EPFL

Thèse n° 9214

Side-channel analysis of isogeny-based key encapsulation mechanisms and hash-based digital signatures

Présentée le 24 janvier 2024

Faculté informatique et communications
Laboratoire de sécurité et de cryptographie
Programme doctoral en informatique et communications

pour l'obtention du grade de Docteur ès Sciences

par

Aymeric GENET

Acceptée sur proposition du jury

Prof. O. N. A. Svensson, président du jury
Prof. S. Vaudenay, Prof. A. Lenstra, directeurs de thèse
Dr J.-Ph. Aumasson, rapporteur
Dr C. Costello, rapporteur
Dr M. Stojilović, rapporteuse

École polytechnique fédérale de Lausanne

2024

What's a man to do
Lost in a world
Without you

Acknowledgements

Hiding secrets is an art that transcends the domain of cryptography.

We're now at the end of my PhD, the time has come for me to acknowledge the people who, like no others, supported me in this crazy journey. First and foremost, without wasting time on strangers, I would like to thank my thesis director, Prof. Arjen Lenstra, who was daring enough to believe in me to write a thesis. Arjen, thank you so much for allowing me to always do what I love all the time and for your never-ending support in every moment of my doctoral studies. You are the one who constantly guided me on the right path, and I owe you everything that I know. I consider myself very fortunate to have had you as a director; I will forever cherish the Thursday lunches and will never forget your help when I had to deal with the administrative rules of the school that got in the way. Next, to Prof. Serge Vaudenay, my thesis co-director and one of my greatest sources of inspiration in cryptography, I would like to express so much gratitude for taking over my supervision. Serge, you are the reason why I wanted to do a PhD in cryptography after you introduced me to this domain. Despite our disagreements, I have learned a lot under your direction, and for that, I will be forever grateful. Thank you.

A special thought goes to all my colleagues from LACAL who made my life as a PhD student full of good memories. Thank you, Benjamin, Dušan, Marguerite, and Novak, your academic commitment's a real inspiration. Then, I want to thank Thorsten for the insights you bring to what I do, as well as Monique, our secretary, for the good talks and the favors you granted me. I'm also indebted to the people I've met during my time in LASEC—home of the most security-thinking persons. Cheers to Abdullah, Andi, Bénédikt, Daniel, Khashayar, Laurane, Loïs, and of course our secretary Martine, for the time we spent together and your help in hard times. You were all the most amazing labmates that anyone could've asked for. Obviously, the lineup wouldn't be complete without the post-docs: Boris, Hailun, Ritam, and Subhadeep; I won't forget your help through hardship. Finally, I would like to acknowledge Kudelski Group for making this thesis a possible joint work between academia and industry. Thank you, Benoît and Joël from management, for allowing such a project. Thank you, Pascal and Stéphane, for tackling any possible complication. And thank you, Adel, Gerrit, Hervé, Nicolas, Pablo, Roman, the other Roman, Sylvain, and Thierry for all your support. At last, thank you Karine for providing a guy like me with your infinite wisdom in cryptography. I owe my thesis to all of you guys.

Acknowledgements

I would also like to acknowledge the work of my collaborators, without whom my thesis just wouldn't be as good. To Denis, Élise, Juliane, Johannes, Luca, Matthias, Nadia, and Simon, I wanna express my sincerest gratitude for the excellent teamwork. I then need, of course, to tell my students a few kind words: Léa, Lucas, Kopiga, Marc, Mathilde, Othman, and Stache; you were the most diligent in your studies. I enjoyed every moment with you and feel proud of how each of your projects turned out. In particular, I want to personally thank you, Natacha. I'm grateful of our collaboration which sparked an entire part of my PhD. All in all, I'm feeling appreciative of everyone with whom I worked. It was an honor collaborating with you.

Gotta also thank the family. Thank you, Mom, Dad, and Henry, for going out of your way to make everything as easy as possible. At last, I want to finish with Aurore, my best friend. Aurore, you were there for me from the start, despite the distance and the fact that you do not always understand everything I tell you. You are gold to my eyes. I do not deserve a friend like you.

Lausanne, September 19, 2023 *M.L.*

Abstract

Current cryptographic solutions will become obsolete with the arrival of large-scale universal quantum computers. To address this threat, the National Institute of Standards and Technology supervises a post-quantum standardization process which involves evaluating candidates in a round-based competition-like format. Among these candidates, two notable schemes were submitted: the isogeny-based key encapsulation mechanism SIKE, and the hash-based digital signature scheme SPHINCS$^+$. While considerable theoretical cryptanalysis has been dedicated to these candidates, relatively little attention has been given to the potential risks associated with their implementation. This thesis aims to address this gap by investigating the vulnerabilities of SIKE and SPHINCS$^+$ to side-channel analysis.

The first part of the thesis focuses on SIKE in relation to power analysis, where we describe three side-channel attacks. The first attack involves a horizontal differential power analysis of the three-point ladder used in the scheme, which recovers the secret scalar used to generate the secret isogeny within a single trace of power consumption through an extend-and-prune method. The second attack applies clustering power analysis to identify all the bits of the secret scalar from a single trace of the three-point ladder by exploiting the leakages of a procedure that swaps two elliptic curve points depending on the difference between two bits of the secret scalar. Lastly, the final attack details a zero-value point attack on the secret isogeny computation, which works by providing a malicious ciphertext that causes many operations, including the j-invariant calculation, to have a zero result based on the value of a single bit of the secret scalar. All attacks were experimentally verified with power traces collected from an STM32F3 running the SIKE implementation recommended for Cortex-M4. Note that while the work in this thesis was being conducted, an independent attack on SIKE resulted in the total security break of the scheme. Our work remains of value despite this attack.

The second part of the thesis focuses on the side-channel analysis of the SPHINCS family. First, we target the seminal SPHINCS-256 scheme through a differential power analysis of its pseudorandom number generator based on BLAKE-256, and describe an attack that recovers at least one 32-bit chunk of the signing key. We successfully conducted an experimental verification of this attack using 10,000 electromagnetic traces collected from a SAM3X8E running a custom Cortex-M3 SPHINCS-256 implementation. Secondly, we adapt an original fault attack on the SPHINCS-256 scheme to SPHINCS$^+$ and analyze its impact on the security of the scheme. This analysis demonstrates that, with high probability, the security of SPHINCS$^+$ significantly drops when a single random bit flip occurs anywhere in the signing process, and

that the countermeasures based on caching the intermediate W-OTS$^+$s offer a marginally greater protection against unintentional faults. Experimental validation of these results was conducted on an STM32F4 running the reference implementation of SPHINCS$^+$ adapted to the Cortex-M4 architecture.

The thesis concludes that the current state of SIKE and SPHINCS$^+$ are vulnerable to side-channel analysis and proposes directions to develop effective countermeasures.

Keywords: Post-Quantum Cryptography, SIKE, SPHINCS-256, SPHINCS$^+$, Differential Power Analysis, Clustering Power Analysis, Zero-Value Point Attack, Fault Attack, Countermeasures.

Résumé

Les solutions cryptographiques actuelles deviendront obsolètes avec l'arrivée des ordinateurs quantiques universels de grande envergure. Pour faire face à cette menace, le National Institute of Standards and Technology[1] supervise un concours de standardisation post-quantique, qui consiste à évaluer des candidats selon un format compétitif à plusieurs rondes. Parmi ces candidats, deux schémas notables ont été proposés : le mécanisme d'encapsulation de clé à base d'isogénies SIKE et le schéma de signature numérique à base de hachage SPHINCS$^+$. Bien que ces candidats aient fait l'objet d'une cryptanalyse théorique considérable, relativement peu d'attention a été accordée aux risques potentiels liés à la mise en œuvre de ces schémas. Cette thèse vise à combler cette lacune en examinant les vulnérabilités de SIKE et SPHINCS$^+$ à l'analyse par canaux cachés.

La première partie de la thèse se concentre sur SIKE en lien avec de l'analyse de consommation, où nous décrivons trois attaques par canaux cachés. La première attaque implique une analyse de consommation différentielle horizontale de l'échelle à trois points utilisée dans le schéma qui permet de récupérer le scalaire secret utilisé pour générer l'isogénie secrète dans une seule trace de consommation grâce à une méthode d'extension et d'élagage. La deuxième attaque applique une analyse de consommation en cluster pour identifier tous les bits du scalaire secret à partir d'une seule trace de l'échelle à trois points, exploitant les fuites d'une procédure qui échange deux points de courbe elliptique en fonction de la différence entre deux bits du scalaire secret. Enfin, la dernière attaque détaille une attaque par point de valeur nulle sur le calcul de l'isogénie secrète, qui fonctionne en fournissant un texte chiffré malveillant qui provoque de nombreuses opérations, y compris le calcul du j-invariant, à aboutir à un résultat nul en fonction de la valeur d'un seul bit du scalaire secret. Toutes les attaques ont été vérifiées expérimentalement avec des traces de consommation récoltées à partir d'un STM32F3 exécutant l'implémentation recommandée de SIKE pour Cortex-M4. Notez que pendant la réalisation de ce travail de thèse, une attaque indépendante sur SIKE a abouti à la rupture totale de la sécurité du schéma. Notre travail reste néanmoins pertinent malgré cette attaque.

La deuxième partie de la thèse se concentre sur l'analyse par canaux cachés de la famille SPHINCS. Tout d'abord, nous ciblons le schéma SPHINCS-256 grâce à une analyse de consommation différentielle de son générateur de nombres pseudo-aléatoires basé sur BLAKE-256 et décrivons une attaque qui permet de récupérer au moins un fragment de 32 bits de la

[1] En français : « Institut national des normes et de la technologie. »

clé de signature. Nous avons réussi à vérifier expérimentalement cette attaque en utilisant 10'000 traces électromagnétiques récoltées à partir d'un SAM3X8E exécutant une implémentation personnalisée de SPHINCS-256 sur Cortex-M3. Ensuite, nous adaptons une attaque originale par faute sur le schéma SPHINCS-256 à SPHINCS$^+$ et analysons son impact sur la sécurité du schéma. Cette analyse démontre que, avec une probabilité élevée, la sécurité de SPHINCS$^+$ diminue considérablement lorsqu'un seul bit aléatoire est inversé n'importe où dans le processus de signature et que les contre-mesures basées sur la mise en tampon des W-OTS$^+$s intermédiaires n'offrent qu'une protection marginalement supérieure contre les fautes involontaires. La validation expérimentale de ces résultats a été réalisée sur un STM32F4 exécutant l'implémentation de référence de SPHINCS$^+$ adaptée à l'architecture Cortex-M4.

La thèse conclut que l'état actuel de SIKE et SPHINCS$^+$ est vulnérable à l'analyse par canaux cachés et propose des chemins pour développer des contre-mesures efficaces.

Mots-clés : Cryptographie Post-Quantique, SIKE, SPHINCS-256, SPHINCS$^+$, Analyse De Consommation Différentielle, Analyse De Consommation En Cluster, Attaque Par Point De Valeur Nulle, Attaque Par Faute, Contre-mesures.

Contents

List of Figures

List of Tables

1 Introduction

Cryptography is the pluridisciplinary study of techniques that aim to achieve various security objectives. These objectives notably include *confidentiality* to prevent unauthorized access to sensitive information, *integrity* to ensure that data has not been altered, *authenticity* to verify the rightful originating sender of data, and *non-repudiation* to prevent that sender from denying the issuance of a message. Such notions are usually accomplished with the use of cryptographic *primitives* (i.e., operations) which constitute the building blocks of larger *cryptosystems* (also called cryptographic *schemes*) used by modern applications to operate securely. For example, a secure digital communication between two parties relies on cryptographic primitives to ensure that the messages exchanged remain confidential, maintain their integrity, are authenticated, and sometimes support non-repudiation.

There are two important subdomains of cryptography that study cryptographic primitives:

- *Secret-key cryptography* studies the primitives that involve a unique (secret) key.
- *Public-key cryptography* studies the primitives that involve two keys of different types:
  - the *public key*, supposedly available at all time to everyone, and
  - the *private key*, confidentially held by a single party.

A *cipher* is a typical example of a cryptographic primitive that can be secret- or public-key. Generally speaking, a cipher describes a technique that protects the confidentiality of a message by turning it into an unintelligible ciphertext through an *encryption* process using an *encryption key*. The resulting ciphertext can only be turned back into its original form with a process known as *decryption* using the corresponding *decryption key*. The cipher is qualified as *secret-key* if the encryption key is the same as the decryption key, and *public-key* if the two keys are distinct (in which case, anyone can encrypt messages that only the holder of the private key can decrypt).

Although secret-key cryptography is the de-facto standard to secure data due to its superior performance, secret-key primitives require managing a key across authorized parties. Take for instance a two-party secure communication in which every message exchanged is encrypted

with a secret-key cipher. In order for the two parties to encrypt and decrypt messages, the two parties need to share the same key which must be kept secret from everyone else. Public-key cryptography overcomes this limitation by enabling the two communicating parties to securely exchange a common secret key through the use of public-key encryption. In this case, one party uses the other party's public key to encrypt a secret key that will then be used with a secret-key cipher for the remainder of the communication. This process is known as *key encapsulation* which, along with the secret-key cipher used, results in a cryptosystem that allows an efficient and confidential communication between any two parties.

In addition to complementing secret-key cryptography, public-key cryptography enables the design of many unique cryptosystems. In particular, public-key cryptography allows the design of *digital signature schemes* that can establish the rightful issuer of a message; thus ensuring authenticity, integrity, and non-repudiation all at once. In such schemes, a signing party digitally signs a message by using their own private key (referred to as the *signing key* in this context) to produce a cryptographic value called *digital signature* that is mathematically associated to the message. The signature can be verified to correspond to the issued message using the signing party's public key through a verification procedure.

As of the writing of this thesis, the most widely used public-key solutions for implementing key encapsulation mechanisms and digital signature schemes are the Rivest–Shamir–Adleman (RSA) algorithm and algorithms based on the Diffie–Hellman (DH) protocol. These solutions are built upon the concept of a *trapdoor permutation*, which is a function where outputs are easy to compute given their inputs, but inputs are difficult to recover from their corresponding outputs unless a certain piece of information (called the *trapdoor*) is known. Such functions are based on mathematical problems that are believed to be computationally difficult to solve. In particular, the trapdoor permutation of RSA relies on the difficulty of factoring large numbers, while the one in DH-based algorithms relies on a hard discrete logarithm problem[1], such as the elliptic curve discrete logarithm problem. The computational intractability of these problems ensures that the data protected by these solutions remains secure.

While cryptosystems based on RSA and DH have been used to protect real-world data for decades, Peter W. Shor has conceived a polynomial-time algorithm that could effectively defeat both these solutions in [Sho94]. The catch is that Shor's algorithm can only be executed on quantum computers; a new type of machine which operates on bits and gates based on the principles of quantum mechanics. Therefore, in order to use Shor's algorithm to break the present instantiations of RSA and DH-based algorithms, a big enough universal quantum computer (i.e., a quantum computer able to accurately perform arbitrary quantum operations with a significant number of quantum bits to represent the parameters involved in the instantiations) needs to be developed. Although a few thousands logical quantum bits would theoretically be sufficient to break, e.g., RSA-1024, the physical implementation of a quantum computer with such a large number quantum bits is challenging, as external disturbances

[1] The discrete logarithm problem in a group G consists of determining an integer $m \geq 1$ such that $x^m = y$, given $x, y \in G$ such that y is in the subgroup generated by x (see [Sil86, XI.4] for the use case with elliptic curves).

introduce errors in the quantum bits, affecting their accuracy. As a result, additional quantum bits need to be implemented to correct the errors caused by environmental perturbations, resulting in the actual requirement of millions of physical quantum bits to achieve the thousands effective quantum bits initially required (see [GE21] for details). At the time of this thesis however, only a few hundreds[2] quantum bits have ever been successfully stabilized with high fidelity. Still, given the rapid growth of quantum computing in the recent years, Shor's algorithm poses a significant threat to current public-key solutions which needs to be addressed.

Post-quantum cryptographic algorithms. Since the advent of large quantum computers will make the current public-key cryptosystems obsolete, replacements resistant against quantum attacks need to be developed. This field of study, called *post-quantum*[3] *cryptography*, aims to complement current cryptographic schemes with solutions that can withstand both classical and quantum attacks. In particular, post-quantum cryptography aims to develop public-key solutions that are not affected by known quantum attacks, such as Shor's algorithm.

To anticipate the eventual emergence of quantum computers, the U.S. National Institute of Standards and Technology (NIST) initiated and supervises a project that seeks to standardize one or more post-quantum public-key cryptographic schemes. For this purpose, NIST publicly solicited post-quantum solutions for key encapsulation mechanisms and digital signature schemes with the aim of identifying potential future standards. The submissions are evaluated in a round-based competition-like process so that NIST could select the solutions that are considered the most appropriate.

In 2017, when the process started, 64 eligible algorithms were received, including 45 key encapsulation mechanisms and 19 digital signature schemes. Among the received candidates, five main families of cryptosystems could be identified:

- Lattice-based cryptosystems
- Code-based cryptosystems
- Hash-based cryptosystems
- Multivariate-based cryptosystems
- Isogeny-based cryptosystems

Five years later, after three rounds of meticulous analysis, four algorithms were selected for standardization, while four others advanced to a fourth round of evaluation, as listed in Table 1.1. Out of the four algorithms selected, three belong to the family of lattice-based cryptosystems (namely, the CRYSTALS suite, consisting of KYBER and DILITHIUM, and FALCON), while the fourth one (SPHINCS$^+$) is a hash-based algorithm. This selection demonstrates the

[2] The current record for the most stabilized quantum bits is held by IBM with their Osprey model that is claimed to feature 433 physical quantum bits (see [IBM22]).

[3] Contrary to what their name could imply, post-quantum cryptographic algorithms are *not* quantum algorithms. They are conventional algorithms that are expected to run on regular (*non*-quantum) computers.

general confidence in lattice-based cryptosystems due to their long history of cryptanalysis. Also, compared to the other families, lattice-based cryptosystems offered the best cost and performance, which encouraged their selection. Among the submissions that advanced to round 4, three candidates belong to the family of code-based cryptosystems (BIKE, Classic McEliece, and HQC), while the last one (SIKE) is an isogeny-based algorithm. These candidates (except for SIKE, which will be discussed later) are still being scrutinized to complement KYBER in the list of selected algorithms as alternative key encapsulation mechanisms that are not based on lattices. As NIST was not satisfied with the non-lattice-based digital signature schemes that were submitted to their process besides SPHINCS$^+$, NIST intends to hold an additional standardization process for post-quantum digital signatures only (see [NIS22]).

Algorithms	Selected	Round-4
Key encapsulation mechanisms	• CRYSTALS-KYBER [Sch+22b]	• BIKE [Ara+22] • Classic McEliece [Alb+22] • HQC [Agu+22] • SIKE [Jao+22]
Digital signature schemes	• CRYSTALS-DILITHIUM [Lyu+22] • FALCON [Pre+22] • SPHINCS$^+$ [Hül+22]	

Table 1.1: Algorithms that passed the third round of the NIST post-quantum project.

The focus of the present thesis revolves around the two outsiders among the selected algorithms and the round-4 submissions; namely SIKE and SPHINCS$^+$.

Supersingular Isogeny Key Encapsulation (SIKE). SIKE is a post-quantum key encapsulation mechanism based on the difficulty of finding the isogeny (i.e., a particular type of mapping) between two seemingly-random elliptic curves; a problem that is believed to be computationally hard to solve for both classical and quantum computers. The scheme works by choosing as the private key a point that generates a confidential isogeny which is then applied to a publicly known elliptic curve to generate the public key. A secret key is encapsulated by the function composition of its corresponding secret isogeny on a party's public key.

SIKE has been submitted to the NIST post-quantum project in 2016 as an improvement over the Supersingular Isogeny Diffie–Hellman[4] (SIDH) protocol which enables two parties to agree on a shared secret via a non-confidential communication channel. The advantages of SIKE over its other post-quantum counterparts are its relatively small keys and ciphertexts, its unique security assumption, and its reliance on well-established elliptic curve procedures. On the other hand, SIKE is relatively slow compared to other post-quantum schemes and, at the time of its submission, had not undergone proper scrutiny due to its recent development.

[4] Despite the name, SIDH does not rely on the hardness of any discrete logarithm problem and is therefore not affected by Shor's algorithm.

When the work in this thesis was conducted, SIKE was still a viable competitor in the NIST post-quantum standardization process. Alas, in 2022, a classical attack by Castryck and Decru in [CD22] completely compromised the security of SIKE, rendering the scheme unsuitable for any practical application. As a result, the authors of SIKE withdrew their scheme from the standardization process. The relevance of our work despite this attack will be discussed later.

SPHINCS-256 and SPHINCS$^+$. SPHINCS is a family of hash-based digital signatures. Hash-based cryptosystems have the particularity of being constructed using only a single primitive called *cryptographic hash function*; a function which turns bitstrings of any size into fixed-size bitstrings in such a way that finding an input that maps to a given output is computationally infeasible. Such a function is applied to secret values to obtain public values which are published as part of the public key. The signing key, on the other hand, consists of the secret values. To sign a message, the signer reveals a specific set of secret values chosen based on the message being signed, while keeping certain others forever confidential to maintain a desired level of security.

As the signer needs to keep track of which secret values have been revealed to preserve security, hash-based digital signature schemes traditionally required to manage an internal state. The SPHINCS family overcomes this limitation by being *stateless*, which means that no state needs to be maintained to ensure its security. This new feature is enabled by the generation of a virtually infinite number of signatures ensuring that the scheme remains secure with overwhelming probability. As a result, SPHINCS schemes can act as drop-in replacements for current digital signature solutions without requiring the additional implementation of a state machine. The original SPHINCS scheme, called SPHINCS-256, achieves statelessness by combining many building blocks, such as one-time signature schemes, few-time signature schemes, and Merkle trees, that may all be constructed with the same hash function.

In 2016, an improvement over SPHINCS-256 called SPHINCS$^+$ was submitted to the NIST post-quantum project, which incorporates several changes that mainly enhance the performance of the scheme. Compared to the other candidates in the standardization process, SPHINCS$^+$ offers the smallest keys, offering an advantage of around two orders of magnitude over, e.g., CRYSTALS-DILITHIUM and FALCON. However, the signature sizes are the largest among the competition, requiring about tens of kilobytes for a single signature, in contrast to the few kilobytes needed by its lattice-based competitors. The signing and verification times are also considered to be extremely slow, particularly when implemented with one of the hash functions approved by NIST.

Nevertheless, as opposed to SIKE, SPHINCS$^+$ is still a surviving candidate that is still going to be standardized by NIST. In fact, NIST considers SPHINCS$^+$ to be a mature enough choice for a post-quantum digital signature solution, since its security depends only on the cryptographic soundness of the underlying hash function whose current implementations are believed to be robust [Gor+22]. As a result, given that SPHINCS$^+$ is soon expected to secure practical data, a thorough investigation of all aspects of its security is crucial to its real-world deployment.

Attacking implementations through side channels. Even though post-quantum algorithms are designed to resist both classical and quantum attacks, these algorithms are intended to be deployed on physical devices. These devices are subject to additional vectors of attack, independently of the algorithm implemented. In particular, *side-channel attacks* exploit the information leakages that result from the physical implementations of cryptographic algorithms. These attacks are particularly relevant to embedded devices, such as smartphones, since these devices are deployed in uncontrolled environments where attackers have physical access. In this context, the information leakages typically include the electrical power consumption of the device which may be linked to the secret values involved in the implemented cryptographic algorithm. An additional vector of side-channel information is obtained by analyzing the behavior of the device when a fault is deliberately injected during the execution of a cryptographic algorithm, as faulty behavior potentially leads to the inadvertent disclosure of secret information. As a result, in addition to ensuring the theoretical security of post-quantum cryptosystems, assessing the security of their implementation in all use-cases is equally important.

Throughout its post-quantum standardization process, NIST consistently requested side-channel evaluation of the submitted candidates [Moo19b; Apo20]. Moreover, resistance to side-channel attacks has been considered to be a crucial factor in the selection of future standards (see [Gor+22]). The security of implementations when deployed on the ARM Cortex-M4 microcontroller is of particular interest, as NIST recommends this microcontroller architecture for embedded implementations of the post-quantum candidates [Moo19a; Kan+19].

Consequently, the present thesis directly responds to NIST's request and evaluates the side-channel security of Cortex-M4 implementations of SIKE and SPHINCS$^+$, as well as a custom Cortex-M3 implementation of SPHINCS-256.

Power analysis. A side-channel power analysis consists of deducing secret values, typically in small portions, using measures of instantaneous power consumption that correspond to executions of a cryptographic algorithm. As such measurements are proportional to the number of physical bits that require to be updated, an attacker can use statistical analysis on the power consumption measurements to gain additional information about the secret values involved in the algorithm.

A multitude of techniques exist to recover any type of information—including confidential information—from measurements of power consumption. The most common technique is differential power analysis [KJJ99; BCO04] which consists of applying statistical techniques to confirm or refute small hypotheses about a secret value. Other methods leverage machine learning algorithms in both unsupervised or supervised fashion to further exploit the statistical link between power consumption and the processed values. Typically, unsupervised algorithms cluster power samples from multiple calls to the same function involving different secret chunks, enabling the recognition of the chunks with the same value (see [Hey+13; PC15]), while supervised algorithms profile the power consumption by creating templates

that are then compared against a target power measurement to extract the secret values from their corresponding execution (see [CRR03; BL12]). More sophisticated attacks attempt to force certain variables to be zero based on small hypothesized secret values, as these values require low energy to process and can consequently be detected in the power consumption (see [Gou03; AT03]).

