_1 \oplus m_1 k_0^* \oplus m_0 k_1^*. \end{aligned}$$

This straightforward approach does not work. If we recall how the selection function was derived we see that the activity d of a sensitive variable depends on its previous value a_0. When one bit is attacked this value is constant and is irrelevant when splitting the set up based on the intermediate value for each key candidate. It is not important how the set is split up, as long as it is split up correct for the correct key candidate which is why it is removed from the selection function. When doing this for five bits at the same time the previous values are still constant but now these values are important. They are unknown so we also need to guess them. To do this we use a different approach.

The first step of the attack is to create 32 bins. Each bin contains the mean trace of all traces with same value of the targeted row after $\lambda(M)$ is computed on the variable part of the nonce.
Next we compute a matrix containing all the values for the sensitive variable. Which is a 32×32 matrix as there are 32 different inputs and key candidates. To do this we exploit the same leakage as in the previous attack on a single bit. For each element in the matrix we compute the output of χ based on the input and the key candidates. With these values we compute the result matrix R with dimensions $K \times T$ where K equals the number of key candidates and T equals the number of samples in each trace. Each element contains the correlation between the values of the sensitive variable for that key candidate and actual power consumption values for that time sample. The row in R with the highest correlation value corresponds to the key candidate with the highest probability to be the correct one.

3.3.5 Result Keyak row

Figure 3.3 shows the result of the attack on a row of the Keyak state. If we compare this result with the result of the attack on a single bit we see this attack is less efficient as the success probability is lower for a similar amount of traces. To compare these results we use the single bit approach where we do the attack five times on each bit of the row independently and view it as a success if all five the attacks result in the correct key hypothesis without taking inconsistent key bit values suggested by neighboring sensitive bits into account. The success probability of the attack on the row is nearly the same as the of the attack on a single bit. This means that both attacks are equivalent. The basic attack on a single bit is already efficient as increasing the size of the sensitive variable does not reduce the number of traces required for the attack.

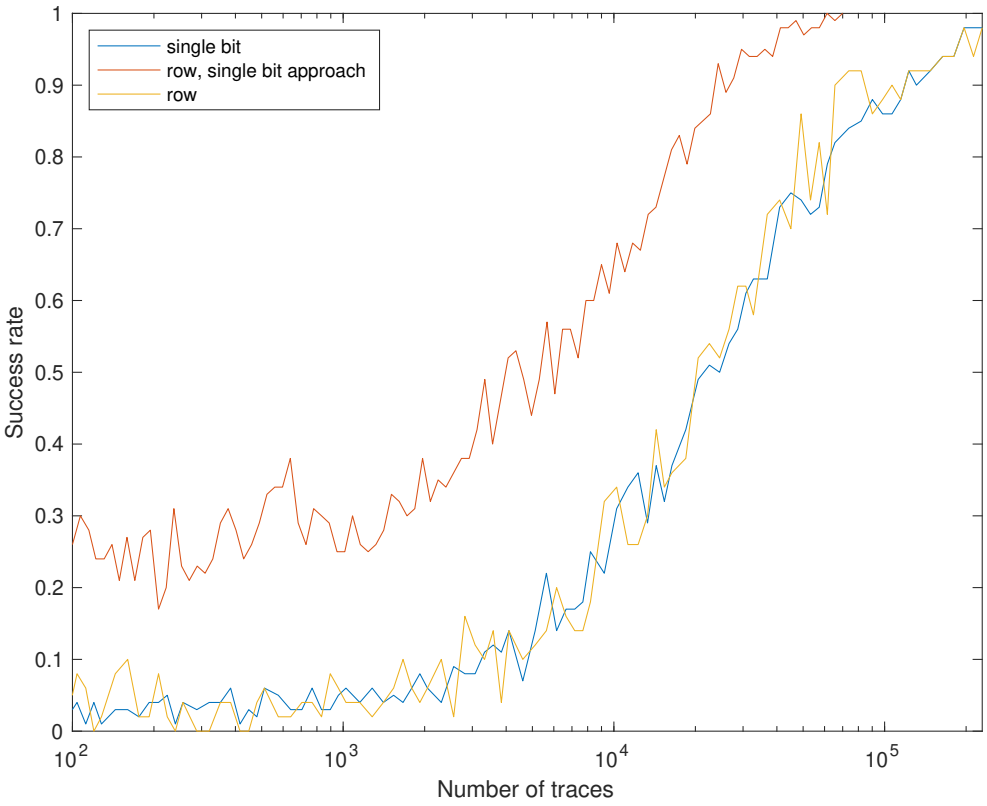


Figure 3.3: Result of attack on row compared with attack on single bit

3.4 Ascon

Like Keyak, Ascon [102] is a CAESAR candidate. From the list of recommended parameters we use Ascon-128, so from this point when we refer to Ascon, we mean Ascon-128. This instance of Ascon uses 128-bit keys and nonces, the block size of the input data is 64-bit.

The authenticated encryption algorithm Ascon uses a duplex construction. Figure 3.4 shows the encryption process of Ascon. It takes four inputs, plaintext P_i, associated data A_i, nonce M and a key K. The block cipher produces two outputs, the ciphertext C_i and a tag T. The permutation is denoted by p^a and p^b. The values $a = 12$ and $b = 8$ equal the number of rounds. The nonce is public. The tag is used during decryption to authenticate the ciphertext. During decryption the algorithm uses the tag as an input and produces the plaintext as output if the tag is valid. If a tag is invalid no output is returned.

The internal state of the algorithm consists of 320 bits which is split up into five registers of 64 bits, named x_0 to x_4. During the initialization of the algorithm, a 64-bit constant IV is written in register x_0. Registers x_1 and x_2 contain the key and the variable nonce is stored in registers x_3 and x_4. The associated data and plaintext are

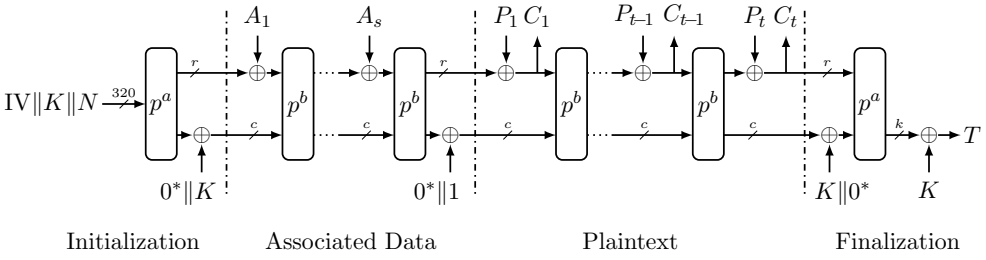


Figure 3.4: Ascon encryption

padded to have a length of a multiple of 64 bits. When the internal state is initialized p^a is applied. Next, the optional associated data is absorbed into the state. After this the plaintext is absorbed into the state. After each block of associated data and plaintext p^b is applied except for the final block of plaintext. During the finalization of the part of the algorithm p^a is again applied and a tag is produced.

The round function or permutation used in Ascon consists of three steps. The first part is the addition of a round constant to register x_2 on low indices which can be computed as follows for each round i,

$$RC_i = \texttt{0xF} - i||\texttt{0x0} + i.$$

Where || means the concatenation of the two parts.

The second part is a non-linear five-bit S-box with algebraic degree two, which takes one bit from each register shown in Figure 3.6(a) and replaces it with the output of the S-box. It is a variant of the S-box used in Keyak. In Figure 3.5 we can see similarities between the S-box of Ascon and Keyak. The designers of Ascon added a few XOR's and a not.

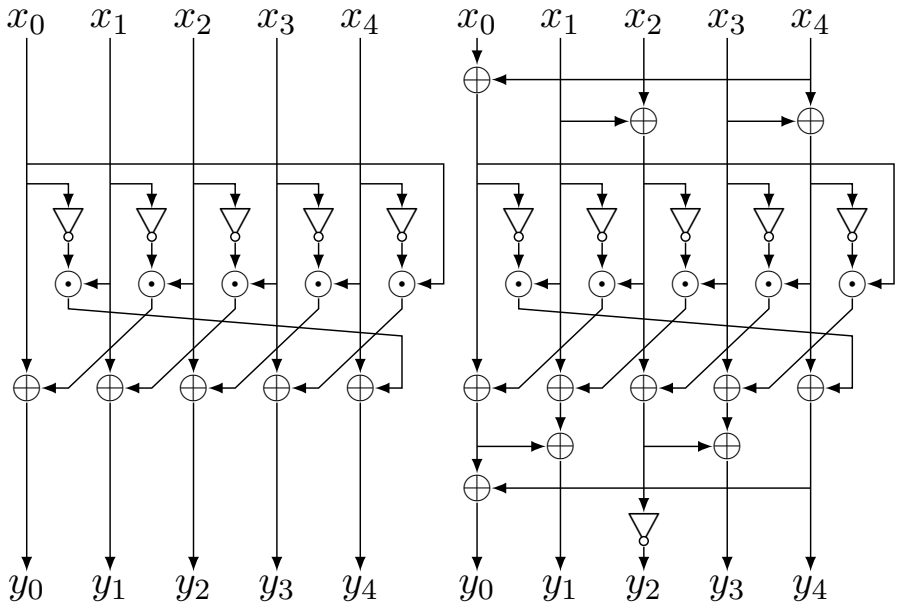
The final step is the linear diffusion layer where each register is rotated twice and XOR'ed with itself which is called $\Sigma_i(x_i)$. Below are the expressions of the linear diffusion layer.

$$\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28)\\
\Sigma_1(x_1) &= x_1 \oplus (x_1 \ggg 61) \oplus (x_1 \ggg 39)\\
\Sigma_2(x_2) &= x_2 \oplus (x_2 \ggg \ 1) \oplus (x_2 \ggg \ 6)\\
\Sigma_3(x_3) &= x_3 \oplus (x_3 \ggg 10) \oplus (x_3 \ggg 17)\\
\Sigma_4(x_4) &= x_4 \oplus (x_4 \ggg \ 7) \oplus (x_4 \ggg 41)
\end{aligned}$$

In permutation Keccak-p the non-linear step is at the end of a round where in the permutation of Ascon it is at the start.

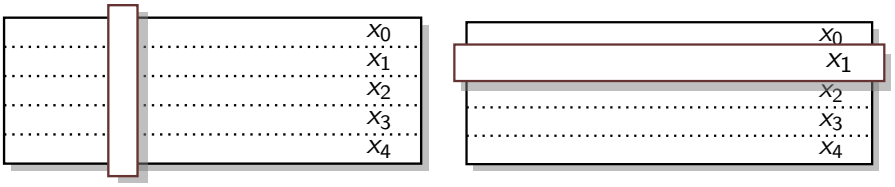
3.4.1 Attack on Ascon

Since we know the contents of the state at initialization, except for the key part. And we can vary the nonce each run, we pick the end of the first round as our point of attack. As sensitive variable we pick a bit of x_0.



(a) The S-box of Keccak-p (b) The S-box of Ascon

Figure 3.5: Comparison between S-boxes



(a) Grouping of bits S-box (b) Input diffusion layer

Figure 3.6: Inputs for the functions of the round function of Ascon [102]

We denote the initialized registers as x_i. In this attack we are only interested in the first round so x'_i denotes the register after the first round. The output of the S-box is specified by y_i.
For the attack intermediate values have to be computed for each different key guess on the nonce. The linear step for x_0 is the following:

$$\Sigma_0(x_0) = x_0 \oplus (x_0 \ggg 19) \oplus (x_0 \ggg 28) \tag{3.2}$$

The output of the non-linear S-box for x_0 can be expressed in the algebraic normal form as follows. Where y_0 is the output and x_i are the input bits.

$$y_0 = x_4x_1 + x_3 + x_2x_1 + x_2 + x_1x_0 + x_1 + x_0$$

This can be rewritten to an expression with a single quadratic term.

$$y_0 = x_1(x_4 + x_2 + x_0 + 1) + x_3 + x_2 + x_0$$

We derive the selection function similar as in [54, Section 3]. We exploit the leakage of the power consumption similar as in the previous attacks. If we rewrite the previous equation.

$$y_0 = x_1(x_4 + 1) + x_1x_2 + x_1x_0 + x_3 + x_2 + x_0$$

All the bits that contribute a constant amount to the activity of the register, namely $x_1x_2 + x_1x_0 + x_2 + x_0$ can be removed.

$$y_0 = x_1(x_4 + 1) + x_3$$

As a result the intermediate value now only depends on one bit from one register of the key and two bits from two registers of the nonce.

$$y_0 = k(m' + 1) + m \tag{3.3}$$

Where $m = x_3, m' = x_4$. If we combine equations (3.2) and (3.3) we get the following selection function for bit 0.

$$\begin{aligned} S_i(M, K^*) = k_0^*(m'_0 + 1) + m_0 + k_1^*(m'_{45} + 1) \\ + m_{45} + k_2^*(m'_{36} + 1) + m_{36} \end{aligned} \tag{3.4}$$

We can generalize this for all bits in the register.

$$\begin{aligned} S_i(M, K^*) = k_0^*(m'_i + 1) + m_i + k_1^*(m'_{i+45} + 1) \\ + m_{i+45} + k_2^*(m'_{i+36} + 1) + m_{i+36} \end{aligned} \tag{3.5}$$

Where the M is the 128-bit nonce split up into two registers m, m' and k_i^* is a bit from a key guess. Additions to index i are done modulo 64 because the bits are located in a 64-bit register. Since there are three key bits in the selection function, there are eight key candidates.

By attacking register x'_0 we obtain only half of the key. To obtain the remaining key bits we need to attack register x'_1. This is the only S-box which has a quadratic

term containing a bit from register x_2. To create a selection function for this register we derive the selection function similar as for x'_0.

In the attack it will not be possible to distinguish the XOR between x_1 and x_2 so their result will be regarded as one term x_{12}.

$$\begin{aligned} S_i(M, K^*) = m_i(k_0^* + 1) + m'_i + m_{i+3}(k_1^* + 1) \\ + m'_{i+3} + m_{i+25}(k_2^* + 1) + m'_{i+25} \end{aligned} \tag{3.6}$$

With the attack on register x'_1 we obtain an XOR between the key bits in register x_1 and x_2. Looking at the remaining output registers of the S-box we have:

$$\begin{aligned} y_2 &= x_4x_3 + x_4 + x_2 + 1 \\ y_3 &= x_4x_0 + x_4 + x_3x_0 + x_3 + x_2 + x_1 + x_0 \\ y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \end{aligned}$$

We can see that y_2 and y_3 have no non-linear terms with a key and a nonce register so they can not be attacked. Register y_4 does have a non-linear term with a key and a nonce register and can be attack. The expression can be rewritten in a similar way.

$$\begin{aligned} y_4 &= x_4x_1 + x_4 + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 + x_1x_0 + x_1 \\ y_4 &= x_4(x_1 + 1) + x_3 \end{aligned}$$

Since we are already able to obtain the whole key by attacking register x'_0 and x'_1 register x'_4 was not attacked.

3.4.2 Results attack on Ascon

Figure 3.7 shows the success probability of the attack on register x'_0 approaching 1 at fifty thousand traces. With this attack we obtain the first half of the key. The success probability of register x'_1 is slightly lower. Using the result from register x'_0 we can obtain the complete key. Looking at the results of the theoretical success probability computed using Section 3.3.2 we observe that it is lower.

If we compare the results from the attack on Keyak with the results from the attack on Ascon we see that it requires less traces to attack Ascon. This is due to the fact that the internal state of Ascon is smaller compared to the internal state of Keyak. When the state in a fully parallel implementation is larger there will be more algorithmic noise which makes it harder to distinguish between correct and incorrect key candidates.

Algorithmic and other noise

In Figure 3.7 we saw that the success probability is lower compared to the theoretical success probability computed using Equation 3.1. This indicates the noise levels are a lot higher in the FPGA that was used compared to the noise of the model. If we compute it for a state with width $b = 1100$ and $h = 8$ we see this is close to the actual result. This means the algorithmic noise $320/1100 \approx 0.29$ which is approximately 29%. This shows the flipping of the registers only makes up 29% of the noise.

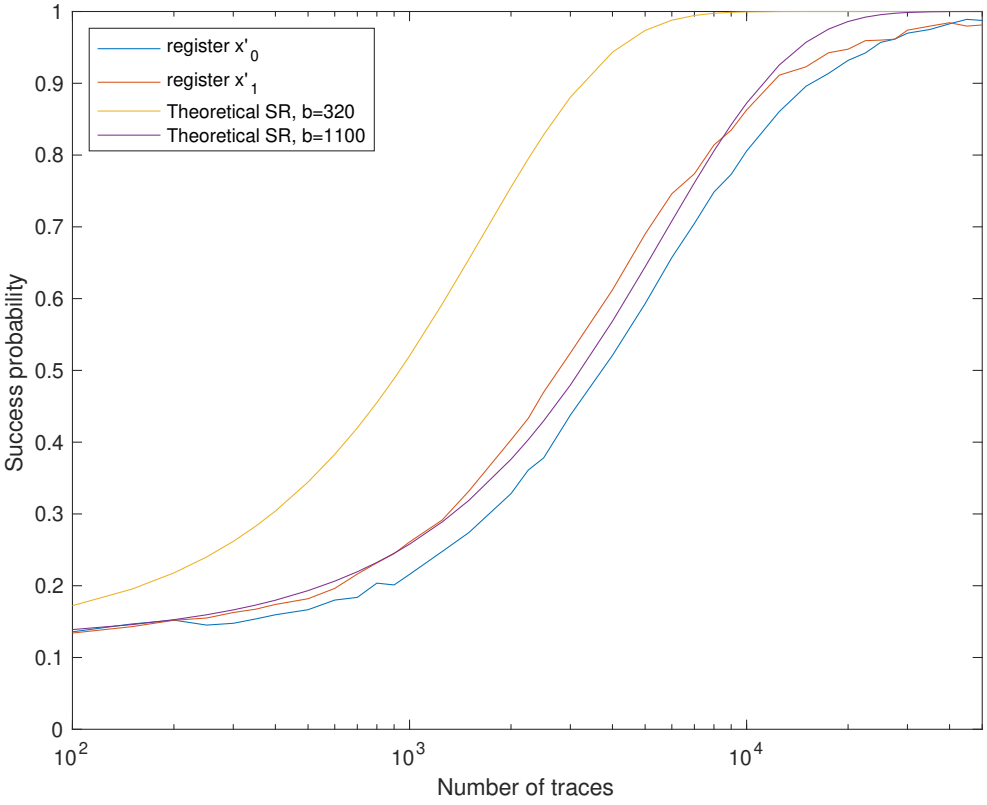
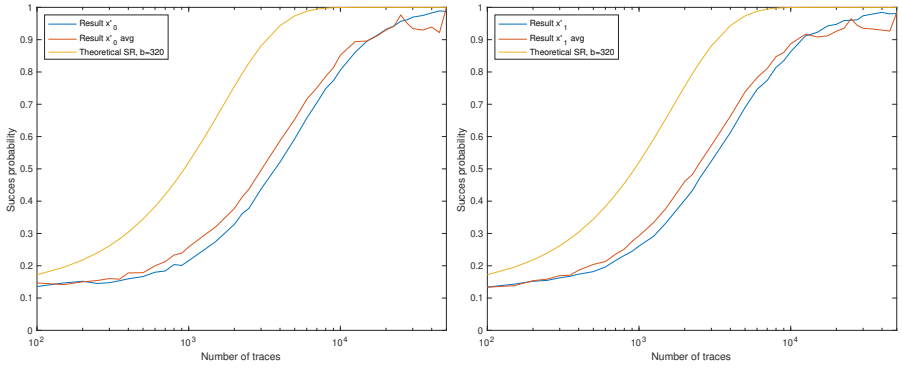


Figure 3.7: Success rate of attack on Ascon bit by bit



(a) Result noise reduction register x'_0 (b) Result noise reduction register x'_1

Figure 3.8: Result attack Ascon with noise reduction by averaging

We observed a lot of the generated noise is other than algorithmic caused by switching of the registers. We assume the other noise is unrelated to the input. This makes it possible to reduce noise by averaging over a set amount of traces with an equal input. We averaged over fifty equal inputs and the results can be observed in Figure 3.8(a) and Figure 3.8(b). Both figures show that the success probability is a bit higher compared to the result where no traces are averaged. The result is still not close to the theoretical curve where the state contains 320 bits. This means that noise other than algorithmic caused by flipping of the registers is still present in the traces. Since it does not reduce by averaging over equal inputs, we believe it is related to the input. The remaining noise is caused by the combinatorial logic. This leaves more possibilities to extend an attack. Perhaps exploiting leakage caused by bits flipping in the registers combined with the combinatorial logic could lead to higher success probabilities.

3.4.3 Threshold Implementations

A technique to protect hardware implementations against first-order DPA is a threshold scheme [214]. To protect against first-order DPA, the number of required shares N is the algebraic degree of the non-linear function plus one. A threshold implementation or TI has three requirements: non-completeness, correctness and uniformity [64]. An implementation is non-complete if at most $N-1$ shares are combined at once. Linear steps can be computed separately on each share and recombined in the end. In the non-linear step shares must be combined. This looks as follows for a TI with three shares.

$$A' = f_a(B, C)$$
$$B' = f_b(C, A)$$
$$C' = f_c(A, B)$$

A threshold implementation is correct if the following holds.

$$f(A \oplus B \oplus C) = f_a(B, C) \oplus f_b(C, A) \oplus f_c(A, B)$$

Not all non-linear functions keep the uniformity of their input shares, this is a problem because it has been proven that if the shares are uniformly distributed, correct and non-complete the implementation is secure against first-order DPA [216]. Both Ascon and Keyak do not have a uniform round function, so random bits must be added to keep the round function uniform. It is possible to reduce the amount of new random bits that are required each round [63] for Keyak. Recently another technique was proposed to increase the state by a few bits to keep it uniform [93]. This way, no new random bits are required each round. Since Ascon uses a variation of Keyak's S-box this works for Ascon as well.
The threshold implementation we use has three shares. One round of the round function is computed in parallel on all three shares each clock cycle.

Power consumption model

We use the same model as in [54] where we compute the Hamming distance between the input and output of a round. This models the switching of a register to leak

information. If the Hamming distance between two bits is 0 it contributes +1 to the power consumption and if it is 1 it contributes -1. A bit in a state can be represented by three bits in the threshold implementation, for 0 there are 000, 011, 101 or 110, for 1 there are 111, 001, 010 or 100. Since the threshold implementation is initialized randomly the occurrence of each triplet is also random. For each triplet in the state we show the contribution of the power consumption in Table 3.1. This results in a mean of 0 and a variance of 3.

Table 3.1: Modeled power consumption of threshold implementation

Bit 0	Contribution	Bit 1	Contribution
000	+3	111	-3
011	-1	001	+1
101	-1	010	+1
110	-1	100	+1

3.4.4 Attack on TI Ascon

In this attack we attack a threshold implementation. A fully parallel threshold implementation with an internal state of 320 bits will have a lot of algorithmic noise. Figure 3.9 shows the theoretical success probability of a fully parallel threshold implementation where the width of the state $b = 320$ bits using functions from [54, Section 4C]. We see that the required amount of traces for a success probability higher than guessing the correct key candidate requires over 10^8 traces. To collect this many traces is not feasible in our experimental setting. To be able to obtain the correct key with a feasible amount of traces, we decreased the size of the state to a toy version. To do this we modify the implementation to have a smaller internal state. In Ascon the state consists of five 64-bit registers, to reduce the amount of algorithmic noise we study a toy-sized version of Ascon with 4-bit words. In total the state consists of 20 bits and with three shares there are 60 bits.

The round constant is changed to the following where i is the round number starting from 0.

$$x_2 = x_2 \oplus (\texttt{0xF} - i * \texttt{0x1})$$

The substitution layer is affected as the S-box is shared between the three shares. If we have three shares A, B and C the sharing works as follows. The shares are split up into 5 registers ranging from 0 to 4. We split the S-box up into three parts, first a linear part, then a non-linear part and finally a linear part. Below we show the

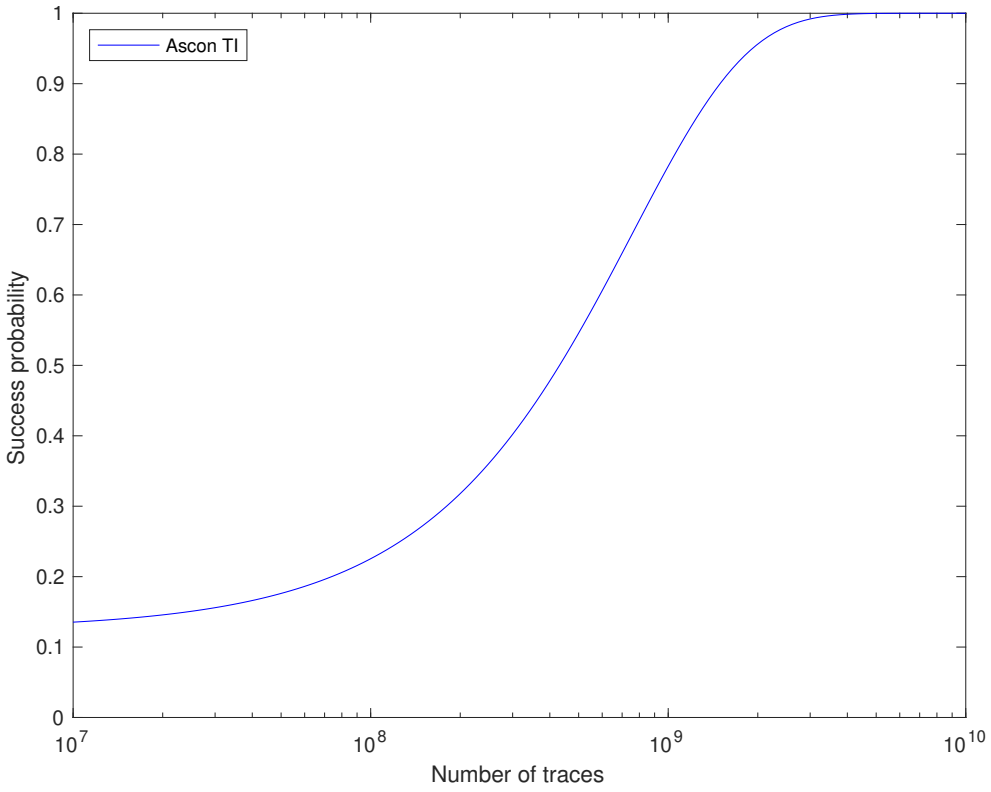


Figure 3.9: Theoretical success probability Ascon TI

computation for share A.

$$
\begin{aligned}
R_{00} &= A_0 \oplus A_4 \\
R_{01} &= A_1 \\
R_{02} &= A_2 \oplus A_1 \oplus RC \\
R_{03} &= A_3 \\
R_{04} &= A_4 \oplus A_3
\end{aligned}
$$

$$
\begin{aligned}
S_{00} &= R_{10} \oplus (\neg R_{11} \wedge R_{12}) \oplus (R_{11} \wedge R_{22}) \oplus (R_{12} \wedge R_{21}) \\
S_{01} &= R_{11} \oplus (\neg R_{12} \wedge R_{13}) \oplus (R_{12} \wedge R_{23}) \oplus (R_{13} \wedge R_{22}) \\
S_{02} &= R_{12} \oplus (\neg R_{13} \wedge R_{14}) \oplus (R_{13} \wedge R_{24}) \oplus (R_{14} \wedge R_{23}) \\
S_{03} &= R_{13} \oplus (\neg R_{14} \wedge R_{10}) \oplus (R_{14} \wedge R_{20}) \oplus (R_{10} \wedge R_{24}) \\
S_{04} &= R_{14} \oplus (\neg R_{10} \wedge R_{11}) \oplus (R_{10} \wedge R_{21}) \oplus (R_{11} \wedge R_{20})
\end{aligned}
$$

Rotate index i, to compute S_{ij} using R_{ij}.

$$
\begin{aligned}
A'_0 &= S_{00} \oplus S_{04} \\
A'_1 &= S_{01} \oplus S_{00} \\
A'_2 &= \neg S_{02} \\
A'_3 &= S_{03} \oplus S_{02} \\
A'_4 &= S_{04}
\end{aligned}
$$

Except for the RC which is only added at one share, the linear steps are equal for register B and C. The linear diffusion layer is also affected, the rotations are computed modulo the size of the registers and result in the following expressions.

$$
\begin{aligned}
\Sigma_0(x_0) &= x_0 \oplus (x_0 \ggg 3) \oplus (x_0 \ggg 0) \\
\Sigma_1(x_1) &= x_1 \oplus (x_1 \ggg 1) \oplus (x_1 \ggg 3) \\
\Sigma_2(x_2) &= x_2 \oplus (x_2 \ggg 1) \oplus (x_2 \ggg 2) \\
\Sigma_3(x_3) &= x_3 \oplus (x_3 \ggg 2) \oplus (x_3 \ggg 1) \\
\Sigma_4(x_4) &= x_4 \oplus (x_4 \ggg 3) \oplus (x_4 \ggg 1)
\end{aligned}
$$

Since $28 \mod 4 = 0$ the two XOR's cancel each other out which results in a shorter linear expression.

$$\Sigma_0(x_0) = (x_0 \ggg 3) \tag{3.7}$$

To attack the y_0 register we can combine Equation (3.3) from the attack on the unprotected implementation and $\Sigma_0(x_0)$ to obtain a selection function.

$$S_i(M, K^*) = k_0^*(m'_{i+1} + 1) + m_{i+1}$$

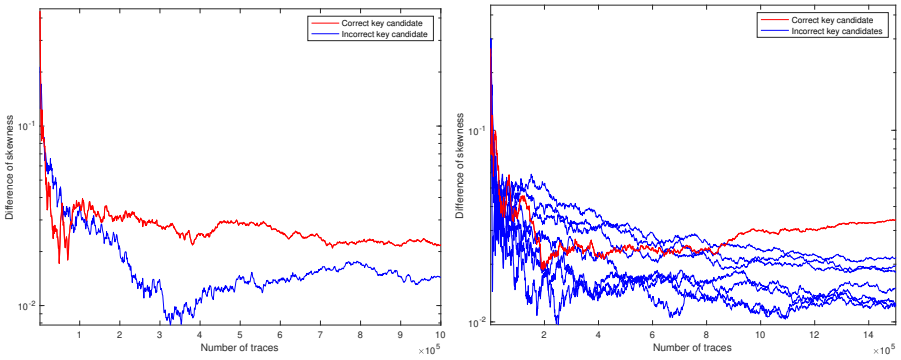
The equation contains only one key bit, as a result there are two key candidates in the attack.

To attack register y_1 we use a similar selection function as in previous attack on the unprotected implementation.

$$
\begin{aligned}
S_i(M, K^*) = {} & m_i(k_0^* + 1) + m'_i + m_{i+3}(k_1^* + 1) \\
& + m'_{i+3} + m_{i+1}(k_2^* + 1) + m'_{i+1}
\end{aligned}
$$

This equations contains three key bits and as a result there are eight key candidates to be considered in the attack.

Since the implementation is protected with a threshold, the mean of the power consumption of the two sets for each key candidate will be equal and a first-order attack with distinguisher like difference of means or the Pearson correlation will not work. Unless there is some unforeseen leakage but we assume this is not the case. As a consequence, a higher order distinguisher is required. In this attack we will use a third order attack, the difference of skewness. The skewness is defined in Section 2.5.1. We simulate traces using the power model discussed in 3.4.3. For the attack we simulate the first twelve rounds of Ascon which corresponds to the initialization phase.



(a) Simulation Ascon register x'_0 (b) Simulation Ascon register x'_1

Figure 3.10: Simulation results for Ascon TI with 20-bit state

3.4.5 Results attack on Ascon TI

Figure 3.10 shows the result of the attack on the simulated traces for register x'_0 and x'_1 where the difference of skewness was used as a distinguisher. From both figures can be observed that it is possible to obtain the correct key candidate with the attack. Register x'_0 required 150.000 traces, where register x'_1 required 900.000. This large difference in traces is explained in Equation 3.7 where two of the three XOR operations use an equal rotation offset so they cancel out. This means instead of having to guess three key bits for this register, we only need to guess a single key bit. This shows the effect of the linear diffusion layer after the S-box in Ascon. Having this linear layer increases the difficulty of the attack as we need to guess two additional key bits. If we compare these results with the attack on the simulated traces in [54] on Keyak we see that required amount of traces scales by the third power with the size of the state. If we would simulate traces for the full Ascon state this number reaches billions as Figure 3.9 suggests. Comparing our results with the results on Keyak we see a similar trend. For this reason we did not simulate traces for a larger state. Also acquiring real traces from the SAKURA-G would be unfeasible.

3.5 FRIET

We showcase the practicality of code embedding with a lightweight authenticated encryption (AE) scheme, called FRIET. It is permutation-based and uses SpongeWrap [57], a mode on top of the duplex [57] construction, similar to CAESAR candidate Ketje [61] NIST lightweight competition submissions Ascon [102], Gimli [48] and Xoodyak [92].

The permutation underlying our AE scheme is called FRIET-PC and it is the result of embedding a code on a permutation FRIET-P. We do not see FRIET (and FRIET-P) as the ultimate fault-attack resistant design but rather as a proof of concept, quite competitive with modern AE schemes (and permutations).

In this section we specify the mode and provide its security claim.

3.5.1 The Permutation AE Mode SpongeWrap

We adopt the AE mode proposed in the paper that introduced the duplex construction and its modes [57], namely SpongeWrap. SpongeWrap has the nice property that it supports AE in *sessions*. A session AE scheme converts sequences of messages, each consisting of (optional) associated data AD and plaintext P, both bit strings of arbitrary length, into a sequence of cryptograms, each consisting of possible associated data, ciphertext C (the enciphered plaintext) and a tag T. The session aspect is related to the tag T: this is not only computed on the associated data and ciphertext of its own cryptogram, but the full sequence of cryptograms that were generated since the start of the session. In other words, a session AE scheme is *stateful*. One can see session AE as support for intermediate tags.

We do not take SpongeWrap [57] as such, but make three minor modifications. First, in the session startup we absorb a dedicated non-secret diversifier D that should be a nonce for sessions started with the same key K. Second, we have the session startup return a tag. Third, we allow for tag lengths longer than the sponge rate. We specify the SpongeWrap mode, with the duplex construction integrated, in Algorithm 2. Here, all parameters are arbitrary-length bit strings with $|X|$ denoting the length of a string X in bits.

SpongeWrap has a b-bit state, with b the width of the underlying permutation f. It has a *block length* ρ and all input strings are first split up into ρ-bit blocks, with the last block possibly shorter. Before a block is absorbed in the state, SpongeWrap appends a domain separation bit to indicate whether the next output will be used as keystream (`1`) or as tag or not at all (`0`). Then the block is padded with a single `1` followed by zeroes. The so-called *duplex rate* r is the size of the part of the state that is directly affected by absorbing, the *outer part*. Due to the domain separation bit and the first bit of the padding, we have $r = \rho + 2$. The remaining part of the state is called the *inner part* and its size is called the *capacity* c. We have $c = b - r = b - \rho - 2$.

The encryption of a message simply consists of splitting AD and P in blocks, padding each block, adding it to the state s and performing the permutation f. Concurrently, each plaintext block is encrypted by bitwise adding to it the outer part of the state at that point. Finally, SpongeWrap squeezes the tag T from the state with a (number of) duplex call(s). Decryption is very similar. After a message has been encrypted or decrypted, one can continue the session with more messages.

The state is initialized by absorbing first the key K and then the diversifier D. For confidentiality the couple (K, D) must be unique per session.

Because it uses the duplex construction, SpongeWrap lends itself quite well to the use of a code-embedded permutation f_C. Actually, we just have to instantiate duplex with the code-abiding permutation f and make some minor modifications:

- The state initialization must set the state to the codeword that encodes the all-`0` vector. For linear codes, this is just the all-`0` vector.

- When absorbing σ, it must first be converted to a valid codeword. If σ is one limb (as it turn out in Friet-PC), it suffices to (bitwise) add it to one limb and the parity limbs that depend on it.

Algorithm 2 SpongeWrap$[f, \rho, \tau]$, with permutation f, block length ρ and tag length τ.

Interface: $T \leftarrow \text{start}(K, D)$
  $s \leftarrow \mathtt{0}^*$ (State s is a persistent data element during the session)
  $\text{absorb}(K, \text{none})$
  $\text{absorb}(D, \text{encrypt})$
  $T \leftarrow \text{squeeze}(\tau)$
  return T

Interface: $(C, T) \leftarrow \text{wrap}(\text{AD}, P)$
  $\text{absorb}(\text{AD}, \text{none})$
  $C \leftarrow \text{absorb}(P, \text{encrypt})$
  $T \leftarrow \text{squeeze}(\tau)$
  return (C, T)

Interface: $P \leftarrow \text{unwrap}(\text{AD}, C, T)$
  $\text{absorb}(\text{AD}, \text{none})$
  $P \leftarrow \text{absorb}(C, \text{decrypt})$
  $T' \leftarrow \text{squeeze}(\tau)$
  if $(T' \neq T)$ **then** **return** error
  return P

Internal interface: $Y \leftarrow \text{absorb}(X, \text{op})$ with $\text{op} \in \{\text{none}, \text{encrypt}, \text{decrypt}\}$
  Let $x[n]$ be X split in ρ-bit blocks, with $n > 0$ and last block possibly shorter
  $Y \leftarrow \epsilon$
  for all blocks of $x[n]$ **do**
    if $(\text{op} = \text{none})$ **then** $b \leftarrow \mathtt{0}$ **else** $b \leftarrow \mathtt{1}$
    if (this is the last block) **then** $b \leftarrow b + \mathtt{1}$
    if $\text{op} = \text{decrypt}$ **then**
      $\text{temp} \leftarrow x[i] + (s \text{ truncated to } |x[i]|)$
      $Y \leftarrow Y \| \text{temp}$
      $\text{duplex}(\text{temp} \| b)$
    else if $\text{op} = \text{encrypt}$ **then**
      $\text{temp} \leftarrow x[i] + (s \text{ truncated to } |x[i]|)$
      $Y \leftarrow Y \| \text{temp}$
      $\text{duplex}(x[i] \| b)$
    else
      $\text{duplex}(x[i] \| b)$
  return Y

Internal interface: $Z \leftarrow \text{squeeze}(\ell)$ with ℓ the requested length of the output Z
  $Z \leftarrow \epsilon$
  while $|Z| < \ell$ **do**
    $Z \leftarrow Z \| (s \text{ truncated to } \rho \text{ bits})$
    $\text{duplex}(\mathtt{0})$
  return Z truncated to ℓ bits

Internal interface: $\text{duplex}(\sigma)$ with $|\sigma| \leq \rho$
  $s \leftarrow s + \sigma \| \mathtt{1} \| \mathtt{0}^*$
  $s \leftarrow f(s)$

- Before using the outer part of the state as tag or keystream, one must check whether the state is a valid codeword and return an error if not.

3.5.2 Specification of the Permutations Friet-PC and Friet-P

In this section, we specify FRIET-P, the permutation implemented in FRIET. Besides, we specify FRIET-PC, its embedding by the linear code $[4, 3, 2]_2$ with parity bit the sum of the 3 native bits. As the propagation properties of FRIET-PC are most relevant, we introduce FRIET-PC first and FRIET-P second.

3.5.3 The Permutation Friet-PC

FRIET-PC has width 384 and has a round function R_i operating on three limbs denoted as a, b, and c. We index the bits of a limb by i ranging from 0 to 127. Limb a and the bits of limb b with indices 0 and 1 form the outer part. The nominal number of rounds is 24 and the round function R_i has 6 steps:

- two non-native limb transpositions τ_1 and τ_2,
- a round constant addition δ_i that is a limb adaptation,
- two mixing steps μ_1 and μ_2 that are limb adaptations,
- a non-linear step ξ, also a limb adaptation.

We specify the FRIET-PC permutation in Algorithm 3 using following notation:

- $x \oplus y$, the exclusive *or* (XOR) of limbs x and y,
- $x \wedge y$, the bitwise logical *AND* of limbs x and y,
- $x \lll n$, the cyclic shift to the left by offset n of limb x. We assume the bits with low indices at the right, so if $y \leftarrow x \lll n$, then $y_n = x_0$

The round constants are in Table 3.2 and the FRIET-PC round function is illustrated in Figure 3.11.

Table 3.2: Round constants rc_i in hexadecimal notation, omitting the leading zero digits

i	rc_i	i	rc_i	i	rc_i
0	1111	8	1001	16	1110
1	11100000	9	100000	17	11010000
2	1101	10	100	18	1010
3	10100000	11	10000000	19	1010000
4	101	12	1	20	1011
5	10110000	13	110000	21	1100000
6	110	14	111	22	1100
7	11000000	15	11110000	23	10010000

Algorithm 3 FRIET-PC

Input: $a, b, c \in \{0,1\}^{128}$
Output: $(a', b', c') \leftarrow$ FRIET-PC(a, b, c)
for Round index i from 0 to 23 **do**
    $(a, b, c) \leftarrow \mathrm{R}_i(a, b, c)$
return (a, b, c)

Here R_i is specified by the following sequence of steps:

$$
\begin{array}{lcll}
c & \leftarrow & c \oplus \mathrm{rc}_i & \delta_i \\
(a, b, c) & \leftarrow & (a \oplus b \oplus c, c, a) & \tau_1 \\
b & \leftarrow & b \oplus (c \lll 1) & \mu_1 \\
c & \leftarrow & c \oplus (b \lll 80) & \mu_2 \\
(a, b, c) & \leftarrow & (a, a \oplus b \oplus c, c) & \tau_2 \\
a & \leftarrow & a \oplus ((b \lll 36) \wedge (c \lll 67)) & \xi
\end{array}
$$

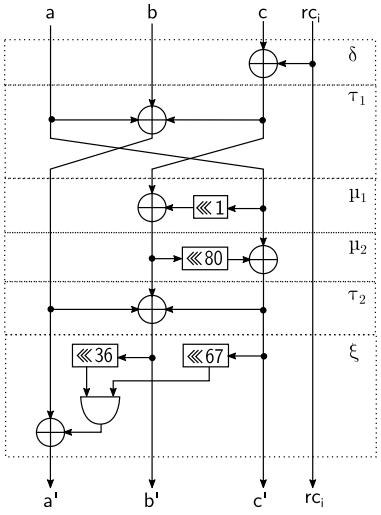


Figure 3.11: Round of FRIET-PC

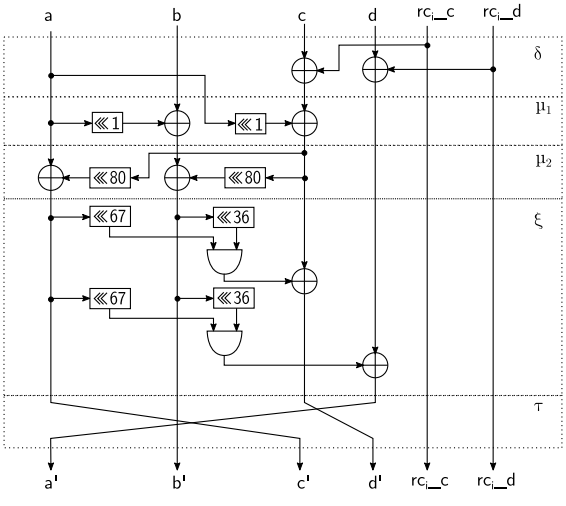


Figure 3.12: Round of FRIET-P

3.5.4 The Round Function of Code-abiding Permutation Friet-P

We build a code-abiding permutation FRIET-P such that its embedding by the parity code $[4, 3, 2]_2$ is FRIET-PC. FRIET-P has width 512, i.e., 4 limbs.

We denote the parity limb d and after any step the parity equation $d = a \oplus b \oplus c$ should be satisfied. It is now straightforward to derive the round function of FRIET-P from the round specification in Algorithm 3 by substituting $(a \oplus b \oplus c)$ by d in limb transpositions steps and duplicating all limb adaptations in d. This results in:

$c \leftarrow c \oplus \mathrm{rc}_i$	$d \leftarrow d \oplus \mathrm{rc}_i$	δ_i
$(a, b, c, d) \leftarrow (d, c, a, b)$		τ_1
$b \leftarrow b \oplus (c \lll 1)$	$d \leftarrow d \oplus (c \lll 1)$	μ_1
$c \leftarrow c \oplus (b \lll 80)$	$d \leftarrow d \oplus (b \lll 80)$	μ_2
$(a, b, c, d) \leftarrow (a, d, c, b)$		τ_2
$a \leftarrow a \oplus ((b \lll 36) \wedge (c \lll 67))$	$d \leftarrow d \oplus ((b \lll 36) \wedge (c \lll 67))$	ξ

We transfer limb transpositions τ_1 and τ_2 to the end and merge them, yielding:

$c \leftarrow c \oplus \mathrm{rc}_i$	$d \leftarrow d \oplus \mathrm{rc}_i$	δ_i
$b \leftarrow b \oplus (a \lll 1)$	$c \leftarrow c \oplus (a \lll 1)$	μ_1
$a \leftarrow a \oplus (c \lll 80)$	$b \leftarrow b \oplus (c \lll 80)$	μ_2
$c \leftarrow c \oplus ((a \lll 67) \wedge (b \lll 36))$	$d \leftarrow d \oplus ((a \lll 67) \wedge (b \lll 36))$	ξ
$(a, b, c, d) \leftarrow (d, b, a, c)$		τ

This sequence of steps is depicted in Figure 3.12 of the FRIET-P round function.

3.5.5 Software Implementation

We implemented and benchmarked FRIET-PC and FRIET-P on an embedded ARM Cortex-M4 microcontroller.

The bitwise logical operations and cyclic shifts on the 128-bit limbs can be implemented very efficiently on the M4's 32-bit architecture using the *bit interleaving* technique [52]. More precisely, we represent every 128-bit limb x as four 32-bit words x_0, x_1, x_2 and x_3 such that the word x_i contains the bits of x with indices congruent to i modulo 4. We also assume that input and output of the permutation are directly mapped to the bit-interleaved format in the state. The bit-interleaving representation offers two main advantages:

- The mixing steps, sum operations and the non-linear layer only require a single register as temporary variable. This allows computing FRIET-PC within the 14 registers that can be freely used.
- The mixing and non-linear steps combine bitwise logical operations with cyclic shifts. The *barrel shifter*, a feature of the Cortex M4, allows computing the shift operations alongside the bitwise Boolean instructions at no extra cost. This reduces the cost of a mixing step in FRIET-PC to 4 XOR operations and that of a non-linear step to 4 XOR and 4 AND operations.

The round constants were chosen such that they could be represented in bit-interleaved representation as the shift of an 8-bit value. As a consequence, the round constant

addition consists in a single XOR operation for Friet-PC and 2 XOR operations for Friet-P. In Friet-PC, the limb transposition takes 8 XOR instructions, while in Friet-P it comes naturally for free. All in all, one round of Friet-PC requires 29 XOR and 4 AND instructions and one round of Friet-P takes 26 XOR, 8 AND and 4 load and store instructions because the 512-bit state does not fit into the registers. To further increase the performance, we fully unrolled the 24 rounds of the permutation. The Friet-PC permutation takes 853 cycles and the Friet-P permutation takes 1163. Hence in this implementation the code embedding results in an overhead of about 36% mostly due to the additional load and store instructions.

We compare our implementations in Table 3.3 with other permutations, ranked by decreasing cycles per byte per round ratio. We also provide the cycles per byte ratios. However, these results should be taken with a grain of salt as, the security margin taken in terms of the number of rounds and the amount of propagation achieved by a single round differs from one permutation to the other.

Table 3.3: Performance Comparison on Cortex-M3/M4

Permutation	Width (bits)	Rounds	Cycles/byte per round	Cycles/byte	Device
Xoodoo [94]	384	12	1.10	13.20	Cortex-M3
Friet-P (this work)	384	24	1.01	24.23	Cortex-M4
Gimli [48]	384	24	0.91	21.81	Cortex-M3
Friet-PC (this work)	384	24	0.74	17.78	Cortex-M4

3.5.6 Fault Attack on the Software Implementation

Here we describe the setup that we use to evaluate the fault resistance of the permutation. We apply electro-magnetic fault injection, which is accomplished by emitting a short EM pulse from a specific position close to the target.

Figure 3.13 shows an overview of the setup. Our target is an STM32F407IG development board containing an ARM Cortex-M4F microcontroller. The xy-table moves a probe across the target with high precision. The VC Glitcher sends a signal so the probe will emit a pulse and it also controls a reset line, in case the pulse was too strong and the board is unable to respond. An oscilloscope is used together with a current probe to measure the power consumption in order to determine a time window where the fault should be injected.

We conducted an electro-magnetic fault injection experiment where we scanned the whole chip. We divided the surface of the chip in a 100 by 100 grid, injected 10 faults per position and repeated this 10 times. This resulted in a total of 1 000 000 faults. For the experiment, we focused on the last round. Table 3.4 shows the fault detection results of the experiment. Each fault has four possible outcomes:

- Normal: no fault has occurred and the device behaves as expected,
- Reset: the EM pulse was too strong and the device was unable to respond so the device was reset,

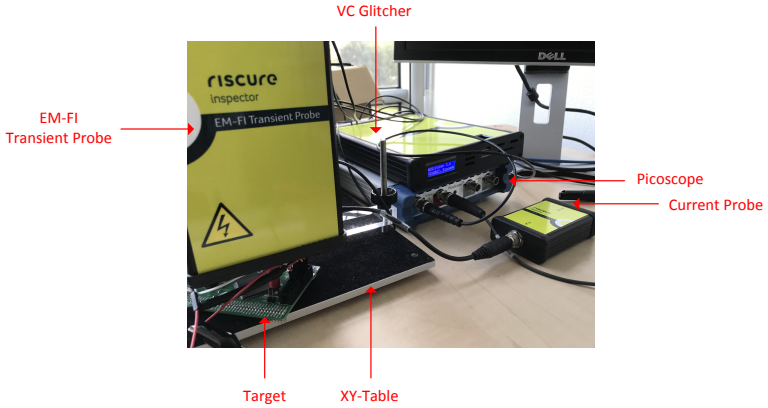


Figure 3.13: The setup.

Table 3.4: Experimental results of 1 000 000 glitches.

Result	Normal	Reset	Detected	Undetected
Number	860488	138916	596	0

- Undetected: a fault occurred that was not detected,
- Detected: a fault occurred that was detected.

The table shows that all faults are detected by our implementation. To achieve this, we added another countermeasure to the implementation. During preliminary experiments, we noticed in a handful of cases that a single glitch was able to modify bits from different words in the same bit-position. To counter this effect, we store the limbs in bit-interleaved format, where the 32-bit words representing limb b, c, and d undergo a circular shift to the left by 1 bit for b, 2 bits for c, and 3 bits for d. The rotated words in each limb ensure a glitch causing a fault in multiple words in the same bit position is still detected. During our fault resistance analysis we did not consider ineffective faults [86].

3.6 Conclusion

In this chapter we performed an attack on a fully parallel unprotected FPGA implementation of Keyak. We improved the attack to be more efficient on the state by attacking a row instead of a single bit. For Ascon we created an attack and applied it on a parallel hardware implementation. To reduce the electrical noise we averaged traces with an equal input. We also created an attack for a threshold implementation of Ascon. We simulated traces for a toy-sized version and attacked those to reduce the required number of traces. We omit results for an attack on simulated traces for Keyak as this can be found in [54, Section 5E].

The attack on Keyak resulted in a lower success probability compared to the attack in the simulated traces. This means more noise is present in the traces compared to

what was taken into account in the model. In the attack on Ascon we tried to reduce the noise by averaging traces. This only reduced the noise slightly which means the greater part of the noise is related to the input. As the model only takes algorithmic noise caused by switching of values in registers into account, this led to the conclusion that a lot of noise present in the traces is algorithmic noise generated by the combinatorial logic.

Our approach to create a more efficient attack on the state, resulted in an attack with a similar success probability, this is due to the attack being equivalent. It did take less time to do the computations for the attack on a row. Creating an attack that is more efficient compared to the attack on a single bit based on the required amount of traces is regarded as future work. We were able to successfully attack the simulated traces of the protected implementation of Ascon. Even though it was a simulation of a toy-sized implementation, it gives insight in the effect of a linear layer after the S-box. The addition of this layer makes it much harder to attack the implementation. To attack real traces of a full implementation that is protected with a threshold would be infeasible.

Towards the end of this chapter we looked at the effectiveness of the countermeasure of the fault resistant permutation Friet. We have seen that the countermeasure was able to detect all faults. To accomplish this we had another small countermeasure that detected faults that caused bit flips in the same position for two different words. It should be noted that these faults are out of scope for the countermeasure as it only claims security on single bit faults. From this we can conclude that the countermeasure is very effective against single bit faults.