The study of side-channel power analyses against post-quantum algorithms is a growing field of interest with ongoing developments in attacks and countermeasures. The most extensive side-channel evaluation of the NIST post-quantum candidates with regard to power analysis is definitely the work by Ueno et al. in [Uen+22] where most of the round-3 key encapsulation mechanisms were found to be vulnerable to a systematic power analysis. Lattice-based key encapsulation mechanisms received the most attention due to their popularity. For example, simple power analysis and differential power analysis models against CRYSTALS-KYBER were developed by Azouaoui et al. in [Azo+22]. Other attacks, based on chosen-ciphertext key recoveries, can be categorized into three different classes, as noticed by Ravi and Roy in [RR21]: attacks based on the validation of the decrypted plaintext [D'A+19; Rav+20; Tan+22; Raj+23], attacks based on the recognition of decryption failures [GJN20], and attacks based on the recovery of the full decrypted messages [Ngo+21; Bac+22; DNG22; Xu+22]. The round-4 code-based cryptosystems were also subject to many side-channel power analyses. In particular, HQC was found to be vulnerable to a decryption failure recognition in [GJ20], to error correction detections in [Sch+20; GLG22a; Sch+22a], to a horizontal differential power analysis in [GLG22b], and to a timing attack that also applies to BIKE in [GLG22a]. Classic McEliece was hit with several message-recovery power analyses in [Lah+19; Col+22], as well as key recoveries first in [GJJ22] through detecting errors in a step of the fast Fourier transform involved in the scheme, then in [Col+23] through a horizontal differential power analysis. In contrast, at the time of this thesis, only one work studied hash-based digital signatures in general with respect to power analysis in [EMY14], where the authors assessed the leakage of a customized Merkle signature scheme, and only one work discussed the potential risks of a side-channel power analysis against SIDH in [KAJ17].

Fault analysis. A fault analysis consists of studying the consequences of introducing a fault during the execution of a cryptosystem. A fault can have various effects, such as skipping microcontroller instructions which typically enables an attacker to bypass security checks, or compromising the integrity of stored values in the attacked device which can lead to side-channel information about secret values. As a result, analyzing the repercussions of deliberate or accidental faults against a cryptosystem is crucial to the security of its implementation.

As with power analysis, there exists a variety of techniques to deliberately inject faults during the execution of a cryptographic algorithm. The most common technique, particularly relevant to embedded devices such as the ARM Cortex-M4, involves injecting a glitch in the voltage or the clock of the device. This approach, known for skipping instructions and corrupting data, is considered somewhat invasive as it requires breaking open the device to access the device's integrated circuit [Bar+12]. On the other hand, a technique known as row-hammer [Kim+14]

has demonstrated the ability to remotely inject data-corrupting faults on web server applications by repeatedly accessing and manipulating data held in RAM, proving the relevance of fault analyses even in controlled environments.

Similarly to power analysis, fault injection in the context of post-quantum cryptography is an increasingly important area of research. Similar to the study with the systematic power analysis, Xagawa et al. in [Xag+21] found that most of the round-3 key encapsulation mechanisms in the NIST post-quantum standardization process were vulnerable to a systematic fault attack that skips a security check. Both CRYSTALS-KYBER and CRYSTALS-DILITHIUM were subject to a fault attack that induces a nonce reuse in [Rav+19b], and to a safe-error attack in [BMR21]. Additional safe-error fault attacks against KYBER were developed in [PP21] and in [HPP21]. DILITHIUM was devastatingly susceptible to a fault attack by Bruiderink and Pessl in [BP18], in which valid and faulty signatures are exploited to extract the signing key when the scheme is configured in deterministic mode. The determinism of DILITHIUM was further exploited by Ravi et al. in [Rav+19a] where the authors show that skipping an addition enables the recovery of a large part of the signing key. Later, another fault attack named signature correction attack has been shown to recover the signing key bit by bit in both deterministic and randomized DILITHIUM in [Isl+22]. FALCON was also found to be vulnerable to fault attacks that retrieve a significant portion of the signing key by forcing the lattice trapdoor sampler to be zero in [McC+19]. Among the round-4 code-based key encapsulation mechanisms, only Classic McEliece has currently been evaluated with respect to fault analysis. In particular, Classic McEliece was attacked through a message-recovery laser injection in [Cay+21], and through a key-recovery laser and fault injections in [Pir+22]. The isogeny-based SIKE is prone to a safe-error fault attack as discovered in [CKM21], and to loop-abort fault injections in [Tas+21; Bel+21; Bee+22]. SPHINCS-256, the ancestor of SPHINCS$^+$, proved to be critically vulnerable to a data-corrupting fault injection in [CMP18], which was experimentally verified in [Gen+18]. The same attack was adapted to a custom FPGA implementation of SPHINCS$^+$ in [Ami+20].

Content of the thesis. The present thesis studies side-channel vulnerabilities and countermeasures on SIKE, SPHINCS-256, and SPHINCS$^+$. These candidates have unconventional security assumptions that have not yet been as thoroughly analyzed as the other post-quantum families in this regard, especially in the context of embedded devices. Analyzing these schemes is important to gain insight into their underlying principles and limitations in order to develop better cryptosystems. Moreover, since SPHINCS$^+$ is expected to become a future standard for securing real-world data in the long term, it is crucial for this scheme to provide a level of security that is consistent with the other candidates.

Even though SIKE has been withdrawn from NIST's post-quantum process due to the attack by Castryck and Decru in [CD22], our side-channel analysis of the scheme gives an important baseline for future isogeny-based implementations. In particular, our work validated the threat of power analysis against isogeny-based cryptosystems and demonstrated that these schemes are subject to unique vectors of attack. Most notably, our power analysis of the isogeny

computation from Chapter 6 led to the side-channel attack by Campos et al. in [Cam+22] which still applies to another isogeny-based key exchange protocol called CSIDH that is not affected by the Castryck–Decru attack. Moreover, our work proposed novel improvements and optimizations of power analysis techniques which may be useful for future attacks. For all these reasons, our work still holds relevance and highlights the usefulness of researching side-channel vulnerabilities even in obsolete cryptosystems.

The current thesis is structured into two parts: the first part covers the isogeny-based key encapsulation mechanism SIKE, and the second part covers the hash-based digital signature schemes SPHINCS-256 and SPHINCS$^+$. Chapter 2 serves as an introduction to side-channel analysis and regroups the various techniques used throughout our work.

Part I. The first part of the thesis is dedicated to the power analysis of SIKE. In Chapter 3, the scheme is introduced, including its mathematical background and algorithms.

Chapter 4 describes a straightforward horizontal differential power analysis of the ARM Cortex-M4 implementation of SIKE. In this attack, an adversary records the power consumption of a single execution of the elliptic curve scalar multiplication involved in SIKE and infers the computing party's private key bit by bit, where the value of each bit is confirmed by computing the Pearson's correlation coefficient of hypothesized results in the double-and-add procedure with their corresponding segments of power consumption. Even though the attack is identical to a horizontal differential power analysis of classical elliptic curve cryptography, our work describes the very first power analysis of SIKE in ephemeral settings which defeats many countermeasures that were initially proposed for SIKE in [Zha+20], such as masking the scalar, or randomly splitting the key. Instead, we suggest randomizing the coordinates as an effective countermeasure with minimal overhead. The chapter also describes and demonstrates in practice many novel improvements that make the attack more effective, such as error-correction techniques based on depth search, and the use of the wavelet transform as a denoising tool.

Following the analysis conducted in the previous chapter, we now suppose that the ARM Cortex-M4 implementation of SIKE is protected with coordinate randomization so that attacks based on differential power analysis are ineffective. In Chapter 5, we show that despite the countermeasure, ephemeral Cortex-M4 SIKE is still vulnerable to a clustering power analysis of the point-swapping function calls that are involved in a single execution of the elliptic curve scalar multiplication. The attack works by clustering all the power samples that correspond to the point-swapping procedure with the k-means algorithm. Since these power samples depend on whether two elliptic curve points were swapped or not, which in turn depend on the difference between two consecutive bits private key, the clustered samples are expected to exhibit two distinct distributions. A successful clustering therefore enables the recovery of the computing party's private-key bits all at once. While the analysis is once again identical to a clustering power analysis in elliptic curve cryptography, our work describes various enhancements of the attack, including clustering in the frequency domain,

processing the traces with a wavelet transform, using a simpler clustering power analysis based on thresholding, and using metrics to prioritize certain keys for key validation. Our attack and the proposed improvements were experimentally verified, and a countermeasure based on splitting the swapping mask into multiple shares is suggested.

As the previous two chapters suggest, the elliptic curve scalar multiplication offers a large surface of side-channel leakage and requires all of its components to be protected against power analysis in order to operate securely in an embedded environment. Supposing the entire scalar multiplication is secure against power analysis using, in particular, coordinate randomization, Chapter 6 addresses the power analysis of other operations in SIKE, such as the isogeny computation. Notably, the chapter describes a zero-value point attack on the isogeny computation which remains applicable despite the randomization of coordinates. However, as opposed to the other chapters, this attack applies only to the semi-static settings of SIKE, and enables the bit-by-bit recovery of the decapsulating party's private key through chosen ciphertexts that force certain computations, such as the j-invariant derivation, to process zero values depending on a single private-key bit. As a result, this attack is enabled by an attacker able to distinguish between zero and nonzero values in the power consumption, which was experimentally verified to be possible against the ARM Cortex-M4 implementation of SIKE using an approach based on collision power analysis. This approach consists of comparing the power measurements with baselines that correspond to executions of the targeted operation (i.e., the j-invariant computation) when this operation is respectively processed with zero and nonzero operands, which can be forced regardless of the decapsulating party's private key.

Part II. The second part of the thesis focuses on the power and fault analyses of the SPHINCS family. Chapter 7 introduces both SPHINCS-256 and SPHINCS$^+$, along with their algorithms.

In Chapter 8, the hash function BLAKE-256, used in SPHINCS-256 to hide the secret values, is attacked through a straightforward differential power analysis. The analysis is performed in practice on a custom Atmel Cortex-M3 implementation of the scheme by collecting electromagnetic radiations emitted by the microcontroller. This work is the first case of a successful side-channel attack against hash-based digital signatures which were so far considered to be "naturally resistant to most kinds of side-channel attacks" (see [Hue+18, §1]). In particular, this is the first attack against an implementation of SPHINCS and shows the importance of securing the implemented hash function against power analysis for the scheme to be deployed in an embedded environment. We argue that a simple countermeasure based on shuffling the operations in the BLAKE-256 hash function makes the attack harder to mount.

While the previous chapter demonstrates a power analysis against a specific implementation of the underlying hash function involved in SPHINCS-256, Chapter 9 describes a fault attack on the successor SPHINCS$^+$ that is agnostic of the hash function used. The attack extends the work by Castelnovi, Martinelli, Prest from [CMP18] and shows, in particular, that any combination of faulty signatures is enough to attack SPHINCS$^+$. Our work then presents a thorough analysis of the repercussions of a fault injection in SPHINCS$^+$ in view of finding an adequate

countermeasure. However, the work concludes that besides detecting a fault by redundantly computing the signature multiple times, no satisfactory countermeasure exists to this day, as even the caching countermeasures that were initially suggested by the present author in [Gen+18] come short when applied to SPHINCS$^+$; a result that was experimentally verified on the SPHINCS$^+$ reference implementation adapted to the ARM Cortex-M4 architecture.

At last, the thesis closes in Chapter 10 with a retrospective of the work conducted.

Personal bibliography. The works published during the authorship of this thesis are listed below in chronological order, along with their respective chapter in the dissertation (if included). The publications involved in this dissertation are highlighted in boldface.

[Kan+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. "Differential Power Analysis of XMSS and SPHINCS". in: *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Apr. 2018, pp. 168–188. DOI: 10.1007/978-3-319-89641-0_10.	Chapter 8
[Gen+18] Aymeric Genêt, Matthias J. Kannwischer, Hervé Pelletier, and Andrew McLauchlan. "Practical Fault Injection Attacks on SPHINCS". in: *Kangacrypt 2018, Australian Workshop on Offensive Cryptography* (2018).	–
[GGK21] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. "Full Key Recovery Side-Channel Attack Against Ephemeral SIKE on the Cortex-M4". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 228–254. URL: https://doi.org/10.1007/978-3-030-89915-8_11.	Chapter 4
[GK22] Aymeric Genêt and Novak Kaluđerović. "Single-Trace Clustering Power Analysis of the Point-Swapping Procedure in the Three Point Ladder of Cortex-M4 SIKE". in: *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 164–192. DOI: 10.1007/978-3-030-99766-3_8. URL: https://doi.org/10.1007/978-3-030-99766-3_8.	Chapter 5

[Feo+22] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluđerović, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. "SIKE Channels". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (2022). https://tches.iacr.org/index.php/TCHES/article/view/9701, pp. 264–289. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i3.264-289. Chapter 6

[Gen23] Aymeric Genêt. "On Protecting SPHINCS+ Against Fault Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.2 (2023). https://tches.iacr.org/index.php/TCHES/article/view/10278, pp. 80–114. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i2.80-114. Chapter 9

2 Side-channel attacks

A *side channel* is a source of information that inadvertently leaks during the implementation of a cryptographic algorithm. Side channels usually result from physical variables (such as the execution time of the algorithm) which are sometimes linked with the information processed by the cryptosystem. In conjunction with the public material of the cryptosystem, the information measured from side channels can be exploited by attackers to undermine the security of the cryptographic scheme; resulting in a *side-channel attack*.

The current thesis considers the *power consumption* of a cryptographic implementation on a target device as the side channel of interest. With access to the power supply of a device that runs cryptographic algorithms, an adversary can typically mount two types of attacks:

- *power analyses*, which consist of recovering processed information through measurements of the power consumed by the device,
- *fault analyses*, which consist of deducing information by analyzing the effects of a disturbance (also known as a *glitch*) introduced into the power supply of the device.

Moreover, the analyses are considered to be *passive* when the adversary does not need to interact with the device, and *active* otherwise.

This chapter outlines all the side-channel power-based techniques used in our work.

2.1 Power analysis

In a power analysis, the adversary attempts to recover secret information processed by an electronic device by exploiting its relation with the instantaneous power consumption (i.e., the amount of power consumed at any instant of time). Such a relation is (partly) due to the structure of modern transistors that electronic devices use to physically store and manipulate bits. As a transistor requires energy to flip its internal state, the current drain is expected to change according to the bit transition (from 0 to 1, and vice versa) that occurs in a register as a result of an operation, and to increase in magnitude with the number of bits that are updated.

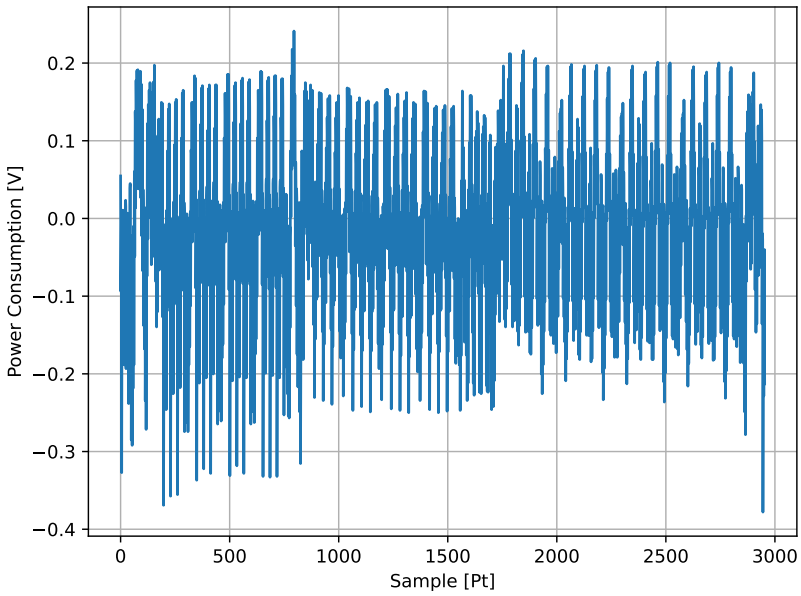


Figure 2.1: Example of a power trace sampled at a rate of 29.54 MHz for a period of 0.1 ms.

In the scope of this thesis, we consider two types of power analyses:

- *vertical* power analyses, which target a *fixed* secret value across different executions of the attacked algorithm by collecting *multiple* power measurements that correspond to multiple executions of the *same* operation on, possibly, different data,
- *horizontal* power analyses, which target an *ephemeral* secret value using a *single* power measurement that corresponds to *multiple* operations. These operations must be similar to allow the segmentation of the power measurement into multiple ones to simulate a vertical power analysis.

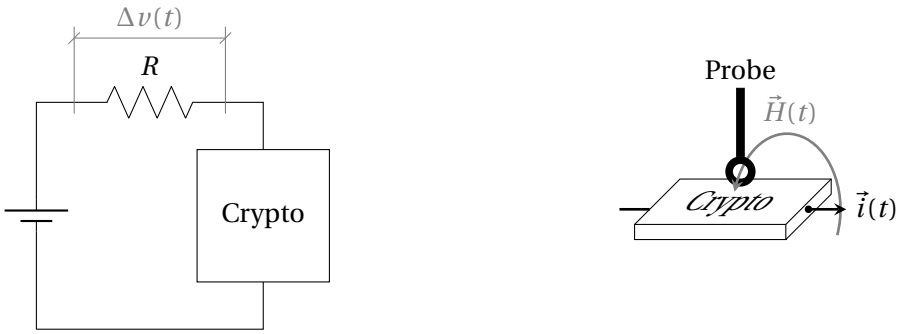
A typical power analysis involves three main steps:

1. **Collection of power consumption**
   The adversary collects traces of power consumption that correspond to (supposedly) known cryptographic operations.
2. **Power consumption processing** (optional)
   The adversary processes the traces of power consumption with signal processing techniques in preparation of secret extraction.
3. **Statistical analysis**
   The adversary applies statistical methods to extract secret information from traces of power consumption.

2.1.1 Traces collection

To conduct a power analysis, the first step involves collecting samples of instantaneous power consumption during the relevant operations. One common way to accomplish this is by "shunting" the cryptographic device, that is, creating a circuit with a resistor of known resistance in series to the voltage collector (or ground) and to the device, as shown in Figure 2.2a. Then, the voltage drop across the shunt resistor is sampled at a fixed rate (using, e.g., an oscilloscope), as this difference depends[1] on the power consumption of the device.

An alternative method of collecting power samples is to measure the ElectroMagnetic (EM) field emitted by a device by placing a local near-field probe on the chip of the device, as shown in Figure 2.2b. According to Ampère's circuital law, the magnitude of the EM field is proportional to the variations of the current flowing through the circuitry of a device, which therefore provides an indirect[2] measurement of the power consumption. To identify the best position, the entire surface is scanned during said operations.



(a) Shunted cryptographic device. Power traces are measures of $\Delta v(t)$.

(b) Probed cryptographic device. Power traces are measures of $\|\vec{H}(t)\|$.

Figure 2.2: Setups to measure power traces.

Compared to measuring voltage differences through a shunt resistor, collecting EM samples is a bit more invasive as the attacker needs access to the device chip. However, since the collection is limited to a local part of circuitry, the EM samples measured are usually more significantly linked to the cryptographic operations as they do not contain the energy contribution of other non-leaking parts of the circuitry. Despite providing better results, power analysis with EM samples requires a much faster sampling rate to accurately capture the electric current variations and, in turn, more sophisticated equipment.

A *power trace* $T = (s_i \in \mathbb{R})_{0 \leq i < M}$ refers to a collection of $M > 0$ (instantaneous) power[3] samples of a cryptosystem that were measured at a given sampling rate during a time window. Figure 2.1 illustrates a power trace when the samples are depicted in a graph.

[1] Assuming the voltage supply is constant, $p(t) = v(t) \times i(t)$, and since $i(t) = v(t)/R$, we have that $p(t) = v(t)^2/R$.

[2] Actually, $p(t) = -dU(t)/dt$ where $U(t)$ is the total electromagnetic energy within a volume which is related to the variations of electric fields according to Poynting's theorem (see [Jac99, §6.8]).

[3] Due to their relation with energy (see footnotes above), power samples are actually voltage samples of a signal.

Given a power trace $T = (s_i \in \mathbb{R})_{0 \leq i < M}$, a *sample point* (also referred to as a sample or timing location) corresponds to a timing index $0 \leq i < M$ in a power trace.

In our experimental verifications, we use the two measurement setups mentioned above: one based on power consumption, and another on electromagnetic emanations.

2.1.1.1 Power measurement setup (ChipWhisperer)

ChipWhisperer [New21a] is an open-source toolkit which features embedded devices that are deliberately exposed to power analysis as a means to experimentally analyze algorithms, attacks, and countermeasures. In particular, ChipWhisperer provides programmable devices based on ARM Cortex-M4 microcontrollers which are recommended by NIST for evaluating the implementations of post-quantum candidates [Moo19a]. As our attacks are mostly theoretical, we therefore took advantage of the convenience of ChipWhisperer to provide proofs of concept. Still, all of our attacks are applicable with additional efforts of marginal complexity on real-world devices.

While the ChipWhisperer framework includes standalone tools to directly sample the power consumption of the target device, we still use a digital oscilloscope to collect power traces. An oscilloscope is a scientific instrument that enables measuring and visualizing electronic signals, as well as other features to analyze the captured waveforms. We made this choice mainly to overcome the limitations of memory of the ChipWhisperer toolkit, but also to avoid its built-in features, such as the synchronization with the clock of the target device, that would make the traces collection too unrealistic in some scenario.

Consequently, the full list of equipment used to collect power traces with the ChipWhisperer framework is the following:

- The ChipWhisperer toolkit [New21a], that includes:
  - A (NAE-CW308T-)STM32F3 board which features an ARM Cortex-M4 microcontroller as the DUT (Device Under Test).
  - A ChipWhisperer-Lite board which is solely used to communicate with the DUT in serial through USB.
  - A ChipWhisperer (NAE-)CW308 UFO board which serves as an intermediary board that enables the ChipWhisperer-Lite and the DUT to exchange signals.
- A high-definition oscilloscope with at least the following specifications:
  - An analog bandwidth of 500 MHz.
  - A sampling rate of 250 samples per microsecond (i.e., 250 MS/s).
  - A resolution of 10 bits per sample.
  - A memory of 50,000 samples per acquisition.
- A general-purpose computer which runs an operating system compatible with the ChipWhisperer framework [New21b].

The STM32F3 is plugged into the CW308 UFO, which is itself connected to the ChipWhisperer-Lite with a 20-pin cable. The oscilloscope measures the power consumption through a passive probe connected to the `SHUNTL`[4] pin on the CW308 UFO, and whose measurement is triggered by reacting to the active-high `GPIO4/TRIG`[4] pin also with a passive probe (both probes are grounded to the `GND` pins on the CW308 UFO). The computer is simply connected to the ChipWhisperer-Lite with a USB to micro-USB cable.

2.1.1.2 EM measurement setup (Arduino)

Arduino [Ard05] is an open-source electronics platform that provides both hardware and software solutions for designing embedded systems. In particular, the company manufactures embedded boards which are equipped with Cortex-M microcontrollers. Our setup involves the Arduino Due model because the board offered enough flash memory and RAM to implement the studied algorithms. Note that the board features a SAM3X8E Cortex-M3 CPU, which is a generation of microcontrollers prior to the Cortex-M4 that was recommended by NIST, simply because we conducted our analysis with it before NIST's recommendation.

Our EM measurement setup, only used in Chapter 8, therefore comprises the following:

- An Arduino Due microcontroller which features an Atmel SAM3X8E Cortex-M3 CPU.
- A local near-field microprobe with a frequency range of 2.5 MHz to 6 GHz.
- A high-definition oscilloscope with at least the following specifications:
  - An analog bandwidth of 2.5 GHz.
  - A sampling rate of 10 samples per nanosecond (i.e., 10 GS/s).
  - A resolution of 8 bits per sample.
  - A memory of 25,000 samples per acquisition.
- A general-purpose computer which runs an operating system compatible with the Arduino framework [Ard12].

The near-field microprobe was connected to the oscilloscope and placed at the position shown in Figure 2.3. This position was found by experimental exploration and likely corresponds to the position of the Cortex-M3 circuitry due to the strong emissions observed. The Arduino Due microcontroller was connected to the computer with a USB to micro-USB cable.

2.1.2 Traces processing

Once collected, the power traces can undergo an optional (pre-)processing phase with various transforms to make the power analysis more effective. For instance, these transforms can

[4] We refer to the official NAE-CW308 UFO datasheet to find the mentioned pins: http://media.newae.com/datasheets/NAE-CW308-datasheet.pdf.

Figure 2.3: Position of an EM microprobe on a SAM3X8E Cortex-M3 microcontroller at which strong EM radiations could be collected.

exhibit leakages that were initially not present in the raw data, or make the power traces cleaner by reducing the noise.

2.1.2.1 Fourier transform

A (discrete) *Fourier transform* is a decomposition of a (discrete) signal into a representation that exhibits information about the frequency components of the signal. The representation is obtained by projecting the signal s_x ($0 \leq x < M$) onto the (discrete) orthogonal Fourier basis $\{e^{2\pi i x f/M}\}_{0 \leq f < M} \in \mathbb{C}^M$ (where $i = \sqrt{-1}$ is the imaginary unit):

$$\hat{s}_f = \sum_{x=0}^{M-1} s_x e^{-2\pi i x f/M} \qquad (0 \leq f < M).$$

In power analysis, the signal corresponds to the power consumption, and the frequency coefficients are associated to the samples from operations being performed at fixed intervals. Exploiting the power leakages in the frequency domain is a well-studied process and is shown to exhibit many advantages, such as the robustness against timing misalignments (see [Agr+03]).

2.1.2.2 Wavelet transform

A (discrete) *wavelet transform* is a multi-level filter bank parameterized by a wavelet function $\psi(t)$ which decomposes a (discrete) signal into frequency bands. As opposed to the Fourier transform, the wavelet transform gathers information both from the frequency and the timing contents by iteratively correlating the signal with $\{1/\sqrt{2^y}\psi((t-2^y x)/2^y)\}_{(x,y)\in\mathbb{Z}^2}$.

A single step of the filter bank separates an input signal s_x ($0 \leq x < M$) into two sub-signals of respectively low and high frequencies:

1) the *approximations*: $a_x = \sum_{k=-\infty}^{\infty} s_k L_{2x-k}$

2) the *details*: $d_x = \sum_{k=-\infty}^{\infty} s_k H_{2x-k}$

where L_x and H_x are respectively low-pass and high-pass filters obtained from the wavelet function $\psi(t)$ (see [Mal08] for the technical details). The filter bank consists of recursing the above formulas with a_x with new filters derived from different scales to reflect the different frequency bands at each level of the transform.

An example of a generic three-level wavelet transform is shown on Figure 2.4. Given f the frequency of s_x and $\ell \geq 0$, $a_x^{(\ell)}$ corresponds to the sub-signal of frequencies $[0, f/2^{\ell+1}]$ and is re-injected into the filters H_x and L_x to ultimately output $a_x^{(2)}$, while $d_x^{(\ell)}$ corresponds to the sub-signals of frequencies $[f/2^{\ell+1}, f/2^{\ell}]$. Note that $d_x^{(0)}$, $d_x^{(1)}$, $d_x^{(2)}$, and $a_x^{(2)}$ cover the entire spectrum of $[0, f]$.

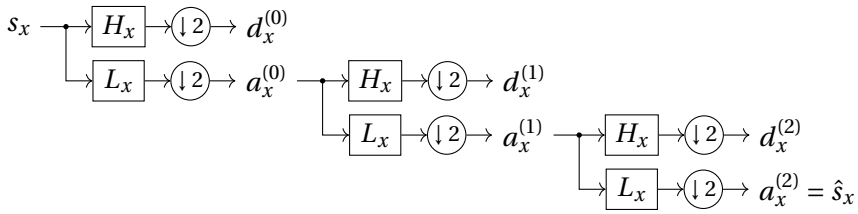


Figure 2.4: A three-level wavelet transform.

In power analysis, the wavelet transform is recognized to refine the quality of the power traces acquired (see [CP05; Sou+21]) by running the analysis on either the approximations or the details alone. This is because the wavelet transform can selectively capture a frequency band of interest while filtering out irrelevant frequencies, resulting in a cleaner signal than the original one.

2.1.3 Traces analysis

The last step of power analysis consists of applying statistical techniques to retrieve secret information using the power traces. These techniques base their effectiveness on the link between processed information and the power consumption, and serve as decision-making tools in the process of recovering portions of secret values.

In this section, we describe all the power analysis techniques that will be used to attack the studied cryptographic algorithms.

2.1.3.1 Differential power analysis

Differential Power Analysis (DPA) is a power analysis technique that aims to deduce fixed secret information across multiple power measurements through statistical metrics. DPA relies on a statistical tool called *distinguisher* to detect the dependency between the power traces and a small portion of secret data. By repeating the dependency detection across all portions, the secret data is eventually fully recovered.

Throughout the years, many distinguishers were explored. The original DPA from Kocher et al. in [KJJ99] uses the *difference of means* which consists of partitioning the power traces into two subsets according to a hypothesized portion of the secret data, and computing the differences of their respective average. Strong differences indicate that the classification is likely correct, which thus confirms the hypothesis.

Correlation is another measure that is commonly used as a distinguisher in DPA; resulting in correlation power analysis [BCO04]. In this type of DPA, the link between power consumption and the processed values is thus quantified with a correlation coefficient. In particular, the power analysis supposes that the power samples are correlated to the Hamming weights of the processed values.

In our work, we exclusively use DPA with correlation as distinguisher (i.e., correlation power analysis).

To assess correlation between the processed values and the power samples, the Pearson's Correlation Coefficient (PCC) is computed. Given $N > 0$ power traces of $M > 0$ samples $T_i = (s_t^{(i)})_{0 \le t < M}$ for $0 \le i < N$, let $\boldsymbol{\tau}(t) = (s_t^{(i)})_{0 \le i < N}$ be a vector of power samples synchronized at a same sample point $0 \le t < M$, and $\mathbf{h} \in \mathbb{N}^N$ a vector of the Hamming weight of the processed values.

$$\mathrm{PCC}(\mathbf{h}, \boldsymbol{\tau}(t)) = \frac{\mathrm{Cov}(\mathbf{h}, \boldsymbol{\tau}(t))}{\sqrt{\mathrm{Var}(\mathbf{h})\mathrm{Var}(\boldsymbol{\tau}(t))}}.$$

The overall attack consists of the following steps:

1. Find an operation in the attacked procedure which involves:
   (a) A (small) portion of a secret value which is the same across all measurements.
   (b) A known input to the cryptographic algorithm.

   In the following, we refer to the result of this operation as the *intermediate value*.
2. Collect N power traces (consisting of M power samples each) that correspond to the computation of the intermediate value with the different inputs to the cryptosystem.
3. Take a guess for the portion of the secret value involved in the intermediate value computation.
4. Compute the vector of intermediate values from the known inputs and the secret value guess, and derive its corresponding vector of Hamming weights $\mathbf{h}$.

5. For each vector of power samples at a same sample point, i.e., $\boldsymbol{\tau}(t)$ for each $0 \leq t < M$, compute PCC($\mathbf{h}, \boldsymbol{\tau}$).
   This results in a vector of PCC at each sample point.

A strong PCC (in absolute value) at any sample point indicates that the guess for the portion of the secret value is valid, while a weak PCC at every sample point can rule out said guess. The number of power traces N should be large enough given the signal-to-noise ratio of the power consumption. Figure 2.5 illustrates the process.

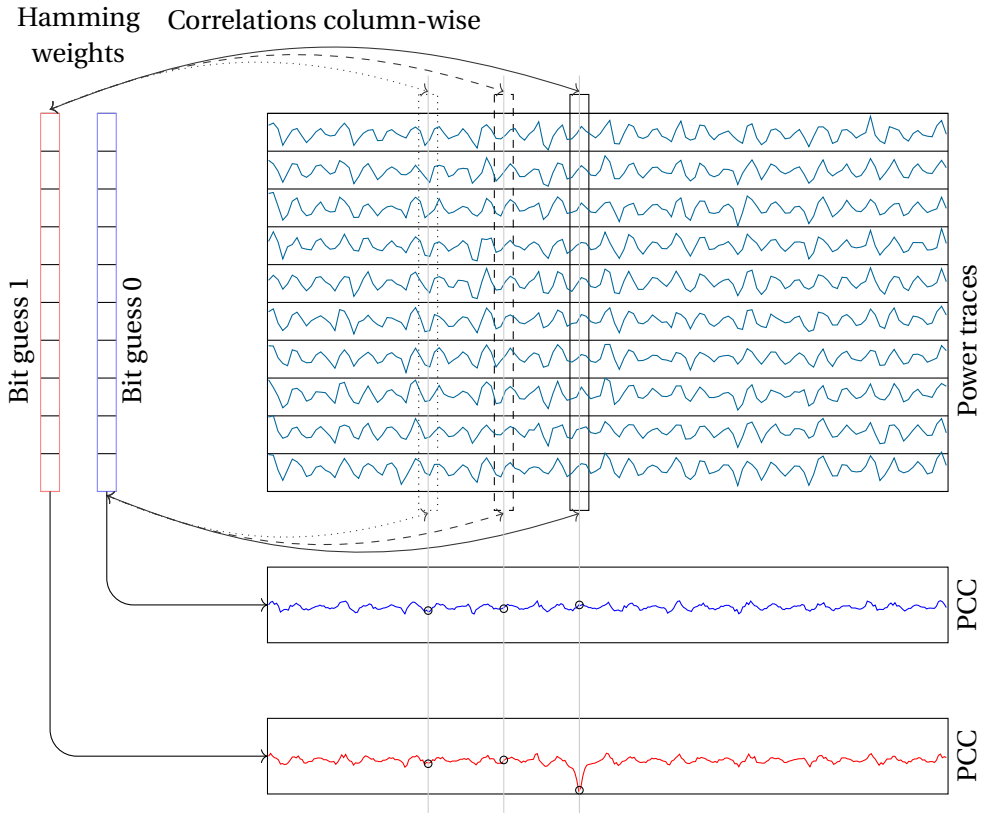


Figure 2.5: Visual representation of a DPA with correlation that reveals a single bit of a secret value. Correlations between two vectors of Hamming weights and the power traces are plotted in the bottom. A strong correlation indicates that the bit value associated to these power traces is 1.

2.1.3.2 Clustering power analysis

Clustering power analysis is a power analysis technique that aims to group together similar samples of power consumption to deduce secret information. The analysis relies on the behavior of the power samples that are expected to follow distinct distributions depending on the value of a small chunk of secret data. As a result, a successful discernment of the power sample distributions leads to the recovery of all the secret chunks and, thus, of all secret data.

Clustering power analysis relies on an underlying classification algorithm to cluster the power samples based on their distribution patterns. The original attack from Heyszl et al. [Hey+13] uses the *k-means algorithm* [Mac67], which works by partitioning a population[5] of N samples into k sets solely based on the values of the samples. Informally speaking, the algorithm starts with k groups of means μ_j ($0 \leq j < k$), and reassigns the samples to the group with the closest mean. As doing so may change the means of the groups, the process is repeated until stability. The full procedure is shown in Algorithm 2.1.

Algorithm 2.1 The k-means algorithm.

Input: $(s_i \in \mathbb{R})_{0 \leq i < N}$: Collection of N samples.

1: Assign each s_i to a cluster j at random ($0 \leq i < N$, $0 \leq j < k$).
2: **repeat**
3: Compute each μ_j as the mean of each cluster ($0 \leq j < k$).
4: Assign each s_i to the cluster $j = \text{argmin}|s_i - \mu_j|$ ($0 \leq i < N$).
5: **until** no μ_j changes (for all $0 \leq j < k$).
6: **return** the final cluster assignments of all s_i ($0 \leq i < N$).

Once the power samples are properly clustered, all the samples in a same cluster are expected to correspond to the same chunk value. Consequently, by mapping all the possible chunk values to each label, we obtain a candidate for the entire secret value that must be validated with public information (such as an encrypted message). If the validation fails, another mapping can be explored.

The overall attack consists of the following steps:

1. Find an operation in the attacked procedure which involves only a small portion of a secret value (which has a known number k of possible values).
2. Collect N power traces (consisting of M power samples each) that correspond to the operation, each time with a different portion of the secret value.
3. Let $\boldsymbol{\tau}(t) = (s_t^{(i)})_{0 \leq i < N}$ be a vector of power samples synchronized at a same sample point $0 \leq t < M$, such that each power sample in the vector corresponds to a different secret portion.
4. Run a clustering algorithm (such as the k-means algorithm) on $\boldsymbol{\tau}(t)$ to label each power sample according to its appropriate cluster.
5. Assign the possible portion values to the labels of the power samples.
6. Combine all the identified portions related to each power sample to obtain a candidate for the secret value.

If the distributions of power samples are distinct enough given the signal-to-noise ratio of the power consumption, there should be at least one candidate at one sample point that corresponds to the secret value. Figure 2.6 illustrates a successful clustering power analysis.

[5] In the scope of our work, the population is one-dimensional.

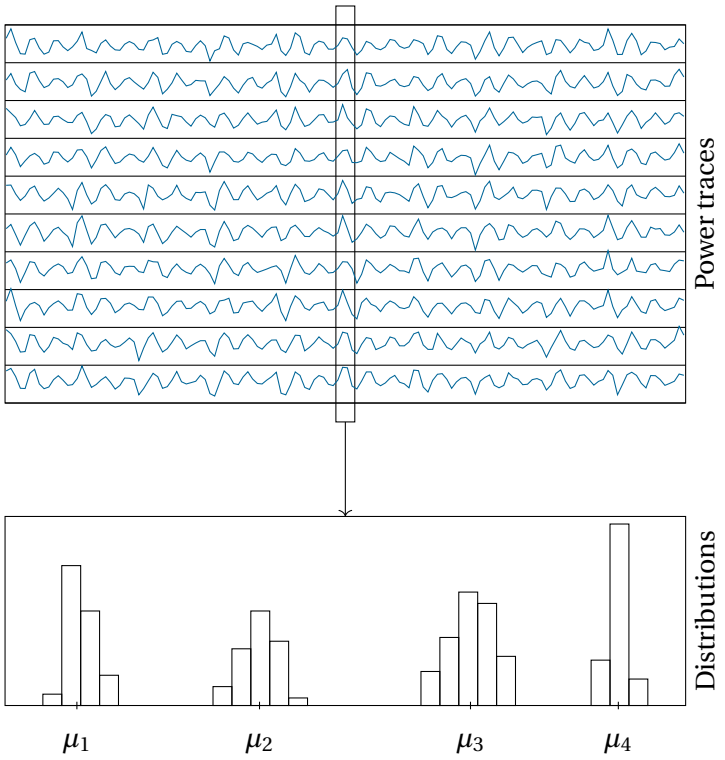


Figure 2.6: Visual representation of a successful clustering power analysis with $k = 4$.

2.1.3.3 Zero-valued power analysis

Zero-valued power analysis is a power analysis technique that aims to deduce secret information by detecting zero values in the power consumption. The analysis relies on the low energy cost of zero-valued operations whose power consumption is expected to exhibit a discernible pattern when sampled. Identifying zero values in the power samples can therefore provide insight into the internal activity of the target device.

In zero-valued power analysis, an adversary leverages the ability to detect zero values in the power consumption by interacting with the device using special inputs that trigger zero-valued operations; resulting thus in an active attack. In particular, the adversary will choose a specific input that causes an operation to process zero operands depending on the value of a small segment of a fixed secret data. By recognizing the zero operands in the power consumption, the adversary can therefore confirm hypothesized portions of a secret value.

There exist many ways to distinguish zero values in a power trace. For instance, [Gou01; AT03] argue that a zero value can be observed by noticing a significant drop of power consumption. In practice, however, this method requires setting a manual threshold based on observing the measured samples.

The overall attack consists of the following steps:

1. Find an operation in the attacked procedure which involves an operand that, given a certain input, can be forced to be zero depending on the value of a small portion of a fixed secret value.
2. Forge a special input that triggers the zero operand depending on a hypothesized value for a small portion.
3. Send the forged input to the device, and collect the power trace that corresponds to the operation.
4. Detect whether the operation was performed with a zero operand or not in the collected trace (with, e.g., a manual inspection of the power samples).

If the zero operand is detected in the power trace, then the hypothesis that led to the forged input can be confirmed (and ruled out otherwise). If applicable, repeating the process over all portions of the secret value recovers the entire secret value.

Collision power analysis.

When the target operation has a known timing and can be forced to process zero and nonzero values regardless of the secret data, a more efficient approach can be employed based on *collision power analysis* [SWP03; MME10]. This technique removes the hassle of detecting zero values manually. In this case, the values processed in a trace are detected by comparing the trace against two baselines (i.e., templates). The two baselines correspond to power traces relating the same execution as the target trace but in which processed values are known to be zero and nonzero.

In practice, a collision power analysis is typically mounted using Pearson's Correlation Coefficient (PCC). This technique correlates a target power trace $T = (s_i \in \mathbb{R})_{0 \leq i < M}$ with a baseline $B^{(b)} = (s_i^{(b)} \in \mathbb{R})_{0 \leq i < M}$ (for $b \in \{0,1\}$) by computing:

$$\mathrm{PCC}(T, B^{(b)}) = \frac{\mathrm{Cov}(T, B^{(b)})}{\sqrt{\mathrm{Var}(T)\mathrm{Var}(B^{(b)})}}.$$

The greater the value of the coefficient, the more correlated the trace is to the baseline. As a result, a zero value is detected when the corresponding trace has a greater correlation coefficient with the zero-valued baseline than with the nonzero one.

To implement collision power analysis in the overall attack, the collection of the two baselines should be added before the first step, so the zero-value detection can be performed with the PCC between the collected traces and the baselines.

2.2 Fault analysis

In the context of electronic devices, a *fault* refers to any kind of disruption that causes the device to deviate from its intended behavior. In the case of a cryptosystem, these faulty behaviors may accidentally reveal sensitive information that compromises the security guarantees of the cryptographic algorithm. As faulty behaviors may occur naturally or be deliberately induced, studying cryptosystems in presence of errors—commonly known as *fault analysis*—is therefore vital.

A common type of faulty behavior with electronic devices is *data corruption* which typically occurs due to environmental circumstances (such as electromagnetic disturbances) that disrupt the data processing or the storage mechanisms of the device. A classic example of a fault attack that leverages the corruption of data is the attack due to Boneh, DeMillo, and Lipton in [BDL97] in which a faulty RSA signature along with its message enables the recovery of the signing key.

Data-corrupting faults come in different kinds. For example, such a fault can be *latent* in case the faulty behavior is due to the introduction of a static alteration in the device, or *transient* in case the faulty behavior is due to a dynamic hitch caused by external factors. Moreover, the faulty behavior induced by a data-corrupting fault is characterized by many features, such as:

- The *granularity* of the data impacted (a single bit vs. whole words).
- The required *control* (a window of instructions in a procedure vs. a specific operation).
- The required *impact* (a bit flip vs. an entire value stuck at zero).

These types and characteristics are achieved through different techniques. For instance, overheating the device is an example of a low-granularity, low-control, and low-impact transient fault injection, while shooting with a focused ion beam on the circuitry of the device injects latent faults that achieve high granularity, high control, and high impact (see [Bar+12]).

Fault injection technique. To conduct a fault analysis, a fault requires to be injected in the device during the execution of a cryptographic operation. One common way to accomplish this is by injecting a glitch (using, e.g., a pulse generator) into the power supply of a device. This technique is known to corrupt the device memory and skip a small number of instructions without damaging the device [Bar+12]. We qualify the output of a device as *faulty* when the derivation of such output is affected by a fault.

In our experimental verifications, we use a setup based on the ChipWhisperer technology to collect faulty results.

Fault injections setup (ChipWhisperer). In addition to the power analysis capabilities introduced in Section 2.1.1.1, the ChipWhisperer toolkit also offers low-cost voltage glitching

technology which enables transient fault injections of precise control, but unpredictable granularity and impact. In particular, the embedded devices featured in the ChipWhisperer toolkit are also exposed to fault analysis, and the one we use is based on the Cortex-M4 which is the microcontroller recommended by NIST for evaluating the implementations of post-quantum candidates [Moo19a]. Since our work is mostly theoretical and does not rely on a sophisticated fault model, we took advantage of the convenience of the ChipWhisperer to provide our proof of concept. Still, our attacks are applicable with additional efforts of marginal complexity on real-world devices.

Our fault framework, exclusively used in Chapter 9, includes:

- A (NAE-CW308T-)STM32F4 board which features an ARM Cortex-M4 microcontroller as the Device Under Test (DUT).
- The Chipwhisperer-Lite Level 2 starter kit:
  - A ChipWhisperer-Lite board which is used both to communicate with the DUT in serial through USB, and inject faults.
  - A ChipWhisperer (NAE-)CW308 UFO board which serves as an intermediary board that enables the ChipWhisperer-Lite and the DUT to exchange signals.
- A general-purpose computer which runs an operating system compatible with the ChipWhisperer framework [New21b].

The STM32F4 is plugged into the CW308 UFO, which is itself connected to the ChipWhisperer-Lite with a 20-pin cable. The `GLITCH`[6] port of the ChipWhisperer-Lite is connected to the `VOUT`[6] port of the CW308. The computer is simply connected to the ChipWhisperer-Lite with a USB to micro-USB cable.

[6] We refer to the official datasheets to find the mentioned pins. NAE-CW308 UFO: http://media.newae.com/datasheets/NAE-CW308-datasheet.pdf, ChipWhisperer-Lite: https://media.newae.com/datasheets/NAE-CW1173_datasheet.pdf.

Isogeny-based cryptography Part I

3 SIKE

Isogeny-based cryptography is a branch of cryptography in which security is based on the hardness of computing isogenies between two given elliptic curves. Informally speaking, an isogeny is a map between two elliptic curves that preserves certain algebraic properties. Isogeny-based cryptography enables a party to conceal secret information into an elliptic curve through the application of a secret isogeny to a public elliptic curve, leading thus to various applications, such as key exchanges, digital signatures, and zero-knowledge proofs.

The most famous isogeny-based cryptosystem is the Supersingular Isogeny Diffie–Hellman (SIDH) protocol which allows two parties to agree on a common secret key through a non-confidential communication. In this scheme, the secret parameters consist of the respective isogenies of the communicating parties (starting from a same preimage curve), while the public parameters consist of the respective image curves of the two isogenies (in addition to special points). By applying their own isogeny to the curve they receive, the two parties obtain a similar elliptic curve which can serve as common secret material in the derivation of a shared secret key.

One of the main advantages of isogeny-based cryptography is its presumed resistance to quantum attacks. As a result, an improved version of SIDH called SIKE (which stands for Supersingular Isogeny Key Encapsulation) was submitted to the NIST post-quantum standardization process. Compared to the other candidates, SIKE offered relatively small keys and ciphertexts, and benefited from years of existing research in Elliptic Curve Cryptography (ECC). However, SIKE had slower runtimes and—because isogeny-based cryptography is a rather new branch of cryptography—had yet to establish the same level of trust as its other post-quantum counterparts (in particular, as lattice-based cryptography).

At the time the work reported in this thesis was conducted, SIKE was still a surviving candidate of the NIST post-quantum standardization process, in which the scheme had advanced to a fourth round of evaluation. Alas, in 2022, the trust suspicion was confirmed when Castryck and Decru's discovery in [CD22] dealt a fatal blow to the SIKE which ended its viability for good. The analysis uses a result due to Kani [Kan97] as an efficient decision tool to correctly guess

the secret isogeny. As a result, SIKE is now considered insecure and should not be used for any practical purpose. Nevertheless, our work on side-channel analysis could inform future cryptographic designs, and so still holds relevance.

History. The concept of an isogeny-based cryptosystem was first considered in [Cou06] by Couveignes. In his note, initially written in 1997, Couveignes describes a key exchange based on the hardness of finding an isogeny between two elliptic curves as a prime example of a hard problem that is independent of any discrete logarithm problem. Nine years later, in [RS06], the same cryptosystem was independently rediscovered by Rostovstev and Stolbunov who present a basic isogeny-based ElGamal encryption and emphasized its quantum resistance (Stolubnov generalized this idea in [Sto10]). This property inspired others to develop novel cryptosystems based on the same mathematical problem, such as an authenticated key agreement [HCH11], a cryptographic hash function [CLG09], and even a random number generator [DJJ10].

So far, isogeny-based cryptosystems considered mostly a specific type of elliptic curves that is called ordinary. However, in [CJS14], Childs, Jao, and Soukharev describe a groundbreaking subexponential quantum attack that can recover the secret isogenies of ordinary curves by exploiting the bijective relationship between the endomorphism ring and the possible isogenies. This weakness was addressed by De Feo and Jao in [DJP11] (the work was later extended with Plût in [DJP14]) who introduced supersingular curves instead of ordinary to make the endomorphism ring non-commutative. While this modification thwarts the Childs–Jao–Soukharev quantum attack, the use of supersingular curves made the key agreement non-commutative as well, which was restored using additional auxiliary points. This, together with various improvements, resulted in the believed to be quantum-resistant isogeny-based variant of Diffie–Hellman called SIDH.

Although the use of auxiliary points in conjunction with supersingular curves resulted in a viable isogeny-based key exchange, this additional information also introduced several vulnerabilities. Most notably, in [Gal+16], Galbraith et al. describe an active attack in which forged auxiliary points lead to the recovery of a private-key bit depending on whether the resulting shared key is still the same between the two parties (which is usually detected with the subsequent secret communication). This attack can be extended to all the private-key bits and proves to be difficult to prevent due to the lack of proper public-key validation. As a response, Jao et al. introduced SIKE [Jao+20] which addresses the above attack by applying the Fujisaki–Okamoto transform (enhanced by Hofheinz, Hövelmanns, and Kiltz in [HHK17]) to SIDH as a generic way to detect forgeries through re-encryption. SIKE was submitted as a candidate for a post-quantum key encapsulation mechanism substitute in the post-quantum standardization process held by NIST and advanced through four rounds of evaluation.

Unfortunately, and as mentioned above, SIKE and SIDH have met their demise due to the efficient key-recovery attack of Castryck and Decru as described in [CD22] (which was improved shortly after in [Wes22], [Rob22], and [Mai+23]). The attack exploits the information obtained from the auxiliary points to confirm small hypothesized components of a secret

isogeny, using a theorem due to Kani as described in [Kan97]. As a result of this attack, all cryptographic isogeny-based schemes that rely on auxiliary points, such as SIDH proofs of knowledge, which were originally proposed in [DJP14] and made into a digital signature scheme in [Yoo+17], have become obsolete. However, other isogeny-based cryptosystems that do not rely on auxiliary points are still surviving. This includes all the key exchanges based on the original Couveignes–Rostovstev–Stolbunov scheme, such as CSIDH [Cas+18], OSIDH [CK20], and CSI-FiSH [BKV19], but also digital signature schemes such as SeaSign [DG19] (improved in [DPV19]), and SQISign [Feo+20].

3.1 Background

This chapter recalls the basic notions of algebraic geometry to introduce SIKE as well as the necessary material to understand our analyses. A reader interested in a full overview of isogeny-based cryptography is advised to read [Feo17], or [Cos19].

3.1.1 Notation

Table 3.1 summarizes the different notations used throughout the following chapters.

Table 3.1: Notations for Chapters 3—6.

Expression	Meaning
$\mathbb{F}_p$	The finite field of (prime) characteristic p.
$\overline{\mathbb{K}}$	The algebraic closure of the field $\mathbb{K}$.
$E(\mathbb{K})$	Elliptic curve defined over the field $\mathbb{K}$.
$\#E$	Number of points in an elliptic curve E.
X_P, Y_P, Z_P	Given a point P, the projective coordinates of P.
x_P, y_P	Given a point P, the affine coordinates of P.
$u \oplus v$	The bitwise eXclusive OR (XOR) of two bitstrings[1] u and v.
$u \,\&\, v$	The bitwise AND of two bitstrings[1] u and v.

3.1.2 Elliptic curves

Definition 3.1.1. Let $p > 3$ be a prime number such that $p \equiv 3 \pmod 4$. We denote by $\mathbb{F}_{p^2}$ the degree-2 extension of $\mathbb{F}_p$ which we define as follows:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[x]/(x^2+1).$$

All elements of $\mathbb{F}_{p^2}$ can be written as $u + i\,v$ where $u, v \in \mathbb{F}_p$ and $i = \sqrt{-1}$.

Definition 3.1.2 (Elliptic curve)**.** Let $p > 3$ be a prime number and $a, b \in \mathbb{F}_{p^2}$ such that $4a^3 + 27b^2 \neq 0$. An *elliptic curve* (in short Weierstraß form) is the set of points in the projective space

[1] The bitstrings are supposed to have the same length. If not, zero bits are usually appended to the shortest bitstring to match the length of the longest bitstring.

of dimension 2 over $\mathbb{F}_{p^2}$ that satisfies a Weierstraß equation, i.e.:

$$E = \{[X : Y : Z] \in \mathbb{F}_{p^2}^3 : Y^2Z = X^3 + aXZ^2 + bZ^3\}.$$

We define $\mathcal{O} = [0 : 1 : 0]$, i.e., the only point on the line $Z = 0$, and refer to it as the *point to infinity*.