Chapter 4

Physical Attacks on Real-World Implementations

4.1 Introduction

Since its invention in the late 80's independently by Koblitz [170] and Miller [204] Elliptic Curve Cryptography (ECC) has established itself as the default choice for classical public-key cryptography, in particular for constrained environments. Especially lightweight Internet of Things (IoT) applications and resources sparse platforms such as RFID tags and sensor nodes consider ECC exclusively for their (exceptional) public-key requirements. This does not come as surprise knowing that working in fields with size 160 bits or so is considered to be at least as secure as RSA using around 1200 bits [106]. This property often results in implementations of smaller memory/area footprints, lower power/energy consumption etc.

A recent initiative is to seriously consider and consequently standardize some post-quantum cryptosystems, i.e., those that could survive a prospect of having a quantum computer that (if built) would break all classical public-key cryptosystems. However, this does not (yet) make research on ECC obsolete as there is still a number of years to go, before the actual transition to post-quantum cryptography might occur.

Research on ECC has evolved from the first proposals to numerous works on protocols, algorithms, arithmetic, implementations aspects including side-channel security etc. Especially, looking into different curves and representations has become a resourceful topic for various optimizations. Twisted Edwards curves [107] were proposed by Bernstein and Lange [45, 49] featuring a complete point operation formulas that is proven to be more efficient and secure with respect to side-channel leakages.

All together, the easiness of constant-time implementations and performance boost, together with somewhat reduced confidence in NIST-standardized curves have made many users transitioning to Edwards curve based protocols including OpenSSH, Tor, TLS, Apple AirPlay, DNS protocols etc [284].

In particular, Edwards-Curve Digital Signature Algorithm (EdDSA) is very popular in real-world application of cryptography. An instance of EdDSA using Edwards Curve25519 called Ed25519 is used among others, in Signal protocol (for mobile phones), Tor, SSL, voting machines in Brazil etc. There is an ongoing effort to stan-

dardize the scheme, known as RFC 8032.

EdDSA including Ed25519 is claimed to be more side-channel resistant than ECDSA [47], not just in terms of resisting software side-channels i.e. featuring constant timing. The authors rely on the idea to "generate random signatures in a secretly deterministic way" such that "different messages lead to different, hard-to-predict values of ephemeral key r". This aims at the known algorithms using lattice methods to compute the long-term ECDSA key from the knowledge of as few as 3 bits of the ephemeral key r for a few hundred of signatures [213]. This knowledge can be typically obtained from side-channel attacks or from non-uniformity of the distribution from which r is taken, so the authors of EdDSA rightfully point at the fact that the "deterministic feature" results in no obvious side-channel leakage exploits. They also state that "no per-message randomness is consumed", making this additionally attractive due to the notoriously expensive generation of random numbers.

In this chapter we show that, although expensive, one should possibly retreat to randomness as we are able to break Ed25519, implemented in WolfSSL, by using 1st order differential power analysis. Actually, the combination of the key and the message in the hash computation (without randomness) makes it a classic scenario for DPA as proposed in the seminal paper of Kocher et al. [172]. More in detail, although we exploit the non-linearity of modular addition in the hash computation, EdDSA is a perfect target for this kind of attack as it fully breaks the scheme after collecting as few as 4 000 power or EM traces. We give all the details of the attack later in this chapter, including a simple fix that would render the attack infeasible.

Additionally, in this chapter, we present a generic fault injection attack that can be applied to a range of platforms and it is using different sources for fault injection. We demonstrate the pervasiveness of it on a 32-bit micro-controller targeting EdDSA implementation within the lightweight cryptographic library WolfSSL. For our attack a single fault during the scalar multiplication algorithm is required for the full key recovery. We give all the details of the setup where this semi-invasive attack is done by applying minor changes to the supply voltage or using electro-magnetic EM signals as the "glitching" sources.

The two attacks discussed in this chapter are both capable of recovering a secret key that can be used to forge a signature. However, they are very different in nature. The first attack that is discussed is a passive attack where an attacker only needs to monitor the power consumption or the electromagnetic emanations of roughly 4000 signatures. In contrast to the passive attack, the second attack described in this chapter is an active attack. An attacker must actively influence the environment of a device. This can be accomplished by, for instance, altering the supplied voltage or emitting an electromagnetic pulse close to the device. Another main difference between the two attacks is that the active attack only requires a single successful fault to recover the secret key.

4.1.1 Related Work

Ed25519 uses SHA-512, a member of the SHA-2 family, for hashing. SHA-512 is used in many applications, often in HMAC mode. Namely, as SHA-1 collisions were expected for years up to now many implementers started already upgrading to alter-

natives. As a matter of fact, due to the recently found collisions [279] in SHA-1 it is strongly recommended to immediately migrate to SHA-2 or SHA-3.

Several works looked into side-channel vulnerabilities in SHA-1 and SHA-2 hash functions or other symmetric-key primitives using modular addition. McEvoy et al. [198] presented an attack on the compression function of SHA-2. Basically, they present the theory of an attack on an HMAC construction using DPA but a full attack on real traces was not executed. The authors also presented a countermeasure against DPA using masking.

In another attack on the compression function of SHA-2, Belaid et al. [36] target other steps (than McEvoy et al.) and they provide results on simulated traces. The authors also suggest a countermeasure for their specific attack.

In Seuschek et al. [267] the authors discuss an attack on EdDSA. They apply the attack as described in [198, 36]. However, they do not execute the attack on either simulated or real traces.

In this work we exploit another aspect of SHA-512. Namely, our attack is the first one to exploit leakage in the computation of the message schedule of SHA-512 (in contrast to the previous paper where they target the addition of part of the message in the round function). More specifically, we target the modular addition operation and exploit the non-linearity of it to attack EdDSA.

Attacking modular addition was done before by several authors. Zohner et al. [311] attack the modular addition in the hash function Skein using real traces. The authors discuss issues regarding a certain symmetry in the results of an attack on modular addition and present a solution. Namely, the correct result value modified by flipping the most significant bit also shows a correlation. This result is called the symmetric counterpart of the correct result. Lemke et al. [178] and Benoît et al. [38] also attack modular addition in other symmetric ciphers on simulated traces. A similar symmetry in the results was observed.

In our work we actually use the symmetry in the results of the attack in a different manner. More precisely, we use it to reduce the number of traces until the key recovery. Additionally, we provide results of our attack on real traces supporting the hypotheses from the theoretical attack considerations. Except for [311] the previous works only support their theory with simulations.

In 1997, the first differential fault attack on public key system RSA-CRT was introduced by Boneh et al. [69]. The authors presented a theoretical concept together with a possible countermeasure. Later Aumüller et al. [17] show the feasibility of the attack by applying it in practice and presenting another countermeasure.

Considering other public key cryptosystems, the first differential fault attack on an elliptic curve cryptosystem was presented by Biehl et al. [62] in 2000. In the scenario they propose the resulting point is not on the original curve anymore. Hence, as a consequence they validate the point as a countermeasure.

Barenghi et al. [24] describe several potential fault attacks on EC-based signature schemes theoretically. In one of the attacks, a fault is introduced during the computation of the hash function. This value is not public and must be recovered by brute forcing over all possible values. The authors implemented the key recovery part and presented their results for this specific scenario.

Recently, a work by Ambrose et al. [12] outlined several differential fault attacks on deterministic signature schemes. However, the authors present no practical results.

The first differential fault attack on Ed25519 was published by Romailler et al. [252]. The authors used the Arduino Nano, an 8-bit micro-controller as their target platform where a signing operation takes over 5 seconds. They introduced a fault in the output of a hash function which is not public so the requirement for the attack is to brute force this value. This issue is complicated with modern platforms using 32-bit or 64-bit architectures. Therefore, the attack is not so practical for other than 8-bit architectures. In our attack, we introduce a fault during the scalar multiplication which makes it platform-independent.

With the introduction of the Rowhammer attack [168] several papers have been published on injecting faults using software manipulations. Poddebniak et al. [234] used the idea to attack deterministic signature schemes. In their work, they explain how to apply the Rowhammer attack and how to prevent it by presenting several countermeasures. This attack is very different than our attack because the impact of Rowhammer is that an invalid signature is generated, while our attack recovers the relevant part of the key in order to forge a signature.

4.1.2 Contributions

Here we summarize the main contributions of this chapter:

- We present the first side-channel attack on Ed25519 using real traces. To this end, we extract secret information i.e. a key that allows us to forge signatures on any message using the key obtained. The key recovery is successful after collecting a few thousands of power consumption traces corresponding to signature generation.

- We present the first side-channel attack on the message schedule of SHA-512 targeting the modular addition operation within. The ideas are extendable to other similar constructions. In contrast to previous attacks on SHA-512 we target the extension of the message schedule instead of the addition of a message in the round function.

- Our attack breaks a real-world implementation. The traces were generated by an implementation of Ed25519 from the lightweight cryptographic library WolfSSL on a 32-bit ARM based micro-controller. This kind of implementation particularly targets low-cost and/or resource-constrained environments as in the IoT use cases and similar.

- We present a countermeasure against this attack. The countermeasure is a result of a small tweak in EdDSA that would not just make the attack infeasible but also does not add much overhead to implementations. A similar countermeasure where randomness is added was presented in the XEdDSA and VXEdDSA Signature Schemes [283, 14] (more details in Sect. 4.4.5).

- In this chapter we also present a conceptually novel and generic differential fault attack on the deterministic signature scheme Ed25519. We inject the fault in the scalar multiplication operation that is unrelated to the hash computation, such that the attacker does not need to brute force the intermediate result.

- The attack is demonstrated on a real-world implementation of Ed25519 from the lightweight cryptographic library WolfSSL on a 32-bit micro-controller. This kind of implementation particularly targets low-cost and/or resource-constrained environments as in the IoT use cases and similar.
- We show that our attack can be effectively executed using voltage glitching and electromagnetic fault injection.
- Finally, we also establish the fact on the necessity of suitable countermeasures as we show that even the common point validity check countermeasure cannot counteract the attack.

4.1.3 Organization of this chapter

In this chapter we have mentioned related previous work and specified our contributions. The rest of the chapter is organized as follows. In Sect. 4.2, we provide background information required for the remainder of the chapter. Section 4.3 gives the ingredients of the side-channel attack and dissect the methodology from attacking the signature scheme down to DPA on modular addition. In Sect. 4.4 we present the practical attack on a 32-bit ARM architecture running WolfSSL and some caveats that had to be overcome before turning the idea into a practical attack. We present the results of the attack with a technique to reduce the number of traces. In Sect. 4.4.5 we present a countermeasure. Section 4.5 describes the fault injection attack and Sect. 4.6 show the results. In Sect. 4.6.4, we describe a countermeasure for the fault attack. Section 4.7 concludes the chapter.

4.2 Background

4.2.1 EdDSA

EdDSA [47] is a digital signature scheme. The signature scheme is a variant of the Schnorr signature algorithm [263] that makes use of Twisted Edwards Curves. The security of ECDSA depends heavily on a good quality randomness of the ephemeral key, which has to be truly random for each signature. Compared to ECDSA, EdDSA does not need new randomness for each signature as the ephemeral key is computed deterministically using the message and the auxiliary key that is derived from the private key. The security depends on the secrecy of the auxiliary key and the private scalar. This does not create a new requirement as we need to keep a private key secret anyway.

In Ed25519, a twisted Edwards curve birationally equivalent to Curve25519 [43] is used. Ed25519 sets several domain parameters of EdDSA such as:

- Finite field F_q, where $q = 2^{255} - 19$
- Elliptic curve $E(F_q)$, Curve25519
- Base point B
- Order of the point B, l

Table 4.1: Our Notations for EdDSA

Name	Symbol
Private key	k
Private scalar	a (first part of $H(k)$)
Auxiliary key	b (last part of $H(k)$)
Ephemeral scalar	r

- Hash function H, SHA-512 [237]
- Key length $b = 256$

For more details on other parameters of Curve25519 and the corresponding curve equations we refer to Bernstein [47].

To sign a message, the signer has a private key k and message M. Algorithm 4 shows the steps to generate an EdDSA signature.

Algorithm 4 EdDSA key setup and signature generation

Key setup.
1: Hash k such that $H(k) = (h_0, h_1, \ldots, h_{2b-1}) = (a, b)$
2: $a = (h_0, \ldots, h_{b-1})$, interpret as integer in little-endian notation
3: $b = (h_b, \ldots, h_{2b-1})$
4: Compute public key: $A = aB$.
Signature generation.
5: Compute ephemeral private key: $r = H(b, M)$.
6: Compute ephemeral public key: $R = rB$.
7: Compute $h = H(R, A, M)$ and convert to integer.
8: Compute: $S = (r + ha) \mod l$.
9: Signature pair: (R, S).

The first four steps belong to the key setup and are only applied the first time a private key is used. Notation $(x, \ldots, y)$ denotes concatenation of the elements. We call a the private scalar and $b = (h_0, h_1, \ldots, h_{2b-1})$ the auxiliary key (see Table 4.1). In Step 5 the ephemeral key r is deterministically generated.

To verify a signature (R, S) on a message M with public key A a verifier follows the procedure described in Algorithm 5.

Algorithm 5 EdDSA signature verification

1: Compute $h = H(R, A, M)$ and convert to integer.
2: Check if the group equation $8SB = 8R + 8hA$ in E holds.
3: If group the equation holds, the signature is correct.

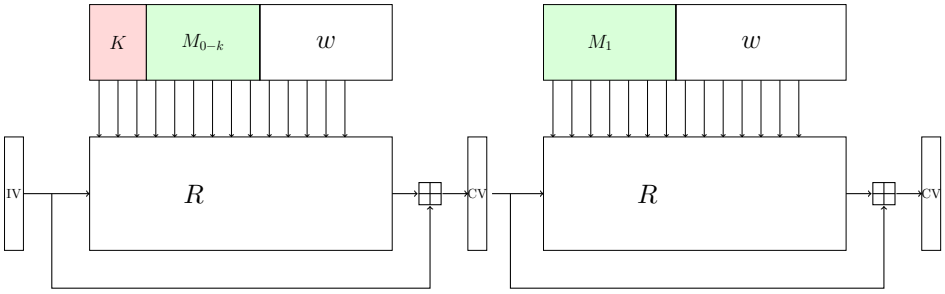


Figure 4.1: SHA-512 hashing of K and M.

4.2.2 SHA-512

SHA-512 is a member of the SHA-2 hashing family, designed by the NSA. The hash functions from the SHA-2 family are named after their digest length. SHA-512 is used several times in the Ed25519 signature scheme. SHA-2 is based on its predecessor SHA-1 and with SHA-1 being broken, implementations change in their usage of hash function from SHA-1 to SHA-2 or SHA-3 [51].

SHA-2 is a Merkle-Damgård construction that uses a compression function based on a block cipher by adding a feed-forward according to Davies-Meyer, see Algorithm 6. Merkle-Damgård iteratively updates a chaining value (CV), this value is initialized to a fixed initial value (IV). The message is padded and split up into blocks. In each iteration a message block is processed. The digest is the value of the CV after all message blocks have been processed. Figure 4.1 shows an overview of the generation of the ephemeral scalar where the auxiliary key and the message are hashed. The letter K denotes the auxiliary key b, M_i the input message, w the remaining message schedule words and R the compression function. M_0 is the fragment of the message that is in the same block as the key and M_1 a fragment in the second block. We assume here a relatively short message.

The compression function has two inputs, the chaining value CV_i and message block M_i. The compression function produces an updated chaining value CV_{i+1}. All the variables in SHA-512 are 64-bit unsigned integers (words). The additions are computed modulo 2^{64}. The algorithm consists of a data path and a message schedule. The data path transforms the CV by iteratively applying 80 rounds on it. The message expansion takes a $16 \times 64 = 1024$-bit message block and expands it to a series of 80 *message schedule words* w_i, each of 64 bits. Each message block consists of 16 64-bit words, that are the first 16 message schedule words. Next, the remaining message schedule words are generated using the 1024-bit message block so there is a word for each round. On a message block 80 rounds are applied, in each round a round constant and a message schedule word is added. As a result a 512-bit message digest is produced.

The compression function of SHA-512 is explained in detail in Algorithm 7 using the notation described in Table 4.2.

Algorithm 6 Merkle Damgård

Input: Message M with $0 \leq$ bit-length $< 2^{128}$
Output: Hash value of M

Pad message M by appending an encoding of the message length
Initialize chaining value CV with constant IV
Split padded message M into blocks
for all blocks M_i **do**
    $\text{CV}_{i+1} \leftarrow \text{CF}(\text{CV}_i, M_i)$
return H $\leftarrow$ CV

Algorithm 7 SHA-512 Compression function

Input: CV_i, M_i
Output: $\text{CV}_{i+1} = \text{CF}(\text{CV}_i, M_i)$

Message expansion
for $i = 0; i < 16; i{+}{+}$ **do**
    $w[i] \leftarrow m[i]$
for $i = 16; i < 80; i{+}{+}$ **do**
    $\sigma_0 \leftarrow (w[i\text{-}15] \ggg 1) \oplus (w[i\text{-}15] \ggg 8) \oplus (w[i\text{-}15] \gg 7)$
    $\sigma_1 \leftarrow (w[i\text{-}2] \ggg 19) \oplus (w[i\text{-}2] \ggg 61) \oplus (w[i\text{-}2] \gg 6)$
    $w[i] \leftarrow \sigma_1 + w[i\text{-}7] + \sigma_0 + w[i\text{-}16]$
$H_0, \cdots, H_7 \leftarrow \text{CV}_i$
Copy chaining value to data path
$a \leftarrow H_0, \ldots, h \leftarrow H_7$
for $i = 0; i < 80; i{+}{+}$ **do**
    $\Sigma_1 \leftarrow (e \ggg 14) \oplus (e \ggg 18) \oplus (e \ggg 41)$
    $\Sigma_0 \leftarrow (e \ggg 28) \oplus (e \ggg 34) \oplus (e \ggg 39)$
    $ch \leftarrow (e \wedge f) \oplus ((\neg e) \wedge g)$
    $maj \leftarrow (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$
    $T_1 \leftarrow h + \Sigma_1 + ch + k[i] + w[i]$
    $T_2 \leftarrow \Sigma_0 + maj$
    $h \leftarrow g$
    $g \leftarrow f$
    $f \leftarrow e$
    $e \leftarrow d + T_1$
    $d \leftarrow c$
    $c \leftarrow b$
    $b \leftarrow a$
    $a \leftarrow T_1 + T_2$
Davies-Meyer feed-forward
$H_0 \leftarrow H_0 + a, \ldots, H_7 \leftarrow H_7 + h$
return $\text{CV}_{i+1} \leftarrow H_0, \ldots, H_7$

Table 4.2: Notation for SHA-512

Name	Symbol
Bitwise right rotate	$\ggg$
Bitwise right shift	$\gg$
Bitwise and	$\wedge$
Bitwise xor	$\oplus$
Bitwise not	$\neg$
Addition modulo 2^{64}	$+$
Message schedule word	$w[i]$
Message word	$m[i]$
Message block	$M[i]$
State of the data path	H_i
Compression function	CF

4.3 The Attack Components - SCA

In this part we elaborate on our strategy and the hierarchy of the attack. Following a top-down approach we examine the Ed25519 signature algorithm looking for vulnerabilities. The way it is composed leads us to identifying the weakness of the modular addition operation in the SHA-512 part.

We start off by explaining what value we need to recover from Ed25519 and how to use it to generate forged signatures. Next, we explain how we recover this value by attacking SHA-512. Finally, we apply DPA on modular addition. To reduce the complexity of the attack we use a divide-and-conquer technique to divide 64-bit key words into 8 bit substrings.

4.3.1 Attacking Ed25519

We describe a key-recovery attack on Ed25519 by measuring the power consumption of 4 000 signature computations.

We attack the generation of the ephemeral key to retrieve the auxiliary key b. This allows us to compute the ephemeral key r. Once we know the auxiliary key, we extract the private scalar by applying the following computations on an arbitrary signature performed with the key.

1. Compute $r = H(b, M)$.
2. Compute $h = H(R, A, M)$.
3. Compute $a = (S - r)h^{-1} \mod l$.

We can use the private scalar a with any message and any auxiliary key b to generate forged signatures. This is because r, in signature verification is only used in R which is part of the signature.

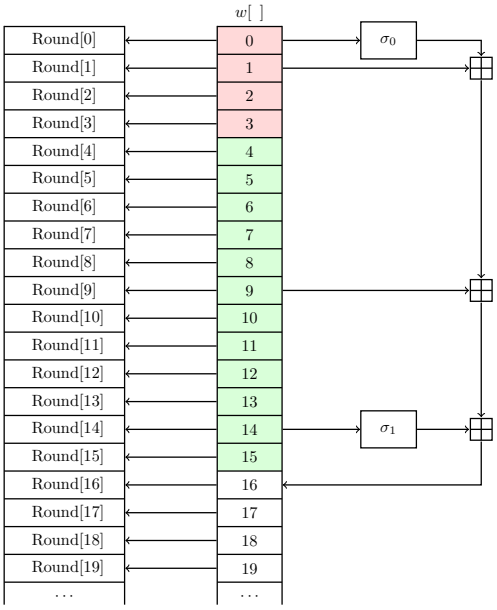


Figure 4.2: Single step of message schedule SHA-512

4.3.2 Attack on SHA-512

The message is appended to the auxiliary key and together this is hashed to compute the ephemeral key. In our attack we assume the message has at least length 512 bits. In this way the first 4 message schedule words contain the constant auxiliary key, the next 8 words contain the variable message and the remaining 4 words can contain more message words or constant padding.

To be able to attack the auxiliary key b, we are looking for steps in the algorithm where a word that only depends on the message part of the input block is added to a constant unknown key-dependent word. If we look at Algorithm 7, we can see that elements from the message schedule are added in two places, namely in message schedule line 6 and in data path line 14. The extending of the message schedule in line 6 seemed like a viable option, as from round 16 to 19 unknown words are added to known words. It depends on the implementation how this can be attacked.

The implementation that we attacked is in WolfSSL, which is a lightweight C-based TLS/SSL library that targets IoT and embedded devices. To determine how to attack the implementation and how to model the leakage we looked at the computation of $w[16]$ in the message schedule, see Fig.4.2. The figure shows a single step to compute the message schedule that is applied recursively to the remaining words.

$$w[16] \leftarrow \sigma_1(w[14]) + w[9] + \sigma_0(w[1]) + w[0] \tag{4.1}$$

σ_0 and σ_1 apply linear transformations that transform a word by taking a word, shifting it over three different offsets and XOR'ing these. They do not play a role in our attack. Of these four words on the right hand side of (4.1), word $w[14]$ and $w[9]$

are part of the message therefore variable and known (green). Word $w[1]$ and $w[0]$ are part of the auxiliary key value so constant and unknown (red). With the attack we are able to recover $\sigma_0(w[1]) + w[0]$. To be able to recover the words separately, we introduce 4 auxiliary variables that represent the key-dependent part of the message expansion word computation. Knowledge of these four variables allows reconstructing the key.

$$w[17] \leftarrow \sigma_1(w[15]) + w[10] + \sigma_0(w[2]) + w[1] \tag{4.2}$$
$$w[18] \leftarrow \sigma_1(w[16]) + w[11] + \sigma_0(w[3]) + w[2] \tag{4.3}$$
$$w[19] \leftarrow \sigma_1(w[17]) + w[12] + \sigma_0(w[4]) + w[3] \tag{4.4}$$

We call the unknown parts $k_{16}, \ldots, k_{19}$, corresponding to the message schedule entries $w[16], \ldots, w[19]$ respectively.

$$k_{19} = w[3] \tag{4.5}$$
$$k_{18} = \sigma_0(w[3]) + w[2] \tag{4.6}$$
$$k_{17} = \sigma_0(w[2]) + w[1] \tag{4.7}$$
$$k_{16} = \sigma_0(w[1]) + w[0] \tag{4.8}$$

Equation (4.3) uses the result of (4.1). Since we can obtain k_{16}, we can compute $w[16]$ and consider it to be known. This also applies to (4.4). In (4.4), $w[19]$ only uses one unknown word as input, so $k_{19} = w[3]$. Once we know $w[3]$, there is only one unknown word in (4.7), word $w[2]$. Thus we can compute it. The remaining unknown words are computed in a similar way. The words $w[0], \ldots, w[3]$ correspond to auxiliary key $b = (h_b, \ldots, h_{2b-1})$.

4.3.3 DPA on Modular Addition

To attack a full addition we need to guess 64 unknown bits. This leaves us with 2^{64} possible candidates. As it is not feasible to correlate the traces with this number of key candidates, we apply a divide-and-conquer strategy similar to the one in [311]. We pick an 8-bit part of the computation result called the sensitive variable.

We start the attack on a 64-bit word with the least significant 8 bits of the words. We craft the selection function $S(M, k^*)$ as follows for k_{16}, where M is part of the input message $(w[9], w[14])$ and k^* is the key byte we make a hypothesis on.

$$S(M, k^*)_{k_{16},\ \text{bit}\ 0-7} \leftarrow ((\sigma_1(w[14]) + w[9]) \mod 2^8) + k^* \tag{4.9}$$

Next, we create the table V containing all possible intermediate values by adding $k^* \in \{0, \ldots, 255\}$ to each 8-bit message. The addition of k^* is not reduced by 2^8, that means the intermediate values have a length of at most 9 bits. The trace set contains T traces, each trace consists of N time samples and there are 256 key candidates. With table V we model the power consumption by computing the Hamming Weight of each intermediate value and store them in table $H = T \times K$. To find the correct key candidate we compute the Pearson correlation of each column of traces with each column of H. The result is stored in table $R = K \times N$. When a sufficient amount

of traces is used, the row with the highest absolute value corresponds to the correct key candidate. We store the value in k'_{16} (the recovered key bits) with the remaining bits 0.

When we know the least significant byte of k_{16} by applying the attack, we use it to obtain the next byte as follows.

$$S(M, K^*)_{k_{16},\text{ bit }8-15} \leftarrow (((\sigma_1(w[14]) + w[9] + k'_{16}) \gg 8) \mod 2^8) + k^*$$

We add k'_{16} to the messages, shift the result 8 bits to the right and compute modulo 2^8 such that the MSB of the previous result is taken into account. We compute the previous steps again and store the key corresponding to the highest correlation value in k'_{16}. We repeat these steps to obtain the remaining 6 bytes of k_{16}. The remaining words of the auxiliary key, k_{17}, k_{18} and k_{19} are obtained in a similar way as k_{16}.

4.4 Experimental Setup and Results - SCA

4.4.1 Setup

For our attack we use the Piñata[1] development board by Riscure as our target. The CPU on the board is a Cortex-M4F, working at a clock speed of 168MHz. The CPU has a 32-bit Harvard architecture with a three-stage pipeline. The board is programmed and modified such that it can be targeted for SCA. The target is the Ed25519 code found in the WolfSSL library, version 3.10.2.

The physical leakage of the device that we exploit is the dependency of the current to the data it is processing. To measure this we use a device called the Current Probe[2] by Riscure. The Current Probe provides us with a clean signal we can exploit.

The oscilloscope to measure the output of the Current Probe is a Lecroy Waverunner z610i. The oscilloscope is triggered by an I/O pin on the target board. We set the pin to a high signal just before SHA-512 is called and to a low signal right after it finishes. Although the clock speed of the CPU is 168MHz, the oscilloscope is set to sample at a rate of 250MS/s. With these settings we captured the traces that we attacked. Figure 4.3 shows a photo of the setup.

4.4.2 Input Correlation

To determine where the computations leak we compute the correlation of values that we know and that are going to be used in the sensitive variable. If we look at Fig. 4.4(a), we see the correlation of the measured power consumption with the Hamming weight of $w[9]$. The same approach was applied for $\sigma_1(w[14])$. For $w[9]$ we observe peaks in the correlation and for $\sigma_1(w[14])$ we only observe noise. The value $w[9]$ is directly loaded from the memory to a register while $\sigma_1(w[14])$ is not loaded from the memory, but $w[14]$ is and has the linear computation σ_1 applied afterwards. We only observe correlation with values directly loaded from the memory. This lead

[1] Piñata board. Accessed: 18-04-2017. Url: `https://www.riscure.com/security-tools/hardware/pinata-training-target`

[2] Current Probe. Accessed: 18-04-2017. Url: `https://www.riscure.com/benzine/documents/CurrentProbe.pdf`

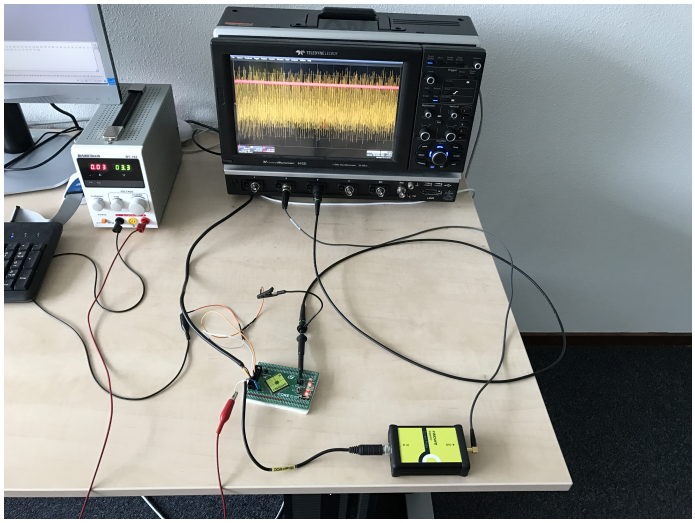
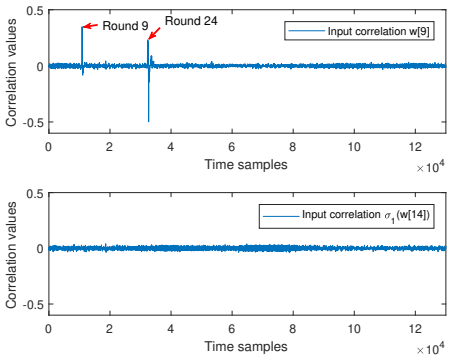
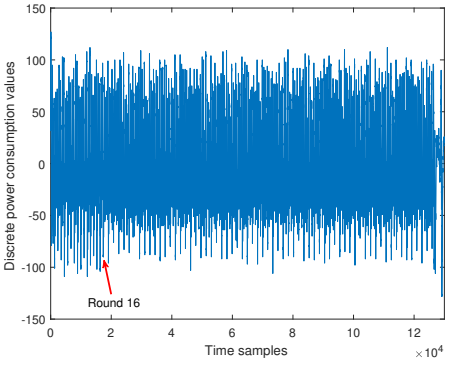


Figure 4.3: Experimental setup



(a) Input correlation of $\sigma_1(w[14])$ and $w[9]$.



(b) Power trace of compression function computation.

Figure 4.4: Input correlation and power trace figures

us to the conclusion that the memory bus provided us with the highest observed leakage.

If we look at Fig. 4.4(b) we see a power trace of the compression function computation where the message expansion is computed. Each negative peak corresponds to a round. The first 16 rounds are shorter as in WolfSSL the message schedule does not happen before the compression rounds start, but on the fly. The time samples in Fig. 4.4(b) correspond to time samples in Fig. 4.4(a), thus we can relate the peaks to the round where they appear. The first peak is when word $w[9]$ is used in the round function at round 9 and the second peak at round 24 when $w[9]$ is used to compute $\sigma_0(w[9])$. There is no input correlation at round 16. The value could be cached and

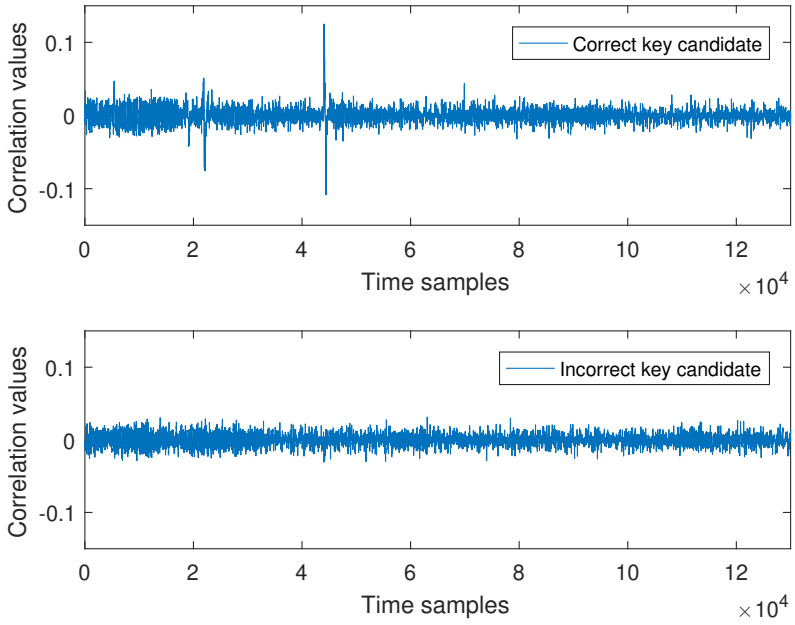


Figure 4.5: Pearson correlation of a correct and an incorrect key candidate.

therefore does not appear on the memory bus.

Since the Hamming weight of values on the memory bus provide the best leakage, we choose to attack values that are loaded or stored from a register to the memory or visa versa. That means in (4.1), $w[16]$ leaks and from that we can recover k_{16}.

4.4.3 Results of the attack

In Fig. 4.5 we see the correlation of the correct key candidate with the traces. Peaks are visible corresponding to the rounds when the value is stored and loaded. The figure also shows the correlation result for an incorrect key candidate where no correlation occurs.

When we plot the highest correlation value for each key candidate we see a similar effect as in other attacks on modular addition where the Pearson correlation is also used. We also see high correlation values for the symmetric counterpart of the correct key candidate. In Fig. 4.6 we can observe this with high peaks for the correct key candidate 68 and for its symmetric counter part key candidate 196. In the symmetric counterpart of the key candidate only the most significant bit is different. As all papers describing an attack on modular addition mention this symmetry it seems unavoidable. Compared to the work [311] we can clearly distinguish the correct key candidate from the incorrect ones.

In Fig. 4.7 we see the success probabilities of the attack on the unknown words $k_{16}, \ldots, k_{19}$. For each data point in the figure we ran the attack 100 times with a certain amount of traces. In Fig. 4.7, the attack was considered successful if all 64 bits of a word were recovered correctly by applying the attack on a byte 8 times. The

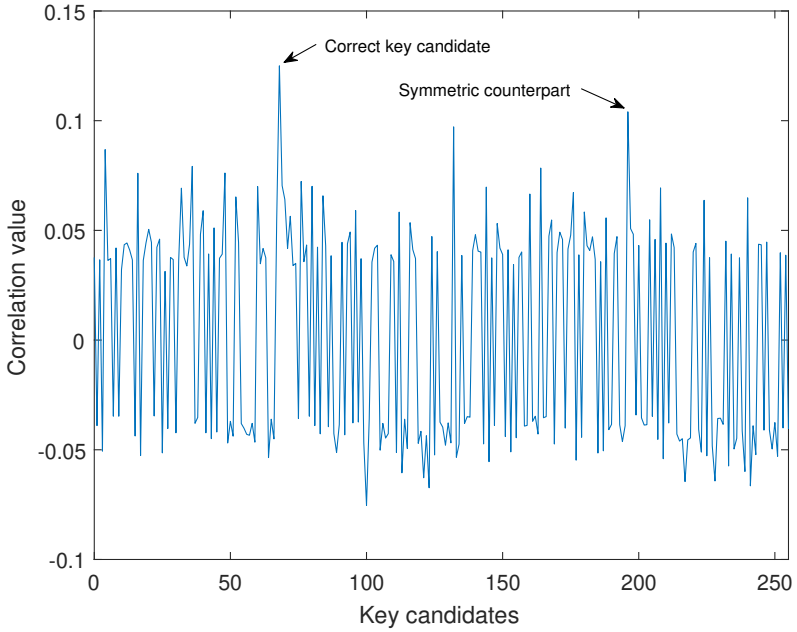


Figure 4.6: Correlation result of the least significant byte of k_{16}, with correct key candidate 68.

figure shows that the success probability of the attack rapidly increases when more than 1000 traces are used. At around 4 000 traces the success probability approaches one making this a practical attack.

4.4.4 Reducing the Number of Traces

Although we can clearly distinguish the correct key candidate from Fig. 4.6, we use the symmetry of the result to increase the success probability of our attack such that less traces are required for a successful attack. The most significant bit is the hardest to attack and requires the highest number of traces to distinguish. If we overlap the bytes that we attack by one bit, the most significant bit in one attack will be the least significant bit in the next attack. Using this overlap technique we find all bits of a word except for the most significant bit. In the attack on Ed25519 we attack four words, that means we need to brute force four bits, so 16 possibilities. We do this by recomputing a valid signature with each possible key. We compare the computed signatures with the valid one we have, the key corresponding to the valid signature is the correct one.

We also overlapped the result with more bits. With 2,3 and 4 bits overlap we need to brute force four bits for each word. This means we need to brute force 2^{16} possibilities.

Figure 4.8 shows the results of the different overlap sizes for the different words that we need to attack to recover the key. As we can see, overlapping bits results in

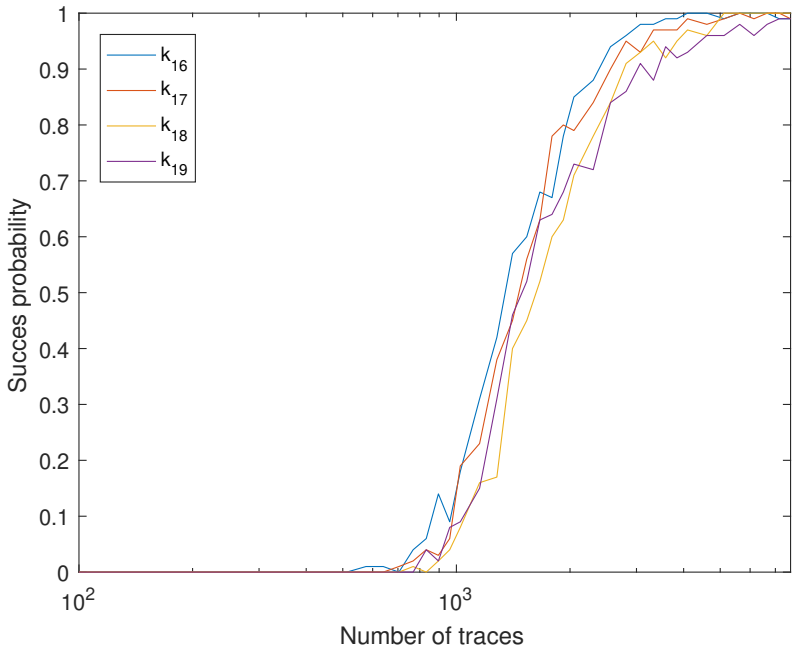


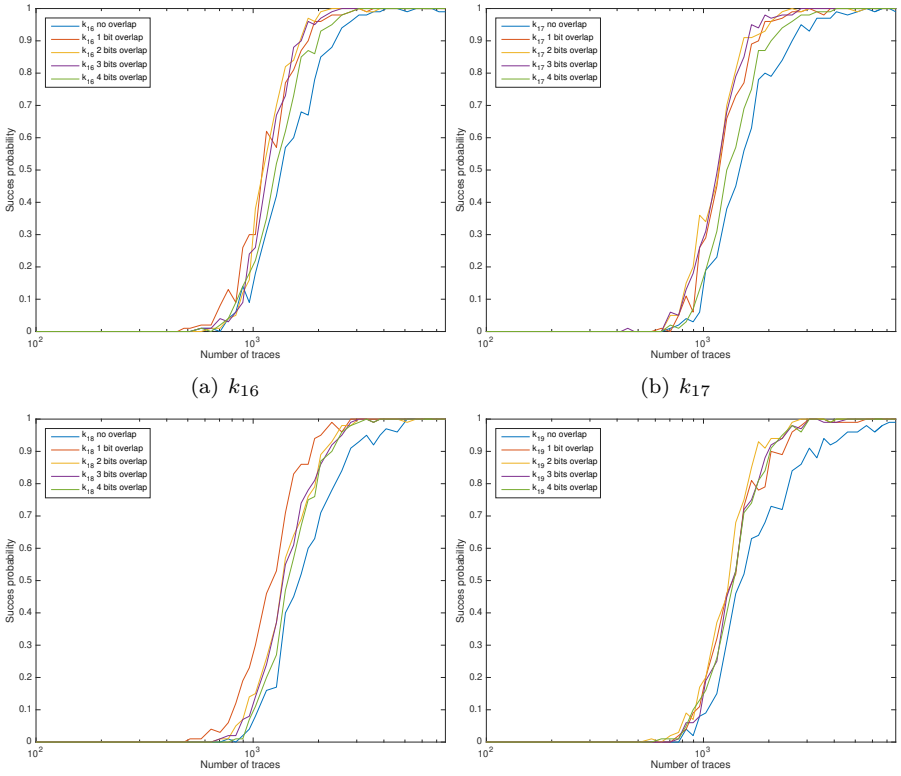
Figure 4.7: Success probability of the attack

a higher success probability. The difference between the amount of overlapped bits seems minimal and not consistent for each word. We already saw that we needed the highest amount of traces to distinguish the most significant bit correctly. Any amount of overlapping bits at least overlaps with the most significant bit. This causes the largest increase of the success probability. Overlapping a larger number of bits does not seem to affect the success probability relevantly.

4.4.5 Discussion and Countermeasure

With our presented attack, we are able to obtain the private scalar such that we can forge signatures by collecting the power measurements of only 4 000 signatures. This makes it a very practical attack and implementers of Ed25519 should take this into account.

The default protection would be the implementation of a protected version of SHA-512. Due to the use of boolean and arithmetic operations, the protection of SHA-1, SHA-2 and ARX algorithms in general is complex and could be quite costly [137, 198]. We have an alternative proposal that requires dropping the deterministic signature feature and adding some randomness in the computation of the ephemeral scalar. We need to create a scenario such that an attacker is not able to make a hypothesis on the constant key value. This can be achieved by padding the key with fresh random bits such that the first 1024-bit block is composed only by key and random value, without any bits known to the attacker. The input message will be processed in blocks after that. Fig. 4.9 visualizes how the input should look. The R_0 block would be a random



(a) k_{16} (b) k_{17}

(c) k_{18} (d) k_{19}

Figure 4.8: Success probability of the attack with overlap.

number of 768 bits. We argue that it is also possible to have an R_0 block composed by 128 bits of randomness and pad the rest of the block with 640 bits with a constant value (e.g. all zero).

The XEdDSA and VXEdDSA [283] signature schemes extend Ed25519 to generate a more robust ephemeral private scalar that is sufficiently random. Although XEdDSA and VXEdDSA also add random values into the signature scheme, XEdDSA is still vulnerable to our attack. As they append a random 64-byte sequence to the key and the message, the vulnerability that we exploit remains the same. VXEdDSA is not vulnerable to our attack but it requires several additional scalar multiplications that add to the computation time.

Obviously, this countermeasure kills the deterministic signature properties, but we do not see this as a dramatic problem. The main motivation for the proposal of deterministic signatures was to avoid a poor management of randomness that can introduce security problems [81, 145]. The proposed countermeasure is also not re-introducing the strong security requirement of randomness needed by ECDSA. Basically, even if the same randomness is used to sign two different messages, the attacker will not be able to recover the key as it would be possible with ECDSA.

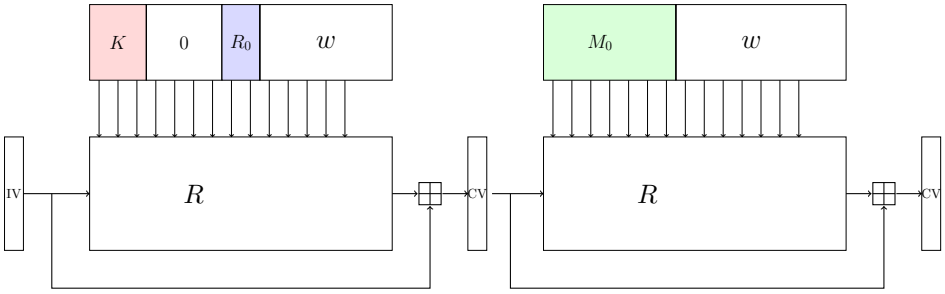


Figure 4.9: Generation of the ephemeral key with a countermeasure.

Additionally we want to highlight that the signature verification procedure remains as is.

As our final comment, in the recent developments of the IETF CFRG group for TLS 1.3, the hash function adopted for Ed448 is SHAKE256. In this case the protection against side-channel attacks such as power and EM based would be easier and pretty robust as explained by Chari et al. [79].

4.5 General Attack Principle - FI

In Ed25519, if an attacker is able to cause a glitch in the computation of the ephemeral public key $R = rB$, or in the computation of the hash $h = H(R, A, M)$ where the same message is signed resulting in R' or h', he can recover the private scalar. Independent of which of the two values is faulty, the hash computation is always faulty as it has R or R' as an input. For a successful attack, we need a correct signature and only a single faulty signature to recover the private scalar a. With private scalar a, a valid signature on any message can be computed as the value of r is arbitrary. From the correct and a faulty signature, the private scalar a can be recovered as follows. The attacker obtains a correct signature (R, S) and a faulty signature (R', S') with the following equations:

$$S = r + ha,$$
$$S' = r + h'a.$$

If we rewrite this, we obtain the following,

$$S - ha = S' - h'a.$$

And we can extract private scalar a

$$a = \frac{S - S'}{h - h'} \tag{4.10}$$

The output of the hash function h is not public so when a fault is injected in the computation of the hash, an attacker must know or be able to compute hash h'. It

can be brute forced as in [252] where the authors use an 8-bit architecture, but their attack does not scale so when a more realistic target is used like a 32-bit or 64-bit architecture, this becomes impractical. Since the ephemeral public key R is part of the signature (hence known), we aim at causing a glitch in the computation of the scalar multiplication $R = rB$. We do not target any particular single bit (or a group of bits), but a fault in any intermediate value of the scalar multiplication is sufficient.

For each execution of the signing algorithm there are three possible outcomes.

- Normal
- Inconclusive
- Successful

A normal outcome denotes the case when no fault occurred and the output is as expected. A successful outcome stands for an induced fault that resulted in the correct key by applying Eq. (4.10). An inconclusive outcome has several possibilities: (i) a fault was induced and a faulty output was produced but the key could not be recovered, (ii) a fault was induced but no output was produced, and (iii) a fault was induced but no output was produced and the device stopped working. At this point the device had to be power cycled to continue the experiment.

4.6 Experimental Setup and Results - FI

4.6.1 Setup

In this chapter, we consider two types of fault injection; voltage fault injection and electromagnetic fault injection. The setups for voltage and EM fault injection are very similar. Our target is a development board containing a Cortex-M4F, more specifically the STM32F407IG. For our experiments we did not have to decapsulate the chip. A signing operation of Ed25519 from WolfSSL takes roughly 30 ms on this platform. Electronic devices have capacitors to keep the power at a stable level so internal or external fluctuations do not influence the behavior of the device. Since we actually want to cause some fluctuations in the power line to alter the behavior with voltage FI, we removed most capacitors on the board. With EMFI we externally cause the fluctuations in the power plane with short EM pulses so the attack also works without removing the capacitors.

We use the VC Glitcher[3] to power the board and to cause fluctuations in the voltage. We also need the Glitch Amplifier as the VC Glitcher does not provide enough current to power the board.

To generate the EM pulses we use an EMFI Transient Probe that is connected and controller by the VC Glitcher. An xy-table is used to move the EMFI Transient Probe with high accuracy.

The oscilloscope used to visualize the effect of the voltage fluctuations or the EM pulses is a Picoscope5203. A current probe measuring those changes is connected in series with the power line and it is also a part of the setup. Figure 4.10 shows a picture of our experimental setup.

[3] `https://www.riscure.com/security-tools/hardware/`

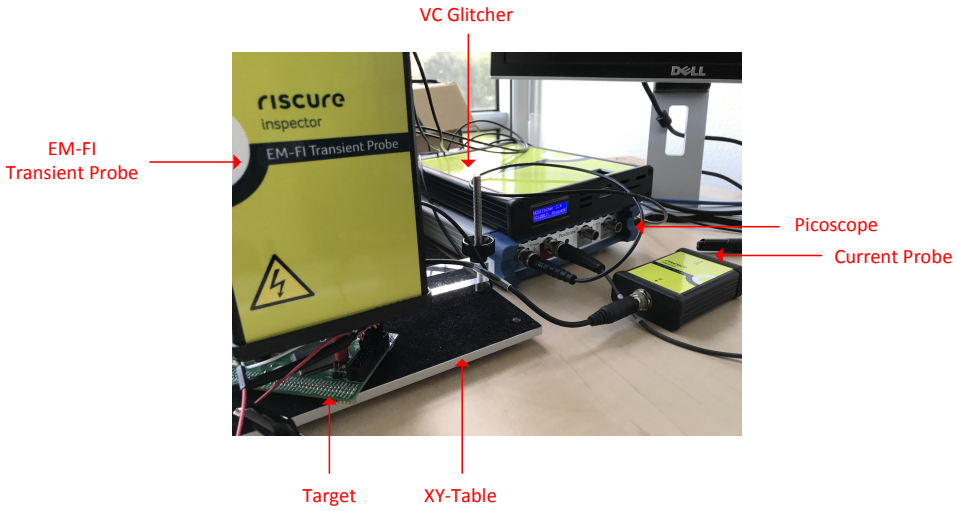


Figure 4.10: This figure shows the experimental setup. In the top left corner we see the EM-FI transient probe and below that the target board which is fixed to the XY-table. In the center with the blue display, we see the VC Glitcher under which is the oscilloscope. The small block on the right is the current probe.

We attack the software implementation of Ed25519 in WolfSSL version 3.11.0. Similar implementations can be found in other cryptographic libraries implementing Ed25519. We added a trigger to the code right before the start of the scalar multiplication. We could also have inserted the trigger before the signature generations starts as the code runs in constant time, so it would merely imply the increase in the offset. The attack is also possible without adding a trigger in the code when using hardware that can generate a trigger based on a pattern in the signal [34].

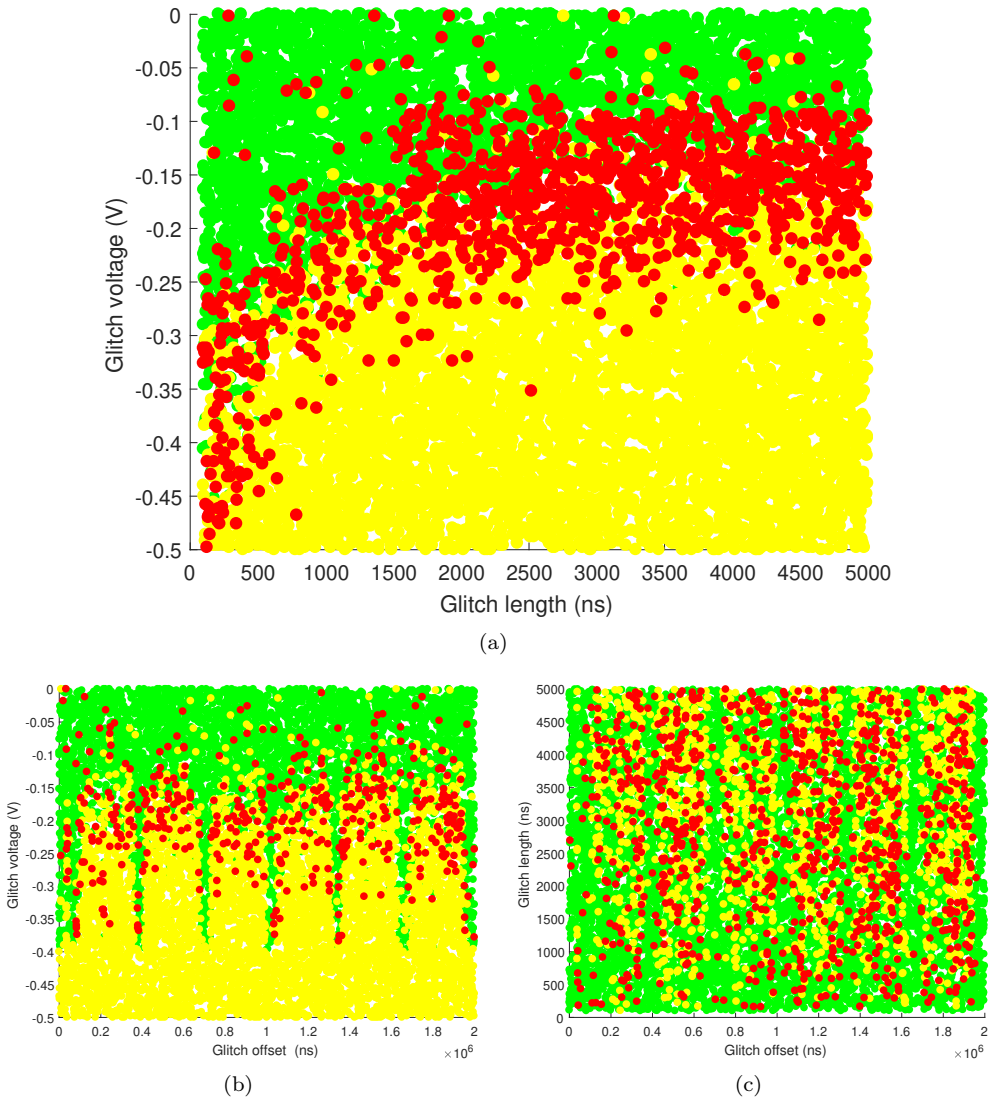
4.6.2 Voltage Fault Injection Results

The first step of the experiment was to continuously under-power the device. Without introducing glitches we were able to achieve a success rate of 44%. When we actively tried to induce glitches using the described parameters, we were able to increase the success rate to 69.95% using an optimal set of parameter values we found. We computed the success rates using 10 000 measurements with those optimal parameters.