The curve can be alternatively defined in the affine plane with affine coordinates defined as $x = X/Z$ and $y = Y/Z$ (for all points such that $Z \neq 0$):

$$E = \{(x, y) \in \mathbb{F}_{p^2}^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}.$$

Definition 3.1.3 (Addition law [Sil86, III.2])**.** Let E be an elliptic curve. The addition of two points $P, Q \in E$ is defined as follows:

1. Let L be the line through P and Q (if $P = Q$, then let L be the tangent line to E and P).
2. Let R be the third (or, if $P = Q$, the other) point of intersection of L with E.
3. Let L' be the line through R and $\mathcal{O}$.

Then L' intersects E at R, $\mathcal{O}$, and a third point. We denote that third point by $P + Q$.

Proposition 3.1.1 (Group structure)**.** *An elliptic curve E forms an abelian group structure under the addition law (3.1.3) and with $\mathcal{O}$ as the identity element.*

Proof. See [Sil86, III, Prop. 2.2]. □

Example. An example of an elliptic curve (exceptionally defined over the field of real numbers instead of $\mathbb{F}_{p^2}$ for visualization purposes) along with its group structure is shown in Figure 3.1.

Definition 3.1.4 (Point negation)**.** Let E be an elliptic curve and $P = [X : Y : Z] \in E$. We define $-P$ as the point R such that $P + R = \mathcal{O}$, i.e., $-P = [X : -Y : Z]$. Consequently, given two points $P, Q \in E$, we have that $P - Q = P + (-Q)$.

Definition 3.1.5 (Scalar multiplication)**.** Let E be an elliptic curve. The *multiplication of $P \in E$ by the scalar $n \in \mathbb{N}$* is defined by repeatedly adding P with itself n times, i.e.:

$$\begin{aligned} [n]P &= \underbrace{P + P + \cdots + P}_{n \text{ times}}, \\ [-n]P &= -([n]P). \end{aligned}$$

Definition 3.1.6 (Point order)**.** Let $E(\mathbb{F}_{p^2})$ be an elliptic curve defined over $\mathbb{F}_{p^2}$ and $P \in E(\mathbb{F}_{p^2})$. The *order of P*, denoted by $\text{ord}(P)$, is the smallest $q > 0$ such that $[q]P = \mathcal{O}$.

Definition 3.1.7 (Generator point)**.** Let E be an elliptic curve and $P \in E$. The *subgroup generated by P* is denoted by $\langle P \rangle = \{[k]P : k \in \mathbb{Z}\}$.

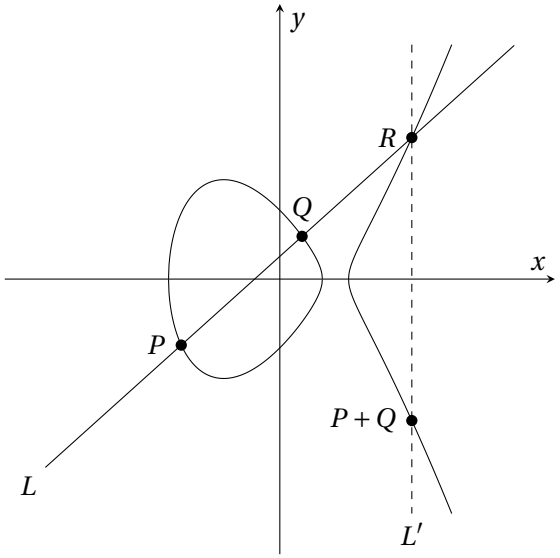


Figure 3.1: Illustration of the addition law on an elliptic curve in the real plane.

Definition 3.1.8 (Torsion subgroup)**.** Let $E(\mathbb{F}_{p^2})$ be an elliptic curve defined over $\mathbb{F}_{p^2}$ and $m \in \mathbb{Z}$ such that $m \geq 1$. The *m-torsion subgroup* of E is defined as the set of all points in $E(\overline{\mathbb{F}}_{p^2})$ whose order divides m, i.e.:

$$E[m] = \{P \in E(\overline{\mathbb{F}}_{p^2}) : [m]P = \mathcal{O}\}.$$

If $E[m] \cong (\mathbb{Z}/(m\mathbb{Z}))^r$, we say that $E[m]$ has rank r.

Definition 3.1.9 (Basis)**.** Let $E(\mathbb{F}_{p^2})$ be an elliptic curve defined over $\mathbb{F}_{p^2}$ and $E[m]$ a torsion subgroup of rank r for $m \in \mathbb{Z}$ with $m \geq 1$. The points $P_1, \ldots, P_r \in E[m]$ form a *basis* of $E[m]$ if the points are linearly independent, i.e., if $[k_1]P_1 + \cdots + [k_r]P_r \neq \mathcal{O}$ for any $0 \leq k_i < \text{ord}(P_i)$ that are not all zero (for $1 \leq i \leq r$). In this case, $E[m] = \langle P_1, \ldots, P_r \rangle = \{[k_1]P_1 + \cdots + [k_r]P_r : k_1, \ldots, k_r \in \mathbb{Z}\}$.

Theorem 3.1.1 (Torsion subgroups structure)**.** *Let E be an elliptic curve defined over a field of characteristic $p > 0$ and $m \in \mathbb{Z}$ with $m \neq 0$. The m-torsion subgroups are entirely characterized and we have:*

1. $E[m] \cong \mathbb{Z}/(m\mathbb{Z}) \times \mathbb{Z}/(m\mathbb{Z})$ *if* $p \nmid m$.
2. *One of the following is true:*
   - $E[p^e] \cong \mathbb{Z}/(p^e\mathbb{Z})$ *for all* $e = 1, 2, 3, \ldots$.
   - $E[p^e] \cong \{\mathcal{O}\}$ *for all* $e = 1, 2, 3, \ldots$.

Proof. See [Sil86, III, Cor. 6.4]. □

Definition 3.1.10 (Supersingular curve)**.** An elliptic curve $E(\mathbb{F}_{p^2})$ over $\mathbb{F}_{p^2}$ is called *supersingular* if $E[p] \cong \{\mathcal{O}\}$ (and *ordinary* if $E[p] \cong \mathbb{Z}/(p\mathbb{Z})$).

Remark 1. There are $\#E(\mathbb{F}_{p^2}) = (p+1)^2$ points in the supersingular curve $E(\mathbb{F}_{p^2})$.

Definition 3.1.11 (Montgomery elliptic curve [Mon87])**.** Let $p > 3$ be a prime number, $\alpha, \beta \in \mathbb{F}_{p^2}$ such that $\beta(\alpha^2 - 4) \neq 0$, and $\mathcal{O}$ the point to infinity. A *Montgomery elliptic curve* (or, simply, a *Montgomery curve*) is an elliptic curve over $\mathbb{F}_{p^2}$ that satisfies the following equation:

$$E = \{(x, y) \in \mathbb{F}_{p^2}^2 : \beta y^2 = x^3 + \alpha x^2 + x\} \cup \{\mathcal{O}\}.$$

In particular, we are interested in Montgomery curves where $\beta = 1$.

Remark 2. Montgomery curves allow for compact representation of points, up to sign, by using only the X and the Z coordinates. In particular, a point $P = [X_P : Y_P : Z_P] \neq [0 : 1 : 0]$ can be represented by a single field element $x_s = X_P / Z_P \in \mathbb{F}_{p^2}$. The value $[X_P : Z_P] = [x : 1]$ uniquely defines $\{\pm P\}$, and we write $P = [x : 1]$.

Remark 3. Let E be a Montgomery curve with $\beta = 1$. Given Q, P, and $Q - P$ in E, the coefficient α of the curve can be recovered from the three points as follows:

$$\alpha = \frac{1 - x_Q x_P - x_Q x_{Q-P} - x_P x_{Q-P}}{4 x_Q x_P x_{Q-P}} - x_Q - x_P - x_{Q-P}.$$

3.1.3 Isogenies

In isogeny-based cryptography, we are interested in morphisms (i.e., maps) between elliptic curves that preserve both their group structures and their properties as algebraic varieties.

Definition 3.1.12 (Homomorphism)**.** Let G_1, G_2 be two additively written groups. A *homomorphism* $\phi : G_1 \to G_2$ is a morphism that preserves group structure, i.e., $\phi(u + v) = \phi(u) + \phi(v)$ for all $u, v \in G_1$.

Definition 3.1.13 (Isomorphism)**.** Let G_1, G_2 be two additively written groups. An *isomorphism* $\phi : G_1 \to G_2$ is a bijective homomorphism, i.e., for all $u_1, u_2 \in G_1$, $u_1 \neq u_2$ implies $\phi(u_1) \neq \phi(u_2)$, and for all $v \in G_2$, there is a $u \in G_1$ such that $\phi(u) = v$. In this case, we say that G_1 is *isomorphic* to G_2.

Definition 3.1.14 (j-invariant)**.** Let $E = \{(x, y) \in \mathbb{F}_{p^2}^2 : y^2 = x^3 + ax + b\} \cup \{\mathcal{O}\}$ be an elliptic curve. The *j-invariant* of E is defined as the following quantity:

$$j = 1728 \frac{4a^3}{4a^3 + 27b^2}.$$

Remark 1. All isomorphic curves share the same j-invariant (see [Sil86, III, Prop. 1.4, (b)] for proof).

Definition 3.1.15 (Kernel)**.** Let $\phi : G_1 \to G_2$ be a homomorphism. The *kernel of ϕ* is the set of preimages in G_1 that are mapped to the identity element of G_2, i.e., $\ker(\phi) = \{u \in G_1 : \phi(u) = 0\}$.

Definition 3.1.16 (Isogeny)**.** Let E_1, E_2 be two elliptic curves. We define an *isogeny* $\varphi : E_1 \to E_2$ as a surjective homomorphism with finite kernel. In this case, we say that E_1 is *isogenous* to E_2.

Example. A trivial isogeny in which $E_1 = E_2 = E$ is the map defined by the scalar multiplication by $m \in \mathbb{Z}$ (in which case, the kernel is simply the m-torsion of E, which is finite):

$$\begin{array}{rccc} \varphi: & E & \to & E \\ & P & \mapsto & [m]P. \end{array}$$

As we are working only on elliptic curves defined over $\mathbb{F}_{p^2}$, we consider only *separable*[2] isogenies. In our case, this means that every isogeny is uniquely defined by its kernel [Sil86, III, Prop. 4.12], and that the mapping can be entirely determined with Vélu's formulas [Vél71].

Definition 3.1.17 (Vélu's formulas [Feo17])**.** Let E be an elliptic curve and G a finite subgroup of E. The separable isogeny $\varphi : E \to E/G$ of kernel G can be written as:

$$\begin{array}{rccc} \varphi: & E & \to & E/G \\ & P & \mapsto & \Big(x_P + \sum\limits_{Q \in G \setminus \{\mathcal{O}\}} (x_{P+Q} - x_Q), \quad y_P + \sum\limits_{Q \in G \setminus \{\mathcal{O}\}} (y_{P+Q} - y_Q)\Big). \end{array}$$

The image curve E/G has equation $y^2 = x^3 + a'x + b'$ where

$$\begin{cases} a' &= a - 5 \sum\limits_{Q \in G \setminus \{\mathcal{O}\}} (3x_Q + a), \\ b' &= b - 7 \sum\limits_{Q \in G \setminus \{\mathcal{O}\}} (5x_Q^3 + 3ax_Q + b). \end{cases}$$

Definition 3.1.18 (Degree of isogeny)**.** Let $\varphi : E_1 \to E_2$ be a separable isogeny. We define the *degree of* φ as the cardinality of its kernel, i.e., $\deg(\varphi) = \#\ker(\varphi)$.

Theorem 3.1.2 (Isogeny composition theorem)**.** *Let $\phi : E_1 \to E_2$ and $\psi : E_1 \to E_3$ be separable isogenies such that* $\ker(\phi) \subset \ker(\psi)$*. Then, there exists a unique isogeny $\lambda : E_2 \to E_3$ satisfying $\psi = \lambda \circ \phi$.*

Proof. See [Sil86, III, Cor. 4.11]. □

Theorem 3.1.2 states that, given their kernel G, every separable isogeny $\varphi : E \to E/G$ can be computed as a composition of isogenies of smaller degree. Suppose $\deg(\varphi) = p_1^{e_1} \cdots p_r^{e_r}$, the kernel G can be decomposed as $G_1 \times \cdots \times G_r \subseteq G$ where $\#G_i = p_i^{e_i}$ (for $1 \le i \le r$), and the overall isogeny can be obtained by the composition chain of $E \to E/G_1 \to (E/G_1)/G_2 \to \cdots \to E/G$.

3.1.4 Key encapsulation mechanism

A key encapsulation mechanism enables a party to confidentially communicate a secret key to another party through an authenticated channel. The construction is asymmetric which means that the two parties will follow distinct procedures.

[2] Separable isogenies are isogenies defined over separable field extensions, i.e., extensions for which the roots of their minimal polynomial are distinct in their algebraic closure (see [Coh02, §7.4] for details).

Definition 3.1.19 (Key encapsulation mechanism)**.** A *key encapsulation mechanism* (KEM) is a triple of probabilistic algorithms $\texttt{KeyGen}(n)$, $\texttt{Encaps}(k, \text{PK})$, $\texttt{Decaps}(c, \text{SK})$ that achieves *confidentiality* of a shared secret key k through public-key encryption.

The algorithms of a key encapsulation mechanism must fulfill the following properties:

- $\texttt{KeyGen}(n)$ generates a random *key pair* (SK, PK) of security parameter n where:
  - SK is a *private key* used to decrypt secret keys.
  - PK is the *public key* corresponding to SK used to encrypt secret keys.
- $\texttt{Encaps}(k, \text{PK})$ encapsulates a secret key k with the public key PK to produce a ciphertext c that corresponds to k.
- $\texttt{Decaps}(c, \text{SK})$ decapsulates the secret key k from c with the private key SK (supposing that c validly corresponds to k).

The security parameter n ensures that there is no probabilistic algorithm that compromises the confidentiality of the key encapsulation scheme in time polynomial nor subexponential in n.

Moreover, we say that the KEM is *ephemeral* if a new key pair is generated for each communication, and *semi-static* if the same key pair can be re-used across all communications (and *fully static* if both the key pair and the encapsulated key are always the same).

3.2 SIDH

The Supersingular Isogeny-based Diffie–Hellman (SIDH) is a key exchange scheme based on the Diffie–Hellman protocol and constitutes the building block of the key encapsulation mechanism in SIKE. The scheme enables two parties (Alice and Bob) to securely agree on a same secret value without prior common secret information. The security of the scheme relies on the hardness of finding an isogeny (without its kernel) between two supersingular isogenous elliptic curves.

In SIDH, Alice and Bob share the image curve of their respective secret isogeny that is generated by a secret kernel, which is itself generated by a secret generator point of a common public curve. By applying their respective isogeny to their counterpart's image curve, Alice and Bob arrive at a same image curve (up to an isomorphism), as illustrated in Figure 3.2.

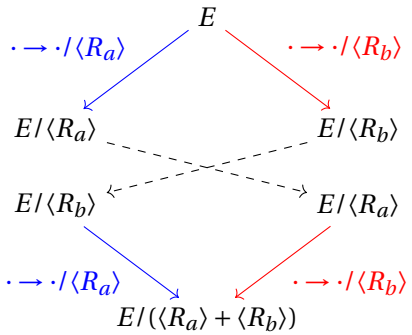


Figure 3.2: Illustration of the SIDH protocol.

3.2.1 Key exchange

Parameters. Alice and Bob parameterize SIDH with the following:

- ℓ_a, ℓ_b : two (small) prime numbers.
- e_a, e_b : two security parameters such that $\ell_a^{e_a}\ell_b^{e_b} - 1$ is prime and $\ell_a^{e_a} \approx \ell_b^{e_b}$.
- $E(\mathbb{F}_{p^2})$: a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ where $p = \ell_a^{e_a}\ell_b^{e_b} - 1$.
- P_a, Q_a : a basis for the ℓ_a-torsion subgroup in $E(\mathbb{F}_{p^2})$.
- P_b, Q_b : a basis for the ℓ_b-torsion subgroup in $E(\mathbb{F}_{p^2})$.

Usually, we use $\ell_a = 2$ and $\ell_b = 3$. This leads to prime numbers $p = 2^{e_a}3^{e_b} - 1$ whose distribution is dense enough for our applications according to the results of Lagarias and Odlyzko in [LO77].

Private computation. Both parties conceal their respective private information through an isogeny with the following steps:

Alice generates her parameters as follows:
1. Choose secret $1 < k_a^{(P)}, k_a^{(Q)} < 2^{e_a}$.
2. $R_a \leftarrow [k_a^{(P)}]P_a + [k_a^{(Q)}]Q_a$.
3. Let $\varphi_a : E \to E/\langle R_a \rangle$.

Alice's private information is $k_a^{(P)}, k_a^{(Q)}$.

Bob generates his parameters as follows:
1. Choose secret $1 < k_b^{(P)}, k_b^{(Q)} < 3^{e_b}$.
2. $R_b \leftarrow [k_b^{(P)}]P_b + [k_b^{(Q)}]Q_b$.
3. Let $\varphi_b : E \to E/\langle R_b \rangle$.

Bob's private information is $k_b^{(P)}, k_b^{(Q)}$.

Public exchange. Both parties exchange their public information through an authenticated channel with the following steps:

Alice sends $(E/\langle R_a \rangle, \varphi_a(P_b), \varphi_a(Q_b))$.

Bob sends $(E/\langle R_b \rangle, \varphi_b(P_a), \varphi_b(Q_a))$.

Key agreement. Both parties agree on a common secret through an isogeny composition with the following steps:

Given $(E/\langle R_b\rangle, \varphi_b(P_a), \varphi_b(Q_a))$, Alice computes the common secret as follows:

1. $S_a \leftarrow [k_a^{(P)}]\varphi_b(P_a) + [k_a^{(Q)}]\varphi_b(Q_a)$.
2. Let $\varphi'_a : E/\langle R_b\rangle \to (E/\langle R_b\rangle)/\langle S_a\rangle$.
3. Let K be the j-invariant of $(E/\langle R_b\rangle)/\langle S_a\rangle$.

Given $(E/\langle R_a\rangle, \varphi_a(P_b), \varphi_a(Q_b))$, Bob computes the common secret as follows:

1. $S_b \leftarrow [k_b^{(P)}]\varphi_a(P_b) + [k_b^{(Q)}]\varphi_a(Q_b)$.
2. Let $\varphi'_b : E/\langle P_a\rangle \to (E/\langle R_a\rangle)/\langle S_b\rangle$.
3. Let K be the j-invariant of $(E/\langle R_a\rangle)/\langle S_b\rangle$.

The j-invariant is the same for both parties, since the curves $(E/\langle R_b\rangle)/\langle S_a\rangle$ and $(E/\langle R_a\rangle)/\langle S_b\rangle$ are necessarily isomorphic to each other. This result arises from $\varphi'_a : E \to (E/\langle R_b\rangle)/\langle S_a\rangle$ (resp. $\varphi'_b : E \to (E/\langle R_a\rangle)/\langle S_b\rangle$) being identical to $\varphi''_a : E \to E/\langle R_b, R_a\rangle$ (resp. $\varphi''_b : E \to E/\langle R_a, R_b\rangle$) due to Theorem 3.1.2 (and since $\langle R_b, R_a\rangle = \langle R_a, R_b\rangle$).

3.3 SIKE

SIKE (Supersingular Isogeny Key Encapsulation) is a KEM which is based on SIDH and the Fujisaki–Okamoto transform[3].

3.3.1 Key exchange

Parameters. SIKE is parameterized with the following:

- $\eta > 0$: a message length.
- $p = 2^{e_2}3^{e_3} - 1$: a prime number parameterized by $e_2, e_3 > 2$ such that $2^{e_2} \approx 3^{e_3}$.
- $\alpha \in \mathbb{F}_{p^2}$: the coefficient of a supersingular Montgomery curve $E_0(\mathbb{F}_{p^2})$ ($\beta = 1$).
- $x_{Q_2}, x_{P_2}, x_{Q_2-P_2}$: the affine x-coordinates of a basis for the 2-torsion subgroup.
- $x_{Q_3}, x_{P_3}, x_{Q_3-P_3}$: the affine x-coordinates of a basis for the 3-torsion subgroup.

Note that SIKE instantiates SIDH with $\ell_2 = 2$, $\ell_3 = 3$, e_2, e_3, $E_0(\mathbb{F}_{p^2})$, P_2, Q_2, P_3, Q_3.

Standard SIKE instances are all instantiated with $\alpha = 6$. The standard parameters for e_2, e_3, and η are shown in Table 3.2. The public generator points P_2, Q_2, and P_3, Q_3 are obtained with a deterministic procedure described in [Jao+20] and are therefore fixed for each parameter set (refer to [Jao+20] for their actual values).

Key generation. A SIKE key pair $(\mathrm{SK}_b, \mathrm{PK}_b)$ is generated with the following steps:

1. Pick $1 < \mathrm{SK}_b < 3^{e_3}$ uniformly at random.
2. $R_b \leftarrow P_b + [\mathrm{SK}_b]Q_b$.
3. Let $\varphi_b : E_0 \to E_b$ be such that $\ker(\varphi_b) = \langle R_b\rangle$.
4. Return $\mathrm{PK}_b = (E_b, \varphi_b(P_a), \varphi_b(Q_a))$.

[3]The Fujisaki–Okamoto transform adapts a protocol in a way that prevents the two parties from cheating (see [FO99] for details).

Note that PK_b is actually encoded as the affine x-coordinates $x_{\varphi_b(P_a)}$, $x_{\varphi_b(Q_a)}$, and $x_{\varphi_b(Q_a)-\varphi_b(P_a)}$, since the coefficient defining E_b can be recovered from these values (see Remark 3).

Instance	Security level	e_2	e_3	η
SIKEp434	1	216	137	128
SIKEp503	2	250	159	192
SIKEp610	3	305	192	192
SIKEp751	5	372	239	256

Table 3.2: Standard SIKE parameter sets, as submitted in the third round of NIST's post-quantum standardization process [Jao+20]. The security levels correspond to security requirements established by NIST (see [NIS16]).

Key encapsulation. Let $F : \mathbb{F}_{p^2} \to \{0,1\}^\eta$, $G : \{0,1\}^\eta \times \mathbb{F}_{p^2}^3 \to \{0,1\}^{e_2}$, and $H : \{0,1\}^\eta \times \mathbb{F}_{p^2}^3 \times \{0,1\}^\eta$ be three cryptographic hash functions[4]. Given a SIKE public key $\text{PK}_b = (E_b, \varphi_b(P_a), \varphi_b(Q_a))$, the scheme encapsulates a secret key with the following steps:

1. Pick $m \in \{0,1\}^\eta$ uniformly at random.
2. Let SK_a be the conversion of $G(m, \text{PK}_b)$ into an integer.
3. $R_a \leftarrow P_a + [\text{SK}_a]Q_a$.
4. Let $\varphi_a : E_0 \to E_a$ be such that $\ker(\varphi_a) = \langle R_a \rangle$.
5. $R'_a \leftarrow \varphi_b(P_a) + [\text{SK}_a]\varphi_b(Q_a)$.
6. Let $\varphi'_a : E_b \to E_{ab}$ be such that $\ker(\varphi_a) = \langle R_a \rangle$.
7. Let j be the j-invariant of the curve E_{ab}.
8. Return $c_0 = (E_a, \varphi_a(P_b), \varphi_a(Q_b))$, and $c_1 = F(j) \oplus m$,
   and let $K = H(m, c_0, c_1)$ be the shared secret key.

Note that c_0 is actually encoded as the affine x-coordinates $x_{\varphi_b(P_b)}$, $x_{\varphi_b(Q_b)}$, and $x_{\varphi_b(Q_b)-\varphi_b(P_b)}$, since the coefficient defining E_a can be recovered from these values (see Remark 3).

Key decapsulation. Let $F : \mathbb{F}_{p^2} \to \{0,1\}^\eta$, $G : \{0,1\}^\eta \times \mathbb{F}_{p^2}^3 \to \{0,1\}^{e_2}$, and $H : \{0,1\}^\eta \times \mathbb{F}_{p^2}^3 \times \{0,1\}^\eta$ be the same cryptographic functions as the ones used during key encapsulation. Given a ciphertext $c_0 = (E_a, \varphi_a(P_b), \varphi_a(Q_b))$ and $c_1 = F(j) \oplus m$ encrypted with PK_b corresponding to SK_b, the scheme decapsulates a secret key with the following steps:

1. Pick $s \in \{0,1\}^\eta$ uniformly at random.
2. $R'_b \leftarrow \varphi_a(P_b) + [\text{SK}_b]\varphi_a(Q_b)$.
3. Let $\varphi_b : E_b \to E'_{ab}$ be such that $\ker(\varphi_b) = \langle R_b \rangle$.
4. Let j be the j-invariant of the curve E_{ab}.
5. Let $m' = F(j) \oplus c_1$.

[4] In practice, F, G, and H are all instantiated with SHAKE256 [NIS15] where all inputs are converted into bits and concatenated with each other.

6. Let SK'_a be the conversion of $G(m', \mathrm{PK}_b)$ into an integer.
7. $R'_a \leftarrow P_a + [\mathrm{SK}'_a]Q_a$.
8. Let $\varphi''_a : E_0 \to E_a$ be such that $\ker(\varphi'_a) = \langle R'_a \rangle$.
9. If $(E'_a, \varphi'(P_b), \varphi'(Q_b)) = c_0$, then
   (a) Let $K = H(m', c_0, c_1)$ be the shared secret key.

   Else
   (b) Let $K = H(s, c_0, c_1)$ be the shared secret key.

Compressed SIKE. Rather than providing the image points for Q_a and P_a (resp. Q_b and P_b) in the public elliptic curve E_b (resp. E_a), the key generation (resp. the key encapsulation) can instead provide elements of $\mathbb{Z}/(2^{e_2}\mathbb{Z}) \times \mathbb{Z}/(2^{e_2}\mathbb{Z})$ (resp. elements of $\mathbb{Z}/(3^{e_3}\mathbb{Z}) \times \mathbb{Z}/(3^{e_3}\mathbb{Z})$) that deterministically generate $E_b[2^{e_2}]$ (resp. $E_a[3^{e_3}]$). This is because these two representations are isomorphic due to Theorem 3.1.1 (see [Aza+16, §4.1] for details). This approach saves around 35% of public key and ciphertext sizes at the cost of slower algorithms. Additional compression techniques that further reduce the sizes of the public key and ciphertext, and improve the runtime of the compression algorithms were developed in [Cos+17; Zan+18].

Implementation. In the scope of this thesis, we specifically consider the SIKE implementation from [Seo+20]. This implementation is the official adaptation to the 32-bit ARM Cortex-M4 of SIKE and is included in the official submission package.

3.3.2 The three-point ladder

The *three-point ladder* is a procedure which takes three points Q, P, $Q-P$ in an elliptic curve to efficiently compute the point $R = P + [\mathrm{SK}]Q$ where SK is the private key of the computing party. In SIKE, the three-point ladder is used to compute the generator point of the secret kernel subgroup.

Algorithm. The three-point ladder algorithm is shown in Algorithm 3.1. The procedure scans the bits of the private key SK from the least to the most significant bit. For each bit, Q is either added to P if the bit is one, or to $Q-P$ if the bit is zero. The point Q is always doubled at the end of each iteration.

Implementation. As SIKE works with Montgomery curves, the implementation of the three-point ladder considered in this thesis (shown in Listing A.1) takes as input only the affine x-coordinates of Q, P, and $Q-P$, which are respectively stored as $[x:1]$ (see Remark 2) in three variables: `R0`, `R`, and `R2`. From these coordinates, the α coefficient of the curve is recovered (see Remark 3). The conditional branching over the bits of SK is done by swapping the point `R` with `R2` depending on the difference between the current and the previous private bits in the for-loop, so `R0` can always be added to `R2`.

Algorithm 3.1 The three-point ladder.

Input: $\text{SK} = (b_0, \ldots, b_{n-1})$ – the bits of the private key.
Input: E – an elliptic curve.
Input: $(Q, P, Q-P)$ – three points in E.
Output: $P + [\text{SK}]Q$.

1: **for each** b_i **in** SK **do**
2: **if** $b_i = 1$ **then**
3: $(Q, P, Q-P) \leftarrow \texttt{xDBLADD}(Q, Q-P, P, E)$. ▷ $(Q, P, Q-P) \leftarrow ([2]Q, P+Q, Q-P)$
4: **else**
5: $(Q, P, Q-P) \leftarrow \texttt{xDBLADD}(Q, P, Q-P, E)$. ▷ $(Q, P, Q-P) \leftarrow ([2]Q, (Q-P)+Q, P)$
6: **return** P.

Double-and-add procedure. The double-and-add implementation (shown in Listing A.2) takes as input the points R0, R, and R2 such that R2 = R0 − R, along with the curve-defining coefficient α (A24 in the source code), and computes ([2]R0, R + R0, R2) which is respectively stored in (R0, R, R2).

Swapping procedure. To perform the conditional swapping of points, a function `swap_points` (shown Listing A.6) takes R and R2 as parameters, as well as a mask that expands the value of the private bit difference on a whole word. Given two consecutive private-key bits b_{i-1}, b_i (for $0 \leq i < n$ with $b_{-1} = b_n = 0$), with 32-bit words, such a mask corresponds to:

$$\texttt{mask} = \begin{cases} \texttt{0x00000000} & \text{if } b_{i-1} \oplus b_i = 0, \\ \texttt{0xFFFFFFFF} & \text{if } b_{i-1} \oplus b_i = 1. \end{cases}$$

Then, each word of the two elliptic curve points (resp. u and v) is processed according to this formula:

$$\begin{aligned} \texttt{tmp} &= \texttt{mask} \,\&\, (u \oplus v), \\ u &= \texttt{tmp} \oplus u, \\ v &= \texttt{tmp} \oplus v. \end{aligned}$$

Such a procedure is constant in timing and execution (see [CS18] for proof).

3.3.3 Strategies

In order to secretly exchange information, SIKE relies on the derivation of a secret isogeny defined by the kernel generated by a given secret point of smooth order. Deriving the isogeny all at once is computationally heavy, as the computations require visiting all the points in the kernel. Instead, the isogeny derivation can be optimized by composing a chain of isogenies of smaller degrees.

Let R_0 be the secret generator point such that $\text{ord}(R_0) = \ell^e$. The goal is to obtain $\varphi : E_0 \to E_0/\langle R_0 \rangle$ from the decomposition of φ into isogenies of degree ℓ. Such isogenies can be derived with Vélu's formulas (see Definition 3.1.17) using (as kernel) the ℓ-torsion of the subgroup

generated by the projection of R_0 in the preimage curve. The procedure therefore aims to efficiently compute ℓ-torsions in each intermediate curve so to navigate from E_0 to $E/\langle R_0 \rangle$ through ℓ-degree isogenies.

In the following, $\varphi_i : E_i \to E_{i+1}$ denotes the isogeny of degree ℓ where $E_{i+1} = E_i/\langle [\ell^{e-i-1}]R_i \rangle$ for $0 \leq i < e$. Each φ_i can be computed only when $[\ell^{e-i-1}]R_i$ is known. Once φ_i is known, the map can be used to project any point from E_i onto E_{i+1} including, in particular, the previous points obtained in the computations that led to $[\ell^{e-i-1}]R_i$. As a result, there are many ways to compute the generator point $[\ell^{e-i-1}]R_i$ in each curve, and so to derive the overall isogeny.

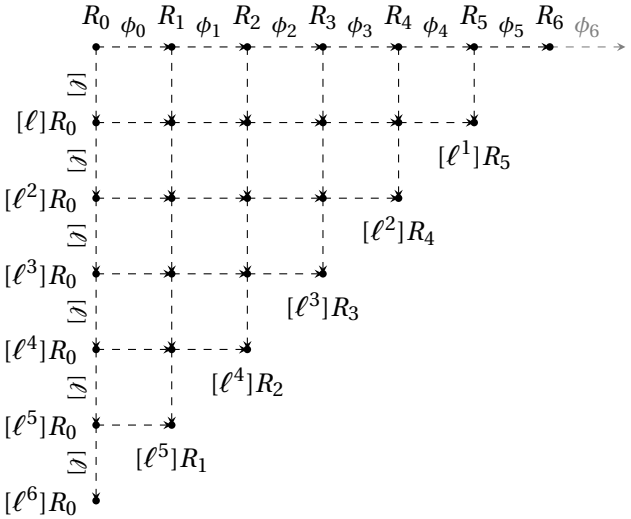


Figure 3.3: Computational structure of $\varphi = \varphi_6 \circ \cdots \circ \varphi_0$.

Figure 3.3 illustrates all the possible ways of deriving the ℓ-degree isogenies from R_0 when $e = 7$. In this figure, each vertex corresponds to a point in an elliptic curve, and each arrow represents an operation. In particular, given a point P on the graph:

- A downwards arrow represents a scalar multiplication by ℓ on P. As a result, each vertical line $0 \leq i < e$ corresponds to operations in E_i, and each vertex on this line corresponds to points in E_i.
- A rightwards arrow represents the application of φ_i to P ($0 \leq i < e$). As a result, each horizontal line $0 \leq j < e$ corresponds to the mapping of the points from E_0 to their counterparts in the different E_i (for $0 \leq i < e - j$).

Note that all extensions of line beyond its endpoint would lead to $\mathcal{O}$. To derive φ, the computing party starts from R_0 in the upper-left corner and needs to reach R_6 in the upper-right corner. The computing party can move freely downwards and upwards, but can move rightwards only when the point at the bottom of the current vertical line is reached. The goal is to walk to the upper-right corner with as few moves as possible.

We call a *strategy* a correct walk from R_0 to R_{e-1} in a graph as the one shown in Figure 3.3. The walk is correct if the computing party follows the rules described above (see [DJP14, §4.2.2] for

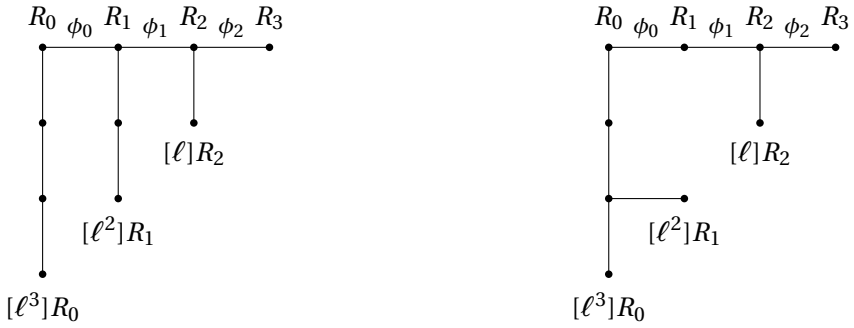
formal details). If followed correctly, a strategy can then be encoded as the set of the visited vertices.

Example. Figure 3.4 shows two concrete examples of a strategy:

(a) A straightforward strategy which simply aims to unlock ϕ_i by multiplying R_i by ℓ^{e-i-1}, and then project R_i onto E_{i+1} to repeat the process (for all $0 \leq i < e$).

(b) An optimized strategy which projects $[\ell^2]R_0$ onto E_1 with φ_0 to obtain $[\ell^2]R_1$, so $[\ell]R_1$ does not need to be visited.

Notice that the graph in Figure 3.4a possesses one more edge than the graph in Figure 3.4b, and would thus require one more operation to execute. On the other hand, Figure 3.4b needs to save $[\ell^2]R_0$ when deriving $[\ell^3]R_0$.



(a) Straightforward strategy.

(b) Optimized strategy.

Figure 3.4: Two different strategies for computing the same isogenies ($e = 4$).

Further optimizations can be considered if, for instance, the computational complexity of the scalar multiplication is lower than the computational complexity of the isogeny application (see [DJP14] for details). In SIKE, optimal strategies are determined once and fixed for each parameter set.

3.3.4 Formulas

As SIKE works with Montgomery curves, certain operations can be performed more efficiently than on a regular elliptic curve. In this section, we give the formulas (as described in [Jao+20]) for these operations.

Let E be a Montgomery curve defined over $\mathbb{F}_{p^2}$ by $\alpha \in \mathbb{F}_{p^2}$ ($\beta = 1$). We denote by $[A : C]$ (for $C \neq 0$) the projective representation of $\alpha = A/C$.

Moreover, given $[A : C]$, we define the additional representations:

- $[A_{24}^+ : A_{24}^-] = [A+2C : A-2C]$ (for $A_{24}^+ \neq A_{24}^-$) which represents $\alpha = 2(A_{24}^+ + A_{24}^-)/(A_{24}^+ - A_{24}^-)$.
- $[A_{24}^+ : C_{24}] = [A+2C : 4C]$ (for $C_{24} \neq 0$) which represents $\alpha = (4A_{24}^+ - 2C_{24})/2C_{24}$.

Point doubling. Given a point $P = [X : Z] \in E$ defined by $[A_{24}^+ : C_{24}]$, the doubling operation is in practice computed as follows:

$$[2]P \quad = \quad [(C_{24}(X^2 - Z^2)^2 : 4XZ(C_{24}(X - Z)^2 + 4A_{24}^+ XZ)].$$

Point tripling. Given a point $P = [X : Z] \in E$ defined by $[A_{24}^+ : A_{24}^-]$, the tripling operation is in practice computed as follows:

$$\begin{aligned}[3]P \quad = \quad & [X(A_{24}^+(X+Z)^4 - A_{24}^-(X-Z)^4 - 2(X^2 - Z^2)(A_{24}^+(X+Z)^2 - A_{24}^-(X-Z)^2))^2 : \\ & Z(A_{24}^+(X+Z)^4 - A_{24}^-(X-Z)^4 + 2(X^2 - Z^2)(A_{24}^+(X+Z)^2 - A_{24}^-(X-Z)^2))^2]\end{aligned}$$

Point addition. Given three points $Q = [X_Q : Z_Q]$, $P = [X_P : Z_P]$, and $Q - P = [X_{Q-P} : Z_{Q-P}]$ in E, the addition of Q with P is in practice computed as follows:

$$Q + P \quad = \quad [Z_{Q-P}(X_P X_Q - Z_P Z_Q)^2 : X_{Q-P}(X_P Z_Q - X_Q Z_P)^2].$$

2-isogeny computation. Given the generator point $R_2 = [X_2 : Z_2] \in E$ such that $\#\langle R_2 \rangle = 2$, the isogeny φ defined by $\ker(\varphi) = \langle R_2 \rangle$ is computed as follows:

$$\begin{aligned}\varphi : \quad E \quad & \to \quad E' \text{ defined by } [A_{24}^+ : C_{24}] = [Z_2^2 - X_2^2 : Z_2^2] \\ [X : Z] \quad & \mapsto \quad [X(XX_2 - ZZ_2) : Z(XZ_2 - ZX_2)].\end{aligned}$$

3-isogeny computation. Given the generator point $R_3 = [X_3 : Z_3] \in E$ such that $\#\langle R_3 \rangle = 3$, the isogeny φ defined by $\ker(\varphi) = \langle R_3 \rangle$ is computed as follows:

$$\begin{aligned}\varphi : \quad E \quad & \to \quad E' \text{ defined by } [A_{24}^+ : C_{24}] = \\ & \qquad [(3X_3 - Z_3)^3(X_3 + Z_3) : (3X_3 + Z_3)^3(X_3 - Z_3)]. \\ [X : Z] \quad & \mapsto \quad [X(XX_3 - ZZ_3)^2 : Z(XZ_3 - ZX_3)^2].\end{aligned}$$

4-isogeny computation. Given the generator point $R_4 = [X_4 : Z_4] \in E$ such that $\#\langle R_4 \rangle = 4$, the isogeny φ defined by $\ker(\varphi) = \langle R_4 \rangle$ is computed as follows:

$$\begin{aligned}\varphi : \quad E \quad & \to \quad E' \text{ defined by } [A_{24}^+ : C_{24}] = [X_4^4 : Z_4^4] \\ [X : Z] \quad & \mapsto \quad [X(XX_4 - ZZ_4)^2(XX_4^4 + XZ_4^2 - 2ZX_4Z_4) : \\ & \qquad Z(XZ_4 - ZX_4)^2(ZX_4^2 + ZZ_4^2 - 2XX_4Z_4)].\end{aligned}$$

j-invariant. When E is a Montgomery curve defined by $[A : C]$ or $\alpha = A/C$, the j-invariant can be computed as follows:

$$j \quad = \quad 256\frac{(A^2 - 3C^2)^3}{C^4(A^2 - 4C^2)} \quad = \quad 256\frac{(\alpha^2 - 3)^3}{\alpha^2 - 4}.$$

4 Horizontal differential power analysis of SIKE

The content of this chapter is based on the work:

[GGK21] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. "Full Key Recovery Side-Channel Attack Against Ephemeral SIKE on the Cortex-M4". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 228–254. URL: https://doi.org/10.1007/978-3-030-89915-8_11

Context. Ever since the second round of their post-quantum standardization process, NIST has repeatedly called for side-channel assessment of the post-quantum candidates [Moo19b; Apo20]. This call was reiterated by the SIKE team who made a public solicitation for cryptanalysis in this regard [Jao19]. In particular, SIKE was lacking evaluation for power analysis at the time, as only one article—by Koziel, Azarderakhsh, and Jao in [KAJ17]—analyzed leakages of supersingular isogeny protocols through power consumption, in which three refined power analysis based on zero-value points on SIDH are presented without practical experiment.

The current chapter directly responds to this need and proposes the first practical power analysis of ephemeral SIKE. Our work complements the first practical side-channel attack on SIKE of [Zha+20] which was published concurrently with our own research. In their study, the authors fully describe a vertical DPA on the three-point ladder of the SIKE key decapsulation procedure, and discuss potential countermeasures. However, since the authors rely on the fact that the private key is fixed across the measurements, their attack is applicable only to the semi-static settings of the SIKE protocol. Our attack extends their results by proposing a horizontal DPA on SIKE in ephemeral settings.

Given that SIKE is now theoretically broken, our analysis has become obsolete. Nevertheless, as our work exploits the power leakages of the elliptic curve scalar multiplication involved in the protocol to recover the private key of one party, the attack complements the previous work that targets the same operation but in the context of ECC, such as [Cla+10], [PZS17], [APS19], and [Bat+23]. In fact, our attack is equivalent to the work mentioned above but applied to the scalar multiplication that is used in SIKE (i.e., the three-point ladder).

Results. The main contribution of this chapter is a full private key extraction through a power analysis of SIKE with only a single trace; breaking thus confidentiality in a passive setting (note that our work was conducted and published prior to the disclosure of the theoretical attack against SIKE by Castryck and Decru [CD22]). Particularly, we target the three-point ladder with a straightforward vertical attack (i.e., with multiple traces and a fixed secret) and show how to extend it to the case of a horizontal attack (i.e., with a single trace and a secret which can therefore be ephemeral). This attack can be applied at any stage of the protocol: key generation, key encapsulation, and key decapsulation.

Our attack is geared towards the recommended implementation of SIKE for the ARM Cortex-M4 [Seo+20]; a low-power and low-cost embedded microcontroller[1] which is recommended by NIST for post-quantum cryptography evaluation due to its widespread use [Moo19a; Kan+19]. In particular, our analysis leverages the leakage model of the microcontroller by incorporating the leakage assessment of the architecture presented by Le Corre et al. in [CGD17].

Finally, we argue how our horizontal power analysis defeats many countermeasures that were mentioned in the power analysis of SIKE as presented in [Zha+20]; namely, starting with a random isomorphic curve, masking the scalar, splitting the key randomly, and using a window-based scalar multiplication. We recommend the well-known projective point coordinate randomization, which counters our attack with a negligible performance overhead (of course, this does not prevent the attack by Castryck and Decru [CD22]).

Outline. The chapter is structured as follows: the horizontal differential power analysis is presented in Section 4.1, which is followed by further enhancements to improve the efficiency of the attack in Section 4.2. The experimental verification of the attack and improvements are explained in Section 4.3. Then, in Section 4.4, a countermeasure is recommended while several others are discussed, so the chapter concludes with Section 4.5.

4.1 Attack description

In this section, we explain how to exploit the link between power consumption and processed data in order to recover private key bits. The attack is tailored to the code Cortex-M4 implementation of SIKE by [Seo+20].

[1] See [ARM10] for the specifications of the ARM Cortex-M4.

4.1.1 Three-point ladder analysis

The main point of attack is the three-point ladder (shown in Algorithm 3.1), as the procedure involves a single bit of the computing party's private key in each iteration of the for loop. The goal is to measure the power consumption of the `xDBLADD` operation and to deduce if the subroutine was executed with or without the `swap_points` at step 7. We may assume that we know the private key up to bit $i-1$ by induction. We also know the starting points $Q, P, Q-P$ since these points are public. Therefore, we may obtain the two possible inputs for `xDBLADD`, and we know how they relate to the value of the i^{th} bit of the private key. The two inputs and their Hamming weights are computed and the power trace of a specific set of instructions within `xDBLADD` is correlated with the Hamming weights. Thanks to differential power analysis, this allows us to distinguish when the i^{th} bit is zero or one.

4.1.1.1 Double-and-add

Despite the involvement of a (random) bit of the private key, `xDBLADD` is a deterministic subroutine. The inputs and outputs of each subprocedure in `xDBLADD` depend only on the original inputs of the subroutine. As a result, an educated guess on the original inputs allows us to infer the results of all the operations involved in `xDBLADD`.

In the Cortex-M4 implementation of SIKE, the subroutine consists of 7 multiplications and 4 squarings of $\mathbb{F}_{p^2}$ elements, as well as multiple field additions, subtractions, and modular reductions. Each $\mathbb{F}_{p^2}$ multiplication and each squaring contains two multi-precision additions of $\mathbb{F}_p$ elements, referred to as "`mp_addfast`". This multi-precision addition is the operation on which our attack is focused. In total, there are $11 \times 2 = 22$ `mp_addfast` functions, out of which only 10 have inputs which differ in case of a `swap_points` at step 6 of the three-point ladder. The code of `xDBLADD`, the squaring, and the multiplication functions can be found in Appendix A (respectively in Listing A.2, Listing A.4, and Listing A.3).

4.1.1.2 Multi-precision addition

In the Cortex-M4 implementation of SIKE, the `mp_addfast` is written in assembly. The function computes the addition of two $\mathbb{F}_p$ elements. Depending on the size of p, each field element is saved in an array of $n \in \{14, 16, 20, 24\}$ 32-bit words. Each `mp_addfast` executes $2n$ load instructions (`LDMIA`), n store instructions (`STMIA`), and n additions (`ADDS`, `ADCS`). These instructions are executed in batches of four consecutive additions, due to the limited number of available registers on the Cortex-M4. The code of the `mp_addfast` function can be found in Listing A.5.

4.1.2 Vertical attack

In a vertical attack against SIKE, we measure multiple executions of the three-point ladder in which Bob's private key is fixed, but the client public key inputs are different. From these traces, we concentrate only on a single `mp_addfast` instruction per `xDBLADD`, i.e., per bit of the private key. Within the `mp_addfast`, we can decide to focus even further on the first addition instruction. We can thus compute the two possible outputs of the first `ADDS` depending on the (timing-constant) `swap_points`, for each public key, and then correlate the two vectors of Hamming weights of these outputs with the power traces using the DPA procedure from Section 2.1.3.1. This process can be repeated for each bit of Bob's private key, as the correctness of each guess depends on the correctness of previous ones, resulting thus in an *extend-and-prune* attack.

4.1.3 Horizontal attack

In the horizontal attack scenario, we are restrained to only one power trace for a single execution of the three-point ladder. The same approach as in the vertical attack cannot be used because there would not be enough data to obtain strong correlations. We can work out this issue and re-obtain "*verticality*" by combining the power traces of all 10 `mp_addfast` functions within each `xDBLADD`. This way, we obtain 10 power traces with which we can correlate pairs of inputs—similarly as in a vertical attack with 10 power traces.

We can further improve this attack. A multi-precision addition takes two $\mathbb{F}_p$ elements as input and gives one as output. Each one of the $2n$ input words of 32 bits is loaded once and then used in the addition instruction, and the n output words of 32 bits are stored. In total, there are $3n$ words which pass through the pipeline registers and whose Hamming distance from the previous word in the pipeline are related to the power consumption.

For each of the $3n$ words, we compute the PCC between the 10 power traces and the 10 pairs of Hamming weights of 32-bit words accounting for the two guesses of the current bit of the private key. For each word, a spike in the correlation is expected at a different position depending on the instruction which uses this particular word. The locations of spikes can be deduced from the shape of the power traces. Once the $3n$ pairs of correlations are computed, we can add them up such that the locations of the expected spikes are aligned. We expect to end up with two correlations for each guess of the private-key bit, with a clear spike in the correlation plot of the correctly guessed value.

4.2 Attack enhancements

In presence of noise in power measurement, the private key guesses may be erroneous. A single wrong guess of a bit of the private key leads to completely inconclusive results, because the following guesses depend on the correctness of the previous bits. Therefore, it is of particular

importance that no erroneous guesses are made in the process of key extraction. We propose two measures to approach this problem.

4.2.1 Depth search

When the guess of a single bit gives inconclusive results, we can proceed by making four guesses for the next two bits in hope of finding a correlation coefficient with a notable spike. In particular we can make a guess for k consecutive bits, obtaining in total 2^k different combinations. For each combination we compute a PCC for each of the k bits. In total there are $2(2^k - 1)$ correlation coefficients, not counting repetitions. We then add up all the PCCs for each k-bit combination and we guess the current SK bit to be the trailing bit of the combination with the strongest correlation. Figure 4.1 illustrates this process on real data.

Selecting the k-bit block that exhibits the strongest sum of PCCs may lead to the correct guess, but this process may still be susceptible to the same inconclusive results as initially encountered. In contrast, when there are strong correlations at the k^{th} bit, the first bit is extremely likely to be correct.

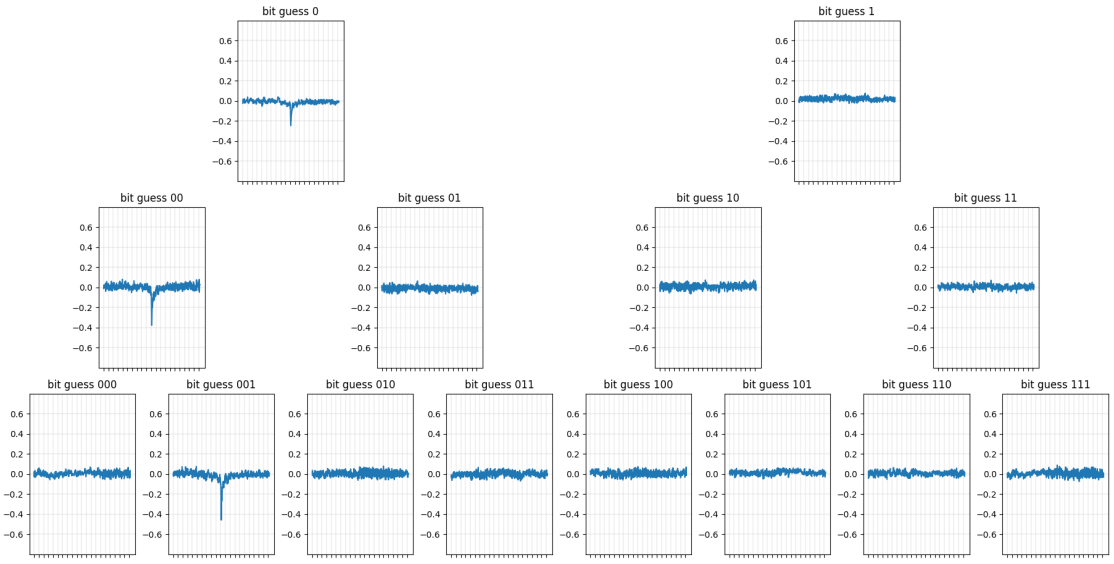


Figure 4.1: Example of a depth search with three bits.

4.2.2 Increasing verticality

We can increase verticality (i.e., the amount of power traces in the horizontal settings) by computing correlations for bits in windows of k. If, for one bit, 10 `mp_addfast` functions can be measured from a single `xDBLADD`, then, for k bits, there will be $k \times 10$ traces of `mp_addfast` functions from the k consecutive `xDBLADD` functions. In total, 2^k hypotheses need to be made (one per bit), and 2^k correlation coefficients are computed for $10k$ power traces.

Finally, rather than performing the attack on contiguous windows of k bits, we select only one bit of Bob's private key to be the trailing bit of the k-bit combination with the strongest correlation. This way, we can re-run the process starting from the bit right afterwards as a way to correct errors due to the potential proximity of strong correlations. This process resembles the error-correction procedure introduced in [Dug+16].

Also, we mention that other operations, such as `fpmul_mont` and `fpsub`, can be measured and combined to increase verticality. While these are dissimilar operations and may leak information differently than `mp_addfast`, they may still add information to the overall selection of Bob's private bits.