To visualize the effects of the glitch parameters, we set the parameters to a constant value except for two. For those two parameters we selected a random value within a certain range. Fig. 4.11 shows the result of the experiment. In Fig. 4.11(a), we vary the glitch length and the glitch voltage parameters. It shows a typical curve of successful glitches as in [77, 232]. A selected set of parameters above the curve means the glitch is not strong enough and the device continues like nothing happened and a selection of parameters below the curve results in a glitch that is too strong and the device stops responding. In Fig. 4.11(b) and Fig. 4.11(c) we vary offset with the glitch

voltage and glitch length and we see a clear pattern that corresponds to an iteration in the scalar multiplication. Each figure contains results of 10 000 measurements.

The results show that inducing exploitable faults is not complicated as even providing a lower voltage results in a reasonably high success rate.



(a)

(b)

(c)

Figure 4.11: Voltage fault injection results, Normal (green), Inconclusive (yellow), Successful (red).

4.6.3 Electromagnetic Fault Injection Results

In this experiment we start by scanning the surface of the chip to determine good positions for a successful fault injection. Figure 4.12 shows the surface of the chip. We divide the x and y-axis up in 100 parts each, resulting in 10 000 positions to scan. We inject a glitch 20 times on each position where the remaining parameters are randomized similar as in the previous section. By manually optimizing the parameters, we were able to achieve a success rate of 99.31%. We did another surface scan with these fixed parameters, the result is shown in Fig. 4.13. The figure shows a heat map, where the color denotes the result of a fault. A color that is a mix between other colors is corresponding to the situations when the resulting faults were mixed.

To scan the surface we performed a total amount of 200 000 measurements. With the best selection of parameters, we used 10 000 measurements to compute the success rate.

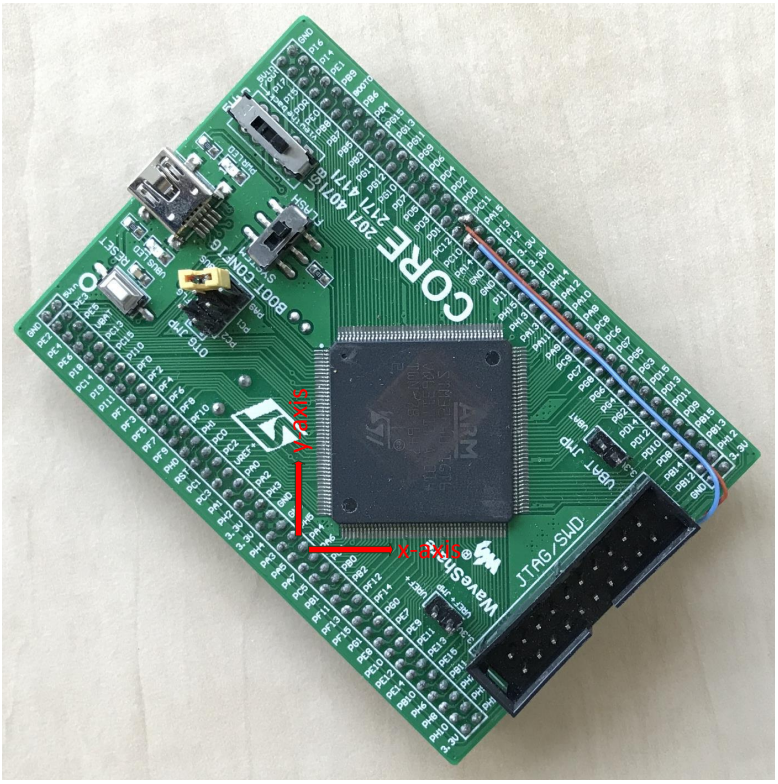


Figure 4.12: Target board

Since we induce a fault in the scalar multiplication, we expected most resulting points would not be on the curve anymore. In WolfSSL there is no check like this implemented, so we added it ourselves to let the signing operation fail in case of a fault as a countermeasure. We scanned the surface again with optimal parameters

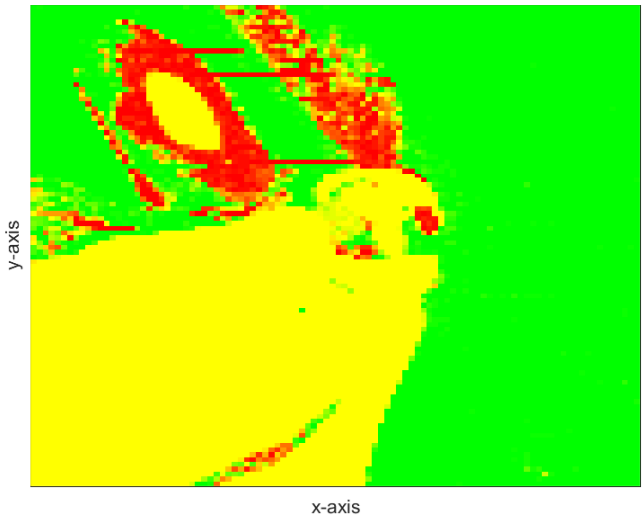


Figure 4.13: Heat map with EMFI results

and as expected not a single fault was successful.

To have a faulty scalar multiplication where the resulting point still is on the curve, we can modify the scalar itself. However, this is only possible while it is used within the scalar multiplication as the original scalar r must be used to compute S that is leading to the key recovery. In the implementation of WolfSSL (similar in other implementations) the scalar is copied and some computations are done to alter its representation. This takes roughly 36μs and gives us plenty of time to emit an EM pulse while the original scalar remains unaltered. Optimizing the parameters resulted in a success rate of 70.15%. Although the success rate with the check on the validity of the resulting point is lower, it is still high enough for the attack to remain practical.

4.6.4 Countermeasures

There are several approaches to count fault injection attacks, both in hardware [53, 162] and in software [17, 23]. Here we discuss some software countermeasures.

A countermeasure is to add redundancy in the implementation. For instance, in a common countermeasure, the implementation could execute the scalar multiplication again and at the end compare both results. If they are not identical, then the fault occurred and the signature should not be released. The signature could also be verified at the end, if the signature is invalid, do not return the signature.

However, there is a problem with adding redundancy as a countermeasure as this introduces a check in the code that an attacker also could try to skip by injecting a glitch. It also penalizes the performance. Although this adds a significant amount

of difficulty to the attack, an attacker only has to be successful once to be able to recover the secret key.

Another solution is to add randomness to the scheme. In [252] the authors propose a countermeasure called "fault infective computations" where 32 random bytes are used together with different implementations of the hash function for each time the hash function is used in the scheme. A second implementation of the hash function adds to the code size and may not be preferred due to resource constraints.

In this attack we exploit that ephemeral scalar r is equal in both signatures. Introducing some randomness in the generation of r counters our attack. New standards where randomness is introduced in the generation of r are proposed like XEdDSA and VXEdDSA [283]. In these schemes 64 random bytes are added, the VXEdDSA scheme also requires several additional scalar multiplications. In Section 4.4.5, we proposed a cheap countermeasure that requires only 16 random bytes that are used in the generation of r which is also effective against this attack. The randomness does not have to be perfect as with ECDSA as long as the bytes remain unknown to an attacker. Signatures generated using this countermeasure are still verifiable and conform the standard.

4.7 Conclusion

In this chapter we presented a side-channel attack and a fault attack on the digital signature scheme Ed25519. By measuring the power consumption of approximately 4 000 signatures we were able to recover the auxiliary key of a signature. We can use the auxiliary key to recover the private scalar that we can use to forge signatures.

We recover the auxiliary key by executing a side-channel attack on SHA-512. We described an attack on the message schedule that is applicable to all applications where a constant secret is hashed together with a variable known input, if the length of the secret is shorter then the block size.

In case of the fault attack, only a single successful fault is required to recover the private scalar. We showed that with both voltage fault injection and electromagnetic fault injection, it is possible to achieve a success rate of nearly 100%.

The attacks we presented poses a real threat to implementation of the signature scheme such as on embedded devices or devices in IoT, if an attacker is able to measure the power consumption or has access to the device. Additionally, we propose a countermeasure to counter these attacks.

Chapter 5

Systematic Evaluation of Countermeasures for Curve25519 Cryptographic Implementations

5.1 Introduction

Elliptic-curve cryptography (ECC) presents the current state of the art in public-key cryptography, both for key agreement and for digital signatures. The reason for ECC getting ahead of RSA is mainly in its suitability for constrained devices due to much shorter keys and certificates, which results in ECC consuming much less energy and power than RSA. Also, ECC has an impressive security record: while certain classes of elliptic curves have shown to be weak [200, 274], attacks against ECC with a conservative choice of curve have not seen any substantial improvement since ECC was proposed in the mid-80s independently by Koblitz [170] and Miller [204]. The best known attack against ECC is still the Pollard rho algorithm [235] and its parallel version by van Oorschot and Wiener [220].

The situation is different for *implementation attacks* against elliptic-curve cryptosystems. Since Kocher put forward the concept of side-channel analysis (SCA) against cryptographic implementations and fault injection (FI) analysis, i.e., the Bellcore attacks, was introduced in the late 90s [171, 172, 69], research into attacks against implementations of ECC and suitable countermeasures has been a very active area. The most common forms of SCA are Simple Power Analysis (SPA) and Differential Power Analysis (DPA) (both introduced in [172]), and profiled attacks. SPA involves visually interpreting power consumption traces over time to recover the secret key while DPA relies on performing a statistical analysis between intermediate values of cryptographic computations and the corresponding side-channel traces.Profiling attacks, most notably template attacks (TAs), were introduced in [80]. Here the adversary first uses a device (for instance, a copy of the attacked device) that is under his/her full control to characterize device leakage; this characterization information is called a *template* for TAs. Then the adversary uses that information to efficiently recover the secret data from the device under attack.

We will review the relevant classes of attacks later in this chapter, but to name a

few recent real-world examples of such implementation attacks against ECC consider the "Minerva" timing attack against multiple implementations of ECDSA [157], or the power-analysis attack recovering the secret key from a TREZOR hardware bitcoin wallet [147]. In fact, the research area has been so active that it produced at least four survey papers reviewing the state of the art in attacks and countermeasures at different points in time [114, 113, 97, 1].

Complete, specific, additive, and public. The survey by Fan and Verbauwhede from 2013 [114] concludes that a side-channel-protected implementation of ECC needs to satisfy three properties:

- the protection mechanisms need to be *complete*, i.e., protect against all relevant attacks;
- they need to be *specific* to the application scenario and resulting attacker model; and
- they need to *additive*, i.e., combined in a way that does not introduce new vulnerabilities.

In this chapter we add a fourth attribute to this list: to enable independent verification of these properties, the implementation needs to be *public*.

Surprisingly, despite the extensive body of literature on implementation attacks on ECC, we are not aware of a single paper describing an implementation fulfilling these four properties or even making somewhat substantiated claims to do so. For a more detailed discussion see the section on related work below.

Note that there are implementations of ECC that claim to fulfill the three properties introduced in [114]. For example, NXP advertises the SmartMX2-P40 *"secure smart card controller"* as supporting *"DES, AES, ECC, RSA cryptography, hash computation, and random-number generation"* and claims that the protection mechanisms are *"neutralizing all side channel and fault attacks as well as reverse engineering efforts"* [218]. However, like essentially all commercial smart cards, all implementation details are kept secret, which actively prevents academic discourse and public evaluation of these claims. As a result of this situation many papers are presenting side-channel attacks against ECC target implementations that never claimed protection against such attacks; for example, see [254, 129].

We do not mean to say that there are no public implementations of ECC including *certain* countermeasures against some specific attacks. On the contrary: most papers describing attacks also suggest countermeasures and some of these papers also present implementations of those countermeasures; see the section on related work below. However, more than 3 decades after the invention of ECC and 2 decades after the invention of side-channel attacks we still don't know the cost of ECC implementations with *complete*, *specific*, and *additive* SCA countermeasures, or if such implementations are even achievable with respect to the budgets in area, memory, power and energy that are typically sparse for low-cost devices.

In this chapter we present the first—to our knowledge—publicly available implementation of a widely used ECC primitive claiming to fulfill the three properties for a concrete real-world application scenario and assuming a rather strong adversarial model on both, side-channel and fault analysis.

The overhead required by our protected implementations is about 36% for the ephemeral implementation and 239% for the static one. While this might seem a relatively hefty price to be paid for security, note that it is rather modest in comparison to securing AES on ARM architectures. In particular, Schwabe et al. [265] show that masked bitsliced AES has an overhead of 458% compared to unprotected bitsliced AES and of 1339% compared to a table-based AES; all on the same ARM Cortex-M4 platform we consider in this chapter. Moreover, we also show in Section 5.5.1 that even our most protected static implementation is more efficient than widely-deployed ECC cryptographic libraries, which provide much less SCA protections.

Security evaluation vs. formal proofs. The current state of the art of formally verifying side-channel protection of asymmetric cryptography is to prove the absence of timing leakage with tools such as ct-verif [11] (on LLVM IR level) or Binsec/Rel [98] (on binary level). Computer-verified proofs of side-channel protections beyond timing-attack resistance are much further in the symmetric-cryptography realm (see, e.g., [206, 32, 111, 26, 25]) and we believe that it is a very important direction of research to extend such results also to asymmetric cryptography. For example, we believe that it is possible to prove that our static X25519 implementation is secure against first-order DPA attacks, but such a proof would require significant advances of the currently available tools for formal verification. For many attacks that are we aim to protect against in this chapter because of their real-world importance—most notably profiling attacks and fault-injection attacks—we are not even aware of a widely accepted sound and complete *definition* of what it means for an implementation to be secure. This is why we follow the current state of the art in practical security evaluation and provide extensive experimental evidence to evaluate the security of our implementations.

Application scenario. Long-term secrets as used in authentication protocols typically form the most precious assets of any security infrastructure. Today, sophisticated communication capabilities are incorporated in more and more systems that formerly used to operate in a stand-alone setting. One prominent example are Internet-of-Things (IoT) devices, serving applications both for industry as well as consumer applications. Most of these devices today do not incorporate dedicated cryptographic hardware designed for adequately protecting long-term-keys.

Moreover, physical access to many devices cannot be effectively restricted, making installations significantly more vulnerable to implementation attacks exploiting all kinds of leakage such as electromagnetic emanation, power consumption etc., than typical for the traditional office or server-room setting. This notably applies to large-scale or decentralized industrial plants such as common in petrochemistry or local drinking-water wells.

In most critical-infrastructure systems, conventional smart-card circuits cannot be used, specifically when considering constraints imposed by power-budget or rough environmental conditions or frequent update requirements.

For instance, wireless modules for off-the shelf industrial sensors often have to operate within a power budget of less than 2 mW [141] clearly exceeded by most off-the-shelf smart-card chips. Appliances for medical use or for production of food and drugs might have to withstand high sterilization temperatures. Conventional smart-card circuits are typically not designed for these environments, forcing implementers

to choose conventional microcontrollers specifically designed for the respective constraints.

Moreover note that in many consumer applications, commercial pressure pushes designers to use conventional microcontrollers for cryptographic operations instead of dedicated smart-card circuitry.

One aspect which should be considered is that customization of typical smart-card chipsets is often impossible. For high-volume applications specialized software, such as java-card applets is common. However, the access to freely programmable circuits is limited. For applications with smaller market volumes tailored solutions mostly could not be realized. The common "turn-key" smart-card products on the other hand often allow only for very limited flexibility regarding the supported cryptographic protocols.

Why consider software only? The side-channel and fault-attack-protected implementation of X25519 that we present in this chapter is a software-only implementation targeting the ARM Cortex-M4 microcontroller. Some would argue that applications that are seriously concerned about side-channel security will rely (at least to some extent) on countermeasures implemented in hardware.

The main reason that we go for a software-only implementation is the application scenario described above, which cannot afford to use specialized hardware[1] Clearly this application scenario calls for an evaluation of how far we can go in software-only countermeasures. We also stress that, while we use this application as a motivation, there are other scenarios —for example, many open-source projects—that cannot afford producing special-purpose hardware.

However, another reason for following a software-only approach is our goal to encourage independent security evaluation of our implementation. Development boards featuring our target platform are widely available for around US$ 20, allowing any side-channel researcher to try to attack our implementation.

Why target X25519? The concrete primitive we target with side-channel and fault-attack protections in this chapter is the X25519 elliptic-curve key-exchange originally proposed under the name “Curve25519” by Bernstein in [43]. To avoid confusion between naming of the specific elliptic-curve used in the protocol and the protocol itself, Bernstein later suggested to call the curve Curve25519 and the protocol X25519 [41].

One reason to target this primitive is its widespread use in many state-of-the-art security protocols like in TLS [217], SSH [4], the Noise Framework [288, Sec. 12], the Signal protocol [281], and Tor [196]. For a more extensive list see [150]. Moreover X25519 is now also considered as primitive for the OPC/UA protocol family [116] by the OPC/UA security working group, specifically targeting IoT applications with both increased protection demands and low computational resources.

Another reason is that in this chapter we make an attempt to settle discussions about the cost of SCA-protected X25519 implementations. On the one hand it is generally acknowledged that the “Montgomery ladder” (see Section 5.2), which is typically used in X25519 implementations, is a good starting point for SCA protected implementations. On the other hand, during standardization of Curve25519

[1] While we concentrate on software only, note that our implementation can be relatively easily modified to use a hardware co-processor for modular arithmetic instead of our assembler arithmetic routines, without modifying higher-level SCA countermeasures.

for TLS, concerns were expressed about the structure of the underlying finite field and group order [182, Sec. 3.1] leading to a significant increase of the cost of side-channel protections. The main argument refers to the presumed need of significantly larger scalar blinding. We show that these concerns can be resolved without excessive performance penalties in comparison to, e.g., curves over fields without structure allowing for fast reduction.

Also the fact that the full group of curve points has a co-factor of 8 has raised concerns in the context of SCA [131]. As one way to alleviate these subgroup concerns it is often suggested to validate inputs [13, 104, 16]; however Bernstein's Curve25519 FAQ [42] states *"How do I validate Curve25519 public keys? Don't."*. For the sake of security we have decided to include a full on-curve check for the static implementation and an early-abort strategy on certain malicious inputs for the ephemeral case.

5.1.1 Related work

There is a vast amount of literature devoted to the topic of side-channel secure implementations of ECC in both, software and hardware. Most related work considers different types of curves, but on the same platform, i.e., the ARM Cortex-M4 microcontroller such as [180] or the same curve but without such a comprehensive side-channel protection [121, 99].

In [180] a range of high-speed implementations are proposed, including: scalar multiplication, ECDH key exchange, and digital signatures on various embedded devices using the Four$\mathbb{Q}$ elliptic curve of Costello and Longa. They also add countermeasures to thwart a variety of side-channel attacks. All implementations are constant time and include countermeasures such as scalar and projective coordinate randomization and point blinding. The implementation with the countermeasures resulted in a slowdown of factor 2 for the ARM platform. To validate the effectiveness of the countermeasures, leakage detection is performed using test vector leakage assessment (TVLA) [33]. They show an improved resistance to SCA, as expected. DPA also failed when countermeasures were deployed. However, active and profiling adversaries are not considered. Hence, this work does not offer a comprehensive evaluation but rather a solid benchmark in evaluating trade-offs for certain classes of side-channel attacks (SPA and DPA).

Fujii and Aranha [121] present an X25519 implementation that is protected against timing attacks by constant-time execution and also randomized projective coordinates and constant-time conditional swaps were implemented. However, the authors do not specify any details about those countermeasures and they do not consider protection against other side-channel attacks.

Considering hardware implementations, Sasdrich and Güneysu performed extensive investigations of costs for countermeasures in hardware, i.e., on an FPGA platform for Curve25519 and Curve448 [257, 258]. In both cases, the Montgomery ladder was deployed to provide a basic protection against timing and Simple Power Analysis (SPA). To offer some DPA protection, point randomization and scalar blinding were added, increasing the amount of look-up tables (LUTs) by 5% and of flip-flops (FFs) by 40% and increasing the overall latency in terms of clock cycles by 45% for Curve448. For Curve25519 the performance penalty was 30% with a similar increase in required LUTs and FFs. In addition, the authors add memory address scrambling

to secure the memory accesses against side-channel attacks and correspondingly make DPA more complex. For this purpose 2^6 different random addresses were used.

A protected implementation of the new complete formulas for Weierstrass curve from [242] is presented in [84]. The implementation is put on an FPGA and evaluated against SCA. More specifically, three different versions were evaluated: (1) an unprotected architecture; (2) an architecture protected through coordinate randomization; and (3) an architecture with both coordinate randomization and scalar splitting. The evaluation is done through timing analysis and TVLA. The results show that applying an increasing level of countermeasures leads to an improved resistance against SCA, but they only consider a side-channel adversary of a limited capacity i.e. being only passive and not capable of profiling.

Availability of the software. We place all software described in this chapter into the public domain. It is available from an anonymous GitHub repository at `https://anonymous.4open.science/r/40fe2d05-17f5-439c-9e89-7a4737e3322e/`.

5.1.2 Contributions

For this chapter, we summarize our contributions as follows:

- We survey state of the art attacks and countermeasures for elliptic curve cryptography.
- We implement and evaluate countermeasures for X25519 in case of ephemeral keys.
- We implement and evaluate countermeasures for X25519 in case of static keys.

5.1.3 Organization of this chapter

Section 5.2 introduces notation and gives the necessary background on X25519 key exchange, the ARM Cortex-M4 microcontroller, and describes our attacker model in detail. In Section 5.3 we consider the (easier) case of protecting X25519 for ephemeral key exchange and in Section 5.4 we build up to the more complex case of protecting X25519 with static keys. In Section 5.5 we describe the lab experiments we carried out to support our claims. Finally, in Section 5.6 we conclude this chapter and give an overview of what we believe to be the most important directions of future work.

5.2 Preliminaries

In this section we introduce the necessary background on the X25519 key-exchange protocol and on the ARM Cortex-M4 microcontroller. Furthermore we introduce our attacker model.

5.2.1 X25519 key exchange

The X25519 elliptic-curve key-exchange protocol is based on arithmetic on the elliptic curve in Montgomery form [205]:

$$E : y^2 = x^3 + 486662x^2 + x$$

over the finite field $\mathbb{F}_p$ with $p = 2^{255} - 19$. The group of $\mathbb{F}_p$-rational points on E has order $8 \cdot \ell$, where ℓ is a 252-bit prime. The central operation is scalar multiplication $\mathsf{smult}(k, x_P)$, which receives as input two 32-byte arrays k and x_P. Each of those arrays is interpreted as a 256-bit integer in little-endian encoding; the integer x_P is further interpreted as an element of $\mathbb{F}_p$. The scalar-multiplication routine first sets the most significant bit of x_P to zero (ensuring that $x_P \in \{0, \ldots, 2^{255} - 1\}$ and sets the least significant 3 bits of k and the most significant bit of k to zero, and the second-most significant bit of k to one (ensuring that $k \in 8 \cdot \{2^{251}, \ldots, 2^{252} - 1\}$). This operation on bits of the input k is often referred to as "clamping", in our pseudocode we denote it by clamp. Subsequently, scalar multiplication outputs the x-coordinate $x_{[k]P}$ of the point $[k]P$ where P is one of the two points with x-coordinate x_P on E (if there are such points) or the quadratic twist of E (otherwise), and where $[k]$ denotes scalar multiplication by k. The scalar multiplication is commonly implemented using the Montgomery ladder [205] using a projective representation $(X : Z)$ of an x-coordinate $x = X/Z$. Pseudocode for this ladder is given in Algorithm 8.

Algorithm 8 The Montgomery ladder for x-coordinate-based scalar multiplication on $E : y^2 = x^3 + 486662x^2 + x$

Input: $k \in \{0, \ldots, 2^{255} - 1\}$ and the x-coord. x_P of point P
Output: $x_{[k]P}$, the x-coordinate of $[k]P$

```
X_1 ← 1; Z_1 ← 0; X_2 ← x_P; Z_2 ← 1, p ← 0
for i ← 254 downto 0 do
    c ← k[i] ⊕ p                                  ▷ b[i] denotes bit i of k
    p ← b[i]
    (X_1, X_2) ← cswap(X_1, X_2, c)
    (Z_1, Z_2) ← cswap(Z_1, Z_2, c)
    (X_1, Z_1, X_2, Z_2) ← ladderstep(x_P, X_1, Z_1, X_2, Z_2)

return (X_1, Z_1)
```

This algorithm uses two sub-routines cswap and $\mathsf{ladderstep}$. The cswap ("conditional swap") routines swaps the first two inputs iff the last input is 1. The $\mathsf{ladderstep}$ routine, on input coordinates $x_{Q-P}, X_P, Z_P, X_Q, Z_Q$, computes $(X_{[2]P}, Z_{[2]P}, X_{P+Q}, Z_{P+Q})$. This computation takes 5 multiplications, 4 squarings, one multiplication by a small constant, and a few additions and subtractions in $\mathbb{F}_{2^{255}-19}$.

The key-exchange protocol uses this smult function and the fixed basepoint $x_B = [9, 0, \ldots, 0]$ and proceeds with straight-forward elliptic-curve Diffie-Hellman.

5.2.2 The ARM Cortex-M4 microcontroller

Our target platform is the ARM Cortex-M4 microprocessor, which implements the ARMv7ME 32-bit RISC architecture [280]. The most relevant features of this platform for the software described in this chapter are the following:

- it features 16 general-purpose 32-bit registers, out of which 14 are freely usable (one is used as instruction pointer and one as stack pointer);

- it features a single-cycle 32×32-bit multiplier producing a 64-bit result, which can be accumulated for free through a fused multiply-accumulate instruction.

The Cortex-M4 features a three-stage pipeline. Recent findings [307, 269, 197] indicate that specific properties of the internal pipeline architecture are highly relevant for the amount of side-channel leakage generated. In the instruction set, input and output operands of the ALU are retrieved from and stored to the register file. However, based on the analysis of [307] and own findings, we presume that the Cortex-M4 features also shortcut data paths which generate additional leakage when the result of a first arithmetic or logic instruction in the pipeline is required as input operand for a subsequent instruction within the pipeline.

Randomness generation. The side-channel countermeasures require a source of uniformly random bytes. Random-number generation is not part of the Cortex-M4, however the specific STM32F407 device that we used for evaluation features a hardware random number generator, which generates 4 random bytes every 40 clock cycles [280]. We use this hardware RNG for all randomness generation. In some contexts we need uniformly random values modulo p or modulo ℓ. In those cases we sample a 512-bit integer and reduce modulo q or ℓ. This approach is also used, for example, for sampling close to uniformly random modulo ℓ in the Ed25519 signature scheme [46, 47]. Compared to a software RNG, the hardware RNG provides faster and higher quality randomness.

Fast X25519 on the Cortex-M4. Multiple earlier papers describe optimized implementations of X25519 on the ARM Cortex-M4 [121, 120]. The speed-record within the scientific literature is currently held by an implementation by Haase and Labrique described in [140]. They report 625 358 cycles for one scalar multiplication on an STM32F407 running at 16 MHz. The optimized routines for field arithmetic from this implementation are in the public domain and available from `https://github.com/BjoernMHaase/fe25519`. They are the starting point for our protected implementations.

5.2.3 Attacker model

Our attacker model is motivated by typical capabilities of a real-world attacker. We assume that an attacker controls the input to the scalar multiplication and obtains the contents of the output buffer after the computation has finished. Furthermore, we assume certain capabilities with respect to the side-channel leakage that becomes available (passive attacker) and FI capability (active attacker).

Passive side-channel attacker. We assume that an attacker can collect sufficiently many (noisy) traces of power or EM leakage together with the chosen input and corresponding output. In addition, we allow for the attacker to generate templates (as in profiled attacks) for self-controlled inputs on a device of the same make and model as the target device. For our experiments we generate templates on the same device as the one we target in the attack.

As mentioned above, instead of proving the side-channel security of our implementation in a certain model (e.g., the noisy leakage model [236]), we use the TVLA methodology (t-test analysis) to show the absence of leakage in traces collected from

an actual device running our implementation. This kind of analysis, although not perfect, is standard for security evaluation labs [153, 302].

A limited fault attacker. A software-only implementation, such as the one we describe in this chapter, cannot be protected against arbitrarily powerful fault attackers. The reason is that such an attacker could simply skip the execution of the program over any dedicated countermeasure or rewrite values in registers and memory to eliminate all effects of fault-attack countermeasures. The fault attacker we consider in this chapter is therefore limited to injecting only one fault per scalar multiplication. We assume that this fault may either skip a short block of consecutive instructions, set an arbitrary register value to zero, or set an arbitrary register value to a random value not controlled by or known to the attacker. These faults are the most common to occur in practice based on our experience and on reviewed literature; see, for example, [286].

Restricting instruction skips to "short" blocks is motivated by the fact that in practice attacks are typically able to skip only up to 2–3 instructions. Furthermore allowing arbitrarily long blocks of code to be skipped would enable trivial attack skipping all cryptographic computations. For similar reasons we also exclude the instruction pointer from the set of "arbitrary" registers that the attacker may fault. Note also that we consider attacks skipping over the function call to our protected implementations out of scope – such attacks need to be protected by the surrounding context.

5.3 SCA-protected ephemeral X25519

In this section we describe our implementation of X25519 protected against side-channel and fault attacks when deployed in an *ephemeral* key-exchange scenario. In such a scenario each secret scalar is used only twice, once in public-key generation and once in the computation of the shared secret key. By itself such an ephemeral key exchange makes little sense in most communication scenarios because communication partners are unauthenticated. However, combined with either signatures in a SIGMA-style protocol [176] or in combination with static key exchanges in, for example, 3DH [194], it is a critical building block to achieve forward secrecy.

The goal of an SCA or FI attacker against such an ephemeral key exchange is to learn the shared secret. This goal can be achieved either by learning the secret scalar or by influencing the scalar multiplication through FI during both key generation and shared-key computation (and scalar multiplication is the main part of that) to an easily guessable value. The advantage of having two instead of just one trace is very small for a passive attacker – it is certainly too few traces to allow any kind of differential attacks (see Section 5.4).

As we focus on protecting the scalar multiplication, we consider generation of the 32-byte secret key as out of scope and assume that it is provided as an input by the user of the library.

In Subsection 5.3.1 we list most relevant passive attacks and in Subsection 5.3.2 we list most relevant active attacks for the ephemeral setting. Due to space constraints we do not list all possible SCA or FI attacks, but only focus on the most relevant ones. For a complete list of attacks we refer the reader to either [114] or [163].

5.3.1 Relevant passive attacks

SPA. Simple power analysis (SPA) attacks were introduced in [172]. Note that we do not emphasize on the word "power", but also include attacks that use a trace of electromagnetic radiation or any other side-channel. Modern definitions of SPA often include any attack that works with side-channel information of a single execution. This definition would include horizontal and certain template attacks. We discuss those separately and hence define SPA attacks in the more classical sense as attacks that visually identify secret-data-dependent control flow. Simply speaking, SPA attacks typically target instruction-dependent leakages that are often data-dependent.

Horizontal (collision) attacks. A class of single-trace attacks is often called *horizontal* attacks. These attacks trace back to [294], and were applied against ECC in [87, 30, 143]. The general idea of horizontal collision is to use key-dependent collisions of the same values across multiple iterations of the scalar multiplication loop. Consider, for example, an attacker able to distinguish whether two finite-field multiplications have an input in common. If this is the case then the attacker can learn whether two scalar multiplication iterations share the same scalar bit. Such single-trace attacks have been described against RSA [294] and ECC [143].

Horizontal correlation attacks. In [87], the authors proposed an attack based on predictions of intermediate arithmetic results within a single trace. Although this method is effective against private-key blinding countermeasures, other mitigation methods, e.g., message or point randomization, prevent this horizontal attack.

Template attacks. Template attacks (TAs), introduced in [80], require a profiling stage in which the adversary estimating the probability distribution of the leaked information to make a better use of information present in each sample. TA is the best attack technique from an information-theoretic point of view but it makes strong assumptions on the attacker, e.g., that they can collect an unbounded number of template traces and the noise is close to Gaussian.

An approach from [199], is to target the internal state of the ladder step after a small number of scalar bits are processed. If a single-trace TA is successful then the number of bits is recovered and the attack can continue on the subsequent bits targeting the same trace.

Another relevant profiling attack is the TA targeting the conditional move (cmov) instruction as introduced in [212]. The cmov instructions is often used to implement cswap. The idea of this attack in our context is to learn about the behavior of cswap and create the corresponding templates (from multiple traces). Subsequently, the templates can be used to recover scalar bits from all cswap operations used in a single scalar multiplication. The attack is complex, but it can be successful even against a single trace.

The third relevant class for TA attacks key-transfer [222]. Note that in most implementations the scalar is copied, from flash to RAM or within RAM, just before the scalar multiplication is executed. This might allow the attacker to template the copy process and try to recover information about the scalar or the session key (i.e., the output of the scalar multiplication) from a single attacked trace.

Deep Learning Attacks. Another class of profiled attacks, similar to TAs, are deep learning (DL) side-channel attacks [75, 167, 189]. These attacks use algorithms like multilayer perceptron (MLP) or convolutional neural networks (CNNs) to recover the secret keys from cryptographic implementations. Their main advantage over TAs is that they do not require pre-processing of leakage traces, require less trace alignment, and make the attack simpler to run. They can be applied to attack ECC as demonstrated in [299, 298, 310]. A similar attack, but unsupervised, is presented in [230].

Online Template Attacks. Online template attacks (OTA) [29] use horizontal techniques to exploit the fact that an internal state of scalar multiplication depends only on the known input and the scalar. Advanced types of those attacks need only one leakage trace and can defeat implementations protected only with scalar blinding or splitting. OTA traces back and extends the doubling and collision attacks [119, 149]. These kind of attacks are thwarted by randomizing the internal state using point blinding and projective coordinate randomization [90] or coordinate re-randomization [212].

Horizontal cmov attacks. Horizontal cmov attacks [211] are similar to TA from [212], but they are unsupervised. They combine a heuristic approach based on clustering with multiple-trace unsupervised points-of-interest selection. Due to the relaxation of the attack setting, these attacks have slightly lower success rate and their complexity is increased in comparison to [212].

5.3.2 Relevant active attacks

Weak curve attacks. The first weak-curve attack on ECC was introduced in [62]. The key observation is that a_6 in the definition of E is not used in the addition. Hence, the addition formula for curve E generates correct results for any curve E' that differs from E only in a_6: $y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a'_6$. Therefore, the adversary can cheat by providing a point P on the insecure curve E'. Then the adversary can solve ECDLP on E' and find out the scalar.

The method of moving a scalar multiplication from a strong curve E to a weak curve E' often requires fault induction. With the help of faults, the adversary can make use of invalid points [62], invalid curves [85], and twist curves [117] to hit a weak curve.

Loop-abort attacks. A rather simple fault-injection attack is to jump out of the scalar multiplication main loop after only a few iterations or completely skip over the loop, which makes the resulting output easily predictable. An attacker against an ephemeral key exchange who is able to jump out of the loop in the same iteration in both key generation and shared-key computation can force the device into generating a weak, easily predictable shared key.

Key Shortening. Another simple fault injection attack is to shorten the scalar. The idea of the attacker is to stop the copy loop of the scalar before it is being used in the scalar multiplication. This way entropy in the scalar is reduced, if the previous memory content is predictable (e.g., if it is zeroed).

Fixing scalar-multiplication output. A straightforward attack to enforce predictable output is to prevent copying of the final result or use a fault to directly write, e.g., zero values, to the output buffer.

5.3.3 Protected implementation

Algorithm 9 presents a pseudocode of our protected implementation of ephemeral X25519. Like most previous X25519 implementations it is running in "constant time", i.e., is not leaking any information about the scalar through timing. Furthermore, we add the following countermeasures: re-randomizing the projective representation in the cswaprr procedure and control-flow counter.

The strategy of our cswaprr, which merges conditional swapping with projective re-randomization, takes into account that memory access leaks significantly more than register operations. We thus fetch input words from memory, conditionally swap and randomize in registers and then store the results back. Randomization here means multiplying both the X and the Z coordinate by a 29-bit random value. We also considered the risk of increased leakage due to shortcut data paths in the core's pipeline. We avoided using a result from an immediately preceding instruction as input for the subsequent operations if operands hold information on the secret scalar.

We moreover take into account that the CPU core always processes 32 bits simultaneously in its single-cycle multiplication and logic instructions. We aim at increasing the noise level by embedding random data into unused operand bits in addition to the confidential information. For instance, conditional moves are implemented on a half-word basis by using two multiplicative masks M_1,M_2 having random data in bits 16 to 31. Bits 15 to 0 are zeroed and the condition (M_1) and negated condition (M_2) is held in bit 0. For inputs A, B, the calculation $A' \leftarrow M_1 \cdot A + M_2 \cdot B; B' \leftarrow M_1 \cdot B + M_2 \cdot A;$ yields the swapped lower half-words of the inputs in bits 0 to 15.

In the following we explain why this implementation is not vulnerable against the attacks from the last two subsections.

SPA attacks are thwarted by using secret-independent control flow. This is rather clearly achieved in the Montgomery ladder, as long as the conditional-swap operation (implemented by us in cswaprr) is carefully implemented without conditional branches.

Using a ladder to compute the scalar multiplication is a commonly recommended countermeasure against "classical" SPA attacks; see, for example, [172].

Horizontal correlation attacks rely on the fact that identical values (i.e., field elements) are used across two or more loop iterations. In our implementation, this class of attacks is thwarted by re-randomizing the projective representation of the two points in the state in each iteration. This re-randomization is merged into the cswaprr operation as detailed above. Each re-randomization is using 29 bits of randomness, which leaves a small chance that occasionally this random value is of a special form that does not fully remove correlation (e.g., the randomizer could be 1 or 2). We do not expect this to be a problem in practice, as an attacker would only very occasionally be able to learn one or two bits of an ephemeral scalar by exploiting these rare correlations.

Horizontal, Template, Deep Learning, and Online Template attacks extract information about the secret scalar by identifying temporary values used in the

Algorithm 9 Pseudocode of side-channel and fault-attack protected ephemeral X25519

Input: A 255-bit scalar k and the x-coordinate x_P of some point P
Output: $x_{[k]P}$

1: $ctr \leftarrow 0$ ▷ Initialize iteration counter
2: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Initialize output buffer to random bytes
3: $k \leftarrow \mathsf{clamp}(k)$
4: $k \leftarrow k/8$ ▷ Divide scalar k by 8 to account for initial 3 doublings
5: `Increase(`ctr`)`
6: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(x_p, 1)$
7: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(X_p, Z_P)$
8: $(X_P, Z_P) \leftarrow \mathsf{montdouble}(X_p, Z_P)$ ▷ 3 doublings to multiply by co-factor 8
9: `Increase(`ctr`)`
10: **if** $Z_P = 0$ **then**
11: **go to** Line 25 ▷ Early-abort if input point is in order-8 subgroup
12: $x_P \leftarrow X_P \cdot Z_P^{-1}$ ▷ Return to affine x-coordinate
13: $X_1 \leftarrow 1, Z_1 \leftarrow 0$
14: $Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$ ▷ Initial randomization of projective representation
15: $k \leftarrow k \oplus 2k$ ▷ Precompute condition bits for cswap
16: `Increase(`ctr`)`
17: **for** i from 252 downto 1 **do** ▷ Main scalar-multiplication loop
18: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, k[i])$ ▷ projective re-randomization merged with cswap
19: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
20: `Increase(`ctr`)`
21: $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ ((X_1, Z_1, X_2, Z_2), k[0])$
22: $x_P \leftarrow X_2 \cdot Z_2^{-1}$
23: `Increase(`ctr`)`
24: **if** `Verify(`ctr`)` **then** ▷ Detected wrong flow, including iteration count
25: $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Set output buffer to random bytes
26:
27: **return** x_P

computation through matching against templates. As a recommended [29] countermeasure against online-template attacks we employ projective randomization [209] with a full-size randomizer chosen uniformly random from $\mathbb{F}_p$. This countermeasure also stops template attacks that are based on exploiting the internal ladder state leakage.

However, that countermeasure does not stop the single-trace attacks. Therefore, the following attacks might be feasible: single-trace TA or DL targeting cswaprr (see [212], for example) and single-trace TA or DL targeting key loading. The ephemeral implementation does not explicitly protects against such attacks – see the evaluation in Section 5.5.3. However, the static implementation adds additional protections against such attack; for details, see Section 5.4.

Weak curve attacks. Since we are implementing Curve25519, many weak-curve attacks are not possible; for example, see the paragraph on small-subgroup attacks in [43, Sec. 3]. However, an attacker can try to insert a point in the order-8 subgroup; such inputs are mapped to the neutral element (represented by $Z_P = 0$) through the three doubling steps in lines 5–7 of Algorithm 9. Lines 9–10 detect this neutral element and abort. In typical X25519 implementations those doublings are performed

at the end of the scalar-multiplication loop, because the lowest 3 bits of the scalar are set to zero through clamping. Moving them to the beginning ensures that the input to the main loop is either of order ℓ or of order 1; even if an attack skips the early-abort through an injected fault.

Loop-abort attacks. In order to protect against loop-abort attacks we employ a flow-counter countermeasure [303]. Specifically, we use a single counter monotonously incremented throughout the scalar multiplication to detect changes in the execution flow. If the value of this counter does not match the expected value at the end of the computation then we return a random value.

Key Shortening. There are two ways how a fault-injection attacker can reduce entropy in the secret scalar: either by setting parts of the key to zero or by skipping over instructions that copy the scalar. We do not implement any particular countermeasure against those attacks for the following reasons. First, as we are on a 32-bit architecture, an attacker can set at most 32 out of the 256 bits to zero with a single fault, which is not sufficient to allow any practical attacks. Second, we have verified that the copying loop is unrolled by the compiler so again, an attacker can control at most 32 bits of the scalar with single FI. We did consider duplicating the scalar-copy operation, but while this would help against fault-injection attacks, it would make SCA easier.

Fixing scalar-multiplication output. In order to prevent an attacker from directly influencing the values in the output buffer, we set the content of this buffer to a random value before starting with the main computation and only copy the result to this buffer if the flow-counter check was successful. An attacker who is restricted to one fault can prevent either randomization or copying of the valid result, but not both – which would be required to obtain a predictable value.

Combined active/passive attacks. It might be possible for an advanced attacker to combine SCA and FI attack as in [112]. This specific attack is disabled by randomization of point coordinates and point blinding. Another strategy for such an attacker could be to use FI to disable some SCA countermeasures and then proceed with SCA. In this chapter we consider such an attacker largely out of scope, but in particular the re-randomization of the projective coordinates in every loop iteration makes life for even such a combined attacker hard: skipping over *one* re-randomization step would not reveal much information and skipping over re-randomization in multiple steps would require more than one fault.

5.4 SCA-protected static X25519

This section describes our implementation of X25519 protected against SCA and FI attacks when deployed in an *static* key-exchange scenario. In this scenario the secret scalar can be used an arbitrary number of times, contrary to the implementation from Section 5.3.

The goal of an attacker against the static key exchange is to either obtain the long-term secret key or the output, i.e., session key. This goal can be either achieved by learning the scalar or by influencing scalar multiplication through FI during both key generation and shared-key computation to an easily guessable value.

The static scalar in our library is protected with various static masks. The scalar and these masks need to be generated by the user and then stored in the library. Afterwards the static masks are automatically updated during scalar multiplication executions and should not be modified or accessed by the user. If possible they should be stored in the memory isolated from the user. The Cortex-M4 does not feature memory isolation, thus for this work we consider that protecting the private key in memory is out of scope.

Below in Section 5.4.1 we list most relevant passive attacks and in Section 5.4.3 we list most relevant active attacks. Note that these lists are extensions of the lists from Section 5.3. Moreover, since the static implementation implements scalar blinding, a countermeasure that randomizes a scalar for each scalar multiplication execution, most of the attacks are effectively reduced to the ephemeral case.

5.4.1 Relevant passive attacks

DPA attacks. With the first publication introducing DPA [172] it was clear that the technique applies to all cryptosystems relying on long-term secrets. The idea is based on the fact that the same secret is used over and over again, which allows the attacker to build a statistical attack. The only requirement is to identify the so-called selection function that takes as inputs some known data and the secret he/she aims at recovering. In a static key exchange, the secret is typically the scalar i.e. the private key and the known input is the point it gets multiplied with. Considering this scenario, the attacker can recover the secret key bit by bit, by simply collecting enough traces for each loop of scalar multiplication and making the hypothesis on intermediate results.

Cross-Correlation Attack. Cross-correlation exploits collisions in subsequent additions and doubling of a scalar multiplication algorithm using many traces [304]. The horizontal correlation attacks from Section 5.4.3 can be traced back to this multi-trace attack.

Special-point attacks. A refined power analysis (RPA) attack exploits the existence of special points: $(x, 0)$ and $(0, y)$. Feeding to a device a point P that leads to a special point $R(0, y)$ (or $R(x, 0)$) at step i under the assumption of processed bits of the scalar will generate exploitable side-channel leakage [136]. A zero-value point attack (ZPA) [9] is an extension of RPA. Not only considering the points generated at step i, a ZPA also considers values of auxiliary registers. For some special points P, some auxiliary registers might predictably have zero value at step i. The attacker can then use the same procedure of RPA to incrementally reveal the whole scalar.

Carry-based attack. The carry-based attack [118] does not attack the scalar multiplication itself but its countermeasures. It relies on the carry propagation occurring when long-integer additions are performed as repeated sub-word additions. For instance, on an 8-bit processor, Coron's first countermeasure, $k' = k + r\#E$, is normally performed with repeated 8-bit additions. Let k_i and $r_i\#E$ denote the i-th sub-word of k and $r_i\#E$, respectively. Note that k_i is fixed and $r_i\#E$ is random in different executions. The crucial observation here is that, when adding k_i to $r_i\#E$, the probability of the carry out $c = 1$ depends mainly on k_i. The adversary can then monitor

the outgoing carry bit of the adder to estimate the probability of $c = 1$. With this probability, the value of k_i can be guessed reliably.

Address-bit DPA. The address-bit DPA attack [90] explores the link between the register address and the key. It was the first time applied to ECC in [155]. This attack is applicable if the addresses of coordinates processed in the ladder step depend in some way on the corresponding scalar bit. Essentially, the scalar bit can be recovered if the attacker can distinguish between data read from different addresses.

Address template attacks. The address leakage mentioned above in the address-bit DPA, can be exploited using a template attack.

Observe that when attacking a single trace address TA is essentially equivalent to the TA targeting the cswap [212] (Subsection 5.3.1), even if the attacked leakage is not coming from the address but from the cswap logic. Therefore, when we talk about address leakage in this chapter, we also mean the cswap leakage.

5.4.2 Relevant active attacks.

Safe-error attacks. The concept of safe-error was introduced in [309, 160]. Two types of safe-error are reported: C safe-error and M safe-error. The C safe-error attack exploits dummy operations which are usually introduced to achieve SPA resistance, like add-and-double-always, for example. The adversary can try to induce temporary faults during the execution of the dummy operation. If the result is unaffected then it means that indeed a dummy operation was affected. Otherwise, in case of different result, the attacker learn that a real operation was affected. This is enough to learn a scalar bit in the attacked iteration. The M safe-error attack exploits the fact that faults in some memory blocks will be cleared. The attack was first proposed in [160] to attack RSA, but it also applies to scalar multiplication. The goal of the attack is to affect memory that is only overwritten if a scalar bit has a certain value, e.g., 1.

Differential fault analysis. The Differential Fault Attacks (DFA) on scalar multiplication [62, 67] use the difference between the correct results and the faulty results to deduce certain bits of the scalar. These attacks require multiple correct and incorrect results of scalar multiplications to learn the static scalar. Since we randomize the scalar, as described later, these attacks are not applicable and we do not detail on them here due to the space constraints.

5.4.3 Protected implementation

The static protected scalar multiplication is presented in Algorithm 10. Similarly to Section 5.3, we employ re-randomizing the projective representation using cswaprr and control-flow counter.

Moreover, to prevent SCA attacks against key transfer, we implement scalar storage blinding: the scalar k is stored as $k_{f^{-1}} = k \cdot f^{-1}$ together with f, which is a 64-bit random blinding. To protect against attacks that use special points as input, we also use static random points R and S for input point blinding, where $R, S = [-k]R$. These blindings f, R, S and $k_{f^{-1}}$ are always securely randomized after each scalar multiplication. They should also be stored securely. In particular, if possible in the

given architecture, they should be stored in a secure memory not accessible to users of the library. That is why these values are marked as secure input in Algorithm 10.

To further improve security, the scalar is randomized with an additional 64-bit value r that multiplicatively masks each scalar multiplication. Then the mask is removed by performing an additional 64-bit scalar multiplication and is not stored.

To circumvent address leakage we follow the approach of [146] and implement the address-randomization technique from [154]. This countermeasure adds additional random cswaprr executions to make the cswaprr sequence in the scalar multiplication independent from the scalar itself. Although this countermeasure is called address-randomization for historical reasons, it also should thwart cswap-like leakage as shown in [146].

Below we explain why our static implementation is not vulnerable against the attacks from the last two subsections.

DPA attacks. To prevent DPA, Coron [90] suggested three countermeasures for an EC-based key exchange: randomizing a point, projective coordinates, and the scalar for every protocol execution. We implement all these methods in our implementation and, as confirmed by the evaluation in Section 5.5, we show its DPA-resistance.

Cross-Correlation Attack. Due to the scalar blinding this attack is essentially reduced to horizontal correlation attack and this is stopped by the projective re-randomization.

Special-point attacks. All special points attacks are stopped by the initial point blinding with $R, S = [-k]R$. This countermeasure ensures that in each scalar multiplication iteration the point used by the ladder is independent from the input point.

Carry-based attack. The carry-based attack is thwarted by the usage of the storage scalar blinding. Essentially the scalar is re-blinded after every usage and therefore, no single guess about k or k_i can be made between multiple blinding computations.

Address-bit DPA. This attack is not possible since scalar is freshly randomized in each scalar multiplication execution.

Address template attacks. Only single-trace attacks are applicable due to the employed randomizations. The single-trace horizontal, template, and DL attacks can directly target whatever instructions involve the scalar bits and are much harder to thwart. To stop them we implement the address-randomization [154] following the approach from [146]. We check its effectiveness in Section 5.5.3.

An alternative countermeasure against address attacks is suggested in [183]. It is shown to be effective, but the proposed cswap implementation does not consider the risk of correlation between memory loads and stores of the unchanged subwords, before and after swapping. We choose the more conservative option of storing conditionally swapped operands only after having them projectively re-randomized first. Thus our implementation avoids the risk of correlation between loaded and stored (swapped) operands.

Safe-error attacks and differential fault analysis. Due to the scalar blinding, these attacks are not applicable.

Algorithm 10 Pseudocode of side-channel and fault-attack protected static X25519

Input: the x-coordinate x_P of P. **Secure Input:** a 64-bit blinding f, blinded scalar $k_{f^{-1}} = k \cdot f^{-1}$, and blinding points $R, S = [-k]R$
Output: $x_{[k]P}$

$ctr \leftarrow 0$ ▷ Initialize iteration counter
$x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Initialize output buffer to random bytes
Copy k_f to internal state while increasing ctr. ▷ Updating ctr in a loop makes sure that copying cannot be affected by faults
$y_P \leftarrow \mathsf{ycompute}(x_P)$
`Increase`(ctr)
$(X_P, Y_P, Z_P) \leftarrow \mathsf{ecadd}((x_P, y_P), R)$ ▷ Point blinding, output of addition of R is projective
$(X_P, Y_P, Z_P) \leftarrow \mathsf{ecdouble}((X_P, Y_P, Z_P))$
$(X_P, Y_P, Z_P) \leftarrow \mathsf{ecdouble}((X_P, Y_P, Z_P))$
$(X_P, Y_P, Z_P) \leftarrow \mathsf{ecdouble}((X_P, Y_P, Z_P))$ ▷ 3 doublings to multiply by co-factor 8
$r \leftarrow \{1, ..., 2^{64} - 1\}$ ▷ Sample 64-bit non-zero random value for scalar blinding
$b \leftarrow \{0, ..., \ell\}$ ▷ Sample blinding factor of non-constant-time inversion
$t \leftarrow r \cdot b \mod \ell$ ▷ Invert using extended binary gcd
$s \leftarrow t^{-1} \cdot b \mod \ell$ ▷ Unblind result of inversion
$k'_{f^{-1}} \leftarrow k_{f^{-1}} \cdot s \mod l$ ▷ Multiplicatively blind scalar $k_{f^{-1}}$
$k' \leftarrow k'_{f^{-1}} \cdot f \mod l$ ▷ Multiplicatively unblind scalar $k'_{f^{-1}}$ with f
`Increase`(ctr)
$x_P \leftarrow X_P \cdot Z_P^{-1}$ ▷ Return to affine x-coordinate
$y_P \leftarrow Y_P \cdot Z_P^{-1}$ ▷ Return to affine y-coordinate
$X_1 \leftarrow 1, Z_1 \leftarrow 0$
$Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$ ▷ Initial randomization of projective representation
$k' \leftarrow k' \oplus 2k'$ ▷ Precompute condition bits for cswap
$a \leftarrow \{0, ..., 2^{253} - 1\}$ ▷ Sample mask for address-randomization
$k' \leftarrow k' \oplus a$ ▷ Mask the scalar
`Increase`(ctr)
$(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, a[252])$ ▷ projective re-randomization merged with cswap based on mask.
for i from 252 downto 0 **do** ▷ scalar multiplication by $k' = k \cdot r^{-1}$
    $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, k'[i])$ ▷ projective re-randomization merged with cswap based on masked k
    if $i \geq 1$ **then**
        $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
        $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, a[i-1])$ ▷ projective re-randomization merged with cswap based on mask
        `Increase`(ctr)
$y_P \leftarrow \mathsf{yrecover}(X_1, Z_1, X_2, Z_2, x_P, y_P)$
$x_P \leftarrow X_2 \cdot Z_2^{-1}$
$X_1 \leftarrow 1, Z_1 \leftarrow 0$
$Z_2 \xleftarrow{\$} \{0, \ldots, 2^{255} - 20\}, X_2 \leftarrow x_P \cdot Z_2$ ▷ Again randomize projective representation
$a' \leftarrow \{0, ..., 2^{65} - 1\}$ ▷ Sample additional mask for address-randomization
$r \leftarrow r \oplus 2r$ ▷ Precompute condition bits for cswap
$r \leftarrow r \oplus a'$ ▷ Mask the random scalar r
`Increase`(ctr)
$(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, a'[64])$ ▷ projective re-randomization merged with cswap based on mask
for i from 64 downto 0 **do** ▷ scalar multiplication by r
    $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, r[i])$ ▷ projective re-randomization merged with cswap based on masked r
    if $i \geq 1$ **then**
        $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{ladderstep}\ (x_P, X_1, Z_1, X_2, Z_2)$
        $(X_1, Z_1, X_2, Z_2) \leftarrow \mathsf{cswaprr}\ (X_1, Z_1, X_2, Z_2, a'[i-1])$ ▷ projective re-randomization merged with cswap based on mask
        `Increase`(ctr)
$Y_2 \leftarrow \mathsf{yrecover}(X_1, Z_1, X_2, Z_2, x_P, y_P)$
$(X_2, Y_2, Z_2) \leftarrow \mathsf{ecadd}((X_2, Y_2, Z_2), S)$ ▷ Remove point blinding, add in $S = [-k]R$
$x_P \leftarrow X_2 \cdot Z_2^{-1}$
`Increase`(ctr)
if `Verify`(ctr) **then** ▷ Detected wrong flow, including iteration count
    $x_P \xleftarrow{\$} \{0, \ldots, 2^{256} - 1\}$ ▷ Set output buffer to random bytes
`Update`(R, S) ▷ Perform double-and-add scalar multiplication with the same 8-bit random number on both R and S
`Randomize`$(k_{f^{-1}}, f)$ ▷ Generate 64-bit random value f, securely compute f^{-1} and update $k_{f^{-1}}$
`Save`$(R, S, k_{f^{-1}}, f)$

return x_P

5.5 Evaluation

This section presents benchmark results of our implementations (Section 5.5.1) and our side-channel evaluation results to support our claims of resistance against DPA and profiled attacks (Section 5.5.3).

5.5.1 Performance Evaluation

We measured clock cycles of the implementations on an STM32F407 Discovery development board and applied the common practice of clocking it down to at 24 MHz for obtaining reproducible cycle counts that are not significantly affected by wait cycles of the memory subsystem [140]. We use the gcc compiler, version 10.2.1 20201103 (release), GNU Arm Embedded Toolchain 10-2020-q4-major. The performance evaluation results are presented in Table 5.1.

We see that both SCA and FI protections come at a significant cost. However, the protections for the ephemeral implementation, mostly the projective re-randomization, only incur a relatively small penalty. The additional protections included in the static implementation, mainly against profiled attacks on key transfer and conditional swap, have a more significant cost even when implemented with efficiency in mind. The total overhead to protect the ephemeral implementation is about 36% and the overhead of protecting the static one is 239% in comparison to the unprotected case.

Table 5.1: Performance Evaluation (using -O2 optimizations).

Implementation / Countermeasure	**Clock cycles:**
Complete unprotected:	683 061
Complete ephemeral:	930 915
Complete static:	2 316 986
Projective re-randomization merged with cswap, cost per iteration (ephemeral & static)	1041
Randomized Addressing (static):	330 481
Update static blinding points (static):	153 911
Update static scalar (static):	199 947
Blinding of the scalar (static):	197 376
Additional 64-bit scalar multiplication (static):	331 742

5.5.2 Performance Comparison

As expected our countermeasures significantly reduce the speed. Still we believe our protected implementations to be highly competitive. Speed-optimized versions of unprotected off-the-shelf-libraries such as BoringSSL, BearSSL or WolfSSL require in between roughly 2 and 2.5 million CPU cycles on our target platform. They are,

thus, more than two times slower than our protected ephemeral and about as fast as our fully protected static implementation.

Table 5.2: Performance Evaluation (using -O2 optimizations).

Library	**CT**	**SCA**	**Clock cycles:**
WolfSSL [305] size:	yes	no	45 930 947
WolfSSL [305] speed:	yes	no	1 974 047
BearSSL [285]:	yes	no	2 576 639
BoringSSL [135]: fixed base	yes	no	1 591 407
BoringSSL [135]: var. base	yes	no	2 516 476
ARM MBed TLS [15]	no	no	6 438 233
This work			
unprotected [140]	yes	no	683 061
ephemeral DH	yes	yes	930 915
static DH	yes	yes	2 316 986

For the purpose of comparison we have also benchmarked our protected implementation against several commonly used cryptographic libraries. We chose the libraries for their wide-spread use (BoringSSL) or because they claimed to be targeting smaller embedded systems (BearSSL, WolfSSL, ARM MBed TLS). We included both, publicly available commercial libraries and open source implementations.

The cycle count results and the essential security features are summarized in Table 5.2. Cycle counts were obtained by using the same ARM Cortex-M4 and parameters as used in our work.

Security properties. None of the other benchmarked implementations exceed the protection level of conventional constant-time execution. According to our assessment, the ARM Mbed TLS library even fails to provide constant execution timing as detailed below.

Regarding the security properties, BoringSSL and BearSSL roughly should be comparable to our unprotected baseline algorithm, while ARM MBed TLS does not even reach that level.

The fixed-basepoint algorithm from BoringSSL uses arithmetics on the Edwards curve and uses large precomputed tables, a strategy which we do not believe suitable for smaller embedded targets due to code size limitations. All other implementations use a Montgomery ladder strategy on the Montgomery curve in projective coordinates. The implementations essentially differ in the way field elements are represented and reduction is carried out and how aggressively optimization for code size and speed are carried out. For instance, the arithmetic of BoringSSL and the speed-optimized strategy for WolfSSL represent field elements by using a 10-limb representation using 10 32-bit words, where field multiplication and reduction is merged together. Instead BearSSL uses a 9-limb representation of 9 words of 31 bits each, resulting in somewhat less memory consumption.

None of, WolfSSL, BearSSL and BoringSSL include any SCA countermeasures. ARM Mbed TLS on the other hand integrates projective randomization also for public input points. As ARM MBed TLS uses an early-abort multiplication strategy that does not execute in constant time we presume that this projective randomization was considered to provide a mitigation for fending off simple timing-based SPA attacks. Unfortunately, as input points are not validated in ARM MBed TLS, this mitigation strategy does not actually work here. Simple timing attacks using the strategy of [131] becomes feasible if the adversary inserts a low order point as multiplications with zero could be distinguished due to the early-abort multiplication strategy. As a result we conclude that the projective randomization substep used in ARM MBed TLS adds additional computational overhead (to the already slow implementation) without actually providing a proper mitigation against timing-based attacks.

We have communicated our results and concerns regarding the timing vulnerability of ARM Mbed TLS to the developers of [15] and suggested different strategies for mitigating or resolving the problem. They acknowledged the issue, and are working on implementing countermeasures.

Speed comparison. The unprotected speed-optimized implementations of BoringSSL, WolfSSL and BearSSL result in cycle counts in between $1\,974k$ and $2,576k$, roughly corresponding to the cycle count of $2\,316k$ that we obtain for our protected static implementation. Our protected ephemeral implementation still outperforms any of these off-the-shelf libraries while still providing stronger SCA and FI protections.

5.5.3 Side-channel Evaluation

In this section we describe our SCA setup and then we present the results of the evaluation of the unprotected, ephemeral, and static implementations. We also discuss the resistance to profiled attacks.

Side-channel analysis setup. We run our side-channel experiments using a Cortex-M4 on an STM32F407IGT6 board clocked at 168 MHz. We measure current with the Riscure Current Probe [245] and we record traces using the PicoScope 3406D oscilloscope. For analysis of the traces we use the Inspector software by Riscure [246].

For all the results presented in this section we compile the code using the standard -O2 optimization flag. However, we perform every test also using no optimizations (-O0) as a double check (since using no optimizations often inflate existing leakage). All results are consistent for both flags with respect to detecting leakage.

We perform the leakage detection in the following scenarios:

- The unprotected implementation, without any countermeasures except constant-time implementations of all operations, including field arithmetic and Montgomery ladder;
- The ephemeral implementation, with projective coordinate re-randomization enabled (see Algorithm 9);
- The static implementation as in Algorithm 10, with all countermeasures enabled (projective re-randomization, randomized addressing, point, scalar, and storage blinding);

- A static implementation modified for the profiled-attacks evaluation with scalar and storage blinding disabled. We test this setting with and without the randomized addressing countermeasure to verify resistance to profiled attacks.

A comparison of power profiles of our standard unprotected, ephemeral, and static implementations is presented in Figure 5.1. As mentioned in Section 5.5.1, the ephemeral implementation is only slightly slower than the unprotected one. In both cases, as marked in red, the implementations consists mainly of a scalar multiplication. In the static case, the trace consist of initial randomizations, two scalar multiplications (marked red), and the final update of the static key and points.

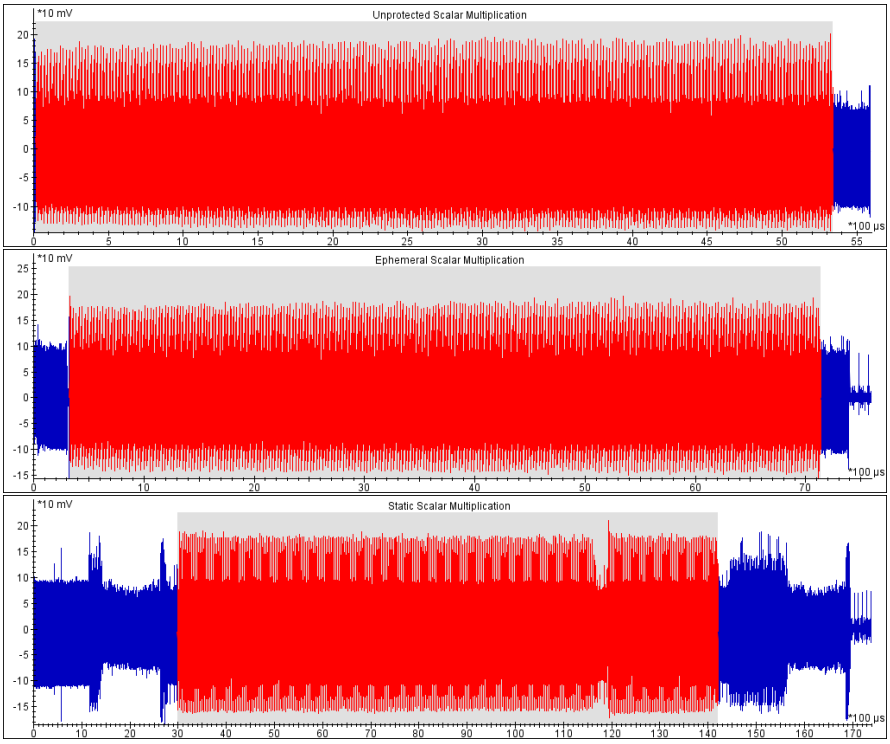


Figure 5.1: Power profiles of unprotected (top), ephemeral (middle), and static (bottom) implementations with scalar multiplications marked with red.

We apply a commonly-used TVLA methodology [133, 156, 33, 290] to our implementations using the aforementioned measurement settings. Following this leakage-evaluation methodology, we use three sets of traces that are equal in size one third of the traces is taken with a fixed input and a fixed scalar (*group* 0), another third with a random input and the fixed scalar (*group* 1), and the last group with the fixed input and a random scalar (*group* 2). If the null hypothesis holds and no leakage is detected then there should be little differences in the Welch's t-test statistics measured between group 0 and 1, and between 0 and 2.

Traditionally, in [156, 290] the TVLA confidence threshold to detect leakage in the t-test statistic was set to 4.5. However, we compute the confidence threshold for our experiments, using the formula from [101] following the approach [223]. The threshold computed this way is more accurate and ensures that false positives are avoided in the leakage assessment. For all our experiments the computed threshold is between 7 and 7.3 and therefore, we assume that peaks above 7 indicate leakage.

In our evaluation we concentrate not only the scalar multiplication, but we also analyze the rest of the trace. Note that we usually expect to see leakage at the beginning of the trace, due to varying input or scalar, and at the end due to varying output.

The leakage is usually detected only around the area that we align on[2]. This is caused by jitter and, especially for the static case, by intended randomization (e.g., for the inverse operations). Therefore, it might happen that if the trace is aligned at the beginning of trace then the leakage is only detected there, even if leakage is present everywhere. To avoid that we align the traces in multiple locations (usually at the beginning, middle, and end of computational blocks). Due to the space constraints, we report only the meaningful results when leakage is detected or no leakage is confirmed.

In the following subsections we always first analyze the difference between groups 0 and 1. The goal is to evaluate the correlation of the implementation leakage to a fixed or a random input point; such leakage, if present, can be used to mount CPA. Secondly we concentrate on groups 0 and 2. The aim is typically to check whether the private key (i.e, scalar) leaks directly; such leakage might be used to mount a template attack.

TVLA of the unprotected implementation. Figure 5.2 depicts the results of the t-tests for groups 0 and 1, and groups 0 and 2. We also aligned the traces at various locations and results were similar. In both cases each group consists of 1000 traces.

The highest peak is reaching 71 for groups 0 and 1, and 65 for groups 0 and 2. Therefore, we conclude that the unprotected implementation leaks significantly.

TVLA of the ephemeral scalar implementation. Similarly to the unprotected implementation we perform the TVLA on the ephemeral implementation, but this time due to the implemented countermeasures, we expect less leakage. Therefore, we also increase the size of each group to 2000 traces.

In Figure 5.3 we present the results of the t-tests for the ephemeral scalar implementation. The leakage is less significant than for the unprotected case. The highest peak is reaching 15 for groups 0 and 2, but it is to be expected since the ephemeral scalar should be different in every execution. As we see if the same scalar is used multiple times then the leakage can be detected.

We also note that the peak reaches 19 for groups 0 and 1. This might be unexpected since we employ the coordinate projective re-randomization. However, we have realized that the leakage might be present due to the using a non-randomized

[2]We align on a distinctive pattern in the trace. We select such a pattern from the first trace and then we match the pattern to all subsequent traces by shifting them horizontally. For matching quality we use Pearson correlation – we simply choose the shift in the trace with the maximum correlation coefficient.

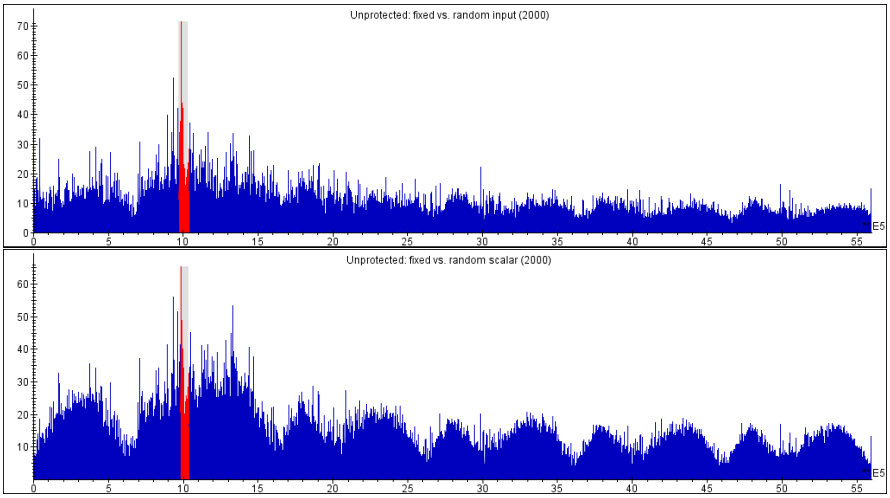


Figure 5.2: TVLA results for the unprotected implementation: fixed vs random point (top) and fixed vs random scalar (bottom). The red color marks the alignment.

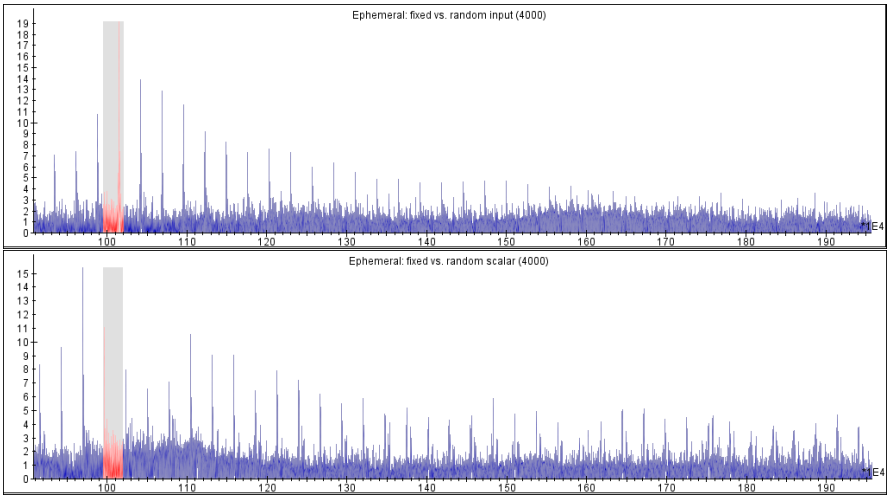


Figure 5.3: TVLA for the ephemeral impl. (0.9ms-2.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

x coordinate of the input point in the ladder step. The input leakage is not exploitable but it would generate t-test peak (since the input is a parameter to TVLA). Essentially, we encounter a well-know limitation of TVLA [33].

To validate the above hypothesis we have modified the implementation by removing the following line of the ladder step: `fe25519_mul(b4,b1,&pState->x0)`, where `&pState->x0` corresponds the affine coordinate of the input x_P. Note that this operation leakage is not exploitable since x_P is public and constant per execution and

the other parameters to `fe25519_mul`, and therefore the output too, are randomized. The results produced by this implementation are incorrect, but we only use it for the evaluation. The result of t-test has shown that the leakage is not present anymore, because the highest peak does not reach even 4.4; the plot of the t-test values is presented in Figure 5.4.

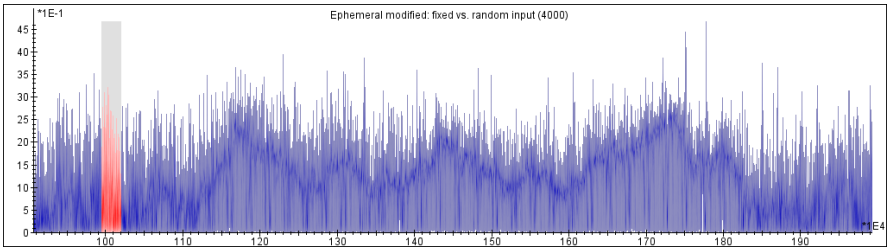


Figure 5.4: TVLA results for the modified ephemeral implementation (0.9ms-2.0ms): fixed vs random point.

We can conclude that the ephemeral implementation is protected against DPA and CPA attacks and most likely against the cross-correlation attacks. However, it might be vulnerable to single-trace profiled attacks.

We recommended to use the same key only once, otherwise more attacks might be possible. Furthermore, it may be possible to learn information about the ephemeral key when it is being copied. However, this would be hard since the ephemeral key should be used only once and copying is done in 32-bit chunks[3].

TVLA of the static implementation. First we run the t-test for the traces collected from the static scalar multiplication that are aligned at the beginning. For groups 0 and 1 we notice leakage at the beginning of the execution, when the input point is being processed. For groups 0 and 2 we detect no leakage (the highest peak is less than 4.6), due to the scalar being stored blinded. The plot of the t-test values for these traces is presented in Figure 5.5.

The static traces are significantly misalignment due to jitter, but also due to the used countermeasures. Therefore, we perform another t-test for which we align the traces at the beginning of the main scalar multiplication and the result is depicted in Figure 5.6; it is zoomed in around the alignment moment. We can see that the peaks in both experiments are below 4.3, and therefore leakage is not detected. We repeated the experiment aligning at various locations: the middle and the end of the main scalar multiplication, and at the beginning of the additional scalar multiplication. No leakage during both scalar multiplications is detected.

However, we have detected leakage in both cases just after the scalar multiplication is finished: around 14.5ms. This leakage is caused by computing the final affine output. After the output is computed, the static key is updated and finally the output is sent. We aligned the traces at the key update procedure and we detected

[3]Since 32-bit words are being copied then we expect that 32-bit Hamming weight may be leaking. If the key is used once then a profiled attack would require a single trace and it might be infeasible due to measurement noise.

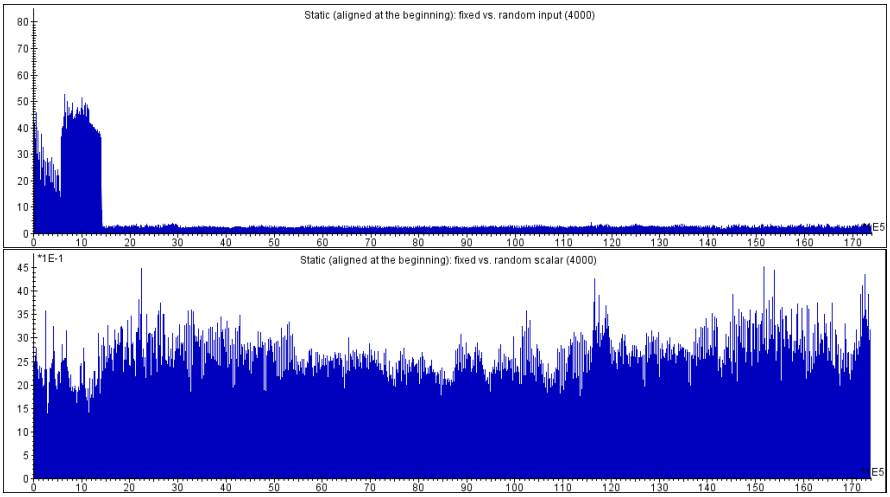


Figure 5.5: TVLA results for the static implementation: fixed vs random point (top) and fixed vs random scalar (bottom).

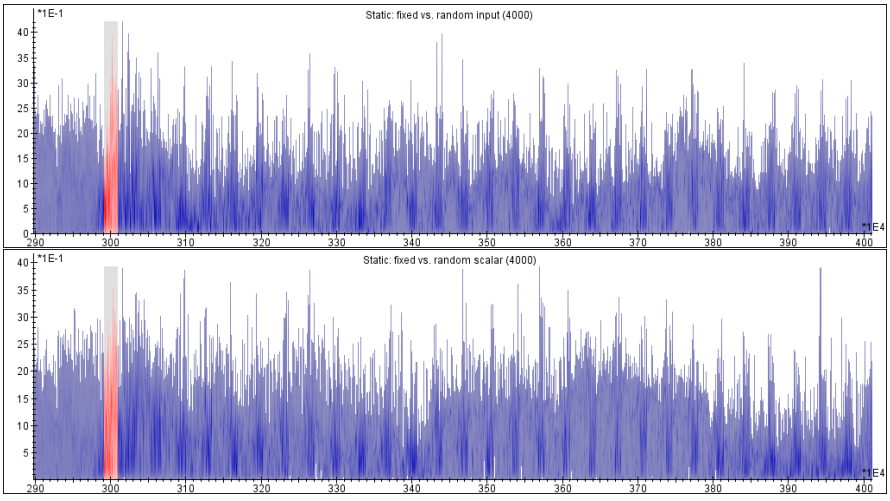


Figure 5.6: TVLA for the static impl. (2.9ms-4.0ms): fixed vs random point (top) and fixed vs random scalar (bottom).

no leakage. At the end of the execution we discovered the leakage corresponding to sending the output (17ms). Such output leakage might be used for a sing-trace attack to recover the output point (and effectively the session key). Although this may be possible, this cannot be avoided since the library returns the unmasked output.

In this section we show that the static implementation is resistant to standard attacks like DPA/CPA, cross-correlation, and OTA.

Profiled attacks. The first most relevant profiling attack is a TA targeting the cswap introduced in [212]. Note that the blinded scalar and the additional 64-bit scalar blinding would need to be recovered from a single trace for the attack to succeed.

We verified the resistance of our implementation to this attack by performing a TVLA experiment on an implementation where we turned off all scalar randomizations: the scalar and storage blinding. In this setting we checked whether turning on the randomized addressing countermeasure prevents scalar leakage from occurring.

Figure 5.7 depicts the t-test results for groups 0 and 1 for the traces collected from the scalar multiplication with all scalar blindings and address randomization turned off. The highest t-test peak is reaching 12.5. This leakage and the indicated points of interest might be used to recover the scalar from the cswaprr instructions using the aforementioned TA. For groups 0 and 2, as expected, we do not see any leakage, i.e., the peak is under 4.5; the plot of the t-test values for this case is presented in Figure 5.7.

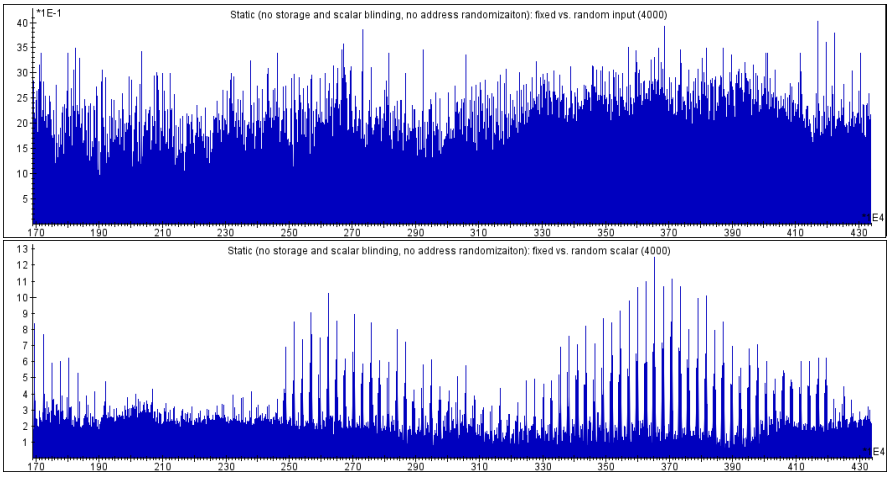


Figure 5.7: TVLA for static impl. (no blindings & address rand.) aligned at 1.65ms (1.7ms-4.3ms): fixed vs random point (top) and fixed vs random scalar (bottom).

To validate whether the address randomization works we turned on this countermeasure. We have discovered that if all scalar blinding countermeasures are turned off, but the address randomization is turned on, no leakage for the TA [212] is present. The plot of the t-test values for this case is presented in Figure 5.8.

Although we have discovered that the leakage that can be exploited by "classical" TA is not present, a DL-based attack might still be theoretically possible. However, we are convinced that the absence of "classical" leakage provide enough assurance even against more complex profiled attacks (including DL).

We consciously decided not to protect the ephemeral scalar multiplication with the address randomization for the sake of efficiency. However, it would be easy to add address randomization in the same way as for the static case.

The second relevant profiled attack, which we have not discussed here yet, is against the scalar transfer, as presented in [299]. Observe that the blinded scalar is

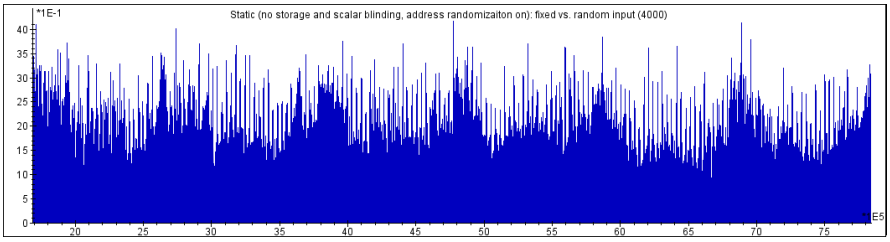


Figure 5.8: TVLA results for the static impl. with blindings turned off and the address rand. turned on (aligned at 1.65ms and zoomed at 1.7ms - 7.8ms): fixed vs random scalar.

copied just before the scalar multiplication. We have verified that the scalar data is being copied word by word in the disassembled code. Therefore, we can expect that even a successful profiled attack on key transfer would recover Hamming weight of 32-bit chunks of the blinded scalar. Furthermore, the attack phase cannot use multiple traces, since a blinded scalar is stored together with the blinding and these values are updated at the end of every scalar multiplication. Therefore, we conclude that it would be hard to mount a profiled attack along the lines of [299] on the scalar transfer.

5.6 Conclusion

We have described software computing the X25519 key-exchange on the ARM-Cortex M4 microcontroller, This software comes with extensive protections against both side-channel and fault attacks. It is, to the best of our knowledge, the first to claim such protections motivated from a real-world application. We present an extensive side-channel and fault-injection analysis and we also perform in-depth side-channel evaluation that shows strong resistance of our implementation.

Chapter 6

Faults and Genetic Algorithms

6.1 Introduction

Embedded security devices such as ID or bank cards, key immobilizers, mobile phones, etc., are omnipresent in our lives and the threats to them directly affect the security and privacy of our data. The attackers often target the weaknesses of implementations rather than the algorithms running on those chips. Basically, these so-called implementation attacks do not focus on the algorithm itself but rather exploit some physical effects. The effects, i.e., physical leakages become available due to the actual implementation of the algorithms on a platform that is typically constrained in terms of area, power, energy, etc. Two well-known types of implementation attacks are side-channel attacks (SCAs) and fault injection (FI) attacks. Side-channel attacks are passive, non-invasive attacks where the device under attack operates within specified conditions and the attacker simply observes the physical leakages produced. Fault injection attacks are, on the other hand, active, more invasive attacks where the attacker inserts faults (e.g., by glitching some parameters like voltage, power, clock, etc.) in order to disrupt the normal behavior of the algorithm.

Side-channel attacks received a lot of attention in the last few decades where we saw successful exploitation of several side channels like timing [171], power consumption [172], and EM emanation [239]. Researchers devised various strategies to use that information and deliver powerful attacks that are even capable of recovering the secret keys. Not surprisingly, many of those strategies in the last few years are based on machine learning [179, 233] and deep learning [75].

When considering fault injection attacks, the situation is somewhat different in terms of attack methodology and analysis techniques. In certain scenarios, just one fault can lead to a total break of a cryptosystem as described in the first paper of Boneh et al. [69]. While it was clear that one fault can have devastating consequences on the security of a system, it was not immediately clear how complicated can be to insert the faults of interest. The first work that illustrates a practical fault injection attack on an actual RSA cryptosystem was done by Aumüller et al. [17]. They describe the engineering efforts behind a successful fault injection and possible countermeasures. Fault injection is often possible through glitching techniques. There are several sources of glitches possible, such as laser pulses, electrical glitches, and

electromagnetic radiation. A fault injection attack is successful if after exposing the device under attack to a specially crafted external interference, the device shows an unexpected behavior, i.e., a fault, which can be exploited by an attacker. Here, the challenge lies in selecting the appropriate parameters for a fault to succeed. If those parameters are not well chosen, the target will respond in a way that does not permit an actual fault analysis attack. When considering various sources of faults, we encounter different number of parameters and corresponding ranges. In general, the search space size of possible parameter values is large and relatively few points in the search space result in faults that can be used by an attacker. Consequently, an interesting follow-up question is: how to find suitable parameter values or intervals, or more precisely, how to efficiently find the correct values in the search space? Surprisingly, the main options available so far are to use either random search or some sort of exhaustive search. This is mainly due to the fact that a complete exhaustive search is usually not feasible so the attacker should focus on specific regions with a certain precision, and perform a so-called grid search.

In our work, we start from this observation and turn to some well-known machine learning strategies to deal with what is basically an optimization problem with a large number of parameters. Another motivation comes from relevant experiences with side-channel analysis and machine learning. In principle, if it is possible to make SCA more powerful by using machine learning (and more generally, artificial intelligence) one would expect the same for fault injection.

In this chapter, we shed some light on the above mentioned questions and observations. We discuss how one could use a special type of metaheuristics called genetic algorithms in order to find parameter values resulting in faults for electromagnetic fault injection (EMFI). A somewhat similar research direction is followed in several previous works [77, 232, 231], but there the authors considered power glitching, which is only a subset of the search space we have to deal with for EMFI. In addition, they attacked a PIN checking mechanism on a smartcard. In our research, we use the faults obtained via a novel technique to mount an algebraic fault attack on a SHA-3 implementation where we consider pulsed EMFI only resulting in a total of 5 parameters. We emphasize that our version of search algorithm differs significantly from previous works as detailed in the rest of this chapter. Finally, our code is available as an open source implementation[1].

6.1.1 Related work

Although a vast amount of work has been done on fault injection itself, see, e.g., [173, 69, 253, 5, 219, 195], only a small fraction of it concerns parameter optimization.

In [188], the authors developed an EMFI susceptibility criterion, which they used to rank the points of the chip surface depending on how sensitive they are to fault injection. The underlying assumption for the criterion is the Sampling Fault Model, described in [221]. The criterion itself is a combination of Power Spectral Density (measuring emitted power at the clock frequency) and Magnitude Squared Incoherence (measuring how linked the emitted signal is to the data being processed). They use a grid scan (in 2 spatial dimensions) to measure all the points and rank them

[1]Github: `https://github.com/geneticemfaults/geneticemfaults`

according to the criterion; a share α of the highest-ranking points are kept for further scanning; the rest is thrown away. They are able to reject over 50% of the chip surface (75% in their best case), while keeping 80% of the points causing faults. However, by *fault*, they mean any perturbation of the normal behavior of the algorithm.

In [77], the authors apply several different methods to the problem of parameter optimization for the supply voltage (VCC) glitching. They reduce the dimensionality of the problem by splitting the search into two stages. In the first stage, they look for the best (glitch voltage, glitch length) combination. In the second stage, 10 most promising (voltage, length) combinations are tried at each point in the time range (which is divided into 100 instants). All parameters not explicitly specified are set as random. The methods are compared at the first stage to random search, FastBoxing and Adaptive zoom&bound algorithms, and a genetic algorithm. This approach is a "smart" search in 2D with a grid search in 1D.

That work is extended in [231] where the authors use a combination of a genetic algorithm and local search (called memetic algorithm) in order to find faults even more efficiently. The authors consider power glitching with 3 parameters and are interested in a fast characterization of the search space.

6.1.2 Contributions

Here we summarize the main contributions of this chapter:

- We use well-known machine learning techniques called genetic algorithms to optimize parameter selection.
- We use this technique in combination with EMFI on Cortex-M4 running SHA-3.
- We apply our methodology to SHA-3 to recover state with a reduced number of glitches.

6.1.3 Organization of this chapter

The rest of this chapter is organized as follows. First, we provide background information on genetic algorithms in Section 6.2. Next, we show our experimental setup in Section 6.3 where we also explain the different fault injection parameters and note the issues with the resulting parameter space. In Section 6.4, we describe the search algorithm, that uses a genetic algorithm. In Section 6.5, we present the results of our experiments. Finally, in Section 6.6, we conclude the chapter.

6.2 Background

6.2.1 Genetic Algorithms

Evolutionary algorithms represent population-based metaheuristic optimization techniques inspired by biological evolution and phenomena like mutation, recombination, and selection [148, 108, 18]. The solutions in the population compete and in that process improve their goodness as evaluated by a fitness function. Evolutionary algorithms often perform well in many types of problems because they ideally do not

make assumptions about the underlying solutions' landscape. Today, there are many types of evolutionary algorithms, but probably the best known ones are genetic algorithms (GA). An instance of a genetic algorithm maps a real optimization problem to the natural concepts as follows:

1. The objective function (which we are optimizing) becomes the fitness function,

2. and a solution (a point in the solution space) becomes an individual in the population.

The general pseudocode of an evolutionary algorithm is given below. Note that this is general enough to cover any type of evolutionary algorithm, including genetic algorithms.

Input: Parameters of the algorithm
Output: Optimal solution set§
1: $t \leftarrow 0$
2: $P(0) \leftarrow CreateInitialPopulation$
3: **while** $TerminationCriterion$ **do**
4: $\quad t \leftarrow t + 1$
5: $\quad P'(t) \leftarrow SelectMechanism\ (P(t-1))$
6: $\quad P(t) \leftarrow VariationOperators(P'(t))$
7: $Return\ OptimalSolutionSet(P)$

6.2.2 Keccak/SHA-3

Here we revisit Keccak again, see chapter 3 for more details. In this chapter, we apply our attack on a cryptographic hash function, which is the new SHA-3 standard [51], and also known as Keccak. The Keccak main function is a sponge construction with a permutation as its core operation. Keccak is a cryptographic primitive that can be used in different modes (such as keyed and unkeyed) to compute hash values, MACs, or to encrypt/decrypt data.

The core permutation named Keccak-$f[b]$ is defined by its width b and in our case, we use the full width where $b = 1\,600$. The permutation is described as a sequence of operations on a state a. The state is a three-dimensional array of elements in $GF(2)$. There are 5 rows and 5 columns, each of length 64, i.e. $a[5, 5, 64]$. Keccak-$f[b]$ is an iterated permutation over 24 rounds, the round function R is defined as follows:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

In total there are five different steps that modify the state. An omitted index implies that the statement is valid for all values of that index.

$$\begin{aligned}
\theta: \quad & a(x,y,z) && \leftarrow a(x,y,z) + \sum_{y'=0}^{4} a(x-1,y',z) + \sum_{y'=0}^{4} a(x+1,y',z-1), \\
\pi \text{ and } \rho: \quad & a(y,2x+3y) && \leftarrow \mathrm{rot}(a(x,y), r(x,y)), \\
\chi: \quad & a(x) && \leftarrow a(x) + (a(x+1)+1) \cdot a(x+2), \\
\iota: \quad & a(0,0) && \leftarrow a(0,0) + RC
\end{aligned}$$

All operations are carried out in $GF(2)$. The function $\mathrm{rot}(W,i)$ is a bitwise cyclic shift operation, where the constants $r(x,y)$ are rotation offsets. The value RC is the round constant, there is a different value for each round. For more details, see [51].

6.3 Experimental Setup

For the target, we use a Cortex-M4 STM32F407IG (Riscure "Piñata") board running a C implementation of SHA-3. This implementation is taken from the WolfSSL library[2]. The board communicates with a PC by a serial interface and is powered by an external power supply. We use a Riscure EM probe and the VC Glitcher device that controls it. The whole setup is controlled by the code that is written in Python 2.x. For interfacing the Riscure equipment, we use Python bindings for the VC Glitcher C API (which is a 32-bit DLL). Figure 6.1 shows a photo of the setup.

The on-board code provides the trigger which signals that the cryptographic operation is in progress. This trigger is used as a reference point for injecting the fault. In case that the board gets stuck in an illegal state after a glitch, it needs to be reset. The only way to reliably reset this particular board is by cutting its power, which can take a significant fraction of a second, depending on the capacitors. We used 100 ms for this event.

The physical dimensions of the chip package are 24×24 mm. Repositioning error of our XYZ table is 0.05 mm, which gives us a spatial grid of at most 480×480 positions. Note that the limiting factor in our case is likely the size of the probe tip, which is much larger.

6.3.1 Parameters

There are multiple parameters one can vary to affect fault probability as follows:

- the spatial position of the probe tip (there are three degrees of freedom; X, Y, Z);
- the moment when the EM pulse fires;
- the pulse intensity;
- the shape of EM probe and the angle w.r.t. the target;
- the shape of EM pulse w.r.t. time

[2]WolfSSL, an embedded SSL/TLS library. Available at: `https://www.wolfssl.com/`

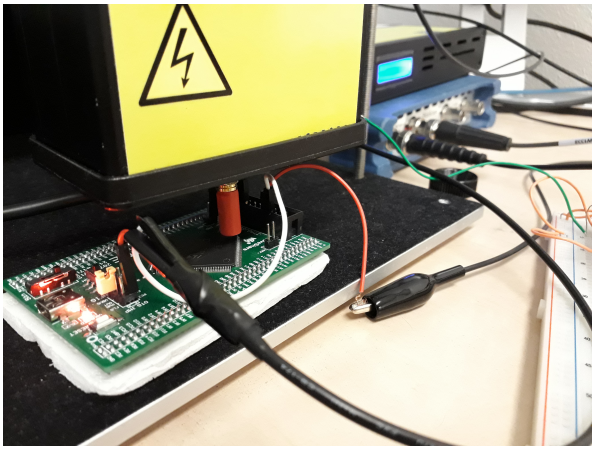


Figure 6.1: The photo of the setup.

In our experiments, we consider only a subset of those as listed below:

- We consider a position, as a pair of parameters (X, Y). We do not vary the distance from the board (Z), since this can be largely compensated for by a change in intensity. These parameters are real values in $[0, 1]$ range.
- The glitch intensity regulates the voltage of the pulse. The SDK manual suggests it to be set to a percentage of power used. We use real values for the percentage in the range $[0, 1]$.
- We consider the time offset to be between 367 and 375 µs, because that is where the injection point must be, for the code we are running. We encode the offset at integer value (number of 2 ns ticks).
- A number of repetitions of the pulse is a primitive form of the pulse shape. We set this parameter as an integer in the range $[1, 10]$.

We do not vary pulse duration, and we leave it to a fixed value of 40 ns. Similarly, we do not vary shape and angle of the probe tip, since changing those automatically is (for us) hard enough to make them unsuitable for automatic optimization.

6.3.2 Search Space Size

As mentioned above, it is not possible to conduct an exhaustive search when considering fault injection with realistic targets. Hence, the question that remains is the following: what is the search space size and how could we efficiently sample it? When considering X and Y position, there is a 0.05 mm repositioning error and 24×24 mm chip size, which gives max resolution of 480×480. For the time offset, with a 2 ns resolution and the range between 367 and 375 microseconds there are in total 4 000 different values. We have no good rule for determining the smallest meaningful increment for the glitch intensity, but if we use a 1% increment, that gives us a range

of 100 values. The repetitions are selected to be a random integer value in the range $[1, 10]$, which gives us 10 values.

The total parameter set size is therefore, $480 * 480 * 4\,000 * 100 * 10 \approx 10^{12}$. At ≈ 0.16 seconds per point, this results in 29 203 years to conduct an exhaustive search. Since trying the same parameters multiple times does not necessarily always yield the same response, we conduct 5 measurements for each point. Even if we would completely ignore everything except X, Y, and offset, we would still need 29.2 years to conduct an exhaustive search.

6.4 Search Algorithm

6.4.1 Assumptions

We consider the device to be a black-box. It assumes that the objective function is not a "golf-course-like" (i.e., to be too flat), in which case the lack of a significant gradient in the fitness landscape means that there is no driving bias toward fitness optimum. The reasoning behind is that a very weak EM pulse will not affect the target at all, and we will observe the normal behavior. Conversely, a very strong EM pulse will completely dishevel its operation and even potentially damage it. Consequently, we expect the faulty behavior to occur somewhere between those two extremes, i.e., along the class border. Additionally, offset ranges (min to max offset) are set by the user, based on a rough expectation of the duration of the cryptographic algorithm.

Usually, the objective function guides the optimization algorithm towards better solutions, and the algorithm ends when it finds the best one. Here, we do not want just a single "best" solution since not every fault we find will also be exploitable, and there are situations where more than one solution is required, so we aim to obtain multiple solutions.

6.4.2 GA Objectives

We require our algorithm to have the following two characteristics:

1. Good coverage of the parameter space – since we do not know where the exploitable faults are located, we need to explore the search space efficiently.

2. Speed – we require the algorithm to be fast in finding the faults, otherwise, there is no advantage of using it when compared to random search, for instance.

These requirements are somewhat conflicting with each other. Basically, as most of the parameter space is useless (i.e., has no faults), covering enough space to make sure we did not miss anything important implies potentially wasting a lot of measurements.

Next, we introduce the terminology we use when discussing the search and possible outputs of the algorithm. A point is a distinct set of parameters, i.e., a point in the parameter space. A measurement is the result of a single attempt at glitching the target with those parameters.

When counting the faulty measurements, we distinguish between:

1. The total number of faulty measurements,

2. and the number of distinct faulty responses (i.e., "unique faulty measurements").

The difference is in the following: if a measurement results in a before-seen faulty output, the second one will not count again. As an example, assume we find a set of parameters resulting in a specific fault. Later, we find some other set of parameters that result in the same fault but we do not count that new faulty measurement into distinct faulty measurements. For the exploitability purposes, the second number is more interesting as detailed in the following sections.

We classify the board response in one of the following classes:

- NORMAL: for normal behavior, meaning the board performs as if we did not do anything.
- RESET: the board did not reply at all, requiring a reset to restore to normal operation.
- SUCCESS: the board produces an output/ciphertext/signature/hash different than the correct one.
- CHANGING: for each point we investigate, we perform 5 measurements. If all measurements are in the same class, the point is put into one of the first three classes; otherwise, it goes into the CHANGING class.

Fitness values are set according to the class: SUCCESS has the highest fitness (10), followed by the CHANGING, then RESET (5), and finally, NORMAL (2). CHANGING points' fitness depends on its underlying measurements: a mix of NORMAL and RESET measurements is somewhat better than an all-RESET or all-NORMAL point, but having individual measurements belonging to the class SUCCESS moves the fitness closer to an all-SUCCESS point. We calculate the fitness of a CHANGING point in the following way:

$$fitness_{\mathrm{C}} = 4 + 1.2 * N_S + 0.2 * N_N + 0.5 * N_R. \tag{6.1}$$

Here, N_S represents the number of SUCCESS points, N_R the number of RESET points, and N_N the number of NORMAL points. Finally, factors 0.2 and 0.5 are chosen in analogy to the values for NORMAL and RESET (which are 2 and 5) while the rest of the factors are selected for the scaling reasons. For example, 4 NORMAL and 1 RESET measurements give fitness 5.3, which is higher than the fitness of a RESET point (with all 5 RESET measurements). Similarly, 4 SUCCESS and 1 RESET measurements give fitness 9.3, which is lower than the fitness of a SUCCESS point (with all 5 SUCCESS measurements).

6.4.3 Algorithm Definition

Despite the fact that we use genetic algorithms like similar to some previous works [77, 231], our custom made algorithm is quite different. We discuss the specifics of our design in the following paragraphs. Our algorithm has several parameters to be determined. We selected those values on the basis of our tuning experiments and recommendations from [77, 231].

More in detail, our algorithm consists of two separate phases as follows:

1. The first phase is a genetic algorithm that we run for 20 generations with population size 50.

2. Only when the genetic algorithm is done, we start with the local search, which takes 10 randomly chosen points in the neighborhood of each SUCCESS point. Note that this is a significant difference from related works where both GA and local search worked at the same time. We opted first to concentrate on exploration aspect – GA (to explore various regions of the target) and only after that on exploitability aspect – local search (to concentrate on more promising regions). Naturally, GA itself has also exploitability component that is especially manifested in the crossover operator but we also designed a custom made crossover operator that promotes exploration perspective.

GA Phase

Our algorithm begins with a genetic algorithm that runs for N generations and has a population of M individuals. The initial population is selected uniformly at random within parameter ranges. We aim to maximize the fitness value, which corresponds to having as many as possible SUCCESS points.

The first phase of a GA is selection; we use roulette-wheel selection. In roulette-wheel selection, the fitness function assigns a fitness to possible solutions (in this case, points in parameter-space). This fitness level is used to associate a probability of being selected for each solution: the probability that an individual will be selected is directly proportional to its fitness. (More precisely, it is equal to its share in the overall fitness: for the i-th individual, $P_i = \frac{fitness_i}{\sum_k fitness_k}$.)

Note that, since the selection is done randomly, it is possible that low-quality individuals are picked. This is not troubling since a GA would not function otherwise. However, it allows for the possibility of an excellent, hard-found solution being accidentally lost. A common countermeasure is *elitism*: with elitism, a number of best-ranking individuals (called the *elites*) are always chosen. We use elitism, with the elite size equal to 1, so only the single best individual gets carried over.

We also experimented with a 3-tournament selection as used in [77, 231], but found it too restrictive, since it promotes a convergence too quickly, resulting in solutions being obtained from only a small part of the search space. This went directly against the objective of a good coverage of the search space.

After the selection phase finishes, the crossover can start. The purpose of crossover is to combine the existing solutions to produce better ones. There are many ways to do the crossover. When we imagine our individuals as points – or rather, vectors – in the 5-dimensional parameter space, then each of the parameters corresponds to one element of this vector. When combining two such individuals, a traditional crossover might take some elements from one parent, and the rest from the other parent. The version of crossover that we used instead picks a point in-between the respective parents' elements. This promotes explorability and enables GA to traverse large parts of the search space.[3] The pseudocode for this crossover is given in Algorithm 11, and the pseudocode for the mutation in Algorithm 12. The mutation rate *p_mutation* is

[3]The parent points define an axis-aligned parallelepiped in parameter-space; the parents are placed on the diagonally opposite vertices. In a Hamming cube, these would be the all-zeros and

set to 5%. In the case the parameter gets out of its ranges, it is clipped to the edges of the range.

Algorithm 11 Crossover operator

```
for each/parameter/p do
    child.p = random value in range [parent1.p, parent2.p]
```

Algorithm 12 Mutation operator

```
r_1, r_2 = uniformly random from [−0.5, 0.5]
for param in [x, y, intensity] do
    with probability p_mutation:
        param = param + r_1
with probability p_mutation:
    offset = offset + r_2 * OFFSET_RANGE
with probability p_mutation:
    repetitions = random integer from [1, 10]
```

Local Search Phase

After the GA is done, we use local search to focus on the promising parts of the explored search space as follows: the space around the intersection points (i.e., places where class values change), and the space around any faults that were already found. We define the neighborhood of a point as a cube centered in it, with edge length equal to 0.02. By the length of 0.02 in parameter space, we mean 2% of the range of that parameter. Parameters x, y, and intensity are all within the range $[0, 1]$. For offset, it's 2% of OFFSET_RANGE (which is OFFSET_MAX - OFFSET_MIN). To determine the distance of values, we use the Euclidean distance.

6.4.4 Practical Considerations

Commonly, optimization algorithms (and nature-inspired metaheuristics in particular) rely on a large number of iterations. Another assumption usually made is that the evaluation of possible solution points is uniform. Here, we have expensive measurements, where the cost of evaluation depends not only on the property of the point itself, but also the context of its evaluation. Although our algorithm consists of a genetic algorithm and local search, we denote it often as a genetic algorithm but we always consider it to have also local search phase. We do not call our technique a memetic algorithm since the GA and local search phases are separated.

When considering EMFI, there is the probe tip, which has to physically move to a different point. To do this with a sufficient precision requires a non-negligible amount of time – the exact amount varies depending on the setup, but it can be up to several seconds per measurement. In comparison, a reset requires just a fraction of a second

all-ones vertices. The first crossover variant corresponds to picking one of its vertices, whereas the second crossover variant corresponds to picking a point within it.

Table 6.1: Statistical results for GA and random search.