4.3 Experimental verification

In order to validate the horizontal attack described in Section 4.1, we reproduced the key recovery on a programmable board which runs an adapted version of Cortex-M4 implementation of SIKE from [Seo+20].

4.3.1 Setup

Hardware. We performed our experiments on an STM32F3 as DUT with the ChipWhisperer framework that was described in Section 2.1.1. The oscilloscope was configured with the following settings:

- A bandwidth of 500 MHz.
- A sampling rate of 250 samples per μs.
- A resolution of 8 bits per sample.
- A memory of 50,000 samples per acquisition.

Software. The attacked implementation is the official SIKE implementation adapted for (32-bit) ARM Cortex-M4 microcontrollers [Seo+20], which is part of the official submission package and is constant in timing. We attacked SIKE instantiated with a prime of 434 bits (i.e., `SIKEp434`); a choice that we elaborate in this section.

In our experiment, we wrote a small piece of software that interfaces the serial communication from the ChipWhisperer framework to the SIKE library. The code allows the computer to program the DUT remotely through USB and simulate the key exchange while power consumption is measured.

Concretely, the software uses ChipWhisperer's SimpleSerial protocol [New17] to program different commands to which the DUT reacts. The computer uses these commands to communicate data to the DUT by serially transmitting, first, the byte of the command in ASCII,

then, the data of length specified for the command. When the procedure corresponding to the command ends, the DUT responds with the letter `z` followed by a code returned by the procedure, which concludes the protocol exchange. Two custom commands were introduced in the scope of this experiment:

- The command `k` which sets Bob's private key used in the three-point ladder.
- The command `p` which sends Alice's public key and executes the three-point ladder procedure of SIKE.

We made additional modifications in the SIKE implementation to ease the collection and the pre-processing of the traces. Note that these adjustments were made for efficiency purpose and are by no means necessary for our attack to work. In other words, we emphasize that the attack can be mounted on the original implementation of SIKE presented in [Seo+20] without any difficulty.

The list of adjustments are the following:

- A GPIO pin (`PA12`[2], a.k.a., the trigger) is toggled when the double-and-add operation of the three-point ladder enters into an `mp_addfast` procedure that depends on the swap.
- An idle delay of about 1 millisecond was introduced in between each `mp_addfast` call, and of about 1 second after each loop iteration of the three-point ladder.

Limitations of the software. While the introduction of a trigger GPIO and multiple delays results in an unrealistic attack scenario, we emphasize on the fact that the attack is still possible on an unmodified SIKE implementation. The process of segmenting the power traces, as well as the correlation and Hamming weights computations can be done *offline*, after the power traces have been sampled. In a plain attack, as opposed to our experiment, the traces acquisition will be synchronized on serial communication. Then, the targeted operations need to be identified within the full resulting power trace (e.g., using cross-correlation techniques, as in [Dug+16]), so the sub-power traces corresponding to the attacked instructions can be manually segmented and carefully aligned to perform the DPA. This process is not the main focus of our study and was therefore duly skipped.

Other SIKE instances. As described in Section 3.3, the original SIKE submission offers various levels of security with four different parameters sets; each of which with a prime of unique size (i.e., a p with a bit-length of 434, 503, 610, and 751, see Table 3.2). While instantiating SIKE with a larger prime offers stronger security guarantees against theoretical cryptanalysis, larger instances present a wider attack surface in a single-trace power analysis. This property was also observed by Bos et al. in [Bos+19], and is due to the increased number

[2]We refer to the official NAE-CW308 UFO datasheet to find the mentioned pins: http://media.newae.com/datasheets/NAE-CW308-datasheet.pdf

of instructions executed which, therefore, yield more power measurements. As a result, our attacked instance (`SIKEp434`) is expected to be the hardest to attack with a single trace.

Also, the compressed instances of SIKE are prone to the same horizontal attack, because the starting points of the three-point ladder are deterministically obtained from the compressed public key.

4.3.2 Experiment

4.3.2.1 Traces collection

Our experiment simulated a portion of the SIKE key exchange between Alice (the computer) and Bob (the DUT); namely, the key decapsulation procedure. Our attack scenario can be summarized with the following steps:

1. On the computer, generate Bob's key pair at random, and send Bob's private key to the DUT (with the command `k`).
2. Given Bob's public key, generate Alice's key pair at random.
3. Send a public key to the DUT (with the command `p`) during which the oscilloscope measures the power consumption of:
   - only the *second* `mp_addfast` call involved in steps 6 and 8,
   - and both `mp_addfast` calls involved in steps 16, 17, 18, and 19,

   of the `xDBLADD` procedure (see Appendix A) as used in the three-point ladder.

Once triggered, the oscilloscope was configured to sample the power consumption at a rate of 250 MS/s during a period of 20 μs. As a result, a power trace for a single execution of `mp_addfast` includes 5,000 power samples.

This attack scenario was repeated a total of 460 times to obtain at most 1 million traces. Each of these experiments includes the power traces of the 10 `mp_addfast` calls from the loop iterations for all the 217 bits of Bob's private key. Hence, $460 \times 10 \times 217 = 998{,}200$ different power traces were acquired during that experiment.

For reference, Figure 4.2 (top) shows the average power consumption of an `mp_addfast` execution captured by our oscilloscope.

Traces polishing. Because our initial results turned out to be inconclusive due to a serious level of noise in the acquisition (see top of Figure 4.2), we processed the collected power traces with a denoising technique, in the hope that such a processing would increase the success rate of our DPA.

In our case, we applied a wavelet denoising compression, as described in Section 2.1.2.2, to obtain cleaner power traces. Best results were experimentally obtained when Daubechies

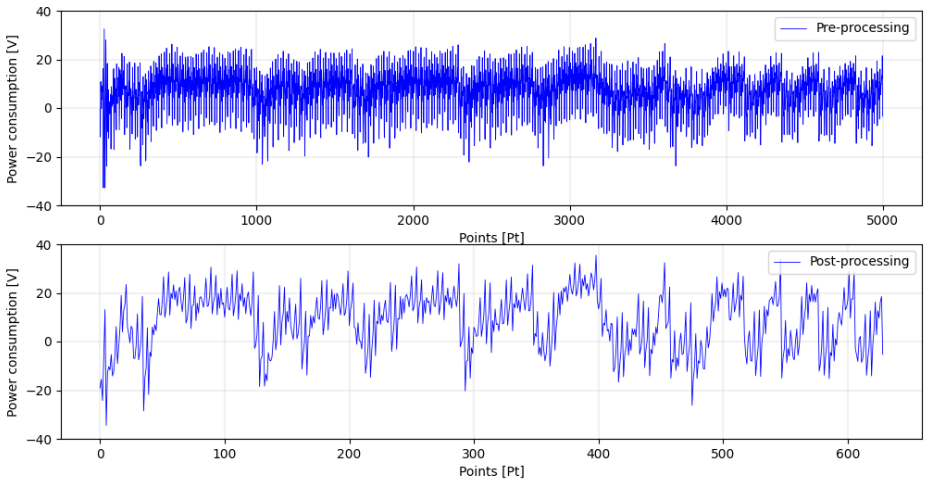


Figure 4.2: Result of the discrete wavelet transform with Daubechies 3 wavelets ('db3').

3 wavelets ('db3') were used recursively three times to reduce the number of samples from 5,000 to 623 (by keeping the approximations). The average of the resulting traces is shown in Figure 4.2.

The denoised traces and public data are made accessible at:
https://github.com/nKolja/SIKE-HPA-2021.

4.3.2.2 Horizontal DPA procedure

Using the denoised power traces, we performed a horizontal DPA on each iteration of the loop in the three-point ladder. Each time, a single bit of Bob's private key is attacked. This process can then be repeated across all the bits of the key.

Since a single bit is hypothesized at each step of the horizontal attack, there are only two hypotheses to consider:

- The points P and $Q-P$ were swapped (the bit is different from the previous bit).
- The points P and $Q-P$ were left un-swapped (the bit is the same as the previous bit).

A strong correlation between the power traces for one loop iteration and the values corresponding to one of the two hypotheses indicates the correctness of the hypothesized bit. As the attack moves forward, a successful recovery of the first bits allows the recovery of the next ones. Therefore, a full-key recovery can be incrementally mounted in an extend-and-prune manner.

Power traces segmentation. Due to the ephemeral settings of the protocol, we have access to only a single trace per loop iteration involving a single bit of Bob's private key. Therefore, in order to apply a classical DPA as described in Section 2.1.3.1, we need to obtain verticality, i.e., find a way to obtain a certain amount of multiple different power samples which are linked to a same portion of the private key. In our case study, we segmented the power trace that corresponds to an iteration of the three-point ladder into 10 different power traces, each of which corresponding to an `mp_addfast` execution, for which, given either hypothesis, the full input and output (and thus, relevant Hamming distances information) are known. As a result, our horizontal DPA will amount to a vertical DPA with 10 power traces and 2 hypotheses.

DPA enhancements. To further improve the success of our attack, we have inspected the targeted function for which the power traces were collected. Particularly, the power traces correspond to the `mp_addfast` function which adds two input $\mathbb{F}_p$ elements and returns a single $\mathbb{F}_p$ element (see Listing A.5). Because, in our experiment, p is 434-bit long, each element is saved as an array of $\lceil 434/32 \rceil = 14$ words of 32 bits. This results in exactly 14 addition instructions, hence 14 leakage points, in a single `mp_addfast` power trace.

Moreover, we considered the leakage model from a Cortex-M4 microcontroller as explained in [CGD18]. Because the power consumption leaks in the Hamming distance between the pipeline registers, we actually obtain *three* leakage points on a power trace per instruction:

(1) the Hamming distance between the first inputs of the current and the previous instruction,

(2) the Hamming distance between the second inputs of the current and the previous instruction, and

(3) the Hamming distance between the output of the current and the previous instruction.

This results in an additional segmentation of $3 \times 14 = 42$ points of leakage. For each point of leakage, a PCC is computed with the 10 `mp_addfast` power traces and the 10 Hamming distances.

We expect each of these PCCs to produce a spike at a different point in time in the correlation plot which we try to recover. The location of the spike corresponds to the position at which the associated 32-bit word is processed by a pipeline register. Each of these leakage points is constant throughout the `mp_addfast` executions and the three-point ladder loop (assuming the power traces are properly aligned, which can be automated using basic peak alignment methods). These positions can even be identified by analysing the spike structure of the power trace (using, e.g., cross-correlation techniques).

Finally, the 42 PCCs at each point of leakage are added together to produce a larger spike. This consists of aligning all correlation plots on their leakage points and adding them together. We expect the difference of added correlation coefficients to be large enough to correctly validate the private bit.

4.3.2.3 Results

Among the 460 trials, our experimental results returned a resounding success rate of 100% in recovering the full key. None of the improvements described in Section 4.2.1 were even required. An example of the corresponding DPA is shown in Figure 4.3 where six bits are shown to be successfully recovered. All the code used to derive our results is shared on:

https://github.com/nKolja/SIKE-HPA-2021.

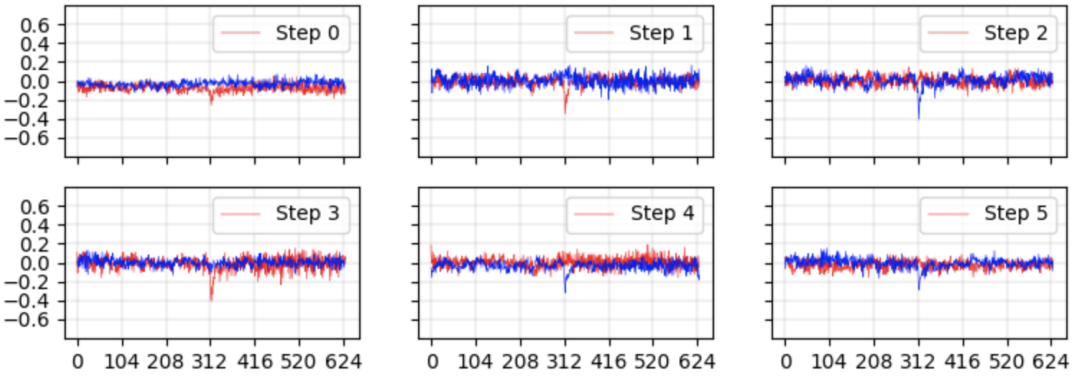


Figure 4.3: Addition of shifted PCC results with 10 segments of a single power trace. Each step corresponds to a different bit. The blue curve corresponds to a bit hypothesis of zero, while the red curve corresponds to bit hypothesis of one.

4.3.2.4 Discussion

This proof of concept shows that, even in ephemeral settings, the official ARM implementation of SIKE is vulnerable to classical power analysis techniques.

4.4 Countermeasures

The attack arises as a consequence of the three-point ladder being a deterministic function with predictable inputs. Each value going through the pipeline registers can be reduced to only two cases. These inputs depend on the public triple x_Q, x_P, x_{Q-P} (which define $Q = [x_Q : 1], P = [x_P : 1]$, $Q - P = [x_{Q-P} : 1]$), the bits of Bob's private key up to the step at which the instruction in question is being executed (which we may assume to be known by induction), and the two possibilities for the current bit of the private key.

4.4.1 Recommended countermeasure

A simple and low-cost countermeasure, which was also mentioned in [Fan+10; Cor99; Zha+20], consists of randomizing the coordinates that define the starting points, i.e., generate three

random non-zero field elements r_Q, r_P, r_{Q-P} and set

$$\begin{cases} Q & = & [x_Q r_Q : r_Q], \\ P & = & [x_P r_P : r_P], \\ Q-P & = & [x_{Q-P} r_{Q-P} : r_{Q-P}]. \end{cases}$$

The increase in complexity comes from generating three random $\mathbb{F}_{p^2}^*$ elements and three field multiplications. This is negligible with respect to the overall cost of the three-point ladder. The execution of the protocol is still correct because the points $Q, P, Q-P$ are not changed, but the input of `xDBLADD`, seen as three pairs of $\mathbb{F}_{p^2}$ elements is now randomized. Since the values r_Q, r_P, r_{Q-P} are secret, we cannot predict the loaded and stored values in the pipeline registers, and thus cannot apply the same attack anymore.

Point randomization is in general still vulnerable to refined power analysis, as shown in [Fan+10; KAJ17]. Such power analysis constitutes in finding a point P such that one of its coordinates is 0, so that randomization would not change this coordinate. Feeding P to the attacked device would lead to some of the coordinates being known in the computation of the ladder. However, the only points that have a zero in the X or Z coordinates are $[0:1:0]$ (i.e, the point at infinity) and $[0:0:1]$, a point of order 2. Neither of these points can be a part of a public key or an input of the three-point ladder, so they can be avoided by a simple sanity check.

4.4.2 Other countermeasures

In addition to the randomized projective coordinates described above, the authors of [Zha+20] proposed a series of countermeasures (based on [Fan+10; JT01]) against DPA on SIKE that we aim to evaluate in the case of a horizontal attack. We will argue that these countermeasures are either too expensive, or do not offer additional protection against horizontal attacks. We also comment on atomic elliptic curve algorithms.

1. **Masking the base point** Q
   The starting point Q is masked with a random point R in order to obtain $Q \leftarrow Q + R$. The final point $P + [\text{SK}](Q+R)$ of the three-point ladder is then adjusted by subtracting $[\text{SK}]R$. Masking the base point prevents both a vertical and a horizontal attacks but cannot be done without leaving the Montgomery representation. As a result, such a countermeasure requires at least a square root computation over the field $\mathbb{F}_{p^2}$, which is very expensive.
2. **Random isomorphic elliptic curve**
   The point Q is mapped to a random elliptic curve E' where the scalar multiplication is computed. The result is then mapped back to the original curve E in order to obtain $[\text{SK}]Q$ which is then added to P.
   Such a countermeasure is unfortunately limiting, since the number of curves of isomorphic to E is low, and finding a non-trivial isomorphism is not trivial. In particular, mapping Q to an isomorphic elliptic curve does not provide enough security against a horizontal attack due to the possibility of testing all isomorphic curves.

3. **Masking the scalar SK**
   The private key SK is masked with a random value r by setting $\mathrm{SK} \leftarrow \mathrm{SK} + r \cdot \mathrm{ord}(Q)$.
   If the masking is different at each execution and big enough, the vertical attack can be conceivably prevented with this countermeasure. However, the horizontal attack is simply extended by $r \cdot \mathrm{ord}(Q)$ bits and recovers a value congruent to the actual SK (mod $\mathrm{ord}(Q)$). Besides, the execution of the three-point ladder would be a factor of $\log(r)$ slower.

4. **Random key splitting**
   The private key SK is divided randomly as $\mathrm{SK} = \mathrm{SK}_1 + \mathrm{SK}_2$. Then two three-point ladders are computed in order to obtain $(P + [\mathrm{SK}_1]Q) + [\mathrm{SK}_2]Q$.
   While splitting SK differently across executions produces measurements of dissimilar operations in a vertical attack, this countermeasure is not effective against a horizontal attack, as both shares can be independently recovered.

5. **Window-based countermeasure**
   Instead of making a binary choice for swapping at each step of the three-point ladder, a 3-bit window is used, and two additions and three doublings are computed per window. While a window-based method increases the complexity of a vertical attack, such a countermeasure is ineffective in the settings of a horizontal attack, as the number of guesses per DPA iteration simply increases from 2^1 to 2^3. Besides, similarly as with the base point masking, this countermeasure is not cost-efficient, as the new ladder will require to leave the Montgomery representation, requiring at least one computation of a square root over $\mathbb{F}_{p^2}$.

6. **Atomic three-point ladder**
   The authors of [CCJ03] propose atomic algorithms for preventing simple side-channel analysis. An atomic algorithm consists of a sequence of instructions that are indistinguishable from a side-channel point of view.
   At the first look, the three-point ladder might seem to be atomic, however the assumption in [CCJ03] that modular operations are side-channel equivalent fails in the Cortex-M4 environment. While we are not able to distinguish a single pair of modular additions with two different inputs, we are able to distinguish 10 tuples of modular additions with two different 10-tuples of inputs, which breaks indistinguishability.

4.5 Conclusion

The chapter describes a DPA on SIKE in ephemeral settings that recovers Bob's entire private key using a single power trace of the three-point ladder in the key decapsulation procedure. The attack was experimentally verified on an STM32F3 which features a Cortex-M4 microcontroller in the context of the ChipWhisperer framework. A countermeasure based on point randomization is finally suggested.

The impact of this attack on the security of SIKE is critical when the reference implementation is adapted in an unprotected manner to a Cortex-M4 microcontroller. This is especially important, because of the exceptionally leaky nature of such microcontrollers, thanks to the findings of [CGD18]. Due to the simplicity of the DPA, countermeasures are required to be deployed when the reference implementation of SIKE is used in an embedded environment. We emphasize on the fact that the three-point ladder attacked in the key decapsulation is not the only point of attack of the SIKE protocol and that *each* use of the three-point ladder (even in the key generation, and key encapsulation) requires to be protected when exposed to power analyses.

However, in the context of SIKE, these considerations are now irrelevant. The attack developed by Castryck and Decru [CD22] has rendered all side-channel attacks against SIKE obsolete as their work enables the recovery of private keys using only public information. Still, the methods introduced in this chapter show an optimization of a well-known attack against an elliptic curve scalar multiplication variant that is not widely recognized. Furthermore, we presented and validated enhancements to the attack, such as the depth search which is novel in itself, and re-explored previously known results, such as the efficiency of the Wavelet transform in DPA.

5 Clustering power analysis of SIKE

The content of this chapter is based on the work:

[GK22] Aymeric Genêt and Novak Kaluđerović. "Single-Trace Clustering Power Analysis of the Point-Swapping Procedure in the Three Point Ladder of Cortex-M4 SIKE". in: *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 164–192. DOI: 10.1007/978-3-030-99766-3_8. URL: https://doi.org/10.1007/978-3-030-99766-3_8

Context. In the previous chapter, we have described a single-trace differential power analysis of SIKE that targets arithmetic operations in the elliptic curve scalar multiplication. This attack, while very powerful, is subject to certain limitations: the analysis relies on the knowledge of the leakage model of the device, an important number of power segments corresponding to the execution of the same instruction needs to be properly partitioned to obtain significant results, and the attack is thwarted if the operations in the three-point ladder are protected against power analysis using, e.g., coordinate randomization (as recommended in Section 4.4).

This chapter presents another single-trace attack on ephemeral SIKE that is effective even when the scheme is hardened with coordinate randomization. This attack—known as clustering power analysis—works by combining samples of power consumption that correspond to the same operation, and that are then categorized into two (or, sometimes, more) classes. A correct classification leads to the recovery of secret information, such as the private key. In particular, the attack targets the point-swapping procedure in the three-point ladder of the key exchange, as the function swaps multiple values only when two consecutive private bits are different, resulting in power samples that are thus expected to follow two distinct distributions (i.e., the values were swapped or not).

The first instance of such an attack was published by Heyszl et al. in [Hey+13], in which the authors used the k-means clustering algorithm on electromagnetic samples collected from multiple probes to recover the private-key bits of an elliptic curve scalar multiplication. Perin et al. extended this approach to a single-probe attack in [Per+14] by combining the classification of multiple private-key bits in a fully-unsupervised process which was experimentally verified on a protected implementation of RSA. The attack was then further improved by Specht et al. in [Spe+15] by pre-processing the traces with a principal component analysis and by using expectation-maximization as a clustering algorithm. Then, Perin and Chmielewski proposed a semi-parametric framework that uses non-profiled learning both as a leakage assessment tool and as a private-key bits recovery tool in [PC15]. A practical unsupervised attack against ECC was mounted on the protected library of μNaCl in [NC17]. Recently, several other libraries were attacked with clustering power analysis on smartphones with low-cost equipment in [Ala+21]. Independent studies which also investigated clustering power analysis but on different parts of the Montgomery ladder obtained similar results in [SH17] and [Shi+19]. Finally, in [Per+21], Perin et al. established the current state of the art in clustering power analysis by using deep learning methods on the traces and the output clusters to predict and correct wrong labelings.

As mentioned in the previous chapter, SIKE is no longer considered secure due to the recent attack by Castryck and Decru [CD22]. Nonetheless, our work still represents a significant improvement in clustering power analysis with respect to the approaches against ECC as described above. For instance, when compared to the work by Nascimento and Chmielewski from [NC17], our attack is shown to successfully extract the key without, in particular, requiring leakage assessment. We also bring many novel improvements, such as clustering in the frequency domain, using the wavelet transform as a denoising tool, and clustering with a simpler method than the k-means clustering algorithm. Still, the other approaches are expected to fully work on the same target.

Results. The chapter exhibits the following contributions:

- A practical single-trace side-channel power analysis attack on the recommended implementation of SIKE for the ARM Cortex-M4 [Seo+20] is described in detail. The most important feature of the attack is that a *single* execution of the key exchange is sufficient, and that only samples from the power domain of the microcontroller are necessary. More specifically, neither electromagnetic sampling, nor profiling, nor previous knowledge of the target device is required. The key recovery is completely *non*-supervised (i.e., no template is ever built) and can be performed without the need for public keys (which are still required to verify whether the key was successfully extracted).
- Clustering the samples is shown to be also possible in the frequency domain, i.e., by first processing the power traces with a Fourier transform. As a result, the attack becomes tolerant to cases where traces are slightly misaligned due to, e.g., bad segmentation.
- Alternatively, the clustering power analysis can be improved by processing the traces with a wavelet transform to compress the power traces by filtering out high frequencies.

Such a transform is shown to be relevant in practice because the compression reduces the number of timing locations to visit.

- A clustering method based on thresholding the distribution of sorted power samples is shown to be sufficient to successfully recover the key. This result shows that the clustering algorithm does not require to be sophisticated, and can be far less complex than the methods proposed so far.
- Finally, a countermeasure based on splitting the masking value into multiple random shares during the swapping procedure is shown to effectively protect against the attack described in the chapter.

Outline. The clustering power analysis is presented in Section 5.1, which is followed by further enhancements to improve the efficiency of the attack in Section 5.2. The experimental verification of the attack and improvements are explained in Section 5.3. Then, in Section 5.4, the suggested countermeasure is described and validated so that the chapter concludes with Section 5.5.

5.1 Attack description

In this chapter, we explain how to the attack which recovers a party's private key by classifying the power samples (in Volts) of a single execution of the three-point ladder in SIKE when such procedure is reinforced with randomized coordinates. The attack is tailored to the code Cortex-M4 implementation of SIKE by [Seo+20].

5.1.1 Point-swapping procedure analysis

The attack targets the three-point ladder of n bits as described in Section 3.3.2. As a result, the attack can be applied at every stage of the protocol. For the sake of simplicity, the chapter assumes that the attack targets the three-point ladder invoked in the key decapsulation.

The attack assumes a passive adversary able to monitor the messages exchanged in the SIKE protocol, and the power consumption of the attacked device.

In the first stage of the attack, the power consumption of the entire three-point ladder is measured with a fixed and fast enough sampling rate. The power samples are then segmented into multiple power traces synchronized at the beginning of each step of the three-point ladder. Moreover, only the segments corresponding to the execution of the `swap_points` functions are considered, each of them ultimately consisting of M samples (typically, a few thousands).

Because `swap_points` masks values with either `0x00000000` or `0xFFFFFFFF` depending on the difference between two consecutive secret bits, the attack attempts to distinguish for each iteration whether the swap occurred or not by gathering the samples at a same location in

Algorithm 5.1 Clustering power analysis against SIKE.

Input: $\{s_i\}_{0\le i<n}$: Collection of all the power samples at a same t.

1: Run the k-means clustering on $\{s_i\}_{0\le i<n}$ (with $k=2$).
2: Let $sk_{-1}=0$.
3: Let $l_i = \begin{cases} 0 & \text{if } s_i \in \text{first cluster,} \\ 1 & \text{if } s_i \in \text{second cluster,} \end{cases}$ $(i \le 0 < n)$.
4: Let $sk_i = l_i \oplus sk_{i-1}$ $(i \le 0 < n)$.
5: **return** $sk = (sk_0, sk_1, sk_2, \ldots, sk_{n-2}, sk_{n-1})$.

all iterations and clustering them with k-means. Since a difference of bits can only be zero (identical) or one (different), only two clusters are considered (i.e., $k=2$). The private key can be entirely reconstructed from the labels at the end of the clustering.