	GA	Random
NORMAL	662.8 (18.9%)	2 995.8 (90.7%)
RESET	496.4 (15.0%)	65.0 (2.0%)
CHANGING	375.2 (11.4%)	232.4 (7.0%)
SUCCESS	1 807.2 (54.7%)	8.8 (0.3%)

(for our board, ≈100 ms to do it reliably). The measurement part itself is even faster – 30 ms or less. Thus, the order in which points are evaluated matters. Even with an optimal routing for any batch of N points, more batches mean more time wasted. For population-based algorithms, this translates to small population sizes not being as efficient as large ones.

Additionally, we may want to get a glimpse of the results even before the scan is finished, especially for long-running scans. In case of a random or grid scan, this means splitting the scan into batches where each covers more or less the whole parameter space, since scanning points in the optimal order results in uneven coverage.

6.5 Results

In this section, we present our results when attacking SHA-3. First, we investigate how well is GA able to find faults (i.e., force the algorithm to output the wrong ciphertext) and then, whether such points can be used in order to obtain the state of the algorithm. For this purpose, we use algebraic fault analysis (AFA), as described in [184]. AFA eliminates the need for analysis of fault propagation as it is needed in differential fault analysis; instead, it uses a SAT solver to recover the state bits from a (clean output, faulty output) pair.

6.5.1 Finding Faults

The duration of GA is determined by the number of faults it finds. We conducted 5 independent runs with 2 074, 2 343, 3 353, 3 606, and 5 132 points, respectively. Each individual run is different due to the stochastic nature of GA as well as the target response. To obtain statistically meaningful results, we report averaged values over all runs and report results in Table 6.1. For GA, on average, in each run, there are 3 301.6 points, which means we conduct 16 508 individual measurements on average. Out of these, 9 700.4 (58.8%) are faulty, and 3 288.4 (19.9%) are unique/distinct. To compare, we use random search with 3 302 points, which represents 16 510 individual measurements. Out of these, 228.2 (1.3%) are faulty, and 160.8 (1.0%) are unique/distinct.

To conclude, when averaged over 5 runs, our GA algorithm gives 42.5 times more faulty measurements, and 20.5 times more distinct faulty ones. The somewhat lower share of distinct measurements for the GA algorithm can be explained by many SUCCESS points being close to each other due to the local search, thus being more likely to cause the same response.

Table 6.2: Random search and GA results for various search stages.

Points	Algorithm	NORMAL	RESET	CHANGING	SUCCESS	#faults	#distinct
500	Random	90.5%	2.0%	7.2%	0.3%	1.3%	0.9%
	GA	63.0%	14.7%	15.8%	6.5%	10.4%	6.3%
1 000	Random	91.0%	2.0%	6.7%	0.3%	1.2%	0.8%
	GA	38.2%	19.8%	16.9%	25.1%	30.6%	19.1%
2 000	Random	90.7%	1.8%	7.2%	0.2%	1.3%	0.9%
	GA	27.1%	17.6%	14.2%	41.1%	46.0%	20.3%

In Table 6.2, we give results for random search and genetic algorithms when considering 500, 1 000, and 2 000 points. Observe how the results for random search do not change significantly with more measurements. At the same time, we observe that GA is very successful already for the smallest case where we use only 500 points and as we add more points, the percentage of SUCCESS points increases.

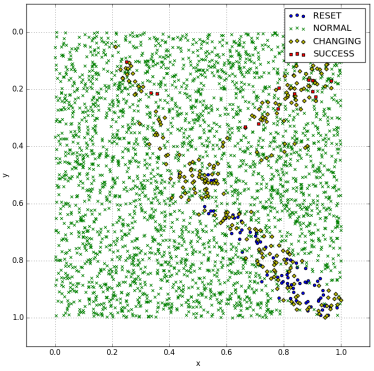
We depict the search space after random search and GA in Figures 6.2(a) until 6.2(f). Figures 6.2(a) and 6.2(b) give results for X and Y parameters. Figures 6.2(c) until 6.2(f) also depict intensity as a parameter. We depict both cases with and without NORMAL points to improve the readability. The number of points for each figure is 3 300 (figures not depicting NORMAL points have fewer points).

Finally, we show the results for GA as it progresses through the evolution process. More precisely, in Figure 6.3(a), we depict the search results for the first 500 points. We can observe that although there are several SUCCESS points, in this phase GA mainly finds NORMAL points spread across the search space. Figure 6.3(b) depicts results for 500–1 000 points range. Here, we can see that GA does not find so many NORMAL points but actually manages to find a significant number of CHANGING and RESET points. We also see a good amount of SUCCESS points. In Figure 6.3(c), we show the results for points between 1 000 and 2 343 (end of search). Here, we see a large number of SUCCESS points where there are one large cluster and three smaller ones. CHANGING and RESET points occur in the same large cluster as the majority of SUCCESS points. NORMAL points occur in a number of small clusters surrounding the main cluster. To conclude, we see that GA is able to find SUCCESS points even with a small number of examined points but its true strength lies when there is a sufficient number of evaluations in order to guide the convergence.

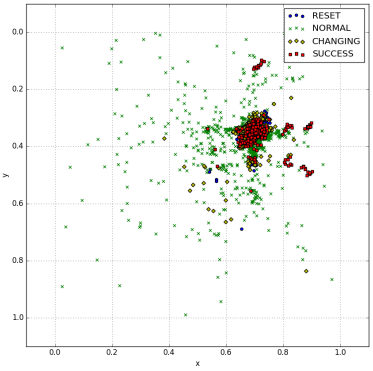
6.5.2 SHA-3 Attack in Practice

To the best of our knowledge, SHA-3 implementation have not yet been attacked in practice. Attacks do exist, but only on simulated data such as [20]. The authors show that Differential Fault Analysis (DFA) can be used to recover the complete state in around 80 faults on average if the attacker is able to inject single-bit faults in the input of the penultimate round (i.e., θ_i^{22}), though they rely on brute-forcing the last few bits. According to [186] (itself an extension of [187]), this is around 500 single-bit random faults for the whole state. The work in [186] generalizes the attack to a single-byte fault model, recovering the state in around 120 random faults.

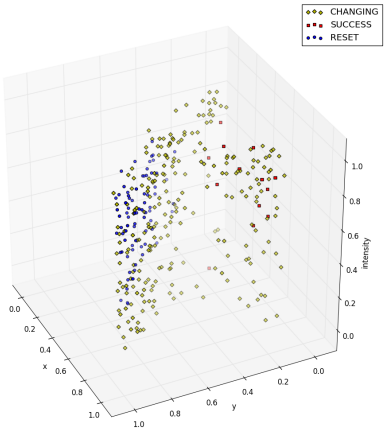
Algebraic Fault Analysis (AFA) seems more promising. Luo et al. manage to bring down the number of faults needed to recover the internal state with SHA3-512



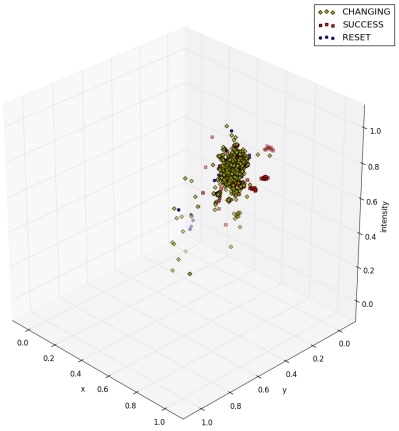
(a) Random search in 2D



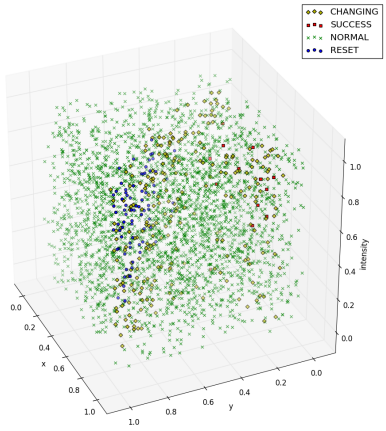
(b) GA and local search in 2D



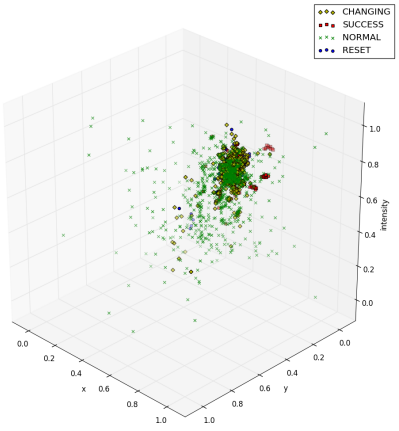
(c) Random search without NORMAL points



(d) GA and local search without NORMAL points

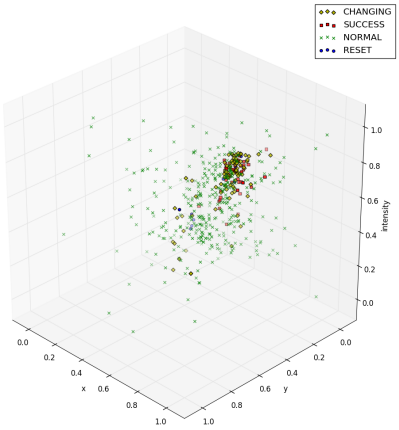


(e) Random search

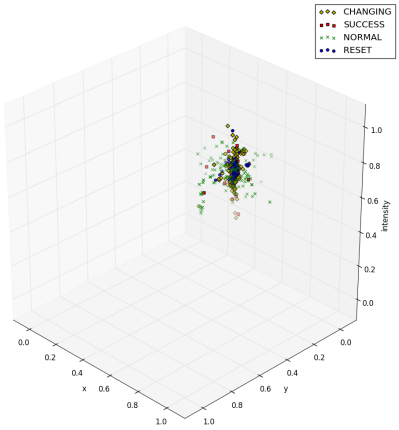


(f) GA and local search

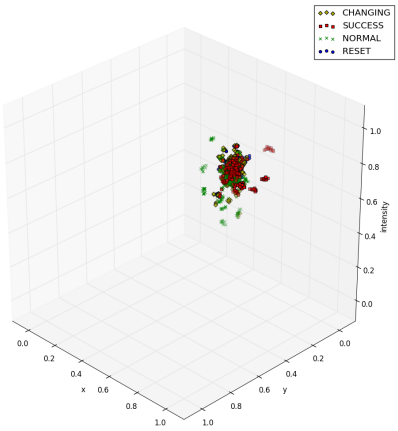
Figure 6.2: Results for GA (with local search) and random search.



(a) Results for points 0–500



(b) Results for points 500–1 000



(c) Results for points 1 000–2 343

Figure 6.3: Results for GA with local search depicting several stages of the search process.

to under 10 with AFA and the 32-bit fault model in the progress described in [187, 186, 184].

Besides being more efficient at recovering state, the several advantages of AFA are as follows,

- It does not require analysis of fault propagation through the algorithm, making it much easier to abstract the internal details.
- We can easily change the fault model, by just changing the appropriate constraints.
- Perhaps most importantly, it works for more relaxed fault models.

The attack in [184] allows the attacker to retrieve the state by injecting multiple faults in the input of the round 22 of Keccak. The faults are allowed to affect up to 1 unit of the state, where units are sized 8b, 16b, or 32b. As in previous work, we use θ_i^{22} as the fault injection point, and χ_i^{22} as the target state to recover. We reused their C++ retrieval code for this purpose.

The general idea behind AFA on SHA-3/Keccak is simple: use a SAT solver to do the work for us, we just need to provide appropriate constraints for it. We start with 1 600 Boolean variables representing the state (θ_i^{22}) and then provide constraints:

1. Fault Model – what kind of a fault do we cause? There is a separate set of (up to) 1 600 Boolean variables ($\Delta\theta_i^{22}$) representing the induced fault. $\theta_i^{22} \oplus \Delta\theta_i^{22}$ is the faulted state, before propagating through the final two rounds of the algorithm. Depending on what the fault model is, we add constraints such as "exactly one bit of $\Delta\theta_i^{22}$ is non-zero", corresponding to a one-bit fault model, or slightly more verbose ones for specifying things such as "we faulted a word-aligned 32-bit word", which corresponds to a 32-bit fault model in [184].

2. Keccak – how does the (faulted) internal state propagate? For Keccak, the internal transformations can be relatively simply encoded as Boolean expressions. This implicitly tells the solver everything it needs to know about fault propagation, regardless of the fault model constraints. There are two cases we consider:
$$H = \iota^{23} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota^{22} \circ \chi \circ \pi \circ \rho \circ \theta(\theta_i^{22}),$$
where H is the correct hash output, and
$$H' = \iota^{23} \circ \chi \circ \pi \circ \rho \circ \theta \circ \iota^{22} \circ \chi \circ \pi \circ \rho \circ \theta(\theta_i^{22} \oplus \Delta\theta_i^{22}),$$
where H' is the faulty hash output.

3. Outputs – which outputs are the concrete ones? We give the SAT solver the actual values of H and H'. After so constraining the SAT solver, we can let it find a solution – an internal state satisfying all the constraints. Once it finds the first such solution, we ban this newly-found solution by adding it as an additional constraint and then we let the SAT solver find another one. This process is repeated until no new solutions can be found.

The bits of the state which are the same in all solutions are the ones we can recover: as for those which take different values in different solutions, their values are not entailed by the combined constraints of the fault model, the algorithm, and the outputs (i.e., the "real" constraints).

Depending on the fault model and the version of SHA-3 (SHA3-512, SHA3-224, etc.), these constraints may or may not be enough to recover part of the state. In this case, additional constraints can be introduced, such as using two faulty hashes H'_1 and H'_2 at a time with a cost of extra Boolean variables and making it harder for the SAT solver (Method II in [184]), or first recovering part of the χ_i^{23} bits (Method III in [184]).

We applied the 32-bit fault model from [184] and Method III. The reason for this is a large number of potential faults to check while a short time for checking the exploitability of the induced faults is often an important factor. We tested the exploitability of all distinct faulty hashes obtained by our evolutionary algorithm, as well as of all those obtained by a random scan. While the share of distinct/unique faulty hashes depends on the size of the scan, the exploitability of a faulty hash does not. For this reason, we calculated the share of exploitable individual faults on all the samples we obtained (with the same hyperparameters).

The results are as follows: GA generated a total of 14 979 distinct faults (out of 82 540 individual measurements); 106 of these were exploitable 32-bit faults, for a share of 0.71%. Random search generated 947 distinct faults (out of 100 000 individual measurements); 110 of these were exploitable 32-bit faults, for a share of 11.61%. When translated into exploitable faults per individual measurement, this gives about 1.41×10^{-3} and 1.13×10^{-3} for GA and random search, respectively – an improvement of 24.6%.

Despite the fact that GA is still significantly more successful than the random search, we observe that actually most of the faults obtained with GA cannot be translated into exploitable faults. This results in a decrease between the performance difference of GA and random search. Still, such results are to be expected: since we never added the constraint of exploitability of faults into GA, it is hard to expect that GA will produce only such faults. Still, this could be addressed by having a fitness function that integrates an analysis of fault exploitability.

6.6 Conclusion

In this chapter, we investigate how genetic algorithms can be used to facilitate faster and more powerful EMFI. When considering the search space size one needs to investigate, it is evident that both random search and exhaustive search should not be the methods of choice. Indeed, our custom-made algorithm is able to find more than 40 times more faults than random search. Those results enable us almost 25% more exploitable faults per individual measurements when considering SHA-3 and algebraic fault attack. To the best of our knowledge, our algorithm is the most powerful currently available technique for finding parameters for EMFI.

Chapter 7

Automatically Fixing Side-Channel Leakage With the Use of Simulators

7.1 Introduction

The seminal work of Kocher [171] demonstrated that interactions of cryptographic implementations with their environment can result in *side channels*, which leak information on the internal state of ciphers. Since then multiple side channels have been demonstrated, exploiting various effects, such as timing [39, 3, 74, 73], power consumption [172], electromagnetic (EM) emanations [239, 124, 76], shared micro-architectural components [128, 289], and even acoustic and photonic emanations [175, 259, 19, 130]. These side channels pose a severe risk to the security of systems, and in particular to cryptographic implementations, and effective side-channel attacks have been demonstrated against block and stream ciphers [138, 240], public-key systems, both traditional [203, 208] and post quantum [226], cryptographic primitives implemented in real-world devices [109, 22], and even non-cryptographic algorithms [27].

Many approaches to protect devices have been suggested, in particular against power and EM attacks. These range from special logic styles that are designed to make leakage data-independent [287, 115, 83], through noise generation to hide the signal [201], to algorithmic changes designed to prevent certain leakage [202]. In particular, *masking* is a common algorithmic countermeasure in which all intermediate (secret-dependent) values in the ciphers are combined with random masks, so that the leakage of one or even a few values does not provide the attacker with enough information to recover the secrets.

The protection afforded by masking is, however, only theoretical. In practice, masked implementations often fail to achieve the promised level of security. One of the most common reasons for leakage from masked implementations is unintended interactions between values in the micro-architecture. For example, the power consumption of updating the contents of a register may depend on a relationship between the values prior to and after the update.

To achieve secure cryptography in the presence of side-channel attacks, cryptographic implementations often go through multiple cycles of leakage evaluation, e.g. as specified in International Standard ISO/IEC 17825:2016(E) [153]. Such a process

Table 7.1: Results of running Rosita to automatically fix masked implementations of AES, ChaCha, and Xoodoo.

	Cycles		Leakage Points	
Function	**Original**	**Fixed**	**Original**	**Remaining**
AES	1285	1479	31	0
ChaCha	1322	2162	238	1
Xoodoo	637	769	38	0

is costly because it requires a high level of expertise and significant manual labor, especially taking into account state-of-the-art side-channel adversaries.

Recently, several works have experimented with tools that provide a high resolution emulation of the power consumption [293]. The results of such emulations are combined with standard statistical tests [33] to perform leakage assessment of software without executing it on the actual hardware [224]. Observing that these tools eliminate the hardware from the leakage evaluation process, we ask the following question:

Can leakage emulators be used for automatic mitigation of (higher-order) side-channel leakage from software implementations?

In this chapter, we answer this question in the affirmative (see Table 7.1 for results), albeit with some caveats. Specifically, we develop an automatic tool, Rosita[1], that uses an emulator to detect leakage due to unintended interactions between values and then rewrites the code to eliminate the leakage. Automating leakage elimination reduces the amount of work required to ensure adherence with the ISO 17825 standard, and to develop secure cryptographic implementations.

To emulate leakage, we develop Elmo*, a leakage emulator that uses the execution engine of the Elmo leakage emulator [197], but improves Elmo in three different directions. We first note that while Elmo detects leakage between consecutive instructions, it fails to detect leakage between instructions that are further apart. To improve the emulation accuracy, we first design and develop novel procedures for detecting leakage between non-consecutive instructions and for identifying the storage components involved in this leakage. We apply these procedures to the Cortex M0 processor, identifying several storage elements. We then modify Elmo to model these elements, achieving an accurate leakage model. Second, we modify Elmo to not only output the instruction that causes the leakage, but also to identify the cause of the leakage. Last, we modify the workflow in Elmo to perform on-line statistical analysis of the generated traces to detect leakage.

In its core, Rosita is a rule-driven code rewrite engine. It uses the output from Elmo* to select rewrite rules and apply them at leaky points. To eliminate leakage, we follow the workflow in Figure 7.1. We repeatedly execute Elmo* to identify code

[1]Available at `https://github.com/0xADE1A1DE/Rosita`.

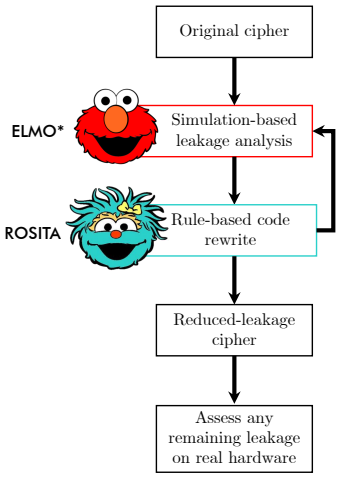


Figure 7.1: Leakage elimination workflow with extended ELMO (ELMO*) and ROSITA.

locations that leak and then invoke ROSITA to rewrite the code. Finally, we test the code produced by ROSITA on the physical device to assess the level of remaining leakage.

We experiment with masked implementations of three ciphers with very different round functions: the AES block cipher [96], the ChaCha stream cipher [40], and the Xoodoo permutation [94], running up to 1 000 000 traces on each. We use the table-lookup based masked implementation of AES-128 by Yao et al. [308]. AES implemented with table lookups tends to be vulnerable to cache-based timing attacks. Recent ciphers however, mostly permutations, can be implemented efficiently with only bitwise Boolean instructions and (cyclic) shifts, e.g., Keccak-p, the permutation underlying SHA-3, [105, 58], Ascon [102], Gimli [48] and Xoodoo [95]. They all have a nonlinear layer of algebraic degree 2 and hence allow very efficient masking. Among those, we chose Xoodoo because it is the simplest of all and it lends itself to efficient implementations for 32-bit architectures. Finally, we consider ChaCha20 as a symmetric-key algorithm that is very different than the other two as a representative of addition-rotation-xor (ARX) ciphers.

As Table 7.1 shows, ROSITA successfully eliminates leakage from the AES and Xoodoo implementations, at a moderate performance impact of less than 21%. For ChaCha, ROSITA is also successful, eliminating all but one leakage point, with a higher performance cost of 64%.

Additionally, in this chapter we present ROSITA++, an extension of ROSITA that performs second-order leakage detection and mitigation. At its core, ROSITA++ extends the leakage detection and root-cause analysis of ROSITA to support second-order analysis. It then uses the ROSITA code rewrite engine to modify the evaluated implementation and eliminate leakage.

Implementing ROSITA++ is far from straightforward. The main appeal of second-order secure implementations is that second-order analysis is significantly more com-

plex than first-order analysis. In particular, we identify three main challenges: the impact of the quadratic increase in trace lengths on the statistical tools used for the analysis, the explosion in the amount of data that needs be processed both due to the increase in trace length and because of the required increase in the number of required traces, and the complexities involved in performing bivariate root-cause analysis.

To address these challenges, we develop statistical and software tools that allow robust and efficient second-order leakage analysis. Our software tools are able to combine and analyze millions of traces each with thousands of sample points and perform bivariate analysis on the combined traces. We believe that these tools are of independent value for the side-channel community and could be used for bivariate analysis in a wide-range of cases.

We assess the effectiveness of Rosita++ with two cryptographic primitives, which represent two very different points in the design space of symmetric cryptography. PRESENT [256] is a popular lightweight block cipher with a traditional substitution-permutation network design. Xoodoo [95, 94], in contrast, is a modern cryptographic primitive that underlies multiple proposed primitives [94]. We implement a three-share version of Xoodoo, building on the non-linear χ layer from Keccak. For PRESENT we extend the two-share implementation of Sasdrich et al. [256] to support three shares.

We show that Rosita++ can remove all leakage detected in the real experiment up to 2 million traces in Xoodoo, in PRESENT all but one leakage point were removed. Further, we find that Rosita++ only requires to emulate 500,000 traces to achieve a protection against a 2 million trace attack on physical hardware.

7.1.1 Contributions

To sum up, the contributions of this chapter are:

- We propose a framework for generating first-order leakage-resilient implementations of masked ciphers by iteratively rewriting the code at leakage points. (Section 7.4)
- We design and implement systematic approaches for identifying leakage through microarchitectural storage elements. We use these approaches to detect multiple sources of leakage in the Cortex M0 processor that Elmo fails to model. We extend Elmo and augment it to model these sources of leakage, achieving an accurate model of leakage. We further augment Elmo to report instructions that leak secret information and the specific cause of leakage for each. (Section 7.4.1)
- We develop Rosita, a code rewrite engine that uses the output of Elmo*, our augmented Elmo, to rewrite leaking instructions and eliminate leakage. (Section 7.4.2)
- We use Rosita to rewrite masked implementations of AES, ChaCha, and Xoodoo. We test the code Rosita produces and show that Rosita eliminates almost all the leakage at up to 1 000 000 traces, with an acceptable performance loss. (Section 7.4.3)

- We explore automated tools for automatic second-order side-channel detection and protection. (Section 7.5.1.)
- We develop statistical and software tools for addressing the challenges.
- We build Rosita++, the first tool to automatically detect and remove unintended second-order leakage from cryptographic implementations. We evaluate Rosita++ on two cryptographic primitives and demonstrate its efficiency. (Section 7.5.2.)
- Rosita, Rosita++ and the associated tools are available as open-source projects.

7.1.2 Organization of this chapter

The rest of this chapter is organized as follows. First we provide more background information and related work about side-channel attack, leakage assessment and leakage emulators in Section 7.2. Next, we describe our experimental setup that we use for the evaluation in Section 7.3. In Section 7.4, we describe Rosita and we evaluation the tool using real traces. In Section 7.5, we describe Rosita++, some of the challenges when doing second-order side-channel evaluations, and we do an evaluation using real traces. In Section 7.6, we describe some of the limitations of the tools. Finally, in Section 7.7, we conclude the chapter.

7.2 Background

7.2.1 Side-Channel Attacks

When software runs on hardware, it affects the environment it executes in. This effect can be manifested as variations in power consumption, electromagnetic emanations, temperature, and state of various CPU components. As these variations correlate with the operation of the algorithm, monitoring these variations discloses information about the internal state, and as such provides a 'communication' channel that transfers information from the software being observed to the observer. These unintended communication channels are often known as *side channels*.

In 1996, Kocher [171] noted that the information acquired through a timing side channel may reveal secret information processed by cryptographic algorithms. Since then a significant effort has been dedicated to analyzing and eliminating side-channel leakage, particularly in the context of cryptographic implementations [172, 239, 124, 31, 192].

Protection against side-channel attacks depends on both the channel and the attacker capabilities. The *constant-time* programming style [50, 164], which avoids secret-dependent branches and table lookups, has proven effective against attacks that depend on the cryptographic operation timing or on its effect on micro-architectural components [39, 73, 128]. However, when the leakage correlates with the data values being processed, constant-time programming is not sufficient to protect the implementation.

One of the main approaches to protect cryptographic implementations against side channels that leak information on data values is *masking* [79, 202]. In a nutshell,

t-order masking represents each internal value v using $t+1$ values $v_0, v_1, \ldots, v_t$ such that the *masks* $v_1, \ldots, v_t$ are chosen uniformly at random, and v_0 is set such that $v = v_0 \oplus v_1 \oplus \cdots \oplus v_t$. Consequently the leakage of up to t values does not disclose any information to the attacker, and the implementation is secure in the t-probing model [152]. In practice, due to the complexities involved in higher-order masking, most masked implementation are first order, where each value is represented by a mask and a masked value.

Although theoretically secure, naive masking often fails to provide the required protection. The main cause is that side-channel leakage correlates not only with the values being processed, but also with the changes in the logical values of internal components, resulting in unintended interactions between values in the processor. Past research has identified two main sources of such leakage: transitional effects and glitches.

Transitional effects are caused when the logical value of a register or even of a long wire, such as a bus, changes from one to zero or vice versa. Changing the value draws more power than maintaining the value unchanged. Consequently, when changing the value of a register, the power consumption corresponds with the Hamming distance between the old and new values [21].

In contrast with transitional effects, which correspond to changes in logical values, *glitches* are temporary changes in electrical signals caused by signal timing differences. Because signals take time to propagate through the circuit, it takes time for the logical values to stabilize during a cycle. Until the signals stabilize, they may fluctuate between signal levels, leaking information that does not correspond just to the logical function computed [193, 82].

For masked implementations, unintended interactions, such as transitional effects and glitches can be disastrous. For example, suppose that a program that processes a secret value $v = v_0 \oplus v_1$ contains two consecutive instructions, where the first instruction uses v_0 and the second uses v_1. Internally, executing the first instruction would place v_0 on the bus, and executing the second instruction would change the contents of the bus to v_1. The transitional effect of changing the contents of the bus draws power which corresponds to the Hamming distance between v_0 and v_1, which is the Hamming weight of the secret value v.

Balasch et al. [21] demonstrate that unintended interactions due to transitions halve the number of intermediate values the adversary needs to acquire. However, as mentioned above, algorithms that use high-order masking are significantly more complex in terms of resources they require, than simple first-order masking. Moreover, Gao et al. [127] demonstrate that glitches may further reduce the security level of masked implementations.

Thus, the common practice, from the practitioners' point of view, is to use first-order masking, and to combine it with ad-hoc countermeasures for unintended interactions. Then, the implementation of the cryptographic software typically undergoes leakage assessment to find code locations that leak information. If leakage is detected, the operator applies manual modifications to the code to eliminate the leakage, this process repeats until no further leakage is evident.

We now turn our attention to assessing leakage in cryptographic implementations.

7.2.2 Univariate Side-Channel Leakage Assessment

Side-channel leakage assessment measures how vulnerable a device is to side-channel attacks. This is however not an exhaustive assessment, as it is impossible to try all possible attacks on a device. However, a side-channel leakage assessment of a device that handles secret information is valuable to the manufacturers of such devices so that they can guarantee a level of security as the device is designed and manufactured.

In side-channel leakage assessment, the main question that we try to answer is whether the evaluated device shows significant leakage. Therefore, a device must show statistically significant leakage to be classified as insecure. Standards such as International Standard ISO/IEC 17825:2016(E) [153] suggest a methodology called Test Vector Leakage Assessment (TVLA) that was initially presented by Goodwill et al. [134]. The TVLA methodology uses Welch's t-test [300] to detect statistical differences between sample distributions that are measured when the device processes different inputs. Two such configurations exist: the first is the fixed vs. random test and the second is the fixed vs. fixed configuration. The reason for calculating such differences is that a protected cipher implementation should not be emitting any information that would let an evaluator differentiate the data it is processing. If the calculated difference is statistically insignificant the device is regarded as side channel free in the context that it was tested on. It has been demonstrated that the results of t-tests should not be misinterpreted as a single test that decides if a device is secure or not [276]. Specifically, the selected fixed inputs and trace amounts collected from the device only suggest that the t-test failed to find leakage for those values. For different fixed input values or for a higher number of traces significant leakage could be observable.

In the Welch's t-test, A t-value is calculated from the means ($\overline{X}_1$ and $\overline{X}_2$) and variances (s_1^2 and s_2^2) of distributions of collected traces at a given sample point. Given the number of samples in each distribution as n_1 and n_2 the t-value (t) and degree of freedom (v) are calculated as,

$$t = \frac{\overline{X}_1 - \overline{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \qquad v = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1} + \frac{\left(\frac{s_2^2}{n_2}\right)^2}{n_2 - 1}}$$

The t-value from a Welch's t-test tells how significantly different the two means of the distributions are. To measure the significance of the difference the hypothesis of these means of the two distributions are the originating from the same population is rejected with some given amount of confidence. This is process is known as hypothesis testing. Hypothesis testing is the scientific method of ruling out hypotheses by rejecting them based on significant evidence against them. The null hypothesis is the hypothesis that we assume to be correct by default, in TVLA we assume that the device is not leaky, until evidence such as a significant t-value from the Welch's t-test would reject it in favour of the alternative hypothesis. The alternative hypothesis here state that the means of the two sample distributions are statistically different. The threshold value of 4.5 for significant leakage is chosen at a confidence(α) level of 0.00001 under the assumptions that $s_1 \approx s_2$ and $n_1 \approx n_2$, such that the total number of traces ($n_1 + n_2$) is greater than 1,000 [260].

Using naive methods to compute t-values may result in numerical errors due to cancellation effects [78]. Schneider and Moradi [260] demonstrated computational improvements to overcome such issues. They also suggested online calculation of t-values from a single pass which makes the calculations much faster compared to naive methods. Another common concern with the evaluation of masked implementations is the typical t-test threshold of 4.5. This value assumes a single independent t-test. For large numbers of sample points this threshold value is not adequate as the possibility of false positives (i.e. classification of leakage at sample points as significant when there is no actual leakage) increases due to the increased number of tests. Ding et al. [101] discussed this further and proposes a method to increase the t-test threshold according to the degree of freedom of the t-test [260] and number of samples.

7.2.3 Bivariate Side-Channel Leakage Assessment

In contrast to univariate analysis, where only a single sample point is analyzed independently of other points, bivariate analysis takes into account the joint leakage of two sample points. This is similar to using two probes with respect to Ishai et al. [152]'s model. A suitable combination function is used to combine each pair of mean centered sample values, then a first-order t-test is used for the analysis similar to the univariate case [260]. The product of the samples is commonly used as a combination function in literature [260].

Given a trace T with T_i and T_j samples, an artificial sample $T_{i,j}$ is formulated as shown in Equation 7.1 where $i < j$ and μ_i,μ_j are means of all samples at ith and jth sample points.

$$T_{i,j} = (T_i - \mu_i)(T_j - \mu_j). \tag{7.1}$$

7.2.4 Leakage Emulators

Because conducting real experiments for leakage detection is costly, leakage emulation has been adapted as an alternative. To the best of our knowledge, PINPAS [144], which detects leakage in Java-based smart cards, is the first such emulator. Since then, various other methods of emulating leakage have been suggested. Among the most accurate use SPICE [210] to simulate the internal circuits of a CPU down to a transistor level [8]. Its drawback is that transistor-level simulators tend to be very slow. Alternatively, researchers have looked at emulating at the source code level [292] and at machine instruction level [293, 224]. In source code level emulation, the emulator does not have any information about a specific CPU that will be used to run the compiled machine code of a given source code. It emulates leakage having source code as its only input. In instruction level emulation, the emulation is based on the machine code that will be executed on a certain CPU or more generally a specific CPU kind. Recently, advanced instruction level emulators have been introduced that use power and electromagnetic traces from real experiments to make better estimates [266, 197]. Similarly, advanced characteristics of CPUs such as instruction pipelining have found their way into recent leakage emulators [91].

Coco [132] suggests reformulating software testing for leakage as a hardware verification problem. Specifically, Coco uses a cycle-accurate simulator of masked software execution to acquire traces of execution on a target CPU. It then uses

Rebbeca [66] to verify the absence of leakage from the underlying hardware. An advantage of the approach is that leakage can be eliminated at both the software and the hardware levels. However, it requires access to the full netlist of the target processor and relies on manual tagging of masked values.

7.2.5 Testing for Statistical Equivalence of Distributions

A statistical equivalence test tells with a given confidence that means of two distributions are originating from the same population. This is the opposite of what Welch's t-test offers. The null hypothesis of an equivalence test is that the means two distributions are different and we expect to reject it in favour of the alternative hypothesis which state that the means of the distributions are the same with a given confidence level. One such equivalence test is the Two One Sided Test (TOST) [264], it uses two one sided t-tests to determine if the means of two distributions are the same with a given confidence. TOST is a parameterised test that requires a lower bound and upper bound for the mean difference of the two distributions under test as parameters. Two individual t-tests determine whether the mean difference is lower than the upper bound and whether it is higher than the lower bound with a given confidence (α). The passing of both t-tests show that, the mean difference is between the lower and upper bounds with the given confidence.

However, TOST in its original form has a limitation when it comes to the evaluation of the mean differences of two distributions: when these mean differences are close or equal to the boundary values, the TOST concludes that the distributions are not equivalent. This happens due to the t-value of the individual t-tests resulting in values closer to 0 when the mean differences are close to boundary values. In the paradigm where TOST is commonly used (e.g. in drug test trials), the boundaries are regarded as the worst values that the mean difference can get. However, in equivalence testing for engineering, we expect a test which accepts boundary values and also the values which are closer to the boundaries.

TOST can be used in two paradigms referred to as bio-equivalence and quality engineering [225]. For this purpose, Pardo [225] demonstrates the following formulas that compute new boundaries ($\overline{X}_H$ and $\overline{X}_L$) given a target mean difference (μ), where s and n are standard deviation and cardinality of the mean differences distribution. t_α is the one sided t-test value at a confidence value of α.

Selecting a critical region with α confidence such that $\overline{X}_H$ is higher than μ,

$$\overline{X}_H = \mu + t_\alpha \frac{s}{\sqrt{n}} \tag{7.2}$$

Selecting a critical region with α confidence such that $\overline{X}_L$ is lower than μ,

$$\overline{X}_L = \mu - t_\alpha \frac{s}{\sqrt{n}} \tag{7.3}$$

The limitation is overcome using the confidence interval of $\overline{X}_L$, $\overline{X}_H$ instead of $-\mu$, μ as mean difference boundaries.

7.2.6 Automatic Approaches to Handling Side-Channel Leakage

Due to the numerous problems and pitfalls with countermeasures against side-channel attacks as previously discussed, researchers developed several automated approaches for handling side-channel leakage. The approaches can be grouped into three categories, simulation-based, code analysis, and hardware-assisted.

Simulation-based Approaches

Veshchikov [292] presents the SILK simulator, which simulates a high level abstraction of the source code of an algorithm that generates traces. Another simulator, MAPS [91] targets the Cortex-M3 and bases its leakage properties on the Hardware Description Language (HDL) source code. The simulator mainly focuses on leakage caused in the pipeline.

These two simulators only automate the generation of traces. Hence, they are basically assist the leakage evaluation process and speed it up.

Code Analysis

Barthe et al. [26] describe how to automatically verify higher-order masking schemes and present a new method based on program verification techniques. The work of Wang and Schaumont [295] explains how formal verification and program synthesis can be used to detect side-channel leakage, prove the absence of such leakage and modify software to prevent such leaks. However, both of these works remain limited in the ways they model the hardware and actual implementations.

Closer to ours are works that, although sacrificing generality, address the problem of "fixing" the leakage from a specific device. Papagiannopoulos and Veshchikov [224] perform an in-depth investigation of device specific effects that violate the *independent leakage assumption* (ILA) [241]. They also provide an automated tool that can detect such violations in AVR assembly code.

Another method to eliminate timing side channels in software was proposed by Wu et al. [306]. Their method requires a list with secret variables as input and produces code that is functionally equivalent to the original code but without timing side channels. In a recently published work Wang et al. [296] describe a type-based method for detecting leaks in source code. They implemented their mitigations in a compiler and evaluated their method. Eldib and Wang [110] propose a method to add countermeasures to source code that masks all intermediate computation results such that all intermediate results are statistically independent.

Agosta et al. [6] introduce a framework to automate the application of countermeasures against Differential Power Analysis (DPA). Their approach adds multiple versions of the code preventing an attacker from recognizing the exact point of leakage.

Hardware-Assisted Masking

Implementing masking within the processor promises a way of avoiding unintended interaction between masked values. Masking apply to the processor as a whole [139, 207], or only to a part of the instruction set [166, 126].

7.3 Evaluation Setup

To evaluate Elmo, we compare its output with leakage assessment of the code on the real hardware. Our evaluation setup is shown in Figure 7.2.

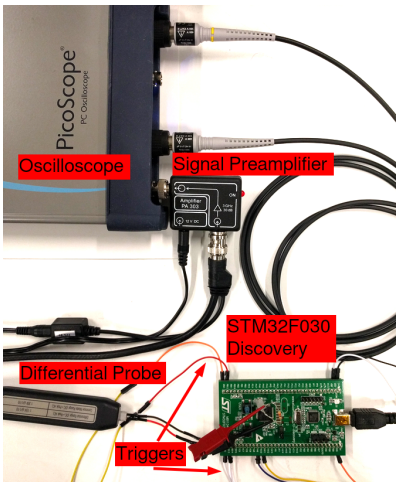


Figure 7.2: Evaluation Setup.

We evaluate Elmo with the same STM32F030 Discovery evaluation board used in McCann et al. [197]. Following the instructions of McCann et al., we disconnect one of the two power inputs of the System on Chip (SoC) and attach a 330 Ω shunt resistor to the second power input. To avoid switching noise, we use a battery to power the evaluation board.

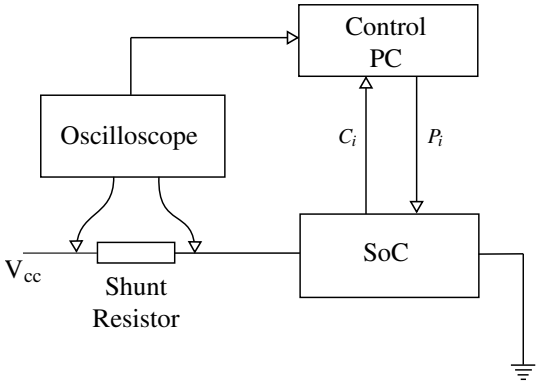


Figure 7.3: Evaluation Setup — Circuit Diagram

We use a PicoScope 6404D with a Pico Technology TA046 differential probe connected to the oscilloscope via a Langer PA 303 preamplifier, to measure the voltage drop across the shunt resistor as a proxy for the power consumption of the SoC.

See circuit diagram in Figure 7.3. For Rosita++, we additionally filter our lower frequencies by adding a 400kHz high pass filter.

We sample every 12.8 ns, which, with a clock rate of 8 MHz, is roughly 9.77 samples per clock cycle. The samples are 8-bit wide and our PicoScope can store up to 2 giga samples before running out of memory.

We use a control PC to orchestrate the experiments. The PC controls the oscilloscope and the STM32F030 Discovery evaluation board. It sends the software to be tested and the data to be used to the evaluation board, and collects the trace data from the oscilloscope. The control PC also generates all of the randomness required for the experiments. As a source we use `/dev/urandom`, which is considered cryptographically secure. The control PC generates the random inputs for the fixed vs. random tests. It further sends a stream of random values to be used for masks by the evaluation board.

Each experiment collects multiple power traces from running the software on the evaluation board. The execution of the software alternates between the fixed and the random cases. Thus, half of the collected traces are for the fixed case and the other half is for the random tests. Fixed and random tests are run randomly interleaved to make sure that the internal state of the device is non-deterministic at the start of each test [260]. To identify the start and end of the segment that we monitor, we use the output pins of the device to trigger the trace collection and to mark the end of the points of interest. Later, we use these trigger points to filter out traces with clock jitter.

To detect leakage, we employ non-specific TVLA. That is, we check the distribution of the values at each trace point and use the Welch t-test to check if the samples in the fixed and in the random traces are drawn from the same distribution. Following the common practice in the domain, we use a t-test value above 4.5 or below -4.5 as an indication of leakage. We validate the setup using the example from McCann et al. [197], getting results similar to theirs, for more details see Section 7.4.1.

7.4 Rosita

Rosita aims to automate the process of producing first-order leakage-resilient software. Specifically, we focus on reducing the manual effort required for ensuring conformance with the ISO 17825 standard. We assume that the underlying algorithm employs a protection technique, such as masking. However, unintended interactions between data, introduced in the execution of the software, can break the independent leakage assumption [241] and leak secret information through a physical side channel, such as the power channel.

We consider two sources of interactions. In architectural interaction, the program overwrites a register with a related value, leaking information through transitional effects. Microarchitectural interactions occur due to transitional effects and glitches within the microarchitecture. For example, when a value of a pipeline stage register is overwritten. We do not handle cases where the application of masking is incorrect, either due to a programmer error or due to compiler optimizations. Similarly, we do not protect against attacks that expose the full state of the cipher [174].

To fix unintended interactions, implementers typically go through a manual, iterative process whereby the software is installed on the target device, the leakage is measured, and fixes are applied to the machine code, until the leakage is reduced to an acceptable level for the target use case.

This process, naturally, requires a significant level of expertise both in setting up and conducting the experiment to assess the leakage and in fixing the software to reduce the leakage. Moreover, because the assessment requires a large number of encryption rounds on relatively low-performing devices, and a number of repetitions in repairing the leakage and evaluating, the process is time consuming.

Rosita automates this process as shown in Figure 7.1. To produce leakage-resilient cryptographic software, we start with a (masked) implementation of the cryptographic primitive. We use cross-compilation to produce both the assembly code (if the original code is in a high-level language) and the binary executable for the target device. The binary executable is then passed to a leakage emulator, in our case Elmo*, a modified version of Elmo [197], to perform leakage assessment. This assessment identifies the leakage and the machine instructions that cause it. Rosita processes the output of Elmo*, together with the assembly code. It applies a set of rules that replace leaky assembly instructions with functionally-equivalent sequences of instructions that do not leak. Afterwards, the produced assembly program is assembled and fed back to Elmo* and the process repeats until no further fixes can be applied, at which time Rosita produces a report indicating the remaining leakage, if any. In all of our experiments, Rosita terminates within a small number of rounds when it detects no further leakage. The rules that Rosita applies are incremental. Hence, Rosita is guaranteed to converge within a finite number of rounds, when all rules are applied in all program positions.

Note that our approach makes use of a leakage emulator. Prior static-analysis-based solutions, such as [224, 295, 306, 292, 91], rely on tags that identify the nature of values within the program. For example, in ASCOLD [224] the programmer needs to assign tags to values, e.g. identifying them as random or masked. The main downside of the tagging approach is that any mistake the programmer makes in tagging values can be translated to missed leakage. In contrast, Rosita applies TVLA, using a procedure that extends ISO 17825, to the emulated power trace. As such, Rosita depends neither on the programmer's proficiency nor on specific properties of the masking scheme to detect leakage. Subject to the accuracy of the emulator and the strength of the statistical tools applied, Rosita will detect leakage in the implementation (up to the level which the masking scheme used is meant to protect).

7.4.1 Leakage Emulation

Due to Rosita's reliance on a leakage emulator, care should be taken when selecting one. For this work we select Elmo [197] as a basis because, unlike instruction-level emulators, it is tailored to a specific processor model, while at the same time it does not require detailed design information to build its model. We now describe how Elmo models the device it emulates and the leakage. We then identify limitations for using Elmo with Rosita and describe how we address these and develop Elmo*.

The Elmo Leakage Model

Emulating the hardware at the transistor level would produce the most accurate leakage estimate. However, this is often infeasible, both due to the complexity of such analysis and because the hardware implementation details are not available to the security evaluators and software developers.

Instead, leakage emulators use an abstract model of the device and of its power consumption. The abstract model is significantly simpler than emulating at the transistor level. At the same time, using an abstract model reduces accuracy and may result in missing some leakage. Thus, the *leakage model* presents a trade-off between modeling cost and accuracy.

Elmo's model of the hardware considers bit values and changes in bit values over the Arithmetic Logic Unit (ALU) inputs and outputs and memory instructions. Specifically, each operand is compared to the corresponding operand of the preceding instruction. Power consumption is modeled as linear combinations of bit values or bit changes.

Elmo models 21 instructions that its authors claim cover typical use in cryptography. These 21 instructions are divided into five groups, each modeled separately. To generate the model, power traces are collected while the processor executes sequences of three instructions. Each trace is processed to select a point-of-interest to be used as a representative of the trace. Elmo then performs a linear regression on the data collected in the traces to find the coefficients for the model.

The model itself consists of 24 main components, each modeling a specific part of the architecture. These cover:

- A linear combination of the bit flips between each operand of the current instruction and the corresponding operand of the previous and the subsequent instructions.
- A linear combination of the bit values of the operands of the current instruction.
- The instruction groups of the previous and subsequent instructions.

Elmo provides a pre-computed model of the STM32F030[2] evaluation board which features an ARM Cortex-M0 based STM32F030R8T6 System-on-Chip (SoC).[3]

Validating the Setup. To validate our setup, we reproduce the results of McCann et al. [197]. Specifically, we perform a fixed vs. random test on the code in Listing 7.1, which contains an implementation of one of the steps in the AES encryption known as the ShiftRows operation. Specifically, register `r1` points to the 16 bytes that represent the state of the AES encryption. ShiftRows performs a fixed permutation of these bytes. The implementation loads three four-byte words and uses the `rors` instruction to rotate the bytes, before storing them back to the state.

Listing 7.1: ShiftRows from McCann et al. [197]

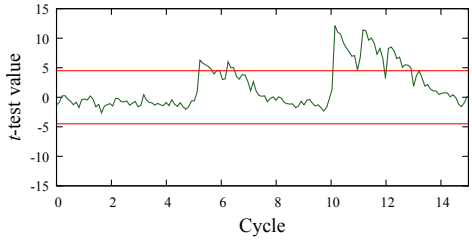
```
ldr   r4, [ r1, #4 ]
```

[2] `https://www.st.com/en/evaluation-tools/32f0308discovery.html`

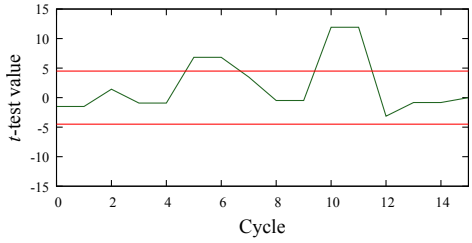
[3] Elmo also provides a model for the Cortex-M4-based STM32F4 Discovery board, which we do not use in this chapter.

```
rors r4, r5
str  r4, [r1, #4]
ldr  r4, [r1, #8]
rors r4, r6
str  r4, [r1, #8]
ldr  r4, [r1, #12]
rors r4, r3
str  r4, [r1, #12]
```

For the fixed vs. random test we collect 2500 traces where the state contains fixed data masked with the same mask value and 2500 traces where the state consists of random values masked with the same mask. A random value for the mask is chosen for each trace. We compare the distribution of the power reading in each sample point between the fixed and the random traces, and calculate the Welch t-test to check the likelihood that the means of the two distributions are the same. As mentioned before, following common practice in side-channel analysis, we consider the means of the distributions different enough to indicate leakage if the absolute value of the t-test value is above 4.5.



(a) Real traces in STM32F030.



(b) Simulated traces from Elmo.

Figure 7.4: Fixed vs. random of the AES ShiftRows operation.

Figure 7.4(a) shows the result of the fixed vs. random test. The horizontal axis shows the time and the vertical axis shows the t-test value. We indicate instruction boundaries with vertical bars, and the t-test threshold of ± 4.5 with horizontal red lines. Comparing the figure to the results of running Elmo on the same code, shown in Figure 7.4(b), we see that Elmo produces a fairly accurate simulation of the leakage.

In particular, our figure resembles Figure 5 of [197], with only minor differences that reflect the different test environment.

Storage Elements and the Elmo Model

The Elmo model of the hardware only looks at interactions between arguments and outputs of successive instructions. However, it overlooks interactions that span multiple cycles. These interactions happen between instruction arguments and values that are stored in storage elements such as registers, memory, or latches.

To evaluate interactions overlooked by Elmo, we design a systematic battery of small sequences of code that aim at highlighting interactions via storage elements between instructions. An example of such code is shown in Listing 7.2. The code aims to check if there is an interaction between the value stored in Line 1 and the value used as the second argument of the `eors` instruction in Line 11. The purpose of the `movs` instructions between the two tested instructions is to eliminate leakage between pipeline stages. The sequence of nine `movs` instructions ensures that the `str` instruction is completed by the time the `eors` instruction enters the pipeline.

Listing 7.2: Evaluating interactions between the `str` and the `eors` instructions.

```
1  str  r1, [r2]
2  movs r7, r7
3  movs r7, r7
4  movs r7, r7
5  movs r7, r7
6  movs r7, r7
7  movs r7, r7
8  movs r7, r7
9  movs r7, r7
10 movs r7, r7
11 eors r3, r4
```

For the test, we collect 10 000 power traces of running the code segment, each run using different random values for the data the code processes. (For example, in Listing 7.2 we randomize `r1`, `r3`, `r4`, `r7`, and the contents of the memory address pointed to by `r2`.) For each run we also record the Hamming distance between the two values we investigate. (In this example, the values of `r1` and `r4`.) We then calculate the Pearson correlation coefficient between the Hamming distance and the values in each point of the trace. A high correlation coefficient indicates that the Hamming distance between the values leaks through the power trace, implying that the first instruction keeps the value it processes in some storage element that interacts with the data processed by the last instruction.

Figure 7.5 shows the Pearson correlation coefficients for two code sequences. One from Listing 7.2, testing leakage from `str` to `eors`, and the other testing leakage from `ldr` to `eors`. As we can see, the code in Listing 7.2 show a pronounced dip in the correlation coefficient around cycle 25, indicating interaction between the values. Conversely, the correlation coefficient when replacing the `str` with `ldr` remains close to zero, indicating no leakage.

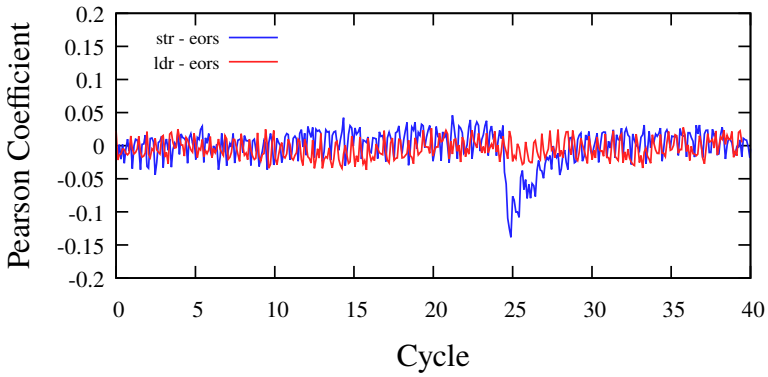


Figure 7.5: Pearson correlation coefficient of interference test.

Dominating Instructions

The methodology we discuss in Section 7.4.1 allows us to find instruction pairs that interact via hidden storage within the processor. However, each such instruction may affect multiple storage elements. To correctly model leakage through these elements, we need to know which instructions affect which storage elements. Because the design details of the processor are not public, we cannot positively identify the storage elements used by an instruction. Instead, we search for *dominating* instructions in pairs, i.e. instructions that set more storage elements than others. For that, we use code sequences similar to Listing 7.3, which checks if `str` dominates `eors`. Specifically, we pick a pair of instructions with interacting storage. In the test code we use two instances of the first (Lines 1 and 9), with the second instruction separating these two instances (Line 5). If the first instruction dominates the second, leakage will be visible at the second instance of the first instruction (Line 9).

Listing 7.3: Checking for a dominating instruction

```
1  str   r1, [r2]
2  movs  r7, r7
3  movs  r7, r7
4  movs  r7, r7
5  eors  r3, r7
6  movs  r7, r7
7  movs  r7, r7
8  movs  r7, r7
9  str   r4, [r5]
```

Theoretically, it is possible to have a pair of instructions that each only affects part of the state set by the other. However, we did not find any such pair.

Findings

We run a broad range of experiments, with (1) some focusing on architecturally known storage elements, such as registers and memory, and (2) others aiming to find micro-architectural storage elements by testing interactions between pairs of instructions. We find several sources of leakage that ELMO does not identify. We note that Gao [125] also identifies many of the issues we find; however, their identification was driven by the iterative tweaking of a cipher, whereas our systematic approach is cipher-agnostic. Of the leakage we find, the first is an architectural issue, whereas the others are microarchitectural. Except where mentioned otherwise, we believe that the cause of all microarchitectural leakage is transition effects, because the leakage corresponds to a change in the logical state. However, without access to the processor implementation details, we cannot rule out the possibility that it is caused by glitches. When the leakage does not correspond to a change in the logical state of the processor we assume that the leakage is due to glitches.

Registers. We find that overwriting a register leaks the (weighted) Hamming distance between the previous value and the new value. This is a significant leakage source, because reusing a register that contains a masked value for another value with the same mask leaks secret information. Unlike Papagiannopoulos and Veshchikov [224], we do not find leakage across different registers.

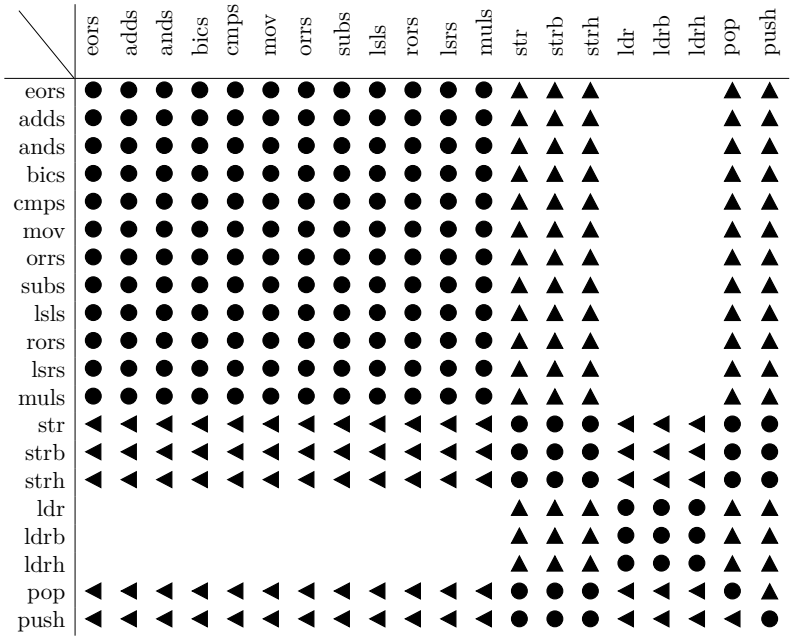
Memory. Writing data to memory interacts with data already stored in the same location. Hence, overwriting one masked value with another may remove the mask, leaking the values.

Instruction Pairs. We analyzed all pairs of instructions for leakage from both arguments. The results for the second argument are summarized in Table 7.2. We see that all instructions set some state, and that most pairs do interact with this state. We now discuss some of our observations about the storage elements used.

Memory Bus. The memory bus seems to have a storage element that stores the most recent value stored to or loaded from the memory. When loading from or storing to memory, the value of the storage element is overwritten, leaking the Hamming distance between the previous and the new value. This leakage differs from the two described above, and happens irrespective of the registers and the memory addresses used. Consequently, when writing to or reading from memory, care should be taken to only access non-secret values or values masked with different masks. We note that the storage element could be the contents of the addressed memory itself, where the power leakage correlates with changing the contents of the memory bus.

It is important to note that the storage element always stores a 32-bit word. Thus, when loading or storing a byte, the whole 4-byte aligned 32-bit word that contains the byte is moved to the storage element. This may create memory interaction between memory operations that seem completely unrelated. For example, consider the code in Listing 7.4. In this example we assume that memory locations 0x300 and 0x400 both contain one secret byte each, both masked with the same mask. The code in this example performs two memory operations, the first stores a byte into address 0x303 and the second reads a byte from location 0x402. We note that none of these locations contains secret data, and the data stored is also not secret. However, the

Table 7.2: State interactions between the second operands of instruction pairs. Triangles point to the dominating instruction. Circles indicate interactions on the same storage.



store operation loads the 32-bit word in memory locations 0x300–0x303 into the memory bus, and the following load operation replaces the contents with the 32-bit word in memory location 0x400–0x403. This causes an interaction between the data in memory locations 0x300 and 0x400, leaking the Hamming distance between the data stored in these locations.

Listing 7.4: Example of word interaction

```
1  movs r3, 0x303
2  movs r4, 0x402
3  movs r7, r7
4  movs r7, r7
5  movs r7, r7
6  strb r5, [r3]
7  movs r7, r7
8  movs r7, r7
9  movs r7, r7
10 ldrb r6, [r4]
11 movs r7, r7
12 movs r7, r7
13 movs r7, r7
```

A further issue in the memory bus is an interaction between the bytes of words

loaded from or stored to memory. Specifically, our analysis shows that when memory data is accessed, consecutive bytes in the word interact with each other. Thus, if a word contains multiple bytes that are all masked with the same mask, loading it from or storing it to memory will leak the Hamming distance between consecutive bytes. We note that due to the memory bus storage element described above, the leakage occurs even if the memory access operations access a single byte of a 32-bit word.

Store Latch. We find that storing a register to memory results in potential interactions between the value of that register and the second argument of subsequent ALU instructions, such as `eors`. However, if the contents of the register changes between the `str` and the ALU instruction, the second argument of the ALU instruction interacts with the *updated* value of the register rather than with its original value.

Listing 7.5: Store latch example.

```
1 str  r5, [r3]
2 movs r7, r7
3 movs r7, r7
4 movs r7, r7
5 movs r5, r2
6 movs r7, r7
7 movs r7, r7
8 movs r7, r7
9 eors r1, r4
```

For example, in the example in Listing 7.5, the code stores the value of `r5` to memory (Line 1). It then updates the value of `r5`, moving the contents of `r2` to it (Line 5). Finally, it calculates the exclusive-or of `r1` and `r4`. Our experiments show leakage in Line 9, which correlates with the Hamming distance between the original values of `r2` and `r4`. Interestingly, we note that the update of the interacting register takes one cycle to become effective. That is, removing Lines 6–8 in 7.5 removes the interaction between the original values of `r2` and `r4`, but leaves an interaction between the original values of `r5` and `r4`.

We believe that the processor maintains a reference to the most recently stored register. This reference is used as an input to a multiplexer that selects the contents of the referenced register. Implementing the `str` instruction requires two cycles [122, Figure 4.6]. In the first, the processor calculates the store address and in the second it performs the store. We believe that, to avoid locking the register file, in the first cycle the processor copies the contents of the register to an intermediate latch, from which it is retrieved in the second cycle. We believe that a glitch on the bus causes interference between the contents of the latch and the second argument of subsequent instructions, explaining the leakage we observe.

Extending the Elmo Model

Recall (Section 7.4.1) that Elmo builds its model using a linear regression from traces collected from sequences of three instructions. To account for the effects of storage elements we identified, we update the model to include a few more components. We call out extension Elmo*.

Whereas the ELMO model treats each operand separately, we also look at combinations of bits across the two operands of the instruction. Because the first operand is typically the destination register, correlating the two operands captures the effect of calculating the result of the operation and overwriting the destination register.

To capture interactions via memory and internal storage elements, we track the contents of these elements, and add model components that correlate with them.

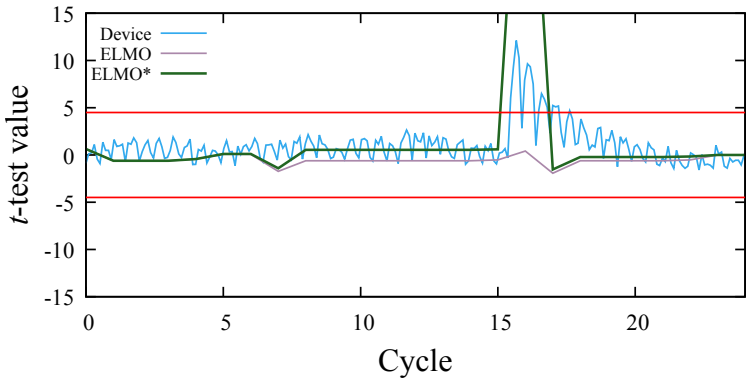


Figure 7.6: Leakage from `str` to `eors`.

In total, our model consists of 25 components. We validate our model by repeating the test cases used for identifying the storage elements. For example, Figure 7.6 shows the real and the emulated leakage from running the code in Listing 7.2 on the real hardware, ELMO, and in ELMO*. We see that our model identifies the leakage that the original ELMO model misses. We note that the leakage in ELMO* has a higher t-test value than the actual leakage. We speculate that the reason is that the model does not suffer from physical noise. As a result, ELMO* requires less traces than a hardware measurement for leakage detection.

Unlike ELMO, ELMO* not only emulates the leaked signal, but also finds which of the components of the model is causing the leak. We, therefore calculate the t-test value not only for the combined emulated signal, but also to separate components of the signal.[4] Thus, for example, we keep track of the t-test value of the part of the signal contributed by each of the instruction operands, by interactions between the instruction result and its operands, and by interactions between the instruction results or operands with values previously stored to or loaded from memory. Using this information, when ELMO* reports that an instruction leaks, we can inspect the components and identify the leakage cause.

7.4.2 Code Rewrite in Rosita

As Section 7.4 describes, the core of ROSITA is a rewrite engine that uses the output of the ELMO* emulator to drive code fixes for leakage. We assume that the original

[4]Implementation note: to reduce memory usage, we calculate the t-test values incrementally, using Welford's algorithm [301].

code is masked, i.e. it does not leak at the algorithm level. However, the translation of the algorithm into machine code and the execution of this machine code can result in unintended and unexpected leakage. In this section, we review the causes of leakage we identify, and describe the fixes the ROSITA applies for each. We begin with a high-level description of ROSITA and proceed with the details of the rewrite rules it applies.

Rosita Design

ROSITA is a rewrite engine that takes the code and the output of ELMO*, and rewrites the code to avoid leakage. To decide which rewrite rule to apply, ROSITA relies on ELMO* to identify the leaking component. (See Section 7.4.1.)

The main strategy ROSITA uses to fix the leakage is to wipe stored state with a random mask. For that, ROSITA dedicates a *mask register* (ROSITA uses register `r7`), which is initialized with a random 32-bit mask. When compiling the software, we use the flag `-ffixed-r7` to direct the compiler not to use the mask register, ensuring that its contents are not modified except by ROSITA. Similarly, we require assembly implementation to not use `r7`.

It is important to note that all of the rewrite rules do not eliminate values used by the program. Thus, if the implementation is not fully masked or uses incorrect randomness, ROSITA will be unable to remove the leakage. Conversely, a success in eliminating a leak is a demonstration that the leak originated from unintended interaction.

Operand Interaction

One of the common forms of unintended interaction is between the operands of successive instructions. Technically, as McCann et al. [197] note, loading an operand to the bus leaks the Hamming distance between the value previously held in the bus and the new value. If both values use the same mask, the Hamming distance between the masked values is the same as that between the original values.

ROSITA identifies such leakage by checking the various t-test values calculated for the operands and their relationship with those of prior instructions. In the case that the leakage is caused by such interaction, ROSITA inserts an access to the mask register, using `movs r7, r7` The instruction moves the contents of the mask register into the mask register, and is therefore functionally a no-op. However, because the value of the mask register goes through the bus, the previous contents of the bus is wiped, removing the interaction between the two masked values.

Register Reuse

Due to the limited number of registers, compilers and programmers often reuse those, e.g. when the old contents are either consumed or stored in memory. A register is typically not emptied after use. Consequently, when new data is loaded into a register, it interacts with algorithmically unrelated data that remains from prior uses of the register.

Papagiannopoulos and Veshchikov [224] note that if the old contents and the new contents are both masked using the same mask value, the difference between

the masked contents, i.e. their exclusive or, is the same as the difference between the unmasked contents. Consequently, when a register is used consecutively for two values with the same mask, it leaks the difference between the values.

To identify this form of leakage, ROSITA checks the t-test value of overwriting register value. Once identified, ROSITA wipes the old contents of the register by copying the contents of the mask register to the destination register of the leaking instruction, as Papagiannopoulos and Veshchikov [224] suggest. For example, suppose that the instruction `movs r3, r4` leaks because both `r3` and `r4` contain values masked with the same mask. To eliminate the leak, ROSITA inserts `movs r3, r7` before the leaking instruction.

Rotation Operations

Rotation operations show interaction between the value pre and post rotation. When a single masked value is rotated, this interaction is unlikely to leak secret data because the mask hides the contents. However, when rotating a word comprised of multiple masked values that all use the same mask, the result of the rotation may align the masked values, effectively nullifying the mask, leaking the difference of the unmasked values.

We propose two approaches to remove this leakage. As an example, suppose that we would like to rotate the register `r2`, whose value is a concatenation of four masked bytes: $(b_1 \oplus m)||(b_2 \oplus m)||(b_3 \oplus m)||(b_4 \oplus m)$. Rotation of `r2` by a multiple of 8 bits would result in leakage of information on the value of the b_i's. For example, assuming `r3` contains the value 8, the instruction `ror r2, r3` would set the value of `r2` to $(b_2 \oplus m)||(b_3 \oplus m)||(b_4 \oplus m)||(b_1 \oplus m)$, and the interaction between the original and the rotated values of `r2` would leak the Hamming weight of $(b_1 \oplus b_2)||(b_2 \oplus b_3)||(b_3 \oplus b_4)||(b_4 \oplus b_1)$.

Word Mask. A straightforward approach for preventing such leakage is to mask the word with our mask register (`r7`), rotate both the word and the mask register and then use the rotated mask to unmask the word. Thus, instead of rotating `r2`, we rotate `r2` $\oplus$ `r7`. As an example, Figure 7.7 shows how ROSITA fixes a `rors r2, r3` instruction that ELMO* indicates is leaking.

```
rors r2, r3                    eors r2, r7
                               rors r2, r3
                               rors r7, r3
                               eors r2, r7
```

Figure 7.7: Masking rotation operations. The leaking `ror` operation on the left is replaced with a masking sequence on the right.

We note that this sequence modifies the contents of our mask register. However, this has no effect on the functionality because the mask register is assumed to be random and there is no long-term dependency on its exact contents.

Partial Rotations. An alternative approach is to combine multiple shifts to avoid rotations of multiples of the data size. For example, a rotation by 8 bits can be replaced with a rotation by 3 bits followed by a rotation by 5 bits.

Rosita employs the word mask approach both because it is more general, i.e. does not depend on the size of the rotation, and because it already has the mask register, which it uses for the other fixes.

Memory Operations

As discussed in Section 7.4.1, there are several effects that can cause interactions between values used in memory operations. These include a storage element in the memory bus that remembers recently accessed memory value and consequently leaks the Hamming distance between the remembered value and the current one on memory access operations, interaction between loaded and stored values and the previous contents they overwrite, and an interaction between bytes in stored words.

When Elmo* indicates that a load instruction leaks due to interaction with the memory bus storage element, Rosita wipes the contents of the bus by pushing the mask register to the stack and popping from the stack to the destination register of the load instruction. Figure 7.8 shows an example of an `ldr` instruction (left) that leaks through interaction of the loaded value with a previously loaded value. To fix this, Rosita inserts a `push` and a `pop` instructions before the load, yielding the code fragment in the right. Popping the mask to the destination of the load instruction also protects against leakage through interaction with the previous value of the destination register.

```
ldr r2, [r3]          push r7
                      pop r2
                      ldr r2, [r3]
```

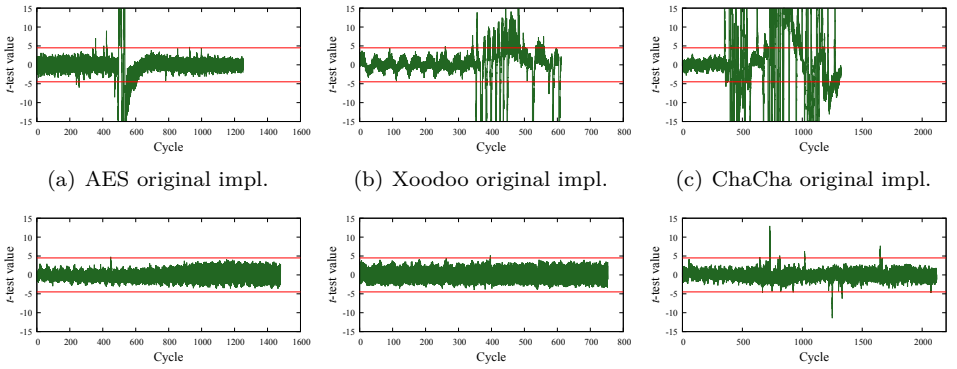
Figure 7.8: A leaking load instruction (left) and the fixed sequence (right).

Due to the more intricate potential interactions, the picture with store instructions is a bit more complex. To overcome interactions with the previous value used on the memory bus and to address possible interactions with the previous contents of memory, Rosita first stores the mask register into the destination location and then performs the required store (See Figure 7.9).

```
str r2, [r3]          str r7, [r3]
                      str r2, [r3]
```

Figure 7.9: A leaking store instruction (left) and the fixed sequence (right).

When byte interaction within the stored data leaks, Rosita stores one byte at a time. In such a case, care should be taken to ensure that these bytes and the operations required for their storage do not create unintended interactions, leading to a relatively long code segment in Listing 7.11 in Section 7.4.2. While this rewrite rule eliminates the leakage, the performance cost of using it is significant. As such, it may be better to avoid stores of words that contain multiple values masked with the same mask. Changing the logic of the cipher is outside the scope of Rosita.



(a) AES original impl. (b) Xoodoo original impl. (c) ChaCha original impl.

(d) AES fixed with ROSITA. (e) Xoodoo fixed with ROSITA. (f) ChaCha fixed with ROSITA.

Figure 7.10: fixed vs. random tests for the three cipher (one fixed input, 1m traces).

Eliminating Byte Interaction in Stores. Figure 7.11 shows an example of the rewrite rule for eliminating interactions in byte stores. The code uses two registers chosen to not conflict with the store, `r0` and `r6` in the example in Figure 7.11. The first is used for selecting the byte to store, while the second is used for the byte. ROSITA uses two stores for each byte to avoid interactions on the memory bus or in the SRAM.

7.4.3 Evaluation

We evaluate ROSITA with masked implementations of three cryptographic primitives. AES [96] is one of the most commonly used ciphers, having been an international standard since 2001. We use the byte-masked implementation of AES-128 by Yao et al. [308].[5] To perform the SHIFTROWS operation of AES, which permutes bytes in the data being encrypted, the implementation uses byte loads and stores. Following the suggestion of Gao [125], we use different masks for each row to avoid leakage through interactions between bytes in memory words.

Efficient software AES implementations on CPUs (without dedicated AES instruction) use table lookups, which makes them vulnerable to cache-based attacks. As an example of a modern primitive, the second cipher we use is the cryptographic permutation Xoodoo. Xoodoo was proposed recently by Daemen et al. in [94] for use in authenticated encryption modes [95]. The optimized and non-masked implementation of Xoodoo we took from [55] and we implemented the 2-share boolean masking scheme of the non-linear layer χ as in [52] ourselves. Implementing the 2-share boolean masking of the linear layer was trivial. In contrast to what [52] mention, we initialize the state with fresh randomness for each trace to keep it consistent with the implementation of AES, even though this is not required.

[5] `https://github.com/Secure-Embedded-Systems/Masked-AES-Implementation/tree/master/Byte-Masked-AES`

```
str r2, [r3]
```

```
push {r6}
push {r0}
movs r0, #0xff
movs r6, r2
ands r7, r7
ands r6, r0
lsls r0, #0
strb r0, [r3, #0]
strb r6, [r3, #0]
movs r6, r7
movs r6, r2
movs r0, #0xff
lsls r0, #8
ands r7, r7
ands r6, r0
lsrs r0, #8
lsrs r6, #8
strb r0, [r3, #1]
strb r6, [r3, #1]
.
.
.
pop {r0}
pop {r6}
```

Figure 7.11: Addressing byte interaction in stores. A leaking store instruction (left) and part of the fixed sequence (right).

The third cipher we consider is ChaCha as a prominent example of an ARX cipher. ChaCha is very efficient in software and widely used in TLS implementations. The challenge is to mask it at low cost and the best result for ARM Cortex-M3 and Cortex-M4 processors was recently published by Jungk et al. [161]. We use their implementation in our experiments.

Fixing Leakage

We first show Rosita's success in fixing the leakage it detects. Figure 7.10(a) shows the results of a non-specific fixed vs. random experiments with 1 000 000 traces of executing the first round of the AES implementation. The figure shows leakage (t-value above the threshold of 4.5) around cycles 500–550, which correspond to the AES ShiftRows operation. As Figure 7.10(d) shows, Rosita detects the leaks and fixes them. This fix does not, however, come for free. The first round now takes 1 479 cycles, compared with 1 285 for the original implementation—a slowdown of 15%. Figure 7.10(b) and Figure 7.10(e) show similar results for ChaCha and Xoodoo with 61% (1322 vs. 2122 cycles) and 18% (637 vs. 753 cycles) slowdowns respectively.

To determine the trend of leakage, we perform the fixed vs. random test on the

hardware with a varying number of traces. Figure 7.12 shows the results for both the original and the fixed implementations. The horizontal axis shows the number of traces used for the fixed vs. random test, and the vertical is the maximum absolute value of the t-test for each of the implementations. As we can see, the original implementations show increasing leakage, significant leakage is visible even with as little as 1 000 traces, and the confidence increases as traces are added. Our fixed implementations show significantly less leakage up to 1 000 000 traces. To remove the remaining leakage, we need to use more than one input. We discuss this issue next.

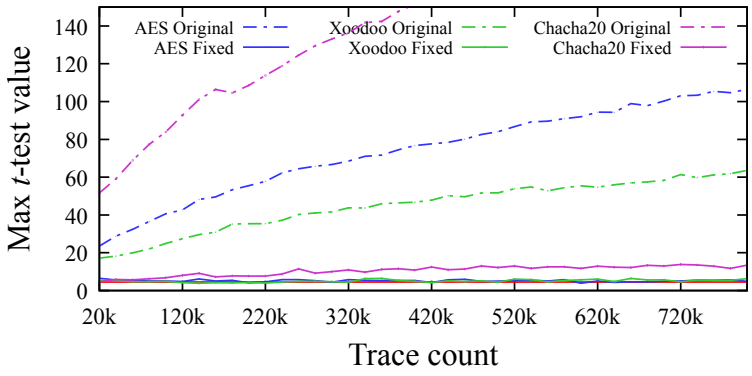
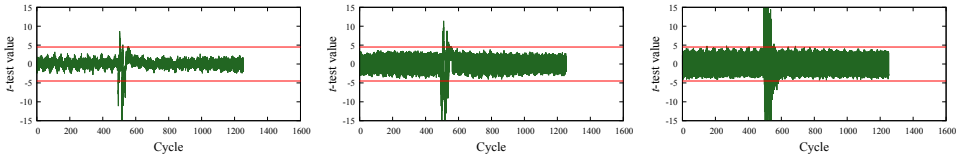


Figure 7.12: t-test value trend.

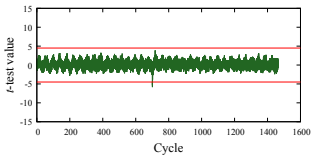
7.4.4 Multiple Fixed Inputs



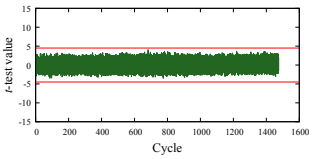
(a) One fixed input – before ROSITA.

(b) Five fixed inputs – before ROSITA.

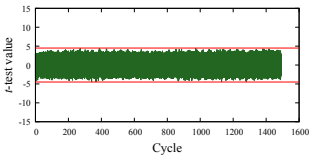
(c) 100 fixed inputs – before ROSITA.



(d) One fixed input – after ROSITA.

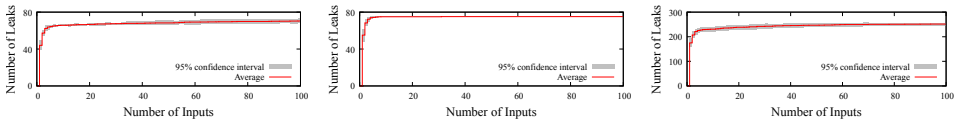


(e) Five fixed inputs – after ROSITA.



(f) 100 fixed inputs – after ROSITA.

Figure 7.13: t-test of masked AES implementation before and after ROSITA, varying the number of fixed vs. random pairs.



(a) AES (b) Xoodoo (c) ChaCha

Figure 7.14: Average number of leaks per fixed inputs.

One of the known limitations of TVLA is that it may miss some leakage if used with only one input [276]. Thus, common operating procedures require running multiple fixed vs. random tests, each with a different fixed input. For example, the ISO 17825 standard requires two fixed inputs. To test the impact of multiple fixed inputs, we perform multiple fixed vs. random tests, each with a different randomly chosen fixed input. To combine the results of multiple experiments, we use the largest absolute t-value calculated for a sample point as a representative for leakage at that point. Thus, if any of the fixed vs. random tests indicates leakage at a point, the combined result will also indicate leakage there. The top row of Figure 7.13 shows the combined results from 1, 5, and 100 fixed inputs for AES. As we can observe, when we increase the number of fixed points, the number of locations that show leakage increases.

Running Rosita with these inputs allows us to fix most of these leakages. Repeating the experiment with the code that Rosita produces and the same fixed inputs we get the t-test values in the bottom rows of the figures. We can observe that Rosita fixes the leakage, at a cost to the performance. Also, when there is a higher number of leakage points, it is required to have more fixed inputs to cover all of them. For ChaCha, 100 fixed inputs were not enough to cover all leakage points as the final result shows 1 leaky point. Table 7.3 compares the performance of the code before and after running Rosita, showing a maximum overhead of 64% for ChaCha.

Table 7.3: Encryption length (cycles) after fixing with Rosita with varying number of fixed inputs.

cipher	Original	1 Fixed	5 Fixed	100 Fixed
AES	1 285	1 464	1 475	1 479
ChaCha	1 322	2 066	2 114	2 162
Xoodoo	637	735	765	769

The number of leakage points identified depends on the fixed inputs chosen for the fixed vs. random test. To better understand the relationship, we want to find out how many leakage points we expect to find for a given number of fixed vs. random inputs. Figure 7.14 shows this for AES, ChaCha and Xoodoo. For a given number of inputs, the plots display the average number of leakage points that Elmo* discovers over 10 selections of inputs. The figure also displays the 95% confidence interval. We see that 10 fixed inputs are enough to find 93% of the leakage points in AES, 92% in ChaCha and 99% in Xoodoo. When we compare the identified leakage points

against the ground truth, we find that many of the discovered leakage points are false positives, explaining the discrepancy between the figures and Table 7.1. To verify that we discover all of the real leakage points, we use Rosita with 100 fixed inputs to fix AES and Xoodoo. We then use the produced code on the hardware with a new set of 100 fixed inputs. We found no evidence for leakage with either AES or Xoodoo but for ChaCha we found one leaky point that had a t-test value of 5.2.

We note that Rosita's success in eliminating leakage demonstrates that the original programs are, indeed, first-order secure, at least semantically. As we mention in Section 7.4.2, the rewrite rules can only fix leakage that stems from unintended interactions.

Performance

The performance bottleneck for Rosita is running Elmo* to generate the simulated traces. We can collect 10 000 traces of AES in 26 seconds, ChaCha in 45 and Xoodoo in 21. In comparison, the code rewrite phase of Rosita takes around 0.1 seconds.

Collecting the same number of 1-round traces from the hardware takes 117 seconds for AES, 220 for ChaCha and 147 for Xoodoo. The collection of traces for Xoodoo from the hardware is slower than for AES because we provide fresh shares for every trace as mentioned above. Hence, the communication dominates the execution time. Thus, Elmo* is 4.5–7 times faster than the real hardware.

We note that the task of collecting traces is ridiculously parallelizable. Hence, on a typical desktop, we can collect traces eight times faster, and with an investment of $1 000 we can double the rate again. In contrast, to parallelize trace collection from the hardware, we would need to replicate the setup, at a cost of over $10 000 per node. Thus, the effective speed of Rosita is about two order of magnitude faster than the hardware.

7.5 Rosita++

Rosita++ aims at detecting and preventing second-order leakage from cryptographic implementations. We assume that the program is nominally second-order secure, i.e. that no combination of two or less values used in the program correlates with secret data. Algorithm 13 shows a simple example of a second-order secure implementation of code that computes the exclusive or of two secret values s and t. Each of the values is represented as three variables. For example s is represented as two random masks s_1 and s_2, as well as a masked value $s_3 = s \oplus s_1 \oplus s_2$ [79]. We note that, assuming the masks are chosen uniformly at random, leakage of any two variables does not allow an attacker to recover any information about the secret values. Thus, the implementation is nominally secure under the 2-probing model [152].

Past work has demonstrated that there is a gap between the nominal security of programs and the level of security provided by a concrete implementation of the program [21, 127, 224, 82, 193]. The main cause is that when the processor executes a program, the leakage corresponds not only with the values processed, but also with the changes in values carried by wires and components. For example, when loading the value of s_2 into its execution pipeline (Line 2), the processor may replace the value of s_1 stored in an internal pipeline register. The leakage from such change correlates

Algorithm 13 Exclusive-or of three-share masked variables

Input: $s = s_1 \oplus s_2 \oplus s_3, t = t_1 \oplus t_2 \oplus t_3$
Output: $u = u_1 \oplus u_2 \oplus u_3 = t \oplus s$
1: $u_1 \leftarrow t_1 \oplus s_1$
2: $u_2 \leftarrow t_2 \oplus s_2$
3: $u_3 \leftarrow t_3 \oplus s_3$

not only with the value of s_2, but also with the changes made in the pipeline register value. Thus, the leakage from Line 2 of the algorithm may correlate with both the value and s_2 and with the bit difference between s_1 and s_2, which equals to $s_1 \oplus s_2$. Similarly, the leakage in Line 3 may correlate both with s_3 and with $s_2 \oplus s_3$. By combining the leakage of $s_1 \oplus s_2$ from Line 2 and the leakage of s_3 from Line 3, the attacker may overcome the nominal second-order security and learn information about s

Past solutions that aim to automate leakage detection [292, 197, 144, 224, 266] and correction [132] focus on first-order leakage. In this section we describe how we adapt Rosita to detect and fix second-order leakage. We first describe Rosita. Then, we outline the main challenges in performing second-order analysis and proceed to describe our approaches for addressing these challenges.

7.5.1 Second-Order Analysis

Challenges for Second-Order Analysis

The core extension required for Rosita++ to support second-order leakage detection and mitigation is support for bivariate analysis. Specifically, instead of looking for instructions that show indication of leakage, we need to look for pairs of instructions that *together* show indication of leakage.

Schneider and Moradi [260] suggest a methodology for performing bivariate analysis. Their approach is to generate artificial bivariate traces from the original univariate traces. For that, the original traces are first preprocessed by calculating the average value for each sample point and subtracting the average from the corresponding sample point in each trace. As shown in Equation 7.1, each point in an artificial trace represents a pair of points in the preprocessed trace, where the value associated with the artificial point is the product of the values for the corresponding points in the preprocessed trace.

Our approach for performing second-order analysis is to replace the use of TVLA in Rosita with the Schneider-Moradi methodology. However, while seemingly straightforward, the approach raises significant challenges.

Challenge C1: Statistical confidence with bivariate traces

The artificial bivariate traces have a artificial sample for each pair of samples in the original traces. Consequently, the length of the bivariate traces grows quadratically with the length of the original traces. That is, for traces of length n, the bivariate traces have a length of $\binom{n}{2} \approx n^2/2$ samples.

The de-facto standard statistical test used in TVLA is to reject the null hypothesis, i.e. report leakage, when the absolute value of the t-test is above a threshold of 4.5, achieving a confidence level of 0.00001. This test, however, fails to account for the multiple comparisons performed in TVLA, where a statistical test is performed independently on each sample point. For a small number of points, the effect of multiple comparisons is negligible. When the trace length increases, multiple comparisons result in false positives, showing leakage where no leakage exists.

Challenge C2: Increased data size

Another issue with bivariate analysis is the increase in the amount of data that needs to be processed compared with first-order univariate analysis. Three factors contribute to this increase. First, due to the effects of noise, the number of traces required for statistical analysis grows exponentially with the order of analysis [79]. Secondly, as discussed, the artificial bivariate traces are significantly longer than the original traces. Thirdly, to increase the statistical confidence while handling Challenge C1 without missing leakage we need to increase the number of traces we process.

Because ROSITA++ repeatedly evaluates implementations, there is a need for efficient methods for handling the increased amount of data with minimal impact on analysis time.

Challenge C3: Bivariate root-cause analysis

The third challenge we face relates to performing the root-cause analysis. ROSITA performs the analysis using a t-test on each of the ELMO* model components. Such an approach can detect univariate leakage. However, detecting bivariate leakage necessitates evaluating pairs of components. A brute-force approach that evaluates a t-test statistics on every pair of components is computationally expensive, particularly considering the increased number of traces, as described in Challenge C2. Thus, new techniques for root-cause analysis are required.

We now discuss how ROSITA++ addresses these challenges.

Achieving Statistical Confidence

As discussed, Challenge C1 is that with the quadratic increase in the number of sample point per trace, the t-test threshold of 4.5 is no longer appropriate. This mostly affects the traces collected from the physical experiment where we collect longer traces (10 times more samples) to reduce the effects of noise. To demonstrate the false positives we collect 500,000 traces of a three-share implementation of Xoodoo (further described in Section 7.5.2) running on a STM32F030 Discovery evaluation board where all of the inputs are drawn uniformly at random. We then split these arbitrarily into two populations, and perform a bivariate t-test analysis, comparing these populations with a threshold of 4.5. As Figure 7.15 shows, despite the populations being sampled from the same distribution, several false positives were present.

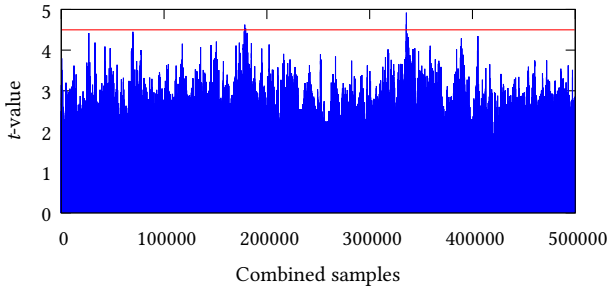


Figure 7.15: A t-test threshold value of 4.5 for a bivariate analysis with 1000 samples with all inputs being random.

For engineers, these false positives are typically of low impact. Experienced engineers can typically identify false positives, e.g. by observing the context. Alternatively, repeating the test would confirm true positives.

Automatic tools, such as Rosita++, do not have the experience or the insight, and must rely on statistical tools for handling false positives. If Rosita++ is used with long code segments these false positives will also be present in its leakage analysis. Therefore, in Rosita++, we adopt the approach of Ding et al.[101], which propose increasing the threshold to ensure that the probability of false positives. Specifically, Ding et al. provide a formula to calculate the threshold given the number of samples and a desired confidence α. We apply the formula to the length of the bivariate trace aiming for a confidence of 0.00001. This ensures that the probability of a false positive error is less than .001%, which we consider negligible. For the traces in Figure 7.15 we would use a threshold of 6.71, which is clearly above the largest peak in the figure. Hence, at this threshold, the analysis does not indicate any leakage, which is expected considering that the two populations are drawn from the same distribution.

Handling Large Datasets

As mentioned, several aspects of bivariate analysis result in a significant increase in the size of data that Rosita++ needs to process. First, for given mean and variance, the Welch t-value grows linearly with the square root of the size of the population. Consequently, when increasing the threshold we need a quadratic increase in the number of traces to achieve the same detection sensitivity. Secondly, the length of the bivariate artificial traces is several orders of magnitude longer than the original univariate traces. Thirdly, due to the effects of noise, detecting higher-order leakage is inherently harder then detecting first-order leakage. The combined effect of these changes is that the amount of data that Rosita++ needs to process is several orders of magnitude larger than that of Rosita. As we need to evaluate the final version of code that is produced by Rosita++ on real hardware the same issue gets even more apparent as we use longer traces as mentioned in Section 7.5.1.

While we are aware of works that have performed analysis at scales similar and even larger than our work [88, 89], we could not find public tools that perform such

analysis, or even performance figures for the analysis. Free tools such as Jlsca[6], Scared[7], SCALib[8] seem to only offer limited capabilities. To address this challenge we developed analysis tools from the ground up. Our analysis tools avoid the overhead of storing the artificial traces (i.e. bivariate combinations) by calculating them on the fly. The tools are multithreaded, allowing a significant speedup, and the data is divided point-wise between the threads, so that each thread only accesses a limited subset of the original traces' samples.

We acknowledge that the approach is fairly straightforward, but we believe that the contribution is important for practical future research into bivariate analysis, and will make our tools available both as independent tools and as part of ROSITA++.

Bivariate Root-Cause Analysis

The third challenge for ROSITA++ is performing root-cause analysis on bivariate traces. The ELMO* statistical model consists of 26 components, each modelling a different micro-architectural effect. When ROSITA performs univariate root-cause analysis, it calculates the Welch t-value for each component separately, where the leaky components are identified by observing high t-values.

While this approach works well for univariate leakage detection, adapting it to bivariate leakage is not trivial. The main reason is that, in bivariate analysis, there is no single cause for leakage.

Recall that a bivariate sample point is a combination of two samples in the original trace. Each of these samples is calculated from the 26 model components. Thus, a bivariate sample is calculated from 52 values, 26 for one emulated sample and 26 for the other. Because we assume that the program is first-order secure, every leak cause requires at least one component from one of the emulated samples and one component from the other. Because there are $26 \times 26 = 676$ pairs of components, searching for the leaky pair is inefficient.

To avoid searching the whole space of pairs of model components, ROSITA++ uses two new methods for finding the components that contribute to the leak. The *component elimination* method tests whether removing a model component removes the leakage. While efficient, this approach may sometimes fail. In the case of such a failure, ROSITA++ reverts to the *Monte-Carlo method*, which tests random combinations of components looking for evidence of component leakage. We now describe these two methods in detail.

Component Elimination. As discussed, the component elimination method aims to identify each leaking model component by checking whether the model leaks after removing the component. Recall that a leak is caused by a pair of components. If we remove one of the components in the leaking pair, the other component does not leak alone.

The component elimination method exploits this behavior. It iterates over each of the 52 model components (26 from each instruction) that determine the bivariate sample, and calculates a hypothetical sample from the values with a reduced model

[6] `https://github.com/Riscure/Jlsca`
[7] `https://gitlab.com/eshard/scared`
[8] `https://github.com/simple-crypto/SCALib`

consisting of the remaining 51 model components. This is done for all emulated traces from the fixed and the random samples. It then tests whether the means of the sampled distributions of the fixed and the random traces are equivalent.

A core observation for this approach is that we cannot use the Welch t-test for determining that the means of the distributions are equivalent. The test is designed to show when means of distributions are different, but a failure to show that they are different does not demonstrate that they are the same. Trying to (ab)use the t-test in revers for demonstrating equivalence highlights the problem. We find that the test can produce unstable results, and may result in a low t-value for combinations that are leaky.

For testing whether means of distributions are equivalent, we use the TOST test [264]. In a nutshell TOST calculates a level of confidence that the difference between means of the compared distributions is outside a predefined range.

To determine the range, we use the approach demonstrated by Pardo [225]. Specifically, using the reduced model we generate two additional sets of traces, both drawn from the 'random' configuration. We calculate the mean difference and variance of these trace sets. To determine the upper and lower boundaries we use Equation 7.2 and Equation 7.3 with the mean difference and variance calculated previously with a confidence level of 0.00001.

While the component elimination method is efficient, it may sometimes fail. For example, if multiple model components leak the same share, removing any one of these components will not eliminate the leak. Similarly, the TOST test may fail to demonstrate the equivalence of the two distribution even when removing a model component eliminates the leak. In those cases, Rosita++ resorts to the alternative Monte-Carlo method. It was used to find root causes in 57 out of 137 queries for Xoodoo and 328 out of 502 queries for Present.

The Monte-Carlo Method. In the Monte-Carlo approach we run a preset number (50 in our case) of random experiments. In each experiment we select a random subset of the model components, and run the t-test on hypothetical traces with only the selected components. For each component we keep track of the number of random experiments it participates in and how many of those experiments indicate leakage. After running the random experiments, Rosita++ picks the components whose inclusion best correlates with evidence of leakage. The preset number of random experiments were selected from a performance analysis done for a code segment from Xoodoo cipher (shown in Listing 7.8) by only using Monte Carlo method to detect and remove leakage. Initially, this implementation had 45 total leakage points. Figure 7.16 shows the reduction of remaining leaky points as we gradually increase the number of Monte Carlo experiments used to find offending components from 10. The number of experiments improves detection of root causes, but after 50 or so experiments the benefits were not significant. We ran the same up to 1000 experiments and did not see significant improvement. Therefore, we settled at using 50 as the preset experiment count for the Monte Carlo experiments.

Code Rewrite

After finding the root cause of the leakage, Rosita++ selects the code-rewrite rule that best match the detected root cause. We use the code-rewrite engine of Rosita,

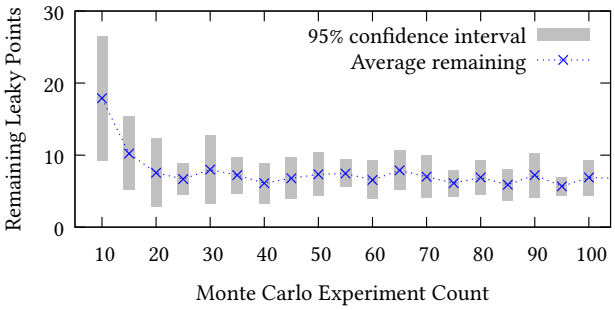


Figure 7.16: Effectiveness in removing leakage of Monte Carlo method for increasing number of experiments

with minor differences to adapt it to root causes that consist of multiple components.

In a nutshell, ROSITA reserves the register `r7`, which it initializes with a random value. When an unintended interaction is detected, the code rewrite engine inserts instructions that use `r7` to eliminate the interaction. For example, when the detected interaction is caused by a pipeline register that is updated by two consecutive instructions, ROSITA inserts the instruction `mov r7, r7`, to buffer between the interacting instructions. Similarly, when the leakage is from an interaction with the memory subsystem, ROSITA inserts the pair of instructions `push {r7}` followed by `pop {r7}`, which wipes the internal state of the memory pipeline.

7.5.2 Evaluation

In this section we evaluate the effectiveness of ROSITA++ in eliminating leakage.

A Toy Example

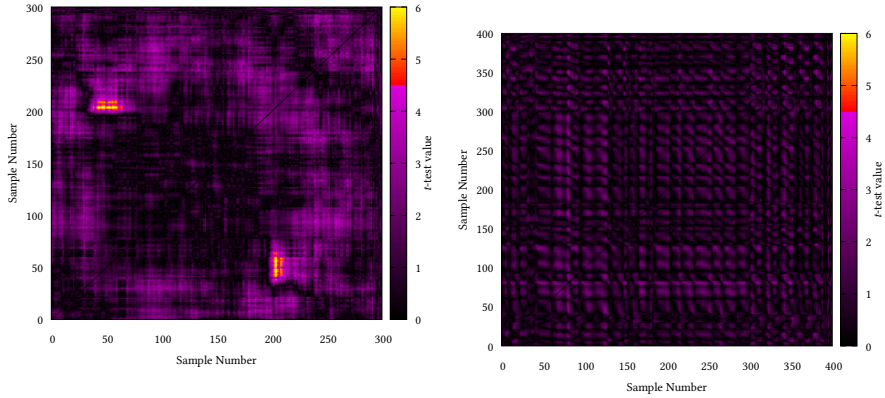
Listing 7.6: A Toy Example

```
1    ; nop padding
2  ldrb r4, [r1]
3  push {r7}
4  pop {r7}
5    ; nop padding
6  ldrb r5, [r2]
7  ldrb r6, [r3]
8    ; nop padding
```

Before we evaluate ROSITA++ on real-world software examples, we demonstrate its effectiveness on a toy example, shown in Listing 7.6. The code presents a typical operation in second-order protected implementation—loading a masked value from memory. Specifically, it assumes that registers `r1`, `r2`, and `r3` contain the addresses of three shares that represent a secret value. The code uses three `ldrb` instructions

to load the masked value into three registers, `r4`, `r5`, and `r6`. We note that the code is nominally second-order secure, because all instructions process at most one share of the secret. However, as we see below, unintended interactions between the load instructions at Lines 6 and 7 result in second-order leakage.

To avoid second-order leakage through a combination of the three load instructions, we separated the first load (Line 2) from the rest of the code. We added the `push` and `pop` instructions in Line 3 and Line 4 to remove interactions between the first load and the following two loads. We further added sequences of nine `nop` instructions (concretely, `mov r7, r7`) to avoid unintended interaction through the processor's pipeline and to achieve a clear temporal separation between the loads. Last, we add short sequences of `nop` instructions around the code to create a temporal separation of the measured code from the triggers.



(a) Before applying code fixes. Leakage is visible around coordinates (50,200) and (200,50).

(b) After applying code fixes. No evidence of leakage is observed.

Figure 7.17: Evaluating a toy example.

In Figure 7.17(a) we see the results of the bivariate leakage analysis performed on the code. The figure is a heat map, where the X and Y axes indicate the samples that are combines to create the artificial bivariate sample. The color of each combined sample indicates the magnitude of the fixed vs. random t-test analysis for the combined sample. We note that because our combination function is symmetric, the figure is symmetric across its main diagonal.

Examining the figure we find that there are two regions that show a t-test value above our threshold of 4.5. These occur at the combinations of samples around 50, which corresponds to Line 2 of Listing 7.6, and sample 200, which corresponds to Line 7.

Running Rosita++ also shows that the combination of Line 2 and Line 7 leaks the secret. The root cause analysis shows that Line 6 and Line 7 interact both through the processor pipeline and through the memory bus.

To fix the leakage, ROSITA++ first inserts the `mov r7, r7` instruction between Line 6 and Line 7, and repeats the analysis to check that the leakage has been eliminated. Finding that there is still leakage through the memory bus, ROSITA++ further adds a combination of `push` and `pop` instructions, producing the code in Listing 7.7. Running the bivariate analysis on the code shows no evidence of second-order leakage. (Figure 7.17(b).)

Listing 7.7: Fixed Toy Example

```
1     ; nop padding
2  mov r7, r7
3  ldrb r4, [r1]
4  push {r7}
5  pop {r7}
6     ; nop padding
7  ldrb r5, [r2]
8  mov r7, r7
9  push {r7}
10 pop{r7}
11 ldrb r6, [r3]
12    ; nop padding
```

Comparing Emulated and Real Traces: To better understand the relationship between emulated and real traces, we compared the leakage observed in the traces in terms of signal-to-noise ratio (SNR). For this experiment we used 20,000 random input traces coming from the emulation and the real experiments; we chose this number because it is sufficient to find leakage in the emulated traces using TVLA. We computed SNR for the leaking values that need to be combined for bivariate analysis: hamming weight (HW) of 4 bytes of $r1$ and HW of 4 bytes of $r2 \oplus r3$. For the real experiments these values were between 0.041 and 0.063 for the bytes of $r1$ and between 0.012 and 0.014 the bytes of $r2 \oplus r3$. We could not compute the SNR directly for the emulated traces since the emulation is deterministic and therefore, noise-free. We added a sufficient amount of noise to the simulated traces to generate a SNR similar to the real experiments. We used Gaussian Noise with means 0 and standard deviation of 0.25% of the signal amplitude for the bytes of $r1$ and 0.1% for the bytes of $r2 \oplus r3$. We are not from where this leakage difference is exactly coming from, but we suspect that we simply found a slight difference between the emulated and the real measurements.

We conclude that if we introduce between 0.1% and 0.25% ratio of noise to the emulated traces then we obtain a similar SNR to the real traces. Moreover, we can use 25 times less traces than in the real experiment to detect leakage using TVLA, since we can detect leakage using emulation with 20,000 traces and we need 500,000 in the real experiments.

Evaluated Cryptographic Implementations

We now turn our attention to more realistic examples. Before performing the evaluation we use ROSITA to detect and eliminate any first-order leakage from the code.

Listing 7.8: Xoodoo code segment under test

$$a_{0,0} = a_{0,0} \oplus (\neg a_{1,0} \wedge a_{2,0}) \oplus (a_{1,0} \wedge b_{2,0}) \oplus (b_{1,0} \wedge a_{2,0})$$
$$b_{0,0} = b_{0,0} \oplus (\neg b_{1,0} \wedge b_{2,0}) \oplus (b_{1,0} \wedge c_{2,0}) \oplus (c_{1,0} \wedge b_{2,0})$$
$$c_{0,0} = b_{0,0} \oplus (\neg c_{1,0} \wedge c_{2,0}) \oplus (c_{1,0} \wedge a_{2,0}) \oplus (a_{1,0} \wedge c_{2,0})$$

We further perform a first-order fixed vs. random TVLA with 2,000,000 traces on the real hardware to verify that no first-order leakage is detected. For the evaluation, we use Rosita++ to detect and correct second-order leakage for 500,000 simulated traces. We then collect 2,000,000 power traces from each of the original and the fixed software, and perform bivariate second-order analysis to identify any leakage. We evaluate two cryptographic implementations, which we describe below.

Xoodoo was proposed by Daemen et al. [94], who provide a reference implementation at Bertoni et al. [55]. We adapted their code to a protected implementation. The implementation is a threshold implementation (TI) [215] using three shares.

TI schemes were proposed by Nikova et al. to prevent the leakage from "glitches" that can occur in hardware implementations [215]. The concept is accomplishing the goal of masking through a number of shares with some additional properties. Specifically, the non-completeness property of threshold implementations enforces that no operation should involve more than two shares.

Xoodoo's state is 48 bytes in length. The state is divided into three equal blocks called *planes*, each consisting of four 32-bit words. $x_{i,j}$ denotes the j^{th} 32-bit word of the i^{th} plane of share x, where $x \in \{a, b, c\}$. Listing 7.8 shows the code segment that we analyze, which forms part of the start of the Xoodoo χ function. Our initial C implementation demonstrated first-order leakage caused by the optimizer merging shares. We therefore manually implemented the code in assembly, ensuring that shares are not merged..

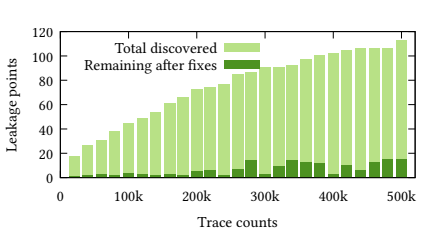
PRESENT is a block cipher that is a substitution permutation network and it was proposed by Bogdanov et al. in [68]. It has a block size of 64-bit and the key can be 80 or 128 bits long. The non-linear layer is based on a single 4-bit S-box facilitating lightweight hardware implementations.

We implemented a masked implementation in software as described in Algorithm 3.2 by Sasdrich et al. [256] for the analysis. The code also specifies a threshold implementation with three shares. This means that, at least in theory, the implementation should not leak in the first order.

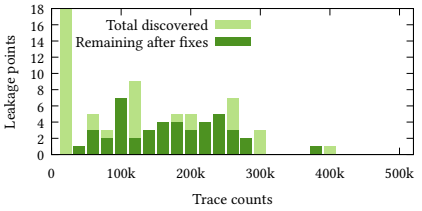
The code segment shown in Listing 7.9 was used for our evaluation. It implements a part of the S-box of PRESENT, involving 3 shares x^1, x^2, x^3 and the lookup table T. The table is an 8-bit to 4-bit lookup table where the inputs are two 4-bit nibbles. Each table lookup used to compute t^i is repeated 16 times to cover the complete 64-bit shares.