5.1.2 Clustering attack

We adapt the clustering power analysis as described in Section 2.1.3.2 to the analysis of the `swap_points` function. Given the n segmented power traces T_i of M power samples each, the procedure consists of three steps:

1. Select a sample location $0 \le t < M$ in the power traces.
2. Cluster with k-means the n power samples at location t throughout the traces T_i (for $0 \le i < n$), and reconstruct the key from the labels.
3. Verify the key obtained.

The second step of the attack is detailed in Algorithm 5.1. The samples s_i must all correspond to the samples at a same point throughout the n segmented power traces. Since the k-means algorithm considers samples from a single timing location in the power traces, the procedure can be repeated with all different positions until a returned key (or its bitwise inverse) successfully decrypts a ciphertext. As a result, the overall attack has a complexity of precisely M executions of k-means and key tryouts.

Unlike the approach in the previous chapter, the rigidity required by the clustering makes the attack a hit or miss: either all bits are sequentially retrieved, or none of the bits can be confirmed for sure.

5.2 Attack enhancements

In a full attack as described in Section 5.1, the adversary needs to pass through all sample locations in the traces and use k-means to recover a deterministic key candidate that eventually needs to be verified. Adopting a better strategy for any of these steps can lead to both faster and more successful results.

This section lists enhancements to speed up the eventual recovery of the key. These can sometimes be combined to improve even further the overall attack.

5.2.1 Enhancing sample selection

The original attack exploits the raw power consumption which requires visiting *all* sample locations. Applying a transform to the power consumption before the clustering step can reduce the number of samples and therefore speed up the attack. Moreover, relating the power consumption to a different domain can exhibit leakage points which may lead to improved results.

5.2.1.1 Fourier transform

Since the power trace captures the operations that periodically swap words in a regular for-loop, some of the frequency coefficients are also expected to follow two distinct distributions. As a result, processing the power consumption with a Fourier transform (as described in Section 2.1.2.1) prior to running the clustering algorithm is expected to give a similar success rate.

Clustering in the Fourier domain exhibits many advantages over clustering in the time domain:

- Due to the Hermitian symmetry ($\hat{s}_f = \hat{s}^*_{-f}$), the upper half of the coefficients is identical to the lower half. Accordingly, the number of locations visited by the clustering algorithm can be halved. Moreover, this number can be reduced by only considering a range of reasonable frequencies (such as all the frequencies below the clock speed of the targeted device).
- As the Fourier analysis treats the frequency components of the signal, the processed signal is tolerant to timing misalignments. Such misalignments are particularly common when monitoring the power consumption for a long time, or when segmenting a long power trace.
- The frequencies of interest (i.e., frequencies at which the information leakage is significant) are expected to be unique to a single device and can therefore be re-used in a subsequent analysis of the same device (resulting in an educated but still unsupervised attack).

5.2.1.2 Wavelet transform

As power traces may contain frequencies that do not contribute to the leakage of information, applying a wavelet transform (as described in Section 2.1.2.2) to the power samples may filter out insignificant frequencies while preserving the important ones. Such a processing is expected to result in a signal that is more susceptible to clustering power analysis.

In our attack, two advantages of the wavelet transform are mainly capitalized upon:

- The wavelet transform downsamples the signal at each level while keeping the lower frequency bands. This process halves the length of a power trace each time and therefore results in fewer timing locations to check in the attack.
- By filtering out higher frequencies, the wavelet transform acts as a post-processing denoiser. Cleaner signals are therefore expected to be output, which anticipates better results.

5.2.1.3 Other transforms

In addition to the Fourier and the wavelet transforms, other transforms that compress the power traces can reduce their number of samples. For instance, principal component analysis [Boh+03] is a technique that reduces the dimensionality of the power traces. Such a transform was also reported to obtain better results in [Spe+15] by selecting the significant principal components.

5.2.2 Enhancing power samples clustering

Since the overall attack needs to run a clustering algorithm several times, an algorithm that clusters the power samples more efficiently leads to faster results.

5.2.2.1 Thresholding

While the k-means algorithm (as described in Section 2.1.3.2) already involves low-complexity computations, the clustering algorithm does not need to be generic and can therefore be tailored to a one-dimensional two-population problem. In particular, the analysis expects two distributions of similar density but shifted mean, which can therefore be separated with an appropriate middle point.

Many solutions exist to find a suitable middle point, such as computing the overall mean of all the samples, or finding the biggest gap between two neighboring power samples. Algorithm 5.2 proposes a clustering which calculates the literal middle point of the distribution by finding the maximum and minimum. Such a solution runs in $\mathcal{O}(n)$, but can be tweaked to present other advantages that are described in the next subsection.

Algorithm 5.2 Thresholding clustering algorithm.

Input: $\{s_i \in \mathbb{R}\}_{0 \leq i < n}$: Set of n samples at a same time t.

1: Compute $d = (\min(\{s_i\}_{0 \leq i < n}) + \max(\{s_i\}_{0 \leq i < n}))/2$.

2: Let $l_i = \begin{cases} 0 & \text{if } s_i < d, \\ 1 & \text{otherwise,} \end{cases} \quad (0 \leq i < n)$.

3: **return** $(l_i)_{0 \leq i < n}$.

5.2.2.2 Other clustering methods

In a noisy environment, rigid clustering methods such as k-means, thresholding, and even expectation-maximization (as used in [Spe+15]) are inadequate due to the two clusters overlapping with each other. Relaxed clustering techniques, such as fuzzy c-means [Dun73], have been reported to successfully overcome these limitations in [NC17; Per+14].

5.2.3 Enhancing key verification

The attack achieves a better performance by reducing the number of key candidates to verify, or by correcting plausible clustering mistakes that could arise due to, for example, samples being too close to the middle point between the two distributions.

5.2.3.1 Majority rule

As noted by [Per+14], a same labeling re-occurring throughout many different locations is likely to be correct. Two majority rules are therefore proposed:

1. A *vertical* majority in which a candidate key occurring multiple times across the timing locations is verified in priority.
2. A *horizontal* majority in which individual key bits are labeled given their majority throughout the clusterings at all timing locations.

In a horizontal majority rule, a threshold can be selected to filter all the bits for which the clusterings give the same results, while the remaining bits can simply be guessed.

5.2.3.2 Educated thresholding

In the clustering power analysis against the three-point ladder of SIKE, two observations can be made:

1. A clustering is successful only when the two sub-distributions are distinct.
2. The number of swaps must always be even.

The first observation stems from the fact that two samples of identical value should always be assigned to the same cluster. Hence, a successful clustering can only be found by splitting the overall distribution between two sample values.

The second observation is due to the fact that the three-point ladder requires the points to always be “unswapped” at the end of the procedure. This means that the sizes of the two clusters are always even which can therefore be used to validate the key found.

As a result, one can design a thresholding algorithm similar to Algorithm 5.2 that first sorts the power samples and then separates the distribution in two, each call at a different threshold starting from a middle point. The threshold can move depending on the distance between the current threshold and the two cluster centers (similarly as in k-means). By iteratively calling such an algorithm, the labels that are more likely to be erroneous can be marked and subsequently flipped in a way that makes sure that the Hamming weight of the labeling bitstring is even. The computational complexity of this new method is $\mathcal{O}(n \log n)$ (owing to the sorting of the sample data).

5.2.3.3 Other post-processing

In case the sample location is known to correspond to a leakage point but the environment is too noisy to perfectly separate the clusters in two (see [NC17; Per+14] for context), methods based on deep learning can still successfully extract the key, as reported by Perin et al. in [Per+21].

5.3 Experimental verification

This section reports a proof of concept for the clustering power analysis described in Section 5.1, in addition to an evaluation of the efficiency of the enhancements proposed in Section 5.2.

5.3.1 Setup

Hardware. Our experimental verification was performed on an STM32F3 as DUT as part of the ChipWhisperer framework described in Section 2.1.1. We program the oscilloscope with the following settings:

- A bandwidth of 20 MHz.
- A sampling rate of 250 samples per μs.
- A resolution of 10 bits per samples.
- A memory of 25,000 samples per acquisition.

Moreover, we measure the power consumption through a 20dB Low-Noise Amplifier (LNA).

Note also that the sampling rate was intentionally made high to showcase the efficiency of the preprocessing transforms.

Software. The software considered is the SIKE implementation for Cortex-M4 of [Seo+20] which needs to be used in a certain way that enables power trace collection. In particular, the program that runs on the DUT waits for the laptop to send the three byte-encoded elliptic

curve points (i.e., the ciphertext) through a serial communication with the ChipWhisperer-Lite. Such a transfer prepares the DUT to run the three-point ladder with a pre-programmed private key which can be triggered anytime.

Coordinate randomization. As the three-point ladder from [Seo+20] does not originally offer protections against power analysis, coordinate randomization was only simulated. In this simulation, three multiplicative field elements are pseudorandomly generated from a pre-programmed seed at the beginning of each iteration. Since only a cheap pseudorandom number generator was required, ChaCha8 [Ber08b] was chosen for this purpose. The X and Z coordinates of the three points are replaced by the product (in $\mathbb{F}_{p^2}$) of their respective value and the pseudorandomly generated elements (one for each point).

Note that while this simulation sufficiently protects the three-point ladder from correlation attacks based on the values of the elliptic points (see, e.g., [Zha+20; GGK21]), the resulting code is not claimed to be secure in a real-life scenario. This implementation is evidently not what was considered by the attack, and the overhead was therefore not measured.

Further modifications. Since the acquisition of power traces is not the main focus of the chapter, the software was further modified to make the experiment easier. Particularly, the software allows an iteration-by-iteration execution of the three-point ladder which toggles a GPIO pin at the beginning of the `swap_points` function. When switched on, the GPIO notifies the oscilloscope to start the collection of power measurements.

Though these modifications create an unrealistic attack scenario, the experiment is *still* practical on unmodified software but requires additional effort of marginal complexity. In a real-world scenario, the adversary first requires to observe the power consumption of the target device by measuring the current through a shunt resistor in series between the microcontroller and the ground (or the voltage collector). The collection of power samples can then be synchronized on communication which requires an oscilloscope with a buffer of a few hundred million samples to capture the consumption of the entire three-point ladder. Finally, the parts which correspond to `swap_points` need to be identified in the collected trace, and then carefully segmented. A reader interested in this process is advised to read [Dug+16].

Source code. The final software on which power traces were acquired can be found here:
https://github.com/AymericGenet/SIKE-clusterswap-2021.

5.3.2 Experiment

5.3.2.1 Traces collection

To collect experimental power traces corresponding to `swap_points` executions, a simulation of an ephemeral SIKE key exchange was conducted:

(1) Program the DUT with a random key and seed.
(2) Generate and send three valid points Q, P, and $Q-P$.
(3) Repeat the following n times:
    (a) Make the DUT execute the next loop iteration.
    (b) Save the power trace from the oscilloscope.

The above was repeated 1,000 times (each time with a different key and seed) for SIKEp434 (hence $n = 218$). An example of a power trace along with its frequency components for SIKEp434 is shown in Figure 5.1. Note that most of the frequency components are zero due to the limiting analog bandwidth of the oscilloscope (20 MHz).

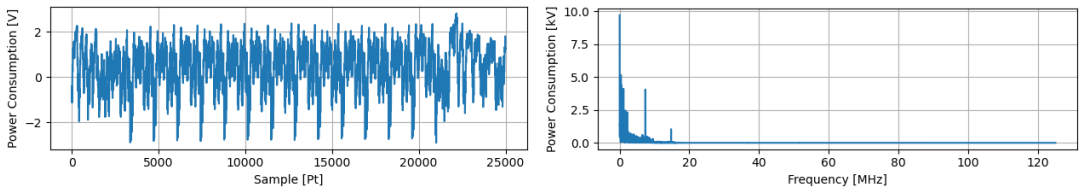


Figure 5.1: Example of one of the n power traces corresponding to `swap_points` in a single iteration of the loop (left) along with its Fourier representation (right).

5.3.2.2 Clustering power analysis procedure

In the next step of the experiment, the n collected traces of each experiment are exploited to attempt a key recovery as explained in Section 5.1 (cf. Algorithm 5.1).

(1) Process (for $0 \le i < n$):
    (a) T_i with an ℓ-level *wavelet transform* ($\hat{T}_i$ of length $\hat{M} = M/2^\ell$),
    (b) $\hat{T}_i$ with a *Fourier transform* ($\hat{F}_i$).
(2) Run the attack on both $\hat{T}_i$ and $\hat{F}_i$:
    (a) Go to the next sample location $0 \le t < \hat{M}$ (resp. $0 \le f < \hat{M}/2$).
    (b) Run clustering on $\{\hat{T}_i[t]\}_{0 \le i < n}$ (resp. on $\{\hat{F}_i[f]\}_{0 \le i < n}$).
    (c) Record the sk_t returned for time t (resp. sk_f).

The above was repeated with $0 \le \ell < 8$ levels of wavelet with a Symlet wavelet of filter length 8 (i.e., `sym4`) to further show the efficiency of the processing transforms. The success rate is calculated through all the timing positions and frequencies over the 1,000 sets of measured traces by comparing the recovered key with the correct key.

5.3.2.3 Results

Out of the 1,000 experiments, across all the levels $0 \leq \ell < 8$, the correct key is *always* found in the set of recovered keys sk_t or sk_f. Table 5.1 and Table 5.2 report various metrics about how often the correct key appears in the two sets of recovered keys. The independent success rates of each timing position and frequency are reported in Figure 5.3. Finally, examples of samples distributions successfully clustered is shown in Figure 5.2 both in timing and frequency.

Table 5.1: Statistics on the total number of timing locations which yield the correct key across the N = 1,000 experiments.

	k-means				Thresholding				
ℓ	min.	max.	$\mathbb{E}(\#t)$	SD($\#t$)	min.	max.	$\mathbb{E}(\#t)$	SD($\#t$)	$\hat{M}$
0	154	341	251.704	30.056	115	289	196.668	29.366	25000
1	72	172	125.611	15.153	58	144	97.923	14.764	12503
2	37	85	62.329	7.646	28	71	48.469	7.582	6255
3	15	44	29.425	4.171	11	36	22.958	3.971	3131
4	8	27	15.494	2.723	5	21	12.127	2.502	1569
5	6	21	13.445	2.371	4	18	10.531	2.195	788
6	2	12	6.036	1.615	1	9	4.033	1.408	397
7	0	5	1.645	0.941	0	5	0.853	0.878	202

Table 5.2: Statistics on the total number of frequencies which yield the correct key across the N = 1,000 experiments.

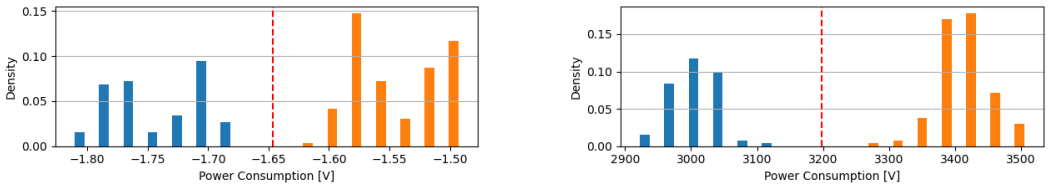
	k-means				Thresholding				
ℓ	min.	max.	$\mathbb{E}(\#f)$	SD($\#f$)	min.	max.	$\mathbb{E}(\#f)$	SD($\#f$)	$\hat{M}/2$
0	18	29	23.625	1.774	17	28	21.965	1.659	12500
1	16	26	20.704	1.630	15	24	19.382	1.584	6251
2	18	27	21.900	1.460	16	27	20.901	1.515	3127
3	15	30	22.641	2.288	13	26	19.993	2.239	1565
4	11	20	15.001	1.429	9	18	13.815	1.488	784
5	6	11	8.611	0.908	5	10	8.063	0.895	394
6	3	7	4.467	0.791	2	7	4.162	0.749	198
7	2	6	4.023	0.800	2	7	3.777	0.747	101

5.3.2.4 Discussion

The above experiment proves that the recommended Cortex-M4 implementation of SIKE from [Seo+20] is vulnerable to low-effort power analyses, even in the case when the implementation is protected with coordinate randomization. As a result, the main objective of the experiment is achieved.

The rest of the discussion focuses on the efficiency of the improvements.

Wavelet efficiency. Contrary to expectations, processing the power traces with the wavelet transform does not improve the success rate (cf. Figure 5.3). While the wavelet transform features noise filtering, information is still lost during the operation as the convolution involved in the transform combines significant power samples with insignificant ones. Nevertheless, the quality of the compression is fitting as the correct key still occurs on average more than once throughout the timing locations, even after several levels of filtering. Therefore, the number of samples to visit can be reduced by a significant factor (up to 2^6 according to our experiments) using this transform.



(a) $t = 1943$ (b) $f = 0.92$ [MHz]

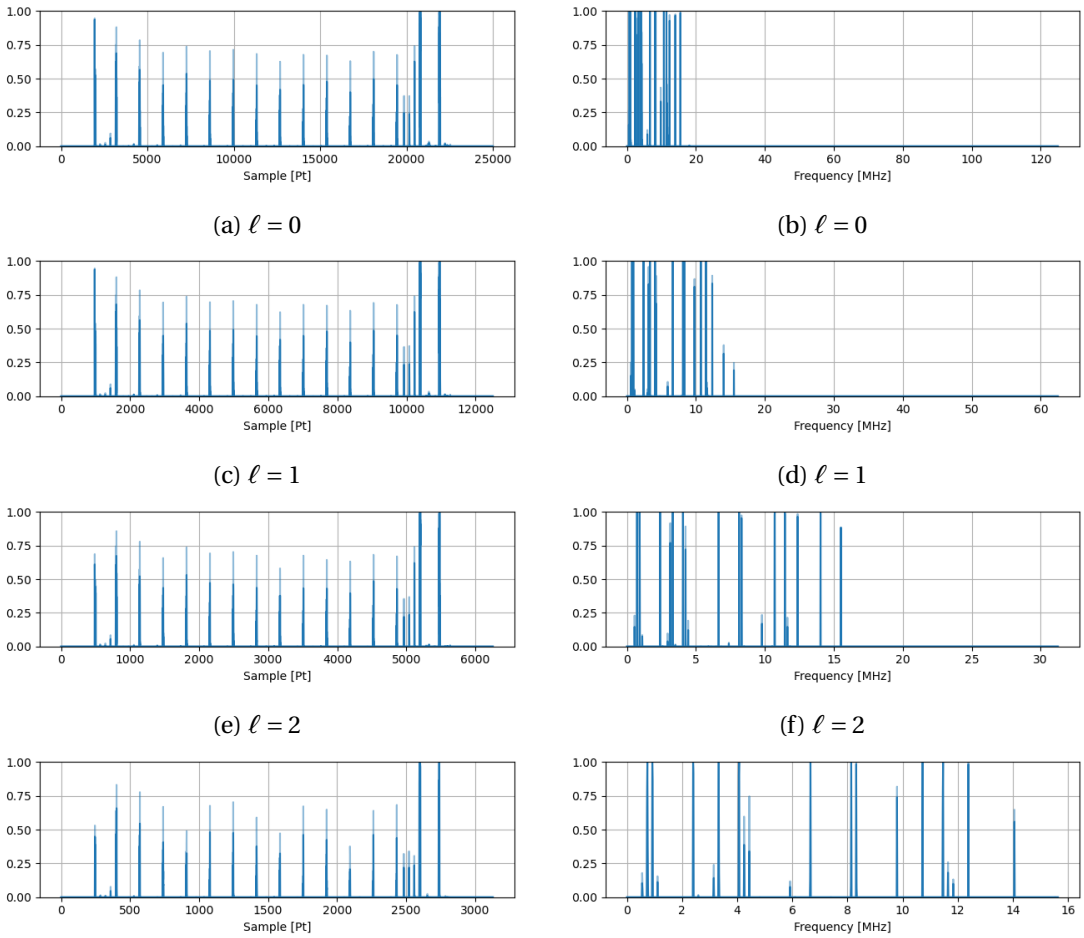
Figure 5.2: Example of a power sample distributions ($\ell = 0$). The threshold (in red) was found by Algorithm 5.2.

As the power trace corresponds to the execution of the `swap_points` function, the sample locations reported in Figure 5.3 correspond to specific instructions of the attacked implementation (shown in Listing A.7). In particular, the very first spike in the figure correspond to the mask computation which expands a secret bit. The regular spikes in the middle correspond to the swap formula performed on each of the 32-bit words of the points. Finally, the last two spikes at the end of the graph correspond to exiting the function. These spikes show all the aspects of the implementation that need protection.

Note that the choice of the wavelet function (`sym4`) was guided by an experimental exploration and that similar results are expected by using a different family or a different filter length.

Fourier efficiency. The Fourier analysis shows remarkable efficiency across all levels of wavelet transforms. Performing a clustering power analysis with frequency components rather than power samples is shown to have a resounding success rate across all experiments. Furthermore, such a success rate is kept throughout the wavelet levels, as the leakage happens at low frequencies which are preserved by the wavelet transform. The most notable observation to make is that clustering in the frequency domain is successful even at the last wavelet level where the same analysis in timing is shown to be inefficient. This proves that even though clustering power samples independently happens to be ineffective, their combination in the frequency domain may be sufficient to perform a successful analysis.

There may be many reasons why low frequencies leak most of the information in the current case. The frequencies of interest are suspected to be subharmonics of the clock speed. For instance, the spikes at 0.92 MHz likely correspond to the pattern of eight instructions in the



(a) $\ell = 0$ (b) $\ell = 0$

(c) $\ell = 1$ (d) $\ell = 1$

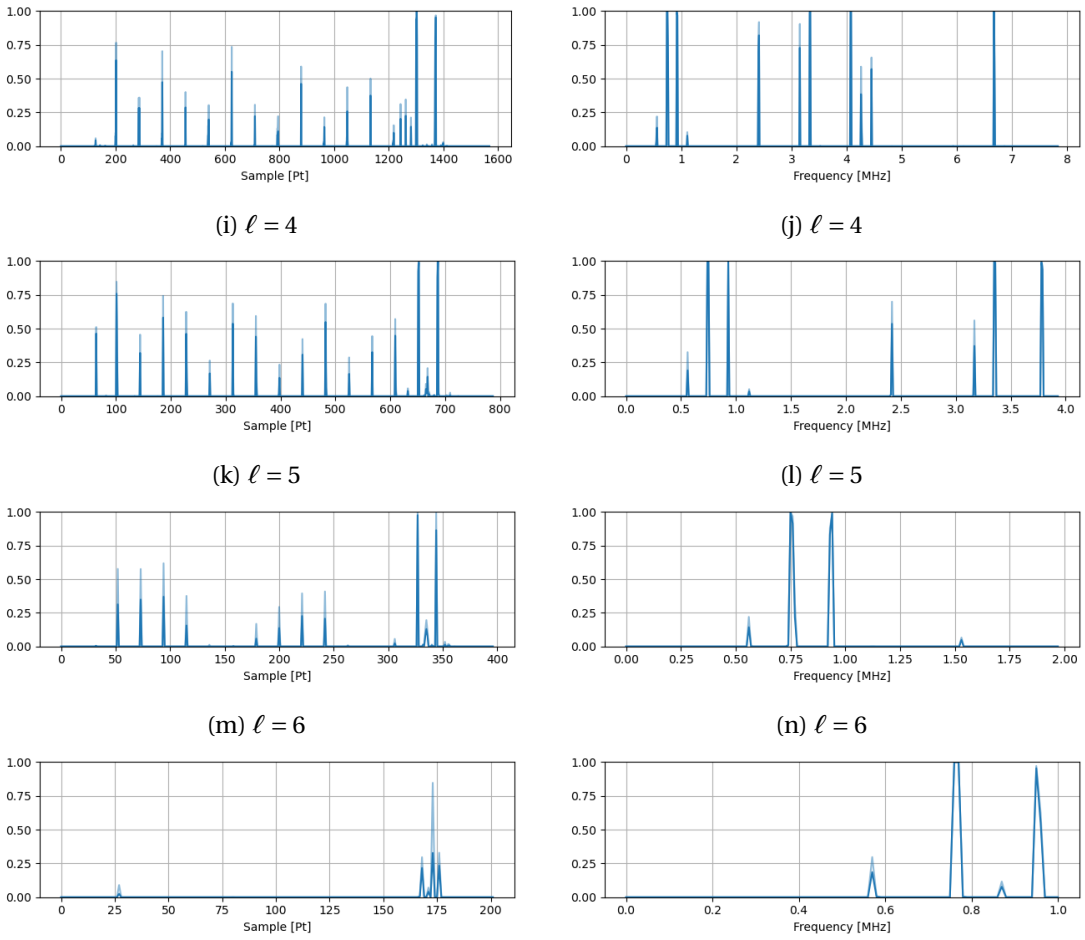
(e) $\ell = 2$ (f) $\ell = 2$

(g) $\ell = 3$ (h) $\ell = 3$

Figure 5.3: Success rate of the clustering power analysis (thresholding in opaque vs. k-means in transparent) at each timing locations (left) and frequencies (right) across different levels of wavelet transforms.

`swap_points` function (see Listing A.7). The same can be said for the spikes at 0.73 MHz and 0.74 MHz, as such frequencies also happen to be a tenth of the clock speed. These frequencies, as well as the other significant ones, may also be due to the consumption of sub-systems in the hardware (e.g., memory) that function at different paces.

Thresholding efficiency. In addition to demonstrating the efficiency of the pre-processing phase, the experiments show that the thresholding proposed in Algorithm 5.2 is almost as successful as k-means. While k-means still obtains better results (cf. Figure 5.3), our experiment with k-means took 29 hours to be performed, while the same analysis with thresholding took only 6.5 hours.



(i) $\ell = 4$ (j) $\ell = 4$

(k) $\ell = 5$ (l) $\ell = 5$

(m) $\ell = 6$ (n) $\ell = 6$

(o) $\ell = 7$ (p) $\ell = 7$

Figure 5.3 (cont.): Success rate of the clustering power analysis (thresholding in opaque vs. k-means in transparent) at each timing locations (left) and frequencies (right) across different levels of wavelet transforms.

Majority rule efficiency. The extremely high occurrence of the correct key in Table 5.1 and Table 5.2 confirms that the vertical majority rule explained in Section 5.2 helps validating the key. In all experiments, the most recorded candidate was always observed to be the correct key. As a result, the correct key is expected to be recovered within the first try-outs as the other candidates were all observed to be either random or close to the correct key.

Other SIKE instances. Note that the success of the clustering is closely connected to the relatively big number of samples available. As more samples are obtained, the distinction between the two clusters becomes easier. However, depending on the noise, additional samples may undermine the success of the overall clustering.