Listing 7.9: Present code segment under test

$$t^3 = \mathrm{T}(x^1, x^2)$$
$$t^2 = \mathrm{T}(x^3, x^1)$$
$$t^1 = \mathrm{T}(x^2, x^3)$$

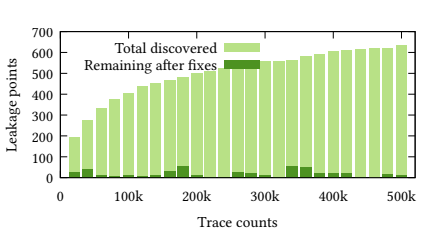


(a) Unprotected cipher as input

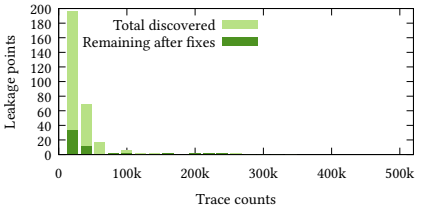


(b) Protected cipher from previous step as input

Figure 7.18: Xoodoo: leakage points discovered before vs. after fixing



(a) Unprotected cipher as input



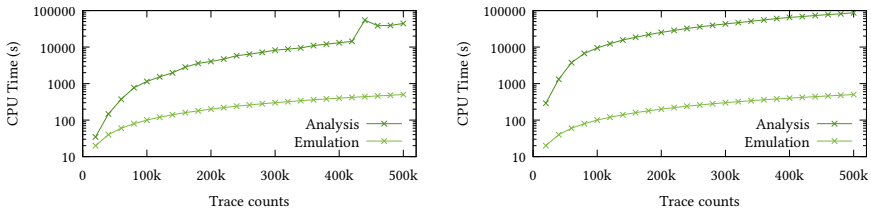
(b) Protected cipher from previous step as input

Figure 7.19: PRESENT: discovered leakage points before vs. after fixing

Fixing Second-Order Leakage

We used ROSITA++ to fix the leakages that were discovered in the code segments introduced above in Section 7.5.2. Specifically, we focus on leakage discovered by a bivariate fixed vs. random t-test only.

To analyze the relationship between the number of traces and leakage discovery, we run ROSITA++ on the unprotected ciphers, varying the number of traces from 20,000 to 500,000 at steps of 20,000. We test these in two scenario. In the first scenario, we use the same unprotected version of the cipher for all steps. The top parts of Figure 7.18, and Figure 7.19 show the number of leakages detected at each step and the number of leakage points remaining after ROSITA++ applies fixes. In



(a) Xoodoo time taken to emulate vs. analysis

(b) Present time taken to emulate vs. analysis

Figure 7.20: Emulation vs. analysis times

Cipher	Unprotected size (cycles)	Protected size (cycles)	Increase
Xoodoo 3 share	56	114	104%
Present 3 share	114	365	220%

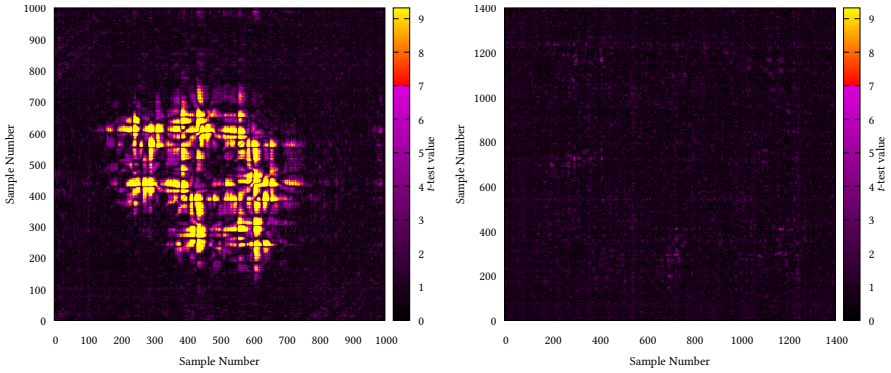
Table 7.4: Overhead added

the second scenario, shown in the bottom parts of the figures, for each step we use a version of the cipher as fixed by Rosita++ in the previous step. Table 7.4 shows the overhead added by the fixes applied at the end of all the runs of the later procedure.

From the results of both, Figure 7.18 and Figure 7.19 it is clear that running emulation at smaller number of traces first and then gradually increasing the number of traces by using previously fixed cipher implementation as input to the next run is a better alternative to running the unprotected cipher once with the target trace count. Figure 7.20 shows the emulation times and fix times when Rosita++ starts from the unprotected version to analyse and apply fixes. There is also a remarkable difference between the time duration that these two approaches take. For Xoodoo, an unprotected implementation took 11 hours and 48 minutes in wall clock time to emulate and apply fixes, but when the same unprotected implementation is run for several times using previous fixed versions as input, the total time consumed was 1 hour and 50 minutes. This pattern applies for the implementation of PRESENT cipher too, where the time duration was 36 hours and 1 hour and 49 minutes. This is caused by the fast discovery of more prominent leakage in the second scenario. In contrast, in the first scenario all leakage discovery takes the same amount of time and the analysis is required to process all traces that were collected, which is huge compared to the trace amounts compared to earlier Rosita++ runs.

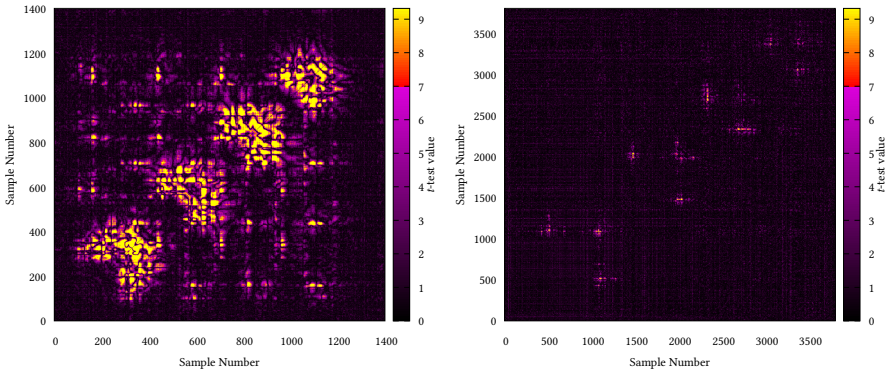
Experiment

As Figure 7.18(b) shows, increasing the emulated trace count from 420,000 to 500,000 did not result in the detection of new leakage points. Therefore, the resulting implementation was picked for running on the real device. On the real device we collected



(a) Before applying code fixes (b) After applying code fixes

Figure 7.21: Xoodoo



(a) Before applying code fixes (b) After applying code fixes

Figure 7.22: PRESENT

2 million traces. Similarly, from Figure 7.19(b) it is clear that at 400,000 there is no further leakage observed until the number of traces that we ran the emulation for. Therefore, we selected the final output from ROSITA++ to be run on the real device.

Figures 7.21(a) and 7.22(a) show the t-test values observed for the unprotected versions of the ciphers and figures 7.21(a) and 7.22(b) show the t-test values for the protected versions by using ROSITA++. Xoodoo and Present was protected using 500,000 emulated traces. The duration that it took to collect traces were the same both protected and unprotected versions of the ciphers, for Xoodoo it was 9 hours 26 minutes and for Present it was 7 hours 48 minutes. This is expected as we collect the same number of samples in both cases but communicate more for Xoodoo due to it requiring more mask bytes. It is clear from the results shown in Figure 7.21(b) that ROSITA++ has been successful in eliminating discovered leakage. However, Figure 7.22(b) show that not all detected leakage has been eliminated. We investigated this and found out that the cause was the interaction of shares through the address bus. Listing 7.10 shows the first leaky segment of the code that is found between samples 1,000 and 1,250. There are two independent leakages happening in this code segment. Both are similar and occur in the `ldr` instruction. The offending instructions are at Listing 5 and Listing 8. The registers used for addressing in these instructions carry one share each. This results in leaking the exclusive-or between the two shares, and this leakage becomes observable as second-order leakage due to the fact that it needs only one share to complete all shares (so the input is leaked). We confirmed this by reproducing the same effect in a separate fixed vs. random experiment which has two shares. It showed significant leakage at 200,000 traces.

Listing 7.10: Leaky code segment of fixed Present

```
1  adds r0, r5, r0
2  push {r7}
3  pop {r7}
4  adds r3, r5, r3
5  ldrb r0, [r0, r2]
6  push {r7}
7  pop {r7}
8  ldrb r3, [r3, r1]
9  push {r7}
10 pop {r7}
```

Tools for Leakage Analysis

We now present the performance of our second-order analysis tools. We run the tools on a desktop computer, featuring an Intel Core i9-10900K CPU and 32 GB of memory. We spawn 15 threads and perform bivariate analysis of four cryptographic implementations. For each implementation we use our measurement setup to collect 2M traces from the real experiments, which we analyze to draw the heat maps shown in Figure 7.21 and Figure 7.22. The results are shown in Table 7.5. The amount of threads used can be changed to fit the underlying hardware, the thread count is dependent on the equal sized splits that are done along the sample axis. It is given by $S(S+1) \div 2$ where S is the number of equal sized splits. For our runs, S was set

Trace set	Samples	Wall Clock Time
Xoodoo unprotected	1000	0:38:52
Xoodoo protected	1400	1:11:35
Present unprotected	1400	1:01:36
Present protected	3800	11:27:09

Table 7.5: Bivariate analysis time

at 5. Without parallelisation the workload will be 12.5 times if run in a single thread as 5 out of 15 of the threads do half of the work.

7.6 Limitations

Possibly the main limitation of Rosita and Rosita++ is that, while we have found no evidence for leakage, there is no guarantee that it fixes all leakage. There two main reasons for that:

- **Methodology:** While popular and standardized, non-specific fixed vs. random tests are not a panacea for leakage. Leakage they detect is not necessarily exploitable and the absence of detection does not necessarily mean that no leakage exists. We note, however, that this does not detract from Rosita and Rosita++ achieving its aim of assisting in assessing compliance with the standard.

- **Model Limitations:** Rosita and Rosita++ rely on Elmo* which is only a model of the hardware. Gaps between the model and the real hardware, both in terms of capturing the hardware behavior and in terms of accuracy of the model can result in missed leakage. Moreover, hardware behavior may change over time, e.g. due to a microcode update [243]. For these reasons we recommend that operators do not rely solely on Rosita or Rosita++. It can be used to achieve a high degree of assurance, and is likely to automate a significant part of the work required for compliance, but testing with the real hardware is essential.

A further limitation of Rosita and Rosita++, which, like others, stem from the Elmo* model is its suitability for other processors. Elmo* uses a very simple model of the processor. It is suitable for small, in-order, cacheless microcontrollers, such as the Cortex-M0, AVR processors such as the ATMEGA328p, or small RISC-V processors. However, the model is unlikely to be suitable for more advanced processors. Nonetheless, we believe that Rosita and Rosita++ are important, because the microcontrollers it targets are extremely popular in embedded devices, where they often implement cryptographic functionalities. At the same time, there is very little control of the physical environment of such devices, allowing the attacker unfettered access and enabling the type of attacks we defend against.

7.7 Conclusions

Since the introduction of side-channel attacks, implementation security of embedded devices has been under the immense scrutiny and constant threat of being exploited. Even with theoretically sound measures such as masking, the devices tend to exhibit some leakages in practice due to unintended interactions in hardware. Mostly manual evaluation involving a tedious decision process and applying fixes to such "leaky" implementations have since been adopted. Some automatic countermeasures have also been developed, but to the best of our knowledge, all of them target univariate leakage in first-order masked implementations.

In this chapter, we set out to automate the detection and application of fixes for first-order and second-order secured implementations through univariate and bivariate analysis. We have demonstrated that it is possible to fix almost all detected leakage for several masked cipher implementations using our methodology for root cause analysis. It is a significant improvement over previous automatic countermeasure application methods due to the simplicity of its nature.

Conclusion and Discussion

In this chapter, we conclude this thesis by summarizing the results of the research. In addition, we also list future research directions.

Summary of Results

In this section, we list a summary of the results presented in this thesis.

- In Chapter 3, we presented physical attacks on several symmetric key cryptographic schemes. The side-channel attack Keyak was done on real traces where the attack was based on previous work that used simulated traces. The side-channel attack on Ascon also done on real traces but the attack was created and presented in this chapter. Finally, the chapter presented a new cipher called FRIET that has build-in countermeasures against fault injection attacks. We presented an evaluation of the countermeasure using electro-magnetic fault injection.

- The digital signature scheme Ed25519 is used in many different libraries and applications. As a result we looked into different side-channel vulnerabilities and how to counter them. We presented new physical attacks on the deterministic digital signature scheme Ed25519 in Chapter 4. The side-channel attack exploits deterministic property of the scheme and by collecting with the same private and random data, we attack the derivation of the ephemeral scalar. To accomplish this we present an attack on the message schedule of SHA-512 that allows us to recover the private key. In addition to the side-channel attack, in this chapter we also present a fault injection attack. We show using voltage fault injection and electro-magnetic fault injection how to recover the private key using a single faulty signature. Finally, we present a countermeasure that applies to both physical attacks presented in the chapter.

- Like Ed25519, Curve25519 is also used in many cryptographic libraries and applications. In Chapter 5, we presented and evaluated countermeasures for cryptographic implementations of Curve25519. Key exchange with elliptic curves uses static and ephemeral keys, since they have different security requirements, we present two speed optimized implementations. One implementation is optimized and offers side-channel protection for ephemeral keys and one implementation is optimized and offers side-channel protection for static keys. The implementation for static keys has stronger countermeasures and as a result

has a longer run-time compared to the implementation for ephemeral keys. The evaluation shows the difference in performance of each implementation and compares it to real-world implementations. The side-channel evaluation shows the effect of the stronger countermeasures.

- We combined fault injection and genetic algorithms to optimize the massive search space that is typical with fault injection attacks in Chapter 6. The genetic algorithm allows to reduce the number of injected faults to obtain a good parameter set. In the chapter, we present the genetic algorithm that we used to the search space. We apply the strategy to attack SHA-3 and show the reduction of the number of injected faults that are typically needed for a successful attack.

- In Chapter 7, we presented automated tools that are able to emulate power consumption given a masked implementation. Using the that resulting power consumption, the tools can automatically detect (higher-order) side-channel leakage and fix it in the compiled assembly. We evaluated the effectiveness of the tools by collecting real traces with the same implementations. We show that the tools are able to detect and remove nearly all side-channel leaks.

Future Work

There are several ways of extending the work presented in this thesis.

The attacks on Keyak and Ascon in Chapter 3 can be further optimized to increase the success rate. We also discuss and evaluate the effectiveness of the fault resistance cipher FRIET in that chapter. That part, we could extend by adding side-channel countermeasures and evaluating them while at the same time keeping the effectiveness of the fault resistance in mind.

The work in Chapter 4 could be extended by formalizing the countermeasure that was presented to thwart both the side-channel attack and the fault injection attack.

Chapter 5 could be extended with a detailed investigation of single-trace profiled attacks, in particular, using deep-learning. Another future direction is to consider formal proofs of the side-channel resistance and higher order protection, including the corresponding evaluation.

Since not much research was conducted regarding finding parameters leading to faults, that opens up potential research directions for Chapter 6. We could extend the work with exploring laser fault injection. Additionally, we could further look into the fitness function and how to determine what makes a good neighborhood to consider. More precisely, what is the best resolution for our experiments such that we do not waste fault injection attempts on neighborhoods where no extra information can be learned.

The work in Chapter 7 could be extended by porting to the tools to support other microcontrollers. In addition, the work could also be continued by using other statistical tests that are used to detect side-channel leakage.

Discussion

This thesis has described several side-channel attacks, fault injection attacks and different countermeasures to protect against these attacks. It is clear that a lot of work can still be done to improve the security against side-channel attacks as real world implementations and devices are still being broken as was seen in the Minerva attack on ECDSA nonces [157], and the attack on the Titan U2F key and several other devices that use the same secure element [250].

In this thesis, we introduced side-channel attacks on deterministic signature schemes. In the new post quantum cryptography signature algorithms the same principles for an attack apply as was shown in [72] because they also have deterministic properties.

The use of simulators to determine side-channel leakage and use to it for early evaluation of cryptographic implementations make it a much cheaper option for the semiconductor industry. Not only is it cheaper compared to immediately hiring an evaluation company it is also much faster and allows for more iterative improvements until the final solution is sent to an evaluator for the final side-channel evaluation that is required to obtain a certificate.

Bibliography

[1] Rodrigo Abarzúa, Claudio Valencia Cordero, and Julio López. "Survey for Performance & Security Problems of Passive Side-channel Attacks Countermeasures in ECC". In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 10.

[2] Farzaneh Abed, Christian Forler, and Stefan Lucks. "General classification of the authenticated encryption schemes for the CAESAR competition". In: *Comput. Sci. Rev.* 22 (2016), pp. 13–26. DOI: 10.1016/j.cosrev.2016.07.002.

[3] Onur Aciiçmez, Werner Schindler, and Çetin Kaya Koç. "Improving Brumley and Boneh timing attack on unprotected SSL implementations". In: *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*. Ed. by Vijay Atluri, Catherine A. Meadows, and Ari Juels. ACM, 2005, pp. 139–146. DOI: 10.1145/1102120.1102140.

[4] Aris Adamantiadis, Simon Josefsson, and Mark D. Baushke. "Secure Shell (SSH) Key Exchange Method Using Curve25519 and Curve448". In: *RFC* 8731 (2020), pp. 1–6. DOI: 10.17487/RFC8731.

[5] Anita Aghaie, Amir Moradi, Shahram Rasoolzadeh, Aein Rezaei Shahmirzadi, Falk Schellenberg, and Tobias Schneider. "Impeccable Circuits". In: *IEEE Trans. Computers* 69.3 (2020), pp. 361–376. DOI: 10.1109/TC.2019.2948617.

[6] Giovanni Agosta, Alessandro Barenghi, and Gerardo Pelosi. "A code morphing methodology to automate power analysis countermeasures". In: *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*. Ed. by Patrick Groeneveld, Donatella Sciuto, and Soha Hassoun. ACM, 2012, pp. 77–82. DOI: 10.1145/2228360.2228376.

[7] Michel Agoyan, Jean-Max Dutertre, David Naccache, Bruno Robisson, and Assia Tria. "When Clocks Fail: On Critical Paths and Clock Faults." In: *Lecture Notes in Computer Science*. Vol. 10. Springer Berlin Heidelberg, 2010, pp. 182–193. DOI: 10.1007/978-3-642-12510-2_13.

[8] Manfred Josef Aigner, Stefan Mangard, Francesco Menichelli, Renato Menicocci, Mauro Olivieri, Thomas Popp, Giuseppe Scotti, and Alessandro Trifiletti. "Side channel analysis resistant design flow". In: *International Symposium on Circuits and Systems (ISCAS 2006), 21-24 May 2006, Island of Kos, Greece*. IEEE, 2006. DOI: 10.1109/ISCAS.2006.1693233.

[9] Toru Akishita and Tsuyoshi Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *Information Security, 6th International Conference, ISC 2003, Bristol, UK, October 1-3, 2003, Proceedings*. Ed. by Colin Boyd and Wenbo Mao. Vol. 2851. Lecture Notes in Computer Science. Springer, 2003, pp. 218–233. DOI: 10.1007/10958513_17.

[10] Monjur Alam, Baki Berkay Yilmaz, Frank Werner, Niels Samwel, Alenka G. Zajic, Daniel Genkin, Yuval Yarom, and Milos Prvulovic. "Nonce@Once: A Single-Trace EM Side Channel Attack on Several Constant-Time Elliptic Curve Implementations in Mobile Platforms". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 2021, pp. 507–522. DOI: 10.1109/EuroSP51992.2021.00041.

[11] José Bacelar Almeida, Manuel Barbosa, Gilles Barthe, François Dupressoir, and Michael Emmi. "Verifying Constant-Time Implementations". In: *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. Ed. by Thorsten Holz and Stefan Savage. USENIX Association, 2016, pp. 53–70.

[12] Christopher Ambrose, Joppe W. Bos, Björn Fay, Marc Joye, Manfred Lochter, and Bruce Murray. "Differential Attacks on Deterministic Signatures". In: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 339–353. DOI: 10.1007/978-3-319-76953-0_18.

[13] Adrian Antipa, Daniel R. L. Brown, Alfred Menezes, René Struik, and Scott A. Vanstone. "Validation of Elliptic Curve Public Keys". In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 211–223. DOI: 10.1007/3-540-36288-6_16.

[14] Diego F. Aranha, Claudio Orlandi, Akira Takahashi, and Greg Zaverucha. "Security of Hedged Fiat-Shamir Signatures Under Fault Attacks". In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 644–674. DOI: 10.1007/978-3-030-45721-1_23.

[15] *Mbed TLS*. https://github.com/ARMmbed/mbedtls (accessed 2021-05-05). 2021.

[16] Jean-Philippe Aumasson. *Should Curve25519 keys be validated?* https://research.kudelskisecurity.com/2017/04/25/should-ecdh-keys-be-validated/ (accessed 2020-12-02). 2017.

[17] Christian Aumüller, Peter Bier, Wieland Fischer, Peter Hofreiter, and Jean-Pierre Seifert. "Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasures". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August*

13-15, 2002, Revised Papers. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 260–275. DOI: 10.1007/3-540-36400-5_20.

[18] T Bäck, D.B Fogel, and Z Michalewicz, eds. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing, Bristol, 2000.

[19] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. "Acoustic Side-Channel Attacks on Printers". In: *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*. USENIX Association, 2010, pp. 307–322.

[20] Nasour Bagheri, Navid Ghaedi, and Somitra Kumar Sanadhya. "Differential Fault Analysis of SHA-3". In: *Progress in Cryptology - INDOCRYPT 2015 - 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*. Ed. by Alex Biryukov and Vipul Goyal. Vol. 9462. Lecture Notes in Computer Science. Springer, 2015, pp. 253–269. DOI: 10.1007/978-3-319-26617-6_14.

[21] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. "On the Cost of Lazy Engineering for Masked Software Implementations". In: *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*. Ed. by Marc Joye and Amir Moradi. Vol. 8968. Lecture Notes in Computer Science. Springer, 2014, pp. 64–81. DOI: 10.1007/978-3-319-16763-3_5.

[22] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. "Power Analysis of Atmel CryptoMemory - Recovering Keys from Secure EEPROMs". In: *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer, 2012, pp. 19–34. DOI: 10.1007/978-3-642-27954-6_2.

[23] Alessandro Barenghi, Luca Breveglieri, Israel Koren, Gerardo Pelosi, and Francesco Regazzoni. "Countermeasures against fault attacks on software implemented AES: effectiveness and cost". In: *Proceedings of the 5th Workshop on Embedded Systems Security, WESS 2010, Scottsdale, AZ, USA, October 24, 2010*. ACM, 2010, p. 7. DOI: 10.1145/1873548.1873555.

[24] Alessandro Barenghi and Gerardo Pelosi. "A Note on Fault Attacks Against Deterministic Signature Schemes". In: *Advances in Information and Computer Security - 11th International Workshop on Security, IWSEC 2016, Tokyo, Japan, September 12-14, 2016, Proceedings*. Ed. by Kazuto Ogawa and Katsunari Yoshioka. Vol. 9836. Lecture Notes in Computer Science. Springer, 2016, pp. 182–192. DOI: 10.1007/978-3-319-44524-3_11.

[25] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. "maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults". In: *Computer Security - ESORICS 2019 - 24th European Symposium on Research in Computer Security, Luxembourg, September 23-27, 2019, Proceedings, Part I*. Ed.

by Kazue Sako, Steve A. Schneider, and Peter Y. A. Ryan. Vol. 11735. Lecture Notes in Computer Science. Springer, 2019, pp. 300–318. DOI: 10.1007/978-3-030-29959-0_15.

[26] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. "Verified Proofs of Higher-Order Masking". In: *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Springer, 2015, pp. 457–485. DOI: 10.1007/978-3-662-46800-5_18.

[27] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. "CSI NN: Reverse Engineering of Neural Network Architectures Through Electromagnetic Side Channel". In: *28th USENIX Security Symposium, USENIX Security 2019, Santa Clara, CA, USA, August 14-16, 2019*. Ed. by Nadia Heninger and Patrick Traynor. USENIX Association, 2019, pp. 515–532.

[28] Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. *SCA-secure ECC in software – mission impossible?* Cryptology ePrint Archive, Report 2021/1003. https://ia.cr/2021/1003. 2021.

[29] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. "Online template attacks". In: *J. Cryptogr. Eng.* 9.1 (2019), pp. 21–36. DOI: 10.1007/s13389-017-0171-8.

[30] Aurélie Bauer, Éliane Jaulmes, Emmanuel Prouff, and Justine Wild. "Horizontal Collision Correlation Attack on Elliptic Curves". In: *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*. Ed. by Tanja Lange, Kristin E. Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Springer, 2013, pp. 553–570. DOI: 10.1007/978-3-662-43414-7_28.

[31] Ali Galip Bayrak, Francesco Regazzoni, David Novo, Philip Brisk, François-Xavier Standaert, and Paolo Ienne. "Automatic Application of Power Analysis Countermeasures". In: *IEEE Trans. Computers* 64.2 (2015), pp. 329–341. DOI: 10.1109/TC.2013.219.

[32] Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. "Sleuth: Automated Verification of Software Power Analysis Countermeasures". In: *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*. Ed. by Guido Bertoni and Jean-Sébastien Coron. Vol. 8086. Lecture Notes in Computer Science. Springer, 2013, pp. 293–310. DOI: 10.1007/978-3-642-40349-1_17.

[33] Georg T. Becker, Jeremy Cooper, Elke DeMulder, Gilbert Goodwill, Joshua Jaffe, Gary Kenworthy, Timofei Kouzminov, Andrew J. Leiserson, Mark E. Marson, Pankaj Rohatgi, and Sami Saab. *Test vector leakage assessment (TVLA) methodology in practice*. International Cryptographic Module Conference. 2013.

[34] Arthur Beckers, Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. "Design and Implementation of a Waveform-Matching Based Triggering System". In: *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*. Ed. by François-Xavier Standaert and Elisabeth Oswald. Vol. 9689. Lecture Notes in Computer Science. Springer, 2016, pp. 184–198. DOI: 10.1007/978-3-319-43283-0_11.

[35] Christof Beierle, Gregor Leander, Amir Moradi, and Shahram Rasoolzadeh. "CRAFT: Lightweight Tweakable Block Cipher with Efficient Protection Against DFA Attacks". In: *IACR ToSC* 2019.1 (2019), pp. 5–45. DOI: 10.13154/tosc.v2019.i1.5-45.

[36] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. "Differential Power Analysis of HMAC SHA-2 in the Hamming Weight Model". In: *SECRYPT 2013 - Proceedings of the 10th International Conference on Security and Cryptography, Reykjavík, Iceland, 29-31 July, 2013*. Ed. by Pierangela Samarati. SciTePress, 2013, pp. 230–241.

[37] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. "Deep learning for side-channel analysis and introduction to ASCAD database". In: *J. Cryptogr. Eng.* 10.2 (2020), pp. 163–188. DOI: 10.1007/s13389-019-00220-8.

[38] Olivier Benoît and Thomas Peyrin. "Side-Channel Analysis of Six SHA-3 Candidates". In: *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Springer, 2010, pp. 140–157. DOI: 10.1007/978-3-642-15031-9_10.

[39] Daniel J Bernstein. *Cache-timing attacks on AES*. 2005.

[40] Daniel J Bernstein et al. "ChaCha, a variant of Salsa20". In: *Workshop record of SASC*. Vol. 8. 2008, pp. 3–5.

[41] Daniel J. Bernstein. *25519 naming*. Posting to the CFRG mailing list. https://www.ietf.org/mail-archive/web/cfrg/current/msg04996.html. 2014.

[42] Daniel J. Bernstein. "A state-of-the-art Diffie-Hellman function". In: https://cr.yp.to/ecdh.html (accessed 2021-05-05).

[43] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records". In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. Ed. by Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. Vol. 3958. Lecture Notes in Computer Science. Springer, 2006, pp. 207–228. DOI: 10.1007/11745853_14.

[44] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers". In: *New Stream Cipher Designs - The eSTREAM Finalists*. Ed. by Matthew J. B. Robshaw and Olivier Billet. Vol. 4986. Lecture Notes in Computer Science. Springer, 2008, pp. 84–97. DOI: 10.1007/978-3-540-68351-3_8.

[45] Daniel J. Bernstein, Peter Birkner, Marc Joye, Tanja Lange, and Christiane Peters. "Twisted Edwards Curves". In: *Progress in Cryptology - AFRICACRYPT 2008, First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*. Ed. by Serge Vaudenay. Vol. 5023. Lecture Notes in Computer Science. Springer, 2008, pp. 389–405. DOI: 10.1007/978-3-540-68164-9_26.

[46] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-Speed High-Security Signatures". In: *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 124–142. DOI: 10.1007/978-3-642-23951-9_9.

[47] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures". In: *J. Cryptogr. Eng.* 2.2 (2012), pp. 77–89. DOI: 10.1007/s13389-012-0027-1.

[48] Daniel J. Bernstein, Stefan Kölbl, Stefan Lucks, Pedro Maat Costa Massolino, Florian Mendel, Kashif Nawaz, Tobias Schneider, Peter Schwabe, François-Xavier Standaert, Yosuke Todo, and Benoît Viguier. "Gimli : A Cross-Platform Permutation". In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 299–320. DOI: 10.1007/978-3-319-66787-4_15.

[49] Daniel J. Bernstein and Tanja 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*. Ed. by Kaoru Kurosawa. Vol. 4833. Lecture Notes in Computer Science. Springer, 2007, pp. 29–50. DOI: 10.1007/978-3-540-76900-2_3.

[50] Daniel J. Bernstein, Tanja Lange, and Peter 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*. Ed. by Alejandro Hevia and Gregory Neven. Vol. 7533. Lecture Notes in Computer Science. Springer, 2012, pp. 159–176. DOI: 10.1007/978-3-642-33481-8_9.

[51] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. *The Keccak reference*. http://keccak.noekeon.org/. Jan. 2011.

[52] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, and R. Van Keer. *Keccak implementation overview*. https://keccak.team/papers.html. May 2012.

[53] Guido Bertoni, Luca Breveglieri, Israel Koren, Paolo Maistri, and Vincenzo Piuri. "Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard". In: *IEEE Trans. Computers* 52.4 (2003), pp. 492–505. DOI: 10.1109/TC.2003.1190590.

[54] Guido Bertoni, Joan Daemen, Nicolas Debande, Thanh Ha Le, Michaël Peeters, and Gilles Van Assche. "Power analysis of hardware implementations protected with secret sharing". In: *Proceedings - 2012 IEEE/ACM 45th International Symposium on Microarchitecture Workshops, MICROW 2012*. Institute of Electrical and Electronics Engineers (IEEE), Dec. 2012, pp. 9–16. DOI: 10.1109/MICROW.2012.12.

[55] Guido Bertoni, Joan Daemen, Seth Hoffert, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. *The eXtended Keccak Code Package (XKCP)*.

[56] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles van Assche, and Ronny van Keer. *CAESAR submission: Keyak v2*. 2016.

[57] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications". In: *Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. Lecture Notes in Computer Science. Springer, 2011, pp. 320–337. DOI: 10.1007/978-3-642-28496-0_19.

[58] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Keccak". In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 313–314. DOI: 10.1007/978-3-642-38348-9_19.

[59] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Permutation-based encryption, authentication and authenticated encryption". In: *Directions in Authenticated Ciphers* (2012).

[60] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. "Sponge functions". In: *ECRYPT hash workshop*. Vol. 2007. 9. Citeseer. 2007.

[61] Guido Bertoni, Joan Daemen, Michaël Peeters, Gilles Van Assche, and Ronny Van Keer. *CAESAR submission: Ketje v2, 2016*.

[62] Ingrid Biehl, Bernd Meyer, and Volker Müller. "Differential Fault Attacks on Elliptic Curve Cryptosystems". In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. Ed. by Mihir Bellare. Vol. 1880. Lecture Notes in Computer Science. Springer, 2000, pp. 131–146. DOI: 10.1007/3-540-44598-6_8.

[63] Begül Bilgin, Joan Daemen, Ventzislav Nikov, Svetla Nikova, Vincent Rijmen, and Gilles Van Assche. "Efficient and First-Order DPA Resistant Implementations of Keccak". In: *Smart Card Research and Advanced Applications*. Springer Science & Business Media, 2014, pp. 187–199. DOI: 10.1007/978-3-319-08302-5_13.

[64] Begül Bilgin, Benedikt Gierlichs, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "Higher-Order Threshold Implementations". 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*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8874. Lecture Notes in Computer Science. Springer, 2014, pp. 326–343. DOI: 10.1007/978-3-662-45608-8_18.

[65] Begül Bilgin, Svetla Nikova, Ventzislav Nikov, Vincent Rijmen, and Georg Stütz. "Threshold Implementations of All 3 ×3 and 4 ×4 S-Boxes". In: *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 76–91. DOI: 10.1007/978-3-642-33027-8_5.

[66] Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. "Formal Verification of Masked Hardware Implementations in the Presence of Glitches". In: *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10821. Lecture Notes in Computer Science. Springer, 2018, pp. 321–353. DOI: 10.1007/978-3-319-78375-8_11.

[67] Johannes Blömer, Martin Otto, and Jean-Pierre Seifert. "Sign Change Fault Attacks on Elliptic Curve Cryptosystems". In: *Fault Diagnosis and Tolerance in Cryptography, Third International Workshop, FDTC 2006, Yokohama, Japan, October 10, 2006, Proceedings*. Ed. by Luca Breveglieri, Israel Koren, David Naccache, and Jean-Pierre Seifert. Vol. 4236. Lecture Notes in Computer Science. Springer, 2006, pp. 36–52. DOI: 10.1007/11889700_4.

[68] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. "PRESENT: An Ultra-Lightweight Block Cipher". In: *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, 2007, pp. 450–466. DOI: 10.1007/978-3-540-74735-2_31.

[69] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. "On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract)". In: *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Springer, 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

[70] Paul Bottinelli and Joppe W. Bos. "Computational aspects of correlation power analysis". In: *J. Cryptogr. Eng.* 7.3 (2017), pp. 167–181. DOI: 10.1007/s13389-016-0122-9.

[71] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2.

[72] Leon Groot Bruinderink and Peter Pessl. "Differential Fault Attacks on Deterministic Lattice Signatures". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 21–43. DOI: 10.13154/tches.v2018.i3.21-43.

[73] Billy Bob Brumley and Nicola Tuveri. "Remote Timing Attacks Are Still Practical". In: *Computer Security - ESORICS 2011 - 16th European Symposium on Research in Computer Security, Leuven, Belgium, September 12-14, 2011. Proceedings*. Ed. by Vijay Atluri and Claudia Díaz. Vol. 6879. Lecture Notes in Computer Science. Springer, 2011, pp. 355–371. DOI: 10.1007/978-3-642-23822-2_20.

[74] David Brumley and Dan Boneh. "Remote Timing Attacks Are Practical". In: *Proceedings of the 12th USENIX Security Symposium, Washington, D.C., USA, August 4-8, 2003*. USENIX Association, 2003.

[75] Eleonora Cagli, Cécile Dumas, and Emmanuel 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*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 45–68. DOI: 10.1007/978-3-319-66787-4_3.

[76] Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax, and Hervé Pelletier. "Electromagnetic Side Channels of an FPGA Implementation of AES". In: *IACR Cryptol. ePrint Arch.* 2004 (2004), p. 145.

[77] Rafael Boix Carpi, Stjepan Picek, Lejla Batina, Federico Menarini, Domagoj Jakobovic, and Marin Golub. "Glitch It If You Can: Parameter Search Strategies for Successful Fault Injection". In: *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*. Ed. by Aurélien Francillon and Pankaj Rohatgi. Vol. 8419. Lecture Notes in Computer Science. Springer, 2013, pp. 236–252. DOI: 10.1007/978-3-319-08302-5_16.

[78] Tony F. Chan, Gene H. Golub, and Randall J. Leveque. "Algorithms for Computing the Sample Variance: Analysis and Recommendations". In: *The American Statistician* 37.3 (1983), pp. 242–247. DOI: 10.1080/00031305.1983.10483115. eprint: https://www.tandfonline.com/doi/pdf/10.1080/00031305.1983.10483115.

[79] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. "Towards Sound Approaches to Counteract Power-Analysis Attacks". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed.

by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 398–412. DOI: 10.1007/3-540-48405-1_26.

[80] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.

[81] Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. "A Systematic Analysis of the Juniper Dual EC Incident". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi. ACM, 2016, pp. 468–479. DOI: 10.1145/2976749.2978395.

[82] Zhimin Chen, Syed Haider, and Patrick Schaumont. "Side-Channel Leakage in Masked Circuits Caused by Higher-Order Circuit Effects". In: *Advances in Information Security and Assurance, Third International Conference and Workshops, ISA 2009, Seoul, Korea, June 25-27, 2009. Proceedings*. Ed. by Jong Hyuk Park, Hsiao-Hwa Chen, Mohammed Atiquzzaman, Changhoon Lee, Tai-Hoon Kim, and Sang-Soo Yeo. Vol. 5576. Lecture Notes in Computer Science. Springer, 2009, pp. 327–336. DOI: 10.1007/978-3-642-02617-1_34.

[83] Zhimin Chen and Yujie Zhou. "Dual-Rail Random Switching Logic: A Countermeasure to Reduce Side Channel Leakage". In: *Cryptographic Hardware and Embedded Systems - CHES 2006, 8th International Workshop, Yokohama, Japan, October 10-13, 2006, Proceedings*. Ed. by Louis Goubin and Mitsuru Matsui. Vol. 4249. Lecture Notes in Computer Science. Springer, 2006, pp. 242–254. DOI: 10.1007/11894063_20.

[84] Łukasz Chmielewski, Pedro Maat Costa Massolino, Jo Vliegen, Lejla Batina, and Nele Mentens. "Completing the complete ECC formulae with countermeasures". In: *Journal of Low Power Electronics and Applications* 7.1 (2017), p. 3.

[85] Mathieu Ciet and Marc Joye. "Elliptic Curve Cryptosystems in the Presence of Permanent and Transient Faults". In: *Des. Codes Cryptogr.* 36.1 (2005), pp. 33–43. DOI: 10.1007/s10623-003-1160-8.

[86] Christophe Clavier. "Secret external encodings do not prevent transient fault analysis". In: *(CHES) 2007*. Springer. 2007, pp. 181–194.

[87] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. "Horizontal Correlation Analysis on Exponentiation". In: *Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings*. Ed. by Miguel Soriano, Sihan Qing, and Javier López. Vol. 6476. Lecture Notes in Computer Science. Springer, 2010, pp. 46–61. DOI: 10.1007/978-3-642-17650-0_5.

[88] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. "Higher-Order Threshold Implementation of the AES S-Box". In: *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*. Ed. by Naofumi Homma and Marcel Medwed. Vol. 9514. Lecture Notes in Computer Science. Springer, 2015, pp. 259–272. DOI: 10.1007/978-3-319-31271-2_16.

[89] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. "Masking AES With d+1 Shares in Hardware". In: *Proceedings of the ACM Workshop on Theory of Implementation Security, TIS@CCS 2016 Vienna, Austria, October, 2016*. Ed. by Begül Bilgin, Svetla Nikova, and Vincent Rijmen. ACM, 2016, p. 43. DOI: 10.1145/2996366.2996428.

[90] Jean-Sébastien Coron. "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems". In: *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 292–302. DOI: 10.1007/3-540-48059-5_25.

[91] Yann Le Corre, Johann Großschädl, and Daniel Dinu. "Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors". In: *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Springer, 2018, pp. 82–98. DOI: 10.1007/978-3-319-89641-0_5.

[92] J. Daemen, S. Hoffert, M. Peeters, G. Van Assche, and R. Van Keer. *Xoodyak, a lightweight cryptographic scheme*. csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/round-2/spec-doc-rnd2/Xoodyak-spec-round2.pdf. Apr. 2018.

[93] Joan Daemen. "Changing of the Guards : a simple and efficient method for achieving uniformity in threshold sharing". In: (2016), pp. 1–8.

[94] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. "The design of Xoodoo and Xoofff". In: *IACR Trans. Symmetric Cryptol.* 2018.4 (2018), pp. 1–38. DOI: 10.13154/tosc.v2018.i4.1-38.

[95] Joan Daemen, Seth Hoffert, Gilles Van Assche, and Ronny Van Keer. "Xoodoo cookbook". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 767.

[96] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Information Security and Cryptography. Springer, 2002. DOI: 10.1007/978-3-662-04722-4.

[97] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. "A synthesis of side-channel attacks on elliptic curve cryptography in smart-cards". In: *J. Cryptogr. Eng.* 3.4 (2013), pp. 241–265. DOI: 10.1007/s13389-013-0062-6.

[98] Lesly-Ann Daniel, Sébastien Bardin, and Tamara Rezk. "Binsec/Rel: Efficient Relational Symbolic Execution for Constant-Time at Binary-Level". In: *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 2020, pp. 1021–1038. DOI: 10.1109/SP40000.2020.00074.

[99] Fabrizio De Santis and Georg Sigl. "Towards side-channel protected X25519 on ARM Cortex-M4 processors". In: *SPEED-B Software performance enhancement for encryption and decryption, and benchmarking*. 2016.

[100] Whitfield Diffie and Martin E. Hellman. "New directions in cryptography". In: *IEEE Trans. Inf. Theory* 22.6 (1976), pp. 644–654. DOI: 10.1109/TIT.1976.1055638.

[101] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. "Towards Sound and Optimal Leakage Detection Procedure". In: *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*. Ed. by Thomas Eisenbarth and Yannick Teglia. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 105–122. DOI: 10.1007/978-3-319-75208-2_7.

[102] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. "Ascon v1.2". In: *Submission to the CAESAR Competition* (2016).

[103] Jack W Dunlap. "Combinative properties of correlation coefficients". In: *The Journal of Experimental Education* 5.3 (Jan. 1937), pp. 286–288. DOI: 10.1080/00220973.1937.11010053.

[104] Thai Duong. *Why not validate Curve25519 public keys could be harmful*. https://vnhacker.blogspot.com/2015/09/why-not-validating-curve25519-public.html (accessed 2020-12-02). 2015.

[105] Morris J Dworkin. "SHA-3 standard: Permutation-based hash and extendable-output functions". In: (2015).

[106] *ECRYPT II Key Recommendations*. 2012.

[107] Harold Edwards. "A normal form for elliptic curves". In: *Bulletin of the American mathematical society* 44.3 (2007), pp. 393–422.

[108] A. E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2003. DOI: 10.1007/978-3-662-05094-1.

[109] Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, and Mohammad T. Manzuri Shalmani. "On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme". In: *Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*. Ed. by David A. Wagner. Vol. 5157. Lecture Notes in Computer Science. Springer, 2008, pp. 203–220. DOI: 10.1007/978-3-540-85174-5_12.

[110] Hassan Eldib and Chao Wang. "Synthesis of Masking Countermeasures against Side Channel Attacks". In: *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*. Ed. by Armin Biere and Roderick Bloem. Vol. 8559. Lecture Notes in Computer Science. Springer, 2014, pp. 114–130. DOI: 10.1007/978-3-319-08867-9_8.

[111] Hassan Eldib, Chao Wang, and Patrick Schaumont. "SMT-Based Verification of Software Countermeasures against Side-Channel Attacks". In: *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*. Ed. by Erika Ábrahám and Klaus Havelund. Vol. 8413. Lecture Notes in Computer Science. Springer, 2014, pp. 62–77. DOI: 10.1007/978-3-642-54862-8_5.

[112] Junfeng Fan, Benedikt Gierlichs, and Frederik Vercauteren. "To Infinity and Beyond: Combined Attack on ECC Using Points of Low Order". In: *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 143–159. DOI: 10.1007/978-3-642-23951-9_10.

[113] Junfeng Fan, Xu Guo, Elke De Mulder, Patrick Schaumont, Bart Preneel, and Ingrid Verbauwhede. "State-of-the-art of Secure ECC Implementations: A Survey on Known Side-channel Attacks and Countermeasures". In: *HOST 2010, Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), 13-14 June 2010, Anaheim Convention Center, California, USA*. Ed. by Jim Plusquellic and Ken Mai. IEEE Computer Society, 2010, pp. 76–87. DOI: 10.1109/HST.2010.5513110.

[114] Junfeng Fan and Ingrid Verbauwhede. "An Updated Survey on Secure ECC Implementations: Attacks, Countermeasures and Cost". In: *Cryptography and Security: From Theory to Applications - Essays Dedicated to Jean-Jacques Quisquater on the Occasion of His 65th Birthday*. Ed. by David Naccache. Vol. 6805. Lecture Notes in Computer Science. Springer, 2012, pp. 265–282. DOI: 10.1007/978-3-642-28368-0_18.

[115] Wieland Fischer and Berndt M. Gammel. "Masking at Gate Level in the Presence of Glitches". In: *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 187–200. DOI: 10.1007/11545262_14.

[116] OPC Foundation. *Unified Architecture*. https://opcfoundation.org/about/opc-technologies/opc-ua/ (accessed 2021-05-05). 2008.

[117] Pierre-Alain Fouque, Reynald Lercier, Denis Réal, and Frédéric Valette. "Fault attack on elliptic curve Montgomery ladder implementation". In: *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2008, pp. 92–98.

[118] Pierre-Alain Fouque, Denis Réal, Frédéric Valette, and M'hamed Drissi. "The Carry Leakage on the Randomized Exponent Countermeasure". In: *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Vol. 5154. Lecture Notes in Computer Science. Springer, 2008, pp. 198–213. DOI: 10.1007/978-3-540-85053-3_13.

[119] Pierre-Alain Fouque and Frédéric Valette. "The Doubling Attack - *Why Upwards Is Better than Downwards*". In: *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. Lecture Notes in Computer Science. Springer, 2003, pp. 269–280. DOI: 10.1007/978-3-540-45238-6_22.

[120] Hayato Fujii and Diego F Aranha. "Efficient Curve25519 implementation for ARM microcontrollers". In: *Anais Estendidos do XVIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. SBC. 2018, pp. 57–64.

[121] Hayato Fujii and Diego F. Aranha. "Curve25519 for the Cortex-M4 and Beyond". In: *Progress in Cryptology - LATINCRYPT 2017 - 5th International Conference on Cryptology and Information Security in Latin America, Havana, Cuba, September 20-22, 2017, Revised Selected Papers*. Ed. by Tanja Lange and Orr Dunkelman. Vol. 11368. Lecture Notes in Computer Science. Springer, 2017, pp. 109–127. DOI: 10.1007/978-3-030-25283-0_6.

[122] Stephen Bo Furber. *ARM system-on-chip architecture*. pearson Education, 2000.

[123] Taher El Gamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms". In: *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*. Ed. by G. R. Blakley and David Chaum. Vol. 196. Lecture Notes in Computer Science. Springer, 1984, pp. 10–18. DOI: 10.1007/3-540-39568-7_2.

[124] Karine Gandolfi, Christophe Mourtel, and Francis Olivier. "Electromagnetic Analysis: Concrete Results". In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science Generators. Springer, 2001, pp. 251–261. DOI: 10.1007/3-540-44709-1_21.

[125] Si Gao. *A Thumb Assembly based Byte-wise Masked AES Implementation*. https://github.com/bristol-sca/ASM_MaskedAES. 2019.

[126] Si Gao, Johann Großschädl, Ben Marshall, Dan Page, Thinh Hung Pham, and Francesco Regazzoni. "An Instruction Set Extension to Support Software-Based Masking". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 773.

[127] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. "Share-slicing: Friend or Foe?" In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.1 (2020), pp. 152–174. DOI: 10.13154/tches.v2020.i1.152-174.

[128] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware". In: *J. Cryptogr. Eng.* 8.1 (2018), pp. 1–27. DOI: 10.1007/s13389-016-0141-6.

[129] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. "ECDH Key-Extraction via Low-Bandwidth Electromagnetic Attacks on PCs". In: *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*. Ed. by Kazue Sako. Vol. 9610. Lecture Notes in Computer Science. Springer, 2016, pp. 219–235. DOI: 10.1007/978-3-319-29485-8_13.

[130] Daniel Genkin, Adi Shamir, and Eran Tromer. "RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis". In: *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. Lecture Notes in Computer Science. Springer, 2014, pp. 444–461. DOI: 10.1007/978-3-662-44371-2_25.

[131] Daniel Genkin, Luke Valenta, and Yuval Yarom. "May the Fourth Be With You: A Microarchitectural Side Channel Attack on Several Real-World Applications of Curve25519". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM, 2017, pp. 845–858. DOI: 10.1145/3133956.3134029.

[132] Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. "Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 1294.

[133] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. "A testing methodology for side-channel resistance validation". In: *NIST non-invasive attack testing workshop*. Vol. 7. 2011, pp. 115–136.

[134] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. *A testing methodology for side-channel resistance validation*. 2011.

[135] *BoringSSL*. https://github.com/boringssl/boringssl (accessed 2021-05-05). 2021.

[136] Louis Goubin. "A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems". In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Springer, 2003, pp. 199–210. DOI: 10.1007/3-540-36288-6_15.

[137] Louis Goubin. "A Sound Method for Switching between Boolean and Arithmetic Masking". In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Springer, 2001, pp. 3–15. DOI: 10.1007/3-540-44709-1_2.

[138] Louis Goubin and Jacques Patarin. “DES and Differential Power Analysis (The "Duplication" Method)”. In: *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. Lecture Notes in Computer Science. Springer, 1999, pp. 158–172. DOI: 10.1007/3-540-48059-5_15.

[139] Hannes Groß, Manuel Jelinek, Stefan Mangard, Thomas Unterluggauer, and Mario Werner. “Concealing Secrets in Embedded Processors Designs”. In: *Smart Card Research and Advanced Applications - 15th International Conference, CARDIS 2016, Cannes, France, November 7-9, 2016, Revised Selected Papers*. Ed. by Kerstin Lemke-Rust and Michael Tunstall. Vol. 10146. Lecture Notes in Computer Science. Springer, 2016, pp. 89–104. DOI: 10.1007/978-3-319-54669-8_6.

[140] Björn Haase and Benoît Labrique. “AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT”. In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.2 (2019), pp. 1–48. DOI: 10.13154/tches.v2019.i2.1-48.

[141] Björn Haase and Benoît Labrique. “Making Password Authenticated Key Exchange Suitable for Resource-Constrained Industrial Control Devices”. In: *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Springer, 2017, pp. 346–364. DOI: 10.1007/978-3-319-66787-4_17.

[142] Darrel Hankerson, Alfred J Menezes, and Scott Vanstone. *Guide to elliptic curve cryptography*. Springer Science & Business Media, 2006.

[143] Neil Hanley, HeeSeok Kim, and Michael Tunstall. “Exploiting Collisions in Addition Chain-Based Exponentiation Algorithms Using a Single Trace”. In: *Topics in Cryptology - CT-RSA 2015, The Cryptographer's Track at the RSA Conference 2015, San Francisco, CA, USA, April 20-24, 2015. Proceedings*. Ed. by Kaisa Nyberg. Vol. 9048. Lecture Notes in Computer Science. Springer, 2015, pp. 431–448. DOI: 10.1007/978-3-319-16715-2_23.

[144] Jerry den Hartog, Jan Verschuren, Erik P. de Vink, Jaap de Vos, and W. Wiersma. “PINPAS: A Tool for Power Analysis of Smartcards”. In: *Security and Privacy in the Age of Uncertainty, IFIP TC11 18th International Conference on Information Security (SEC2003), May 26-28, 2003, Athens, Greece*. Ed. by Dimitris Gritzalis, Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sokratis K. Katsikas. Vol. 250. IFIP Conference Proceedings. Kluwer, 2003, pp. 453–457.

[145] Marcella Hastings, Joshua Fried, and Nadia Heninger. “Weak Keys Remain Widespread in Network Devices”. In: *Proceedings of the 2016 ACM on Internet Measurement Conference, IMC 2016, Santa Monica, CA, USA, November 14-16, 2016*. Ed. by Phillipa Gill, John S. Heidemann, John W. Byers, and Ramesh Govindan. ACM, 2016, pp. 49–63.

[146] Johann Heyszl, Stefan Mangard, Benedikt Heinz, Frederic Stumpf, and Georg Sigl. "Localized Electromagnetic Analysis of Cryptographic Implementations". In: *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*. Ed. by Orr Dunkelman. Vol. 7178. Lecture Notes in Computer Science. Springer, 2012, pp. 231–244. DOI: 10.1007/978-3-642-27954-6_15.

[147] Jochen Hoenicke. *Extracting the Private Key from a TREZOR ... with a 70$ Oscilloscope*. https://jochen-hoenicke.de/crypto/trezor-power-analysis/ (accessed 2012-12-02). 2015.

[148] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992. DOI: 10.7551/mitpress/1090.001.0001.

[149] Naofumi Homma, Atsushi Miyamoto, Takafumi Aoki, Akashi Satoh, and Adi Shamir. "Collision-Based Power Analysis of Modular Exponentiation Using Chosen-Message Pairs". In: *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*. Ed. by Elisabeth Oswald and Pankaj Rohatgi. Vol. 5154. Lecture Notes in Computer Science. Springer, 2008, pp. 15–29. DOI: 10.1007/978-3-540-85053-3_2.

[150] IANIX. "Things that use Curve15519". In: https://ianix.com/pub/curve25519-deployment.html (accessed 2021-05-05).

[151] Donald E. Eastlake III and Paul E. Jones. "US Secure Hash Algorithm 1 (SHA1)". In: *RFC* 3174 (2001), pp. 1–22. DOI: 10.17487/RFC3174.

[152] Yuval Ishai, Amit Sahai, and David A. Wagner. "Private Circuits: Securing Hardware against Probing Attacks". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 463–481. DOI: 10.1007/978-3-540-45146-4_27.

[153] *Information technology – Security techniques – Testing methods for the mitigation of non-invasive attack classes against cryptographic modules*. Standard. Geneva, CH: International Organization for Standardization, 2016.

[154] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. "A Practical Countermeasure against Address-Bit Differential Power Analysis". In: *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. Lecture Notes in Computer Science. Springer, 2003, pp. 382–396. DOI: 10.1007/978-3-540-45238-6_30.

[155] Kouichi Itoh, Tetsuya Izu, and Masahiko Takenaka. "Address-Bit Differential Power Analysis of Cryptographic Schemes OK-ECDH and OK-ECDSA". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar.

Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 129–143. DOI: 10.1007/3-540-36400-5_11.

[156] Josh Jaffe, Pankaj Rohatgi, and Marc Witteman Riscure. "Efficient sidechannel testing for public key algorithms: RSA case study". In: (2011).

[157] Jan Jancar, Vladimir Sedlacek, Petr Svenda, and Marek Sýs. "Minerva: The curse of ECDSA nonces Systematic analysis of lattice attacks on noisy leakage of bit-length of ECDSA nonces". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2020.4 (2020), pp. 281–308. DOI: 10.13154/tches.v2020.i4.281-308.

[158] Don Johnson, Alfred Menezes, and Scott A. Vanstone. "The Elliptic Curve Digital Signature Algorithm (ECDSA)". In: *Int. J. Inf. Sec.* 1.1 (2001), pp. 36–63. DOI: 10.1007/s102070100002.

[159] Marc 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*. Ed. by Pascal Paillier and Ingrid Verbauwhede. Vol. 4727. Lecture Notes in Computer Science. Springer, 2007, pp. 135–147. DOI: 10.1007/978-3-540-74735-2_10.

[160] Marc Joye and Sung-Ming Yen. "The Montgomery Powering Ladder". In: *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Springer, 2002, pp. 291–302. DOI: 10.1007/3-540-36400-5_22.

[161] Bernhard Jungk, Richard Petri, and Marc Stöttinger. "Efficient Side-Channel Protections of ARX Ciphers". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 627–653. DOI: 10.13154/tches.v2018.i3.627-653.

[162] Mark G. Karpovsky, Konrad J. Kulikowski, and Alexander Taubin. "Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard". In: *2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June - 1 July 2004, Florence, Italy, Proceedings*. IEEE Computer Society, 2004, pp. 93–101. DOI: 10.1109/DSN.2004.1311880.

[163] Michael Kasper, Richard Petri, Dirk Feldhusen, Max Gebhardt, Georg Illies, Manfred Lochter, Oliver Stein, Wolfgang Thumserang, and Guntram Wicke. *Minimum Requirements for Evaluating Side-Channel Attack Resistance of Elliptic Curve Implementations*. http://publica.fraunhofer.de/documents/N-487544.html (accessed 2021-05-05). 2016.

[164] Emilia Käsper and Peter Schwabe. "Faster and Timing-Attack Resistant AES-GCM". In: *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*. Ed. by Christophe Clavier and Kris Gaj. Vol. 5747. Lecture Notes in Computer Science. Springer, 2009, pp. 1–17. DOI: 10.1007/978-3-642-04138-9_1.

[165] C Kerry and P Gallagher. "FIPS PUB 186-4: Digital Signature Standard (DSS)". In: *Federal Information Processing Standards Publication. National Institute of Standards und Technology* (2013).

[166] Pantea Kiaei and Patrick Schaumont. "Domain-Oriented Masked Instruction Set Architecture for RISC-V". In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 465.

[167] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. "Make Some Noise. Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.3 (2019), pp. 148–179. DOI: 10.13154/tches.v2019.i3.148-179.

[168] Yoongu Kim, Ross Daly, Jeremie S. Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. "Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors". In: *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*. IEEE Computer Society, 2014, pp. 361–372. DOI: 10.1109/ISCA.2014.6853210.

[169] Neal Koblitz. "Constructing Elliptic Curve Cryptosystems in Characteristic 2". In: *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*. Ed. by Alfred Menezes and Scott A. Vanstone. Vol. 537. Lecture Notes in Computer Science. Springer, 1990, pp. 156–167. DOI: 10.1007/3-540-38424-3_11.

[170] Neal Koblitz. "Elliptic Curve Cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209. DOI: 10.2307/2007884.

[171] Paul 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*. Ed. by Neal Koblitz. Vol. 1109. Lecture Notes in Computer Science. Springer, 1996, pp. 104–113. DOI: 10.1007/3-540-68697-5_9.

[172] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. "Differential Power Analysis". In: *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Springer, 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[173] Oliver Kömmerling and Markus G. Kuhn. "Design Principles for Tamper-Resistant Smartcard Processors". In: *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. Ed. by Scott B. Guthery and Peter Honeyman. USENIX Association, 1999.

[174] Thilo Krachenfels, Fatemeh Ganji, Amir Moradi, Shahin Tajik, and Jean-Pierre Seifert. "Real-World Snapshots vs. Theory: Questioning the t-Probing Security Model". In: *CoRR* abs/2009.04263 (2020). arXiv: 2009.04263.

[175] Juliane Krämer, Dmitry Nedospasov, Alexander Schlösser, and Jean-Pierre Seifert. "Differential Photonic Emission Analysis". In: *Constructive Side-Channel Analysis and Secure Design - 4th International Workshop, COSADE 2013, Paris, France, March 6-8, 2013, Revised Selected Papers*. Ed. by Emmanuel Prouff. Vol. 7864. Lecture Notes in Computer Science. Springer, 2013, pp. 1–16. DOI: 10.1007/978-3-642-40026-1_1.

[176] Hugo Krawczyk. "SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in the IKE-Protocols". In: *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*. Ed. by Dan Boneh. Vol. 2729. Lecture Notes in Computer Science. Springer, 2003, pp. 400–425. DOI: 10.1007/978-3-540-45146-4_24.

[177] Benjamin Lac, Anne Canteaut, Jacques J. A. Fournier, and Renaud Sirdey. "Thwarting Fault Attacks using the Internal Redundancy Countermeasure (IRC)". In: *IACR ePrint Archive* 2017 (2017), p. 910.

[178] Kerstin Lemke, Kai Schramm, and Christof Paar. "DPA on n-Bit Sized Boolean and Arithmetic Operations and Its Application to IDEA, RC6, and the HMAC-Construction". In: *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Springer, 2004, pp. 205–219. DOI: 10.1007/978-3-540-28632-5_15.

[179] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. "Power analysis attack: an approach based on machine learning". In: *Int. J. Appl. Cryptogr.* 3.2 (2014), pp. 97–115. DOI: 10.1504/IJACT.2014.062722.

[180] Zhe Liu, Patrick Longa, Geovandro C. C. F. Pereira, Oscar Reparaz, and Hwajeong Seo. "FourQ on Embedded Devices with Strong Countermeasures Against Side-Channel Attacks". In: *IEEE Trans. Dependable Secur. Comput.* 17.3 (2020), pp. 536–549. DOI: 10.1109/TDSC.2018.2799844.

[181] Zhuoran Liu, Niels Samwel, Leo Weissbart, Zhengyu Zhao, Dirk Lauret, Lejla Batina, and Martha A. Larson. "Screen Gleaning: A Screen Reading TEMPEST Attack on Mobile Devices Exploiting an Electromagnetic Side Channel". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[182] Manfred Lochter, Johannes Merkle, Jörn-Marc Schmidt, and Torsten Schütze. *Requirements for Elliptic Curves for High-Assurance Applications*. Workshop record of the NIST Workshop on Elliptic Curve Cryptography Standards. https://csrc.nist.gov/csrc/media/events/workshop-on-elliptic-curve-cryptography-standards/documents/papers/session4-merkle-johannes.pdf. 2015.

[183] Antoine Loiseau, Maxime Lecomte, and Jacques J. A. Fournier. "Template Attacks against ECC: practical implementation against Curve25519". In: *2020 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2020, San Jose, CA, USA, December 7-11, 2020*. IEEE, 2020, pp. 13–22. DOI: 10.1109/HOST45689.2020.9300261.

[184] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. "Algebraic fault analysis of SHA-3 under relaxed fault models". In: *IEEE Transactions on Information Forensics and Security* 13.7 (2018), pp. 1752–1761.

[185] Pei Luo, Yunsi Fei, Xin Fang, A. Adam Ding, David R. Kaeli, and Miriam Leeser. "Side-channel analysis of MAC-Keccak hardware implementations". In: *Proceedings of the Fourth Workshop on Hardware and Architectural Support for Security and Privacy - HASP '15*. Association for Computing Machinery (ACM), 2015. DOI: 10.1145/2768566.2768567.

[186] Pei Luo, Yunsi Fei, Liwei Zhang, and A. Adam Ding. "Differential Fault Analysis of SHA-3 Under Relaxed Fault Models". In: *J. Hardw. Syst. Secur.* 1.2 (2017), pp. 156–172. DOI: 10.1007/s41635-017-0011-4.

[187] Pei Luo, Yunsi Fei, Liwei Zhang, and A. Adam Ding. "Differential Fault Analysis of SHA3-224 and SHA3-256". In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*. IEEE Computer Society, 2016, pp. 4–15. DOI: 10.1109/FDTC.2016.17.

[188] Maxime Madau, Michel Agoyan, and Philippe Maurine. "An EM Fault Injection Susceptibility Criterion and Its Application to the Localization of Hotspots". In: *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*. Ed. by Thomas Eisenbarth and Yannick Teglia. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 180–195. DOI: 10.1007/978-3-319-75208-2_11.

[189] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. "Breaking Cryptographic Implementations Using Deep Learning Techniques". In: *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*. Ed. by Claude Carlet, M. Anwar Hasan, and Vishal Saraswat. Vol. 10076. Lecture Notes in Computer Science. Springer, 2016, pp. 3–26. DOI: 10.1007/978-3-319-49445-6_1.

[190] Antun Maldini, Niels Samwel, Stjepan Picek, and Lejla Batina. "Genetic Algorithm-Based Electromagnetic Fault Injection". In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018*. IEEE Computer Society, 2018, pp. 35–42. DOI: 10.1109/FDTC.2018.00014.

[191] Antun Maldini, Niels Samwel, Stjepan Picek, and Lejla Batina. "Optimizing Electromagnetic Fault Injection with Genetic Algorithms". In: *Automated Methods in Cryptographic Fault Analysis*. Springer, 2019, pp. 281–300.

[192] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.

[193] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. "Side-Channel Leakage of Masked CMOS Gates". In: *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings*. Ed. by Alfred Menezes. Vol. 3376. Lecture

Notes in Computer Science. Springer, 2005, pp. 351–365. DOI: 10.1007/978-3-540-30574-3_24.

[194] Moxie Marlinspike. *Simplifying OTR deniability*. https://signal.org/blog/simplifying-otr-deniability/ (accessed 2021-05-05). 2013.

[195] Honorio Martín, Thomas Korak, Enrique San Millán, and Michael Hutter. "Fault Attacks on STRNGs: Impact of Glitches, Temperature, and Underpowering on Randomness". In: *IEEE Trans. Inf. Forensics Secur.* 10.2 (2015), pp. 266–277. DOI: 10.1109/TIFS.2014.2374072.

[196] Nick Mathewson. *Cryptographic directions in Tor*. Presentation at Real-World Crypto 2016. https://rwc.iacr.org/2016/Slides/nickm-rwc-presentation.pdf. 2016.

[197] David McCann, Elisabeth Oswald, and Carolyn Whitnall. "Towards Practical Tools for Side Channel Aware Software Engineering: 'Grey Box' Modelling for Instruction Leakages". In: *26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017*. Ed. by Engin Kirda and Thomas Ristenpart. USENIX Association, 2017, pp. 199–216.

[198] Robert P. McEvoy, Michael Tunstall, Colin C. Murphy, and William P. Marnane. "Differential Power Analysis of HMAC Based on SHA-2, and Countermeasures". In: *Information Security Applications, 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers*. Ed. by Sehun Kim, Moti Yung, and Hyung-Woo Lee. Vol. 4867. Lecture Notes in Computer Science. Springer, 2007, pp. 317–332. DOI: 10.1007/978-3-540-77535-5_23.

[199] Marcel Medwed and Elisabeth Oswald. "Template Attacks on ECDSA". In: *Information Security Applications, 9th International Workshop, WISA 2008, Jeju Island, Korea, September 23-25, 2008, Revised Selected Papers*. Ed. by Kyo-Il Chung, Kiwook Sohn, and Moti Yung. Vol. 5379. Lecture Notes in Computer Science. Springer, 2008, pp. 14–27. DOI: 10.1007/978-3-642-00306-6_2.

[200] Alfred Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. "Reducing elliptic curve logarithms to logarithms in a finite field". In: *IEEE Trans. Inf. Theory* 39.5 (1993), pp. 1639–1646. DOI: 10.1109/18.259647.

[201] Thomas S Messerges. *Power analysis attacks and countermeasures for cryptographic algorithms*. University of Illinois at Chicago, 2000.

[202] Thomas S. Messerges. "Securing the AES Finalists Against Power Analysis Attacks". In: *Fast Software Encryption, 7th International Workshop, FSE 2000, New York, NY, USA, April 10-12, 2000, Proceedings*. Ed. by Bruce Schneier. Vol. 1978. Lecture Notes in Computer Science. Springer, 2000, pp. 150–164. DOI: 10.1007/3-540-44706-7_11.

[203] Thomas S. Messerges and Ezzy A. Dabbish. "Investigations of Power Analysis Attacks on Smartcards". In: *Proceedings of the 1st Workshop on Smartcard Technology, Smartcard 1999, Chicago, Illinois, USA, May 10-11, 1999*. Ed. by Scott B. Guthery and Peter Honeyman. USENIX Association, 1999.

[204] Victor S. Miller. "Use of Elliptic Curves in Cryptography". In: *Advances in Cryptology - CRYPTO '85, Santa Barbara, California, USA, August 18-22, 1985, Proceedings*. Ed. by Hugh C. Williams. Vol. 218. Lecture Notes in Computer Science. Springer, 1985, pp. 417–426. DOI: 10.1007/3-540-39799-X_31.

[205] Peter L Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of computation* 48.177 (1987), pp. 243–264.

[206] Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. "Compiler Assisted Masking". In: *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. Ed. by Emmanuel Prouff and Patrick Schaumont. Vol. 7428. Lecture Notes in Computer Science. Springer, 2012, pp. 58–75. DOI: 10.1007/978-3-642-33027-8_4.

[207] Elke De Mulder, Samatha Gummalla, and Michael Hutter. "Protecting RISC-V against Side-Channel Attacks". In: *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 45. DOI: 10.1145/3316781.3323485.

[208] Elke De Mulder, Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. "Differential power and electromagnetic attacks on a FPGA implementation of elliptic curve cryptosystems". In: *Comput. Electr. Eng.* 33.5-6 (2007), pp. 367–382. DOI: 10.1016/j.compeleceng.2007.05.009.

[209] David Naccache, Nigel P. Smart, and Jacques Stern. "Projective Coordinates Leak". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 257–267. DOI: 10.1007/978-3-540-24676-3_16.

[210] LW Nagle and DO Pederson. "Simulation program with integrated circuit emphasis (SPICE)". In: *16th Midwest Symposium on Circuit Theory*. 1973.

[211] Erick Nascimento and Lukasz 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*. Ed. by Thomas Eisenbarth and Yannick Teglia. Vol. 10728. Lecture Notes in Computer Science. Springer, 2017, pp. 213–231. DOI: 10.1007/978-3-319-75208-2_13.

[212] Erick Nascimento, Lukasz Chmielewski, David F. Oswald, and Peter Schwabe. "Attacking Embedded ECC Implementations Through cmov Side Channels". In: *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. Springer, 2016, pp. 99–119. DOI: 10.1007/978-3-319-69453-5_6.

[213] Phong Q. Nguyen and Igor E. Shparlinski. "The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces". In: *Des. Codes Cryptogr.* 30.2 (2003), pp. 201–217. DOI: 10.1023/A:1025436905711.

[214] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. "Threshold Implementations Against Side-Channel Attacks and Glitches". In: *Information and Communications Security*. Springer Science & Business Media, 2006, pp. 529–545. DOI: 10.1007/11935308_38.

[215] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. "Threshold Implementations Against Side-Channel Attacks and Glitches". In: *Information and Communications Security, 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006, Proceedings*. Ed. by Peng Ning, Sihan Qing, and Ninghui Li. Vol. 4307. Lecture Notes in Computer Science. Springer, 2006, pp. 529–545. DOI: 10.1007/11935308_38.

[216] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches". In: *Journal of Cryptology* 24.2 (Oct. 2010), pp. 292–321. DOI: 10.1007/s00145-010-9085-7.

[217] Yoav Nir, Simon Josefsson, and Manuel Pegourie-Gonnard. "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier". In: *RFC* 8422 (2018), pp. 1–34. DOI: 10.17487/RFC8422.

[218] NXP. *SmartMX2 P40 family P40C012/040/072 Secure smart card controller*. https://www.nxp.com/docs/en/data-sheet/P40C040_C072_SMX2_FAM_SDS.pdf. 2015.

[219] Colin O'Flynn. "Fault Injection using Crowbars on Embedded Systems". In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 810.

[220] Paul C. van Oorschot and Michael J. Wiener. "Parallel Collision Search with Cryptanalytic Applications". In: *J. Cryptol.* 12.1 (1999), pp. 1–28. DOI: 10.1007/PL00003816.

[221] Sébastien Ordas, Ludovic Guillaume-Sage, and Philippe Maurine. "EM Injection: Fault Model and Locality". In: *2015 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2015, Saint Malo, France, September 13, 2015*. Ed. by Naofumi Homma and Victor Lomné. IEEE Computer Society, 2015, pp. 3–13. DOI: 10.1109/FDTC.2015.9.

[222] David F. Oswald and Christof Paar. "Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World". In: *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*. Ed. by Bart Preneel and Tsuyoshi Takagi. Vol. 6917. Lecture Notes in Computer Science. Springer, 2011, pp. 207–222. DOI: 10.1007/978-3-642-23951-9_14.

[223] Louiza Papachristodoulou, Apostolos P. Fournaris, Kostas Papagiannopoulos, and Lejla Batina. "Practical Evaluation of Protected Residue Number System Scalar Multiplication". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2019.1 (2019), pp. 259–282. DOI: 10.13154/tches.v2019.i1.259-282.

[224] Kostas Papagiannopoulos and Nikita Veshchikov. "Mind the Gap: Towards Secure 1st-Order Masking in Software". In: *Constructive Side-Channel Analysis and Secure Design - 8th International Workshop, COSADE 2017, Paris, France, April 13-14, 2017, Revised Selected Papers*. Ed. by Sylvain Guilley. Vol. 10348. Lecture Notes in Computer Science. Springer, 2017, pp. 282–297. DOI: 10.1007/978-3-319-64647-3_17.

[225] Scott Pardo. *Equivalence and Noninferiority Tests for Quality, Manufacturing and Test Engineers*. 1st ed. Philadelphia, PA: CRC Press LLC, 2013.

[226] Aesun Park, Kyung-Ah Shim, Namhun Koo, and Dong-Guk Han. "Side-Channel Attacks on Post-Quantum Signature Schemes based on Multivariate Quadratic Equations - Rainbow and UOV -". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2018.3 (2018), pp. 500–523. DOI: 10.13154/tches.v2018.i3.500-523.

[227] Conor Patrick, Bilgiday Yuce, Nahid Farhady Ghalaty, and Patrick Schaumont. "Lightweight Fault Attack Resistance in Software Using Intra-instruction Redundancy". In: *SAC 2016*. 2016, pp. 231–244. DOI: 10.1007/978-3-319-69453-5_13.

[228] Philippe Pierre Pebay. *Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments.* Tech. rep. Sandia National Laboratories, Sept. 2008. DOI: 10.2172/1028931.

[229] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. "Improved Higher-Order Side-Channel Attacks with FPGA Experiments". In: *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*. Ed. by Josyula R. Rao and Berk Sunar. Vol. 3659. Lecture Notes in Computer Science. Springer, 2005, pp. 309–323. DOI: 10.1007/11545262_23.

[230] Guilherme Perin, Lukasz Chmielewski, Lejla Batina, and Stjepan Picek. "Keep it Unsupervised: Horizontal Attacks Meet Deep Learning". In: *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2021.1 (2021), pp. 343–372. DOI: 10.46586/tches.v2021.i1.343-372.

[231] Stjepan Picek, Lejla Batina, Pieter Buzing, and Domagoj Jakobovic. "Fault Injection with a New Flavor: Memetic Algorithms Make a Difference". In: *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*. Ed. by Stefan Mangard and Axel Y. Poschmann. Vol. 9064. Lecture Notes in Computer Science. Springer, 2015, pp. 159–173. DOI: 10.1007/978-3-319-21476-4_11.

[232] Stjepan Picek, Lejla Batina, Domagoj Jakobovic, and Rafael Boix Carpi. "Evolving genetic algorithms for fault injection attacks". In: *37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014, Opatija, Croatia, May 26-30, 2014*. IEEE, 2014, pp. 1106–1111. DOI: 10.1109/MIPRO.2014.6859734.

[233] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele 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*. IEEE, 2017, pp. 4095–4102. DOI: 10.1109/IJCNN.2017.7966373.

[234] Damian Poddebniak, Juraj Somorovsky, Sebastian Schinzel, Manfred Lochter, and Paul Rösler. "Attacking Deterministic Signature Schemes Using Fault Attacks". In: *2018 IEEE European Symposium on Security and Privacy, EuroS&P 2018, London, United Kingdom, April 24-26, 2018*. IEEE, 2018, pp. 338–352. DOI: 10.1109/EuroSP.2018.00031.

[235] John M. Pollard. "Monte Carlo methods for index computation (mod p)". In: *Mathematics of Computation* 32.143 (1978). http://www.ams.org/journals/mcom/1978-32-143/S0025-5718-1978-0491431-9/S0025-5718-1978-0491431-9.pdf, pp. 918–924.

[236] Emmanuel Prouff and Matthieu Rivain. "Masking against Side-Channel Attacks: A Formal Security Proof". In: *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. Lecture Notes in Computer Science. Springer, 2013, pp. 142–159. DOI: 10.1007/978-3-642-38348-9_9.

[237] FIPS PUB. *Secure Hash Standard (SHS)*. Tech. rep. NIST, July 2015. DOI: 10.6028/NIST.FIPS.180-4.

[238] NIST FIPS PUB. *180-4," Secure Hash Standard (SHS)," March 2012*.

[239] Jean-Jacques Quisquater and David Samyde. "ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards". In: *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*. Ed. by Isabelle Attali and Thomas P. Jensen. Vol. 2140. Lecture Notes in Computer Science. Springer, 2001, pp. 200–210. DOI: 10.1007/3-540-45418-7_17.

[240] Christian Rechberger and Elisabeth Oswald. "Practical Template Attacks". In: *Information Security Applications, 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers*. Ed. by Chae Hoon Lim and Moti Yung. Vol. 3325. Lecture Notes in Computer Science. Springer, 2004, pp. 440–456. DOI: 10.1007/978-3-540-31815-6_35.

[241] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. "A Formal Study of Power Variability Issues and Side-Channel Attacks for Nanoscale Devices". In: *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*. Ed. by Kenneth G. Paterson. Vol. 6632. Lecture Notes in Computer Science. Springer, 2011, pp. 109–128. DOI: 10.1007/978-3-642-20465-4_8.

[242] Joost Renes, Craig Costello, and Lejla Batina. "Complete Addition Formulas for Prime Order Elliptic Curves". In: *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. Lecture Notes in Computer Science. Springer, 2016, pp. 403–428. DOI: 10.1007/978-3-662-49890-3_16.

[243] Oscar Reparaz, Josep Balasch, and Ingrid Verbauwhede. "Dude, is my code constant time?" In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*. Ed. by David Atienza and Giorgio Di Natale. IEEE, 2017, pp. 1697–1702. DOI: 10.23919/DATE.2017.7927267.

[244] Oscar Reparaz, Lauren De Meyer, Begül Bilgin, Victor Arribas, Svetla Nikova, Ventzislav Nikov, and Nigel P. Smart. "CAPA: The Spirit of Beaver Against Physical Attacks". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*. Ed. by Hovav Shacham and Alexandra Boldyreva. Vol. 10991. Lecture Notes in Computer Science. Springer, 2018, pp. 121–151. DOI: 10.1007/978-3-319-96884-1_5.

[245] Riscure. *Current Probe. Security Test Tool for Embedded Devices*. https://www.riscure.com/product/current-probe/ (accessed 2021-05-05). 2018.

[246] Riscure. *Side Channel Analysis Security Tools*. https://www.riscure.com/security-tools/inspector-sca/. 2021.

[247] Ronald L. Rivest. "The MD5 Message-Digest Algorithm". In: *RFC* 1321 (1992), pp. 1–21. DOI: 10.17487/RFC1321.

[248] Ronald L. Rivest. "The RC5 Encryption Algorithm". In: *Fast Software Encryption: Second International Workshop. Leuven, Belgium, 14-16 December 1994, Proceedings*. Ed. by Bart Preneel. Vol. 1008. Lecture Notes in Computer Science. Springer, 1994, pp. 86–96. DOI: 10.1007/3-540-60590-8_7.

[249] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Commun. ACM* 21.2 (1978), pp. 120–126. DOI: 10.1145/359340.359342.

[250] Thomas Roche, Victor Lomné, Camille Mutschler, and Laurent Imbert. "A Side Journey To Titan". In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 231–248.

[251] Niels Roelofs, Niels Samwel, Lejla Batina, and Joan Daemen. "Online Template Attack on ECDSA: - Extracting Keys via the Other Side". In: *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings*. Ed. by Abderrahmane Nitaj and Amr M. Youssef. Vol. 12174. Lecture Notes in Computer Science. Springer, 2020, pp. 323–336. DOI: 10.1007/978-3-030-51938-4_16.

[252] Yolan Romailler and Sylvain Pelissier. "Practical Fault Attack against the Ed25519 and EdDSA Signature Schemes". In: *2017 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2017, Taipei, Taiwan, September 25, 2017*. IEEE Computer Society, 2017, pp. 17–24. DOI: 10.1109/FDTC.2017.12.

[253] Niels Samwel and Lejla Batina. "Practical Fault Injection on Deterministic Signatures: The Case of EdDSA". In: *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*. Ed. by Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 10831. Lecture Notes in Computer Science. Springer, 2018, pp. 306–321. DOI: 10.1007/978-3-319-89339-6_17.

[254] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. "Breaking Ed25519 in WolfSSL". In: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 1–20. DOI: 10.1007/978-3-319-76953-0_1.

[255] Niels Samwel and Joan Daemen. "DPA on hardware implementations of Ascon and Keyak". In: *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*. ACM, 2017, pp. 415–424. DOI: 10.1145/3075564.3079067.

[256] Pascal Sasdrich, René Bock, and Amir Moradi. "Threshold Implementation in Software - Case Study of PRESENT". In: *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Springer, 2018, pp. 227–244. DOI: 10.1007/978-3-319-89641-0_13.

[257] Pascal Sasdrich and Tim Güneysu. "Cryptography for Next Generation TLS: Implementing the RFC 7748 Elliptic Curve448 Cryptosystem in Hardware". In: *Proceedings of the 54th Annual Design Automation Conference, DAC 2017, Austin, TX, USA, June 18-22, 2017*. ACM, 2017, 16:1–16:6. DOI: 10.1145/3061639.3062222.

[258] Pascal Sasdrich and Tim Güneysu. "Implementing Curve25519 for Side-Channel-Protected Elliptic Curve Cryptography". In: *ACM Trans. Reconfigurable Technol. Syst.* 9.1 (2015), 3:1–3:15. DOI: 10.1145/2700834.

[259] Alexander Schlösser, Dmitry Nedospasov, Juliane Krämer, Susanna Orlic, and Jean-Pierre Seifert. "Simple photonic emission analysis of AES". In: *J. Cryptogr. Eng.* 3.1 (2013), pp. 3–15. DOI: 10.1007/s13389-013-0053-7.

[260] Tobias Schneider and Amir Moradi. "Leakage Assessment Methodology - A Clear Roadmap for Side-Channel Evaluations". In: *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*. Ed. by Tim Güneysu and Helena Handschuh. Vol. 9293. Lecture Notes in Computer Science. Springer, 2015, pp. 495–513. DOI: 10.1007/978-3-662-48324-4_25.

[261] Tobias Schneider, Amir Moradi, and Tim Güneysu. "ParTI - Towards Combined Hardware Countermeasures Against Side-Channel and Fault-Injection Attacks". In: *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. Lecture Notes in Computer Science. Springer, 2016, pp. 302–332. DOI: 10.1007/978-3-662-53008-5_11.

[262] Claus-Peter Schnorr. "Efficient Identification and Signatures for Smart Cards". In: *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Springer, 1989, pp. 239–252. DOI: 10.1007/0-387-34805-0_22.

[263] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptol.* 4.3 (1991), pp. 161–174. DOI: 10.1007/BF00196725.

[264] Donald J Schuirmann. "A comparison of the two one-sided tests procedure and the power approach for assessing the equivalence of average bioavailability". In: *Journal of pharmacokinetics and biopharmaceutics* 15.6 (1987), pp. 657–680.

[265] Peter Schwabe and Ko Stoffelen. "All the AES You Need on Cortex-M3 and M4". In: *Selected Areas in Cryptography - SAC 2016 - 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*. Ed. by Roberto Avanzi and Howard M. Heys. Vol. 10532. Lecture Notes in Computer Science. Springer, 2016, pp. 180–194. DOI: 10.1007/978-3-319-69453-5_10.

[266] Nader Sehatbakhsh, Baki Berkay Yilmaz, Alenka G. Zajic, and Milos Prvulovic. "EMSim: A Microarchitecture-Level Simulation Tool for Modeling Electromagnetic Side-Channel Signals". In: *IEEE International Symposium on High Performance Computer Architecture, HPCA 2020, San Diego, CA, USA, February 22-26, 2020*. IEEE, 2020, pp. 71–85. DOI: 10.1109/HPCA47549.2020.00016.

[267] Hermann Seuschek, Johann Heyszl, and Fabrizio De Santis. "A Cautionary Note: Side-Channel Leakage Implications of Deterministic Signature Schemes". In: *Proceedings of the Third Workshop on Cryptography and Security in Computing Systems, CS2@HiPEAC, Prague, Czech Republic, January 20, 2016*. Ed. by Martin Palkovic, Giovanni Agosta, Alessandro Barenghi, Israel Koren, and Gerardo Pelosi. ACM, 2016, pp. 7–12. DOI: 10.1145/2858930.2858932.

[268] Madura A. Shelton, Lukasz Chmielewski, Niels Samwel, Markus Wagner, Lejla Batina, and Yuval Yarom. "Rosita++: Automatic Higher-Order Leakage Elimination from Cryptographic Code". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi. ACM, 2021, pp. 685–699. DOI: 10.1145/3460120.3485380.

[269] Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. "Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

[270] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*. IEEE Computer Society, 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[271] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. "Friet: An Authenticated Encryption Scheme with Built-in Fault Detection". In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*. Ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 581–611. DOI: 10.1007/978-3-030-45721-1_21.

[272] Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. *Towards Lightweight Cryptographic Primitives with Built-in Fault-Detection*. Cryptology ePrint Archive, Report 2018/729. https://eprint.iacr.org/2018/729. 2018.

[273] Sergei P. Skorobogatov and Ross J. Anderson. "Optical Fault Induction Attacks". In: *Cryptographic Hardware and Embedded Systems - CHES 2002*. Springer Berlin Heidelberg, 2003, pp. 2–12. DOI: 10.1007/3-540-36400-5_2.

[274] Nigel P. Smart. "The Discrete Logarithm Problem on Elliptic Curves of Trace One". In: *J. Cryptol.* 12.3 (1999), pp. 193–196. DOI: 10.1007/s001459900052.

[275] Youssef Souissi, Maxime Nassar, Sylvain Guilley, Jean-Luc Danger, and Florent Flament. "First Principal Components Analysis: A New Side Channel Distinguisher". In: *Information Security and Cryptology - ICISC 2010 - 13th International Conference, Seoul, Korea, December 1-3, 2010, Revised Selected Papers*. Ed. by Kyung Hyune Rhee and DaeHun Nyang. Vol. 6829. Lecture Notes in Computer Science. Springer, 2010, pp. 407–419. DOI: 10.1007/978-3-642-24209-0_27.

[276] François-Xavier Standaert. "How (Not) to Use Welch's T-Test in Side-Channel Security Evaluations". In: *Smart Card Research and Advanced Applications, 17th International Conference, CARDIS 2018, Montpellier, France, November 12-14, 2018, Revised Selected Papers*. Ed. by Begül Bilgin and Jean-Bernard Fischer. Vol. 11389. Lecture Notes in Computer Science. Springer, 2018, pp. 65–79. DOI: 10.1007/978-3-030-15462-2_5.

[277] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. "The World Is Not Enough: Another Look on Second-Order DPA". In: *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and*

Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings. Ed. by Masayuki Abe. Vol. 6477. Lecture Notes in Computer Science. Springer, 2010, pp. 112–129. DOI: 10.1007/978-3-642-17373-8_7.

[278] Data Encryption Standard. "NIST FIPS PUB 46-2". In: *US Department of Commerce* (1993).

[279] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. "The First Collision for Full SHA-1". In: *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*. Ed. by Jonathan Katz and Hovav Shacham. Vol. 10401. Lecture Notes in Computer Science. Springer, 2017, pp. 570–596. DOI: 10.1007/978-3-319-63688-7_19.

[280] STMicroelectronics. *Reference manual – STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced Arm-based 32-bit MCUs*. Tech. rep. RM0090 Rev 17. https://www.st.com/content/ccc/resource/technical/document/reference_manual/3d/6d/5a/66/b4/99/40/d4/DM00031020.pdf/files/DM00031020.pdf/jcr:content/translations/en.DM00031020.pdf. STMicroelectronics, 2018.

[281] Open Whisper Systems. "Signal Protocol". In: https://signal.org/ (accessed 2021-05-05).

[282] Mostafa Taha and Patrick Schaumont. "Side-Channel Analysis of MAC-Keccak". In: *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*. Institute of Electrical and Electronics Engineers (IEEE), June 2013. DOI: 10.1109/hst.2013.6581577.

[283] *The XEdDSA and VXEdDSA Signature Schemes*. Accessed September 9, 2017. 2017.

[284] *Things that use Ed25519*. Accessed September 29, 2017. 2017.

[285] *BearSSL*. https://bearssl.org/ (accessed 2021-05-05). 2018.

[286] Niek Timmers, Albert Spruyt, and Marc Witteman. "Controlling PC on ARM Using Fault Injection". In: *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*. IEEE Computer Society, 2016, pp. 25–35. DOI: 10.1109/FDTC.2016.18.

[287] Kris Tiri and Ingrid Verbauwhede. "A digital design flow for secure integrated circuits". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 25.7 (2006), pp. 1197–1208. DOI: 10.1109/TCAD.2005.855939.

[288] *The Noise Protocol Framework (revision 34)*. https://noiseprotocol.org/noise.pdf. 2018.

[289] Eran Tromer, Dag Arne Osvik, and Adi Shamir. "Efficient Cache Attacks on AES, and Countermeasures". In: *J. Cryptol.* 23.1 (2010), pp. 37–71. DOI: 10.1007/s00145-009-9049-y.

[290] Michael Tunstall and Gilbert Goodwill. "Applying TVLA to Public Key Cryptographic Algorithms". In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 513.

[291] Rajesh Velegalati, Robert Van Spyk, and Jasper van Woudenberg. "Electro Magnetic Fault Injection in Practice". In: *International Cryptographic Module Conference (ICMC)*. 2013.

[292] Nikita Veshchikov. "SILK: high level of abstraction leakage simulator for side channel analysis". In: *Proceedings of the 4th Program Protection and Reverse Engineering Workshop, PPREW@ACSAC 2014, New Orleans, LA, USA, December 9, 2014*. Ed. by Mila Dalla Preda and Jeffrey Todd McDonald. ACM, 2014, 3:1–3:11. DOI: 10.1145/2689702.2689706.

[293] Nikita Veshchikov and Sylvain Guilley. "Use of Simulators for Side-Channel Analysis". In: *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 2017, pp. 51–59. DOI: 10.1109/EuroSP.2017.31.

[294] Colin D. Walter. "Sliding Windows Succumbs to Big Mac Attack". In: *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science Generators. Springer, 2001, pp. 286–299. DOI: 10.1007/3-540-44709-1_24.

[295] Chao Wang and Patrick Schaumont. "Security by compilation: an automated approach to comprehensive side-channel resistance". In: *ACM SIGLOG News* 4.2 (2017), pp. 76–89.

[296] Jingbo Wang, Chungha Sung, and Chao Wang. "Mitigating power side channels during compilation". In: *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, Tallinn, Estonia, August 26-30, 2019*. Ed. by Marlon Dumas, Dietmar Pfahl, Sven Apel, and Alessandra Russo. ACM, 2019, pp. 590–601. DOI: 10.1145/3338906.3338913.

[297] Xiaoyun Wang and Hongbo Yu. "How to Break MD5 and Other Hash Functions". In: *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*. Ed. by Ronald Cramer. Vol. 3494. Lecture Notes in Computer Science. Springer, 2005, pp. 19–35. DOI: 10.1007/11426639_2.

[298] Leo Weissbart, Lukasz Chmielewski, Stjepan Picek, and Lejla Batina. "Systematic Side-Channel Analysis of Curve25519 with Machine Learning". In: *J. Hardw. Syst. Secur.* 4.4 (2020), pp. 314–328. DOI: 10.1007/s41635-020-00106-w.

[299] Leo Weissbart, Stjepan Picek, and Lejla Batina. "One Trace Is All It Takes: Machine Learning-Based Side-Channel Attack on EdDSA". In: *Security, Privacy, and Applied Cryptography Engineering - 9th International Conference, SPACE 2019, Gandhinagar, India, December 3-7, 2019, Proceedings*. Ed. by Shivam Bhasin, Avi Mendelson, and Mridul Nandi. Vol. 11947. Lecture Notes in Computer Science. Springer, 2019, pp. 86–105. DOI: 10.1007/978-3-030-35869-3_8.

[300] Bernard L Welch. "The Generalization of 'Student's' Problem when Several Different Population Varlances are Involved". In: *Biometrika* 34.1/2 (1947), pp. 28–35.

[301] BP Welford. "Note on a method for calculating corrected sums of squares and products". In: *Technometrics* 4.3 (1962), pp. 419–420.

[302] Carolyn Whitnall and Elisabeth Oswald. "A Critical Analysis of ISO 17825 ('Testing Methods for the Mitigation of Non-invasive Attack Classes Against Cryptographic Modules')". In: *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*. Ed. by Steven D. Galbraith and Shiho Moriai. Vol. 11923. Lecture Notes in Computer Science. Springer, 2019, pp. 256–284. DOI: 10.1007/978-3-030-34618-8_9.

[303] Marc Witteman. *Secure Application Programming in the presence of Side Channel Attacks*. Tech. rep. https://www.riscure.com/uploads/2018/11/201708_Riscure_Whitepaper_Side_Channel_Patterns.pdf. Riscure, 2018.

[304] Marc F. Witteman, Jasper G. J. van Woudenberg, and Federico Menarini. "Defeating RSA Multiply-Always and Message Blinding Countermeasures". In: *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*. Ed. by Aggelos Kiayias. Vol. 6558. Lecture Notes in Computer Science. Springer, 2011, pp. 77–88. DOI: 10.1007/978-3-642-19074-2_6.

[305] *The wolSSL crypto library as accessed at april 24th, 2021)*. https://github.com/wolfSSL/wolfssl/tree/master/wolfcrypt. 2021.

[306] Meng Wu, Shengjian Guo, Patrick Schaumont, and Chao Wang. "Eliminating timing side-channel leaks using program repair". In: *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, Amsterdam, The Netherlands, July 16-21, 2018*. Ed. by Frank Tip and Eric Bodden. ACM, 2018, pp. 15–26. DOI: 10.1145/3213846.3213851.

[307] Yan Yan and Elisabeth Oswald. "Examining the practical side channel resilience of ARX-boxes". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers, CF 2019, Alghero, Italy, April 30 - May 2, 2019*. Ed. by Francesca Palumbo, Michela Becchi, Martin Schulz, and Kento Sato. ACM, 2019, pp. 373–379. DOI: 10.1145/3310273.3323399.

[308] Yuan Yao, Mo Yang, Conor Patrick, Bilgiday Yuce, and Patrick Schaumont. "Fault-assisted side-channel analysis of masked implementations". In: *2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018*. IEEE Computer Society, 2018, pp. 57–64. DOI: 10.1109/HST.2018.8383891.

[309] Sung-Ming Yen and Marc Joye. "Checking Before Output May Not Be Enough Against Fault-Based Cryptanalysis". In: *IEEE Trans. Computers* 49.9 (2000), pp. 967–970. DOI: 10.1109/12.869328.

[310] Yuanyuan Zhou and François-Xavier Standaert. *Simplified Single-Trace Side-Channel Attacks on Elliptic Curve Scalar Multiplication using Fully Convolutional Networks*. Proceedings of the 40th WIC Symposium on Information Theory in the Benelux. `https://perso.uclouvain.be/fstandae/PUBLIS/219.pdf`. 2019.

[311] Michael Zohner, Michael Kasper, and Marc Stöttinger. "Butterfly-Attack on Skein's Modular Addition". In: *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May 3-4, 2012. Proceedings*. Ed. by Werner Schindler and Sorin A. Huss. Vol. 7275. Lecture Notes in Computer Science. Springer, 2012, pp. 215–230. DOI: `10.1007/978-3-642-29912-4\_16`.

Summary

Cryptographic implementations that run on devices can leak information through unintended channels. When these devices are out in the field, an adversary can gather this information from these channels and try to recover secret information. Not only is it possible to passively gather information, an adversary with access to a device might also introduce glitches into the environment that can alter the behavior of a device. Doing this carefully might alter the outcome of a cryptographic algorithm which may leak secret information. Protecting cryptographic implementations against these attacks is often costly. When taking into account that devices out in the field are often very constrained in resources and security measures are often absent, it becomes clear that protecting against these attacks is not a trivial task.

This thesis investigates the side-channel security of several cryptographic implementations. We investigate both hardware and software implementations of symmetric and public key cryptographic implementations. Furthermore, we investigate leakage simulation and how this can be applied in automatic leakage evaluation.

First, we investigate side-channel attacks and countermeasures for the cryptographic permutations Keyak and Ascon. And we evaluate the fault resistance of the cryptographic permutation FRIET with built-in countermeasures against faults.

Next, we investigate physical attacks on the digital signature scheme Ed25519. We present a side-channel attack on the hash function SHA-2 that is used to derive an ephemeral key from a long-term secret, where we aim to recover the long-term secret. Additionally, we also look into a differential fault attack to recover the private key. The chapter concludes with a countermeasure against both attacks.

Third, we present two optimized side-channel protected implementations in software of Curve25519. One implementation offers side-channel protection for use with ephemeral keys, the other implementation offers more side-channel protection for using with static keys. Additionally, we evaluate the implementations and demonstrate the effect on the side-channel leakage.

Fourth, we investigate fault injection in combination with genetic algorithms to optimize the massive search space that is typical with fault injection attacks. We apply the strategy to attack the hash function SHA-3 and show the reduction of the number of injected faults that are required for a successful attack.

Finally, we present automated tools that are able to emulate the power consumption given a masked implementation in software. The tool uses simulated power consumption to detect side-channel leakage and automatically fix it in the compiled code. The effectiveness of the tool is evaluated using real traces, collected using the same implementations.

Samenvatting

Cryptografische implementaties die op apparaten worden uitgevoerd, kunnen informatie lekken via onbedoelde kanalen. Wanneer deze apparaten in gebruik zijn, kan een aanvaller deze data verzamelen en geheime informatie proberen te achterhalen. Het is niet alleen mogelijk om passief informatie te verzamelen. Een aanvaller met toegang tot een apparaat kan ook storingen in de omgeving introduceren die het gedrag van een apparaat kunnen veranderen. Als dit zorgvuldig gebeurt, kan dit de uitkomst van een cryptografisch algoritme veranderen waardoor geheime informatie kan lekken. Het beschermen van cryptografische implementaties tegen deze aanvallen is vaak erg kostbaar. Bovendien is de apparatuur waar die implementaties op draaien vaak beperkt in rekenkracht wat het moeilijk maakt om deze apparaten tegen aanvallen te beschermen.

Dit proefschrift onderzoekt de side-channel beveiliging van verschillende cryptografische implementaties. We onderzoeken zowel hardware- als software-implementaties van symmetrische en publieke sleutel cryptografische implementaties. Verder onderzoeken we hoe informatie lekt doormiddel van simulatie en hoe dit kan worden toegepast in een automatische evaluatie.

Eerst onderzoeken we side-channel aanvallen en tegenmaatregelen voor de cryptografische permutaties Keyak en Ascon. Verder evalueren we de foutbestendigheid van de cryptografische permutatie FRIET met ingebouwde maatregelen tegen van buitenaf geïntroduceerde fouten.

Vervolgens onderzoeken we fysieke aanvallen op het digitale handtekening algoritme Ed25519. We introduceren een aanval op de hashfunctie SHA-2 die wordt gebruikt om een tijdelijke sleutel af te leiden van een langetermijngeheim, waarbij we ernaar streven het langetermijngeheim te reconstrueren. Daarnaast onderzoeken we ook een differentiële foutaanval om de privésleutel te achterhalen. Het hoofdstuk wordt afgesloten met een tegenmaatregel tegen beide aanvallen.

Ten derde presenteren we twee geoptimaliseerde beveiligde software implementaties van Curve25519. De ene implementatie biedt beveiliging voor gebruik met tijdelijke sleutels, de andere implementatie biedt meer beveiliging voor gebruik met statische sleutels. Daarnaast evalueren we de implementaties en demonstreren we het effect op de lekkage van het stroomverbruik.

Ten vierde onderzoeken we foutinjectie in combinatie met genetische algoritmen om de enorme zoekruimte die typisch is voor foutinjectieaanvallen te optimaliseren. We passen de strategie toe om de hashfunctie SHA-3 aan te vallen en tonen de vermindering van het aantal geïnjecteerde fouten dat nodig is voor een succesvolle aanval.

Ten slotte presenteren we geautomatiseerde tools die in staat zijn om het stroomver-

bruik te simuleren bij gemaskeerde implementaties in software. De tool gebruikt het gesimuleerde stroomverbruik om lekkage te detecteren en deze automatisch in de gecompileerde code op te lossen. De effectiviteit van de tool wordt geëvalueerd door het stroomverbuik te meten van fysieke apparaten waar diezelfde implementaties op draaien en dat resultaat te vergelijken.

List of Publications

Conference/Journal proceedings

- Niels Samwel and Joan Daemen. "DPA on hardware implementations of Ascon and Keyak". In: *Proceedings of the Computing Frontiers Conference, CF'17, Siena, Italy, May 15-17, 2017*. ACM, 2017, pp. 415–424. DOI: 10.1145/3075564.3079067

- Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. "Breaking Ed25519 in WolfSSL". in: *Topics in Cryptology - CT-RSA 2018 - The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*. Ed. by Nigel P. Smart. Vol. 10808. Lecture Notes in Computer Science. Springer, 2018, pp. 1–20. DOI: 10.1007/978-3-319-76953-0_1

- Niels Samwel and Lejla Batina. "Practical Fault Injection on Deterministic Signatures: The Case of EdDSA". in: *Progress in Cryptology - AFRICACRYPT 2018 - 10th International Conference on Cryptology in Africa, Marrakesh, Morocco, May 7-9, 2018, Proceedings*. Ed. by Antoine Joux, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Vol. 10831. Lecture Notes in Computer Science. Springer, 2018, pp. 306–321. DOI: 10.1007/978-3-319-89339-6_17

- Antun Maldini, Niels Samwel, Stjepan Picek, and Lejla Batina. "Genetic Algorithm-Based Electromagnetic Fault Injection". In: *2018 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2018, Amsterdam, The Netherlands, September 13, 2018*. IEEE Computer Society, 2018, pp. 35–42. DOI: 10.1109/FDTC.2018.00014

- Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. "Friet: An Authenticated Encryption Scheme with Built-in Fault Detection". In: *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*. ed. by Anne Canteaut and Yuval Ishai. Vol. 12105. Lecture Notes in Computer Science. Springer, 2020, pp. 581–611. DOI: 10.1007/978-3-030-45721-1_21

- Niels Roelofs, Niels Samwel, Lejla Batina, and Joan Daemen. "Online Template Attack on ECDSA: - Extracting Keys via the Other Side". In: *Progress in Cryptology - AFRICACRYPT 2020 - 12th International Conference on Cryptology*

in Africa, Cairo, Egypt, July 20-22, 2020, Proceedings. Ed. by Abderrahmane Nitaj and Amr M. Youssef. Vol. 12174. Lecture Notes in Computer Science. Springer, 2020, pp. 323–336. DOI: 10.1007/978-3-030-51938-4_16

- Madura A. Shelton, Niels Samwel, Lejla Batina, Francesco Regazzoni, Markus Wagner, and Yuval Yarom. "Rosita: Towards Automatic Elimination of Power-Analysis Leakage in Ciphers". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021

- Zhuoran Liu, Niels Samwel, Leo Weissbart, Zhengyu Zhao, Dirk Lauret, Lejla Batina, and Martha A. Larson. "Screen Gleaning: A Screen Reading TEMPEST Attack on Mobile Devices Exploiting an Electromagnetic Side Channel". In: *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021

- Monjur Alam, Baki Berkay Yilmaz, Frank Werner, Niels Samwel, Alenka G. Zajic, Daniel Genkin, Yuval Yarom, and Milos Prvulovic. "Nonce@Once: A Single-Trace EM Side Channel Attack on Several Constant-Time Elliptic Curve Implementations in Mobile Platforms". In: *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*. IEEE, 2021, pp. 507–522. DOI: 10.1109/EuroSP51992.2021.00041

- Madura A. Shelton, Lukasz Chmielewski, Niels Samwel, Markus Wagner, Lejla Batina, and Yuval Yarom. "Rosita++: Automatic Higher-Order Leakage Elimination from Cryptographic Code". In: *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*. Ed. by Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi. ACM, 2021, pp. 685–699. DOI: 10.1145/3460120.3485380

Book Chapter

- Antun Maldini, Niels Samwel, Stjepan Picek, and Lejla Batina. "Optimizing Electromagnetic Fault Injection with Genetic Algorithms". In: *Automated Methods in Cryptographic Fault Analysis*. Springer, 2019, pp. 281–300

Preprints

- Thierry Simon, Lejla Batina, Joan Daemen, Vincent Grosso, Pedro Maat Costa Massolino, Kostas Papagiannopoulos, Francesco Regazzoni, and Niels Samwel. *Towards Lightweight Cryptographic Primitives with Built-in Fault-Detection*. Cryptology ePrint Archive, Report 2018/729. https://eprint.iacr.org/2018/729. 2018

- Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. *SCA-secure ECC in software – mission impossible?* Cryptology ePrint Archive, Report 2021/1003. https://ia.cr/2021/1003. 2021

Curriculum Vitae

August 2022 –

SCA Analyst / Infrastructure Engineer
PQShield, The Netherlands

July 2021 – July 2022

Embedded Security Engineer
Intrinsic ID, The Netherlands

July 2019 – December 2019

Intern
Google, United States

October 2016 – June 2021

PhD Candidate
Digital Security Group (DiS)
Radboud University, The Netherlands

September 2014 – August 2016

Master of Science
Computer Science
Radboud University, The Netherlands

September 2011 – August 2014

Bachelor of Science
Computer Science
Leiden University, The Netherlands