Still, similar results (if not better) have been obtained by running the same experiment with the bigger instances of SIKE. The experiments were executed with fewer runs, a fixed wavelet level, and only using the thresholding algorithm. The results are reported in Table 5.3 and prove that the attack is not limited to SIKEp434.

Table 5.3: Statistics on the total number of timing locations and total number of frequencies which yield the correct key across the $N = 10$ experiments with the other instances of SIKE ($\ell = 5$).

		Timing				Frequency			
p	n	min.	max.	$\mathbb{E}(\#t)$	SD($\#t$)	min.	max.	$\mathbb{E}(\#f)$	SD($\#f$)
503	252	8	14	10.5	2.5	9	11	9.7	0.7
610	304	16	31	22.1	5.3	10	14	11.7	1.3
751	378	17	25	22.2	2.7	9	13	10.6	1.5

5.4 Countermeasures

Protecting the point-swapping procedure against clustering power analysis is not obvious, as the attack defeats classical countermeasures of [Cor99] which include coordinate randomization, exponent randomization, point blinding, and even shuffling the for-loop. Moreover, due to the recent study which relies on deep learning [Per+21], even the tiniest bias in the power consumption may lead to a full recovery.

To make the task of protecting against the attack even more challenging, the target CPU of the Cortex-M4 is known to be hard to protect (see [Bat+23]). As the Cortex-M4 appears to leak in the Hamming distance of the two consecutive values in the pipeline registers (see [CGD18]), the countermeasures need not only to consider the Hamming weight of the processed values, but also the Hamming distance between the values used by two consecutive instructions.

In this section, a countermeasure based on thresholding the swapping mask is suggested.

5.4.1 Description

The proposed countermeasure revises the original swapping procedure from Section 3.3.2 in the following sense; instead of computing the value `mask` & $(u \oplus v)$ all at once, the idea is to split this quantity into two shares and add each share separately in a two-stage process (to both u and v). Such a procedure avoids computing values of extreme Hamming distances.

To this end, the swapping mask is replaced by two 32-bit masks: `m1` and `m2` such that their bitwise "xor" is equal to `mask`. In other words, given two consecutive private-key bits $\mathtt{sk}_{i-1}$,

and SK_i (for $0 \leq i < n$ with $\mathrm{SK}_{-1} = \mathrm{SK}_n = 0$):

$$\mathtt{m1} \oplus \mathtt{m2} = \begin{cases} \mathtt{0x00000000} & \text{if } \mathrm{SK}_{i-1} \oplus \mathrm{SK}_i = 0, \\ \mathtt{0xFFFFFFFF} & \text{if } \mathrm{SK}_{i-1} \oplus \mathrm{SK}_i = 1. \end{cases}$$

Given the two masks m1 and m2, the new procedure works as follows:

$$\begin{aligned} \mathtt{tmp1} &= \mathtt{m1} \,\&\, (u \oplus v), \\ \mathtt{tmp2} &= \mathtt{m2} \,\&\, (u \oplus v), \\ u &= (\mathtt{tmp1} \oplus u) \oplus \mathtt{tmp2}, \\ v &= (\mathtt{tmp1} \oplus v) \oplus \mathtt{tmp2}. \end{aligned}$$

Because of the property of $\mathtt{m1} \oplus \mathtt{m2}$, the above procedure swaps u and v in the same sense as the swapping procedure described in Section 3.3.2.

5.4.2 Implementation

The results from Section 5.3 provide insight on the critical points of the procedure that require particular care. Mainly three leaking points were identified:

1. The generation of the masks.
2. The instructions used to perform the swapping operation.
3. Exiting the function.

The third point can be avoided by incorporating the procedure into the code without calling a function, so only the first two points are addressed.

5.4.2.1 Masks generation

Let swap refer to the secret difference of private-key bits (i.e., $\mathtt{swap} = \mathrm{SK}_{i-1} \oplus \mathrm{SK}_i$). The suggested countermeasure involves generating two random masks m1 and m2 that are either equal or bit-wise complement depending on swap. To achieve this, given a random m1, the second mask m2 is derived with the following formula: $\mathtt{m2} = (1 - 2 \cdot \mathtt{swap})(\mathtt{m1} + \mathtt{swap})$. This makes m2 become the bitwise complement of m1 through the representation of negative numbers in the CPU with the two's complement (i.e., $\mathtt{m2} = -(\mathtt{m1} + 1)$ if $\mathtt{swap} = 1$).

Performance. Safely generating these two quantities for each bit processed requires sampling additional randomness. In particular, the multiplication of $(\mathtt{m1} + \mathtt{swap})$ by $(1 - 2 \cdot \mathtt{swap})$ is computed as $u_1(\mathtt{m1} + \mathtt{swap}) - u_2(\mathtt{m1} + \mathtt{swap})$ where $u_1 - u_2 = 1 - 2 \cdot \mathtt{swap}$. In total the mask generation requires at least 8 bytes of entropy (29 bits of which are effective) and introduces an overhead of at least 12 additional instructions when compared to the original mask computation. The code is given in Listing A.8.

5.4.2.2 The swapping operation

Because of the Cortex-M4 leakage model, the order of the operations and of the operands play a critical role in the countermeasure. Particular care has to be taken with store and load instructions, as the power consumption of these procedures leaks sensitive values. As a result, given the two masks `m1` and `m2` generated as before, the implementation of the countermeasure must follow a special order given in Listing A.9.

Performance. As opposed to the original pattern of 8 instructions, such a solution requires 14 instructions per iteration and doubles the numbers of loads and stores which introduces further delay.

5.4.2.3 Benchmarks

We compare the runtimes of our countermeasure against the runtimes of the unprotected version of SIKE. About 62% of the overhead stems from acquiring randomness for mask generation. As we generate a new mask for each swap (so for each word), we use a cheap pseudorandom number generator to limit the impact on performance; namely, the Tiny Mersenne Twister pseudorandom number generator [SM11] seeded with a 64-bit value obtained from a source of true randomness.

The protected `swap_points` is about 5.7 times slower than the unprotected swap. Within the three-point ladder function, the overhead adds up to about 70,000 additional cycles which takes up to 5% of the total computing time of the three-point ladder. When considered as a part of a full execution of SIKE, the overhead due to protecting the swap boils down from about 1% in the key generation and decapsulation to 0.7% in the encapsulation procedure, which is negligible.

Table 5.4: Runtimes (in cycles) of the SIKEp434 implementation with and without the countermeasure on an Intel i9-8950HK CPU @ 2.90GHz with Turbo boost turned off.

Operation	unprotected	protected
Mask generation	1	251
Swapping operation	71	148
Three-point ladder	1,172,432	1,241,721
Key generation	6,083,645	6,153,241
Encapsulation	9,893,673	9,962,113
Decapsulation	10,625,881	10,747,176

5.4.3 Experimental validation

The proposed countermeasure was validated by conducting Welch's t-test [SM16]. Such a test gives a degree of confidence that two classes of power samples are statistically indistinguishable. In the present case, the two classes respectively correspond to whether the points were swapped during the collection of the power traces, or not.

The t values are computed with the following formula:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\sigma_0^2/n_0 + \sigma_1^2/n_1}}$$

where μ_0, μ_1 correspond to the means of the two classes, σ_0^2, σ_1^2 to their variances, and n_0, n_1 to their cardinalities (here, $n_0, n_1 \approx 1000$). A threshold of 4.5 for the t values is set to reject the null hypothesis (see [Din+17]). In other words, a t value greater than the threshold gives evidence that the two distributions are not indistinguishable.

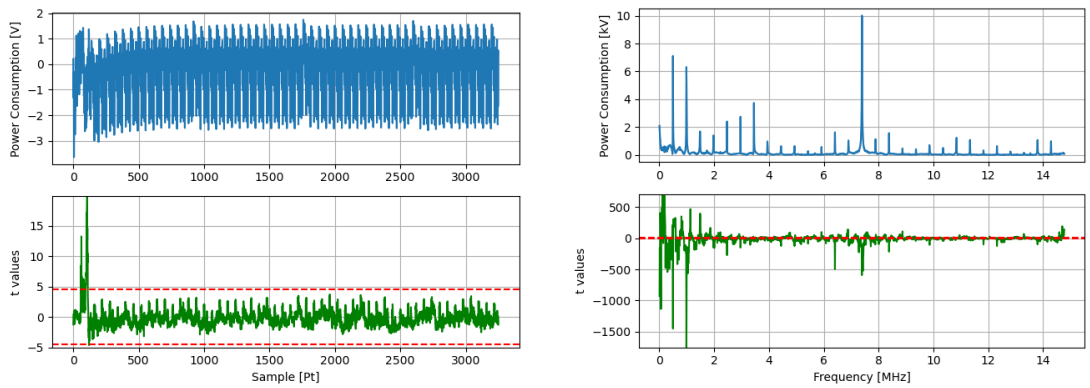
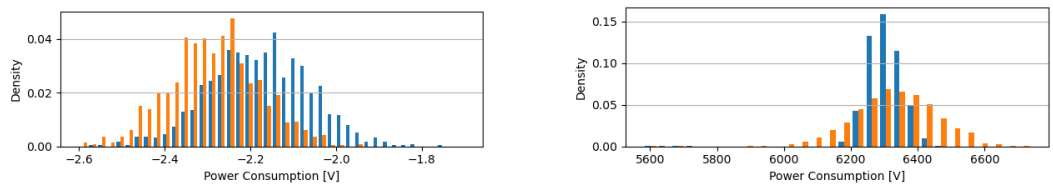


Figure 5.4: t-test of the countermeasure both in timing and frequency. The horizontal lines in red show the threshold above which the null hypothesis is rejected.



(a) $t = 101$ (b) $f = 0.98$ [MHz]

Figure 5.5: Power sample distributions at the locations which produced the highest value in both t-tests.

The results are shown in Figure 5.4. Even though significantly large t values appear in the plots, the attack is still unsuccessful when re-run against the countermeasure as the histograms corresponding to the power samples from the two classes overlap with each other at all points in time and frequency. Figure 5.5 illustrates this by showing the histograms at the highest peaks

of the t-test plots from Figure 5.4. As one can notice, both histograms exhibit a significant variance, which prevents the attack to fully[1] recover the private key. The histograms at all other points showed a similar overlapping.

While such a discrepancy of distributions prevents a successful clustering with the techniques described in this chapter, more sophisticated attacks (such as [Per+14; Per+21]) may still prevail. These attacks may therefore require additional efforts to withstand.

5.4.4 Other countermeasures

In addition to the countermeasure proposed, other techniques are likely to prevent a clustering power analysis of the swapping procedure. Desynchronizing the clock of the target device results in unaligned power traces with different random frequencies, so the attack is expected to be unsuccessful in neither domains. Such a countermeasure might be implemented by interleaving dummy `nop` instructions with the actual instructions of the regular `swap_points` function. Alternatively, swapping pointer addresses rather than values may be effective in the power domain but is shown to succumb to the same attack using electromagnetic radiations in [NC17].

5.5 Conclusion

The chapter described a plain clustering power analysis able to recover the entire private key in a single execution of the three-point ladder in the implementation of SIKE for Cortex-M4. While the attack by Castryck and Decru in [CD22] made our analysis no more pertinent for attacking SIKE, the chapter still demonstrated that, in clustering power analysis, processing the traces with a wavelet transform efficiently reduces the number of timing locations to visit, and that clustering frequency components may succeed even where clustering power samples is inefficient.

While the attack has been experimentally shown to be always successful, the reader must keep in mind that the experiment was performed using the ChipWhisperer framework on a chip that was deliberately made vulnerable to power analysis. However, the countermeasure described completely thwarts the attack even on such a vulnerable chip. If the countermeasure is safe under such defenseless circumstances, then the implementation can be assumed to be safe in a more realistic scenario.

As future work, the experiment could be repeated with a different clock speed to evaluate the evolution of the frequency components. Other improvements using, e.g., multiple samples of a single iteration in a multivariate clustering analysis may also be investigated. Also, the proposed countermeasure requires to be evaluated against other side-channel attacks and improved both in performance and security.

[1] The extent to which private-key bits can still be recovered is left as future work.

6 Zero-value power analysis of SIKE

The content of this chapter is based on the work:

[Feo+22] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluđerović, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. "SIKE Channels". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (2022). https://tches.iacr.org/index.php/TCHES/article/view/9701, pp. 264–289. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i3.264-289

Context. In the previous chapters, we presented two different ways of attacking the three-point ladder in SIKE through side-channel power analysis. While these attacks show that the elliptic curve scalar multiplication is a crucial operation to protect against power analysis, other components, such as the secret isogeny computation, may also be vulnerable to power analysis. Yet, to this day, no study of the power leakages of the secret isogeny computation has been conducted. As a result, the extent to which SIKE is vulnerable to power analysis without targeting the three-point ladder is unclear.

This chapter addresses this inquiry and shows a power analysis of the secret isogeny computation in SIKE, which therefore applies even when the three-point ladder is completely secure against power analysis. The attack is an instance of a ZVP attack, similar to the one from Akishita and Takagi in [AT03], that aims to distinguish the computation of zero values (from nonzero values) through power consumption due to their low energy cost. The idea relies on Goubin's refined power analysis from [Gou03] and consists of using special inputs that cause the device to perform operations with zero operands depending on the value of a bit of a static private key. Confirming or refuting the presence of a certain value being zero with power analysis therefore leads to the recovery of the private-key bit involved. Taking into account the recovered bit, this analysis can usually be repeated for all remaining bits of the private key.

As opposed to the ZVP attacks against SIKE presented by Koziel, Azarderakhsh, and Jao in [KAJ17] which still targets the three-point ladder, our attack is a special case of ZVP that targets the secret isogeny computation. In fact, our analysis shows that special torsion points cause an exception in the computation of the secret isogeny, resembling thus the exceptional procedure attack by Takagi and Izu in [IT03], forcing all subsequent computations to be zero. In particular, the computation of the j-invariant results in a zero value. Our attack therefore captures the power consumption of the j-invariant derivation when these special torsion points are provided, and attempts to detect if the power traces exhibit zero-valued computations within or not. As j-invariants can also be forced to be zero or nonzero regardless of the private key, we decided to distinguish the nature of j-invariants by comparing their power traces to baselines (i.e., templates when $j = 0$, and when $j \neq 0$) that can therefore be collected before starting the attack, in a similar fashion as the online template attack by Batina et al. from [Bat+19]. Moreover, we decided to compare the target power traces with the baselines using collision power analysis, as initially envisioned by Walter in [Wal01] and further improved by others in [FV03] and [Dan+16].

The same attack was simultaneously discovered by Wang et al. in [Wan+22] when the authors uncovered Hertzbleed; a physical weakness of the x86 processors in which the dynamic frequency scaling depends on the data being processed. In particular, when the data is zero, the processing of the x86 becomes faster than when the data is nonzero. This vulnerability enables a timing side channel which was showcased on SIKE using the attack of this chapter, as special torsion points cause an avalanche of zero-valued computations, leading thus to a practical static private key recovery with a timing attack. A month later, in [Adj+22], Adj et al. described a fault attack that forces zero values in the isogeny computation which, if effective, recovers a private-key bit.

Once again, as the attack by Castryck and Decru [CD22] defeats the security of SIKE, our analysis is outdated. Regardless, our work represents a pioneering contribution to the side-channel evaluation of secret isogeny derivations. As a result, Campos et al. conducted another power analysis of the isogeny computation in [Cam+22], which involves a similar key-recovery attack by detecting zero values (as well as values equal to six) in the curve coefficient. As their analysis applies also to all state-of-the-art CSIDH-based implementations, their work is still currently relevant. Also, we showed a novel way of recognizing zero-valued operations through collision power analysis using baselines.

Results. We introduce and confirm in the lab a ZVP attack against the official (uncompressed) SIKE implementation for Cortex-M4 [Seo+20] in semi-static settings and hardened with coordinate randomization. The attack consists of recovering the private key in an extend-and-prune fashion with special inputs that force intermediate values to be zero depending on a single secret bit. Although the possibility of this threat had been postulated previously, to the best of our knowledge, this is the first complete description of such attacks.

The attack targets the isogeny evaluation part of SIKE, which appears to have much greater side-channel leakage than the three-point ladder, permitting key recovery with as few as one trace per bit. In particular, we analyze the behavior of 3- and 4-isogeny evaluation formulas on invalid points, which appears to be novel in itself. We then show that our attack is easily, though not so cheaply, countered by partially validating ciphertexts.

Outline. Section 6.1 introduces our attack in an abstract way, while Section 6.2 reports on their experimental realization. Finally, Section 6.3 discusses the countermeasure, so the chapter can close with Section 6.4.

6.1 Attack description

We provide an attack based on forcing the computing party to evaluate some rational function on an elliptic curve point which has 0 as one of the coordinates, also called *"zero-value point"*. These points are $\mathcal{O} = [1:0]$, $T = [0:1]$ and the undefined point $[0:0]$. The attacking party creates a malicious public key in the key encapsulation procedure, made of three points $(Q, P, Q-P)$ which are used by the target during `Decaps`. The attack forces the computation of such points during the isogeny computation. Our final goal is to see where and how such points can occur, and to force a computing party to compute such points based on a secret bit of their private key. The attack is done adaptively in an extend-and-prune manner, that is, we perform the attack in multiple steps; at each step we recover one bit of the private key by assuming that we know the previous parts.

Preliminaries

We write $E_\alpha(\mathbb{F}_{p^2})$ a supersingular Montgomery curve that is uniquely defined by a coefficient $\alpha \in \mathbb{F}_{p^2}$ (as $\beta = 1$). In Section 3.3.4, we further defined $[A:C]$ (where $C \neq 0$) as the projective representation of α such that $\alpha = A/C$. Recall that we further defined the following notation:

- $[A^+_{24} : A^-_{24}] = [A+2C : A-2C]$ (for $A^+_{24} \neq A^-_{24}$) which represents $\alpha = 2(A^+_{24}+A^-_{24})/(A^+_{24}-A^-_{24})$.
- $[A^+_{24} : C_{24}] = [A+2C : 4C]$ (for $C_{24} \neq 0$) which represents $\alpha = (4A^+_{24} - 2C_{24})/C_{24}$.

Considering invalid internal states caused by malicious ciphertexts, an elliptic curve represented in SIKE may thus fall into one of the following categories:

The *undefined* curve, represented by $[A^+_{24} : C_{24}] = [A^+_{24} : A^-_{24}] = [0:0]$. This does not represent any algebraic object.

The *degenerate* curve, represented by $C_{24} = 0$ and $A^+_{24} = A^-_{24} \neq 0$. This is not, properly speaking, a curve.

The *singular* curves with $\alpha = \pm 2$, corresponding to $A^+_{24} = C_{24} \neq 0$ and $A^-_{24} = 0$, or to $A^-_{24} = C_{24} \neq 0$ and $A^+_{24} = 0$. These are not elliptic curves, because they exhibit a singularity in $(\mp 1, 0)$ and behave often as exceptional points in formulas.

Elliptic curves, for any value $\alpha \neq \pm 2$. These further subdivide into *ordinary* and *supersingular* curves. Of the $p^2 - 2$ possible values for α, only $\approx p/2$ yield a supersingular curve.

The points of the curve E_α are the projective solutions of the equation of a Montgomery curve (see Definition 3.1.11). SIKE drops the information on the y-coordinate, and represents the points as pairs $[X : Z]$, with X and Z not both zero. We shall make a slight abuse of language by calling $[X : Z]$ a point, given that it may correspond to up to two solutions of the Montgomery curve equation. Considering invalid internal states, an elliptic point in SIKE may be one of

The *undefined* point $[0 : 0]$. This does not correspond to any algebraic point.

The point at infinity $\mathcal{O} = [X : 0]$ with $X \neq 0$, the identity of the elliptic group law.

The *distinguished* point $T = [0 : Z]$, with $Z \neq 0$, of order 2. Assuming the curve is well defined, $[2]T = \mathcal{O}$.

The *special* 4-torsion points $[X : \pm X]$, with $X \neq 0$. Assuming the curve is well defined $[2]P = T$ for any such point.

An ordinary point $[X : Z]$ not belonging to any of the above. Assuming the curve is well defined, such a point is *on the curve* if $X/Z + \alpha + Z/X$ is a square in $\mathbb{F}_{p^2}$. Its algebraic properties, such as its group order, depend on all of X, Z and α.

Finally, we write the private key as $\mathrm{SK} = \mathrm{SK}_0 2^0 + \mathrm{SK}_1 2^1 + \ldots$ and we denote $\mathrm{SK}_{<k} = \mathrm{SK}_0 2^0 + \mathrm{SK}_1 2^1 + \cdots + \mathrm{SK}_{k-1} 2^{k-1}$.

6.1.1 Isogeny analysis

Our goal is to manipulate the isogeny kernel generating point R of the target party so that it becomes incompatible with isogeny formulas. This leads either to computation of undefined points $[0 : 0]$ (which propagate indefinitely) or to (heuristically) random computations. The two cases can be easily distinguished through power consumption.

We assume that the computing party derives an isogeny of degree 3^{e_3}. The argumentation can be adapted to 2^{e_2}-isogenies, or more generally to any set of SIKE parameters, as is shown in full generality in the conjoint work of [Feo+22], or in [Kal22, §5.3].

The isogeny algorithm uses a hard-coded strategy, and attempts to compute a 3^{e_3}-isogeny independently of the actual order of the kernel point R. The malicious public key points $(Q, P, Q - P)$, as well as the kernel point R, will be elements of the 2^{e_2}-torsion subgroup $E[2^{e_2}]$ (as opposed to $E[3^{e_3}]$ which is expected by the algorithm). Actually, we will show that there is an exponent $o > 0$ which satisfies so-called *leakage properties*:

- L1 If $\mathrm{ord}(R) \mid 2^{o-1}$ then the isogeny eventually computes undefined points $[0 : 0]$.
- L2 If $2^o \mid \mathrm{ord}(R)$ then the isogeny computes random values.

Algorithm 6.1 Malicious public key generation

Input: E_α – A supersingular elliptic curve.
Input: $\mathrm{SK}_{<k}$ – Known part of private key (k being the index of the bit being guessed).
Input: e' – An exponent s.t. $0 < e' \leq e_2$, $0 \leq k \leq e_2 - e'$.
Output: Malicious public key $\mathrm{PK}_k^j = (Q, P, Q-P)$.

1: Let Q_2, P_2 be two generator points of $E_\alpha[2^{e_2}]$.
2: Suppose that $[2^{e_2-1}]Q_2 \neq T$.
3: $S \leftarrow [2^{e_2-(e'-1)}]P_2$.
4: $Q \leftarrow [2^{e_2-(k+e')}]Q_2$.
5: $P \leftarrow S - [\mathrm{SK}_{<k}]Q$.
6: **return** $\mathrm{PK}_k^j = (Q, P, Q-P)$.

The exponent o depends on the isogeny degree, the tree-traversal strategy, and the order of the target party's point (all being public parameters), and can therefore be precomputed for any set of SIKE parameters.

In this section, we first show how an adversary can control the order of a point in such a way that it depends on the value of a private key bit. Then we show that there exists an exponent o which satisfies the leakage properties (L1, L2).

6.1.1.1 Computing the kernel point

The goal of the attack is to force the target party to compute a point of order $2^{o-1+\mathrm{SK}_k}$, SK_k being the value of the private key bit that we are trying to guess, and o the exponent satisfying leakage properties (L1, L2). We show in Algorithm 6.1 how we can create for any $e' > 0$ a public key PK_k^j such that the targets' party kernel point is of order $2^{e'-1+\mathrm{SK}_k}$.

The kernel generating point R obtained from the public key shown in Algorithm 6.1 satisfies the order constraint as is proved in the following lemma.

Lemma 1. *The kernel generator point $R = P + [\mathrm{SK}]Q$ generated from the public key PK_k^j of Algorithm 6.1 satisfies*

$$\mathrm{ord}(R) = \begin{cases} 2^{e'-1} & \text{if } \mathrm{SK}_k = 0, \\ 2^{e'} & \text{if } \mathrm{SK}_k \neq 0. \end{cases}$$

Proof. Following from Algorithm 6.1, we have $\mathrm{ord}(S) = 2^{e'-1}$ and $\mathrm{ord}(Q) = 2^{k+e'}$. Denote with $Q' = [2^k]Q$, a point of order $2^{e'}$. It follows that

$$P + [\mathrm{SK}]Q = S + [\mathrm{SK} - \mathrm{SK}_{<k}]Q = S + [\mathrm{SK}_k]Q' + [\mathrm{SK}_{k+1}2]Q' + \ldots = S + [\mathrm{SK}_k]Q' + Q''$$

where Q'' is a point of order dividing $2^{e'-1}$ and independent from S, by construction. Therefore, if $\mathrm{SK}_k = 0$ then $R = S + Q''$ is of order $2^{e'-1}$ because S is of said order, and S and Q'' are

independent. On the other side, if $\text{SK}_k \neq 0$ then R is of order $2^{e'}$ because it is the sum of $[\text{SK}_k]Q'$ of order $2^{e'}$, with $S + Q''$ of order $2^{e'-1}$. □

6.1.1.2 Computing the isogeny

In this subsection we will prove the existence of the exponent o which satisfies the leakage properties (L1, L2).

The 3^{e_3}-isogeny is computed by means of a hard-coded sequence of sub-algorithms which include point tripling, 3-isogeny computation, 3-isogeny evaluation, and saving and loading a point. The order in which these steps are executed is encoded in a strategy as explained in Section 3.3.3.

The kernel point provided to the isogeny is of incompatible order, which leads to irregular behaviors. During the execution of the 3^{e_3}-isogeny, geometric structure will be lost, and we will essentially work with random points on random elliptic curves. We show when irregular behavior starts, and what types of unexpected behavior can happen.

The 3-isogeny in SIKE satisfies the following properties:

- P0 If the kernel point is of incorrect order, then the image point (resp. curve) is arbitrary and does not share any known geometric relation with the preimage point (resp. curve).
- P1 For any point $R = [X : Z]$ we have $R + T = [Z : X]$.
- P2 If the image of $[X : Z]$ is $[U : V]$, then the image of $[Z : X]$ is $[V : U]$.
- P3 If the input point is equal to the kernel point, then the output is $\mathcal{O}$.
- P4 If the kernel point is $[X : Z]$ and the input is $[Z : X]$, then the output point is T.
- P5 The image of $\mathcal{O}$ is $\mathcal{O}$.
- P6 The image of T is T.

Property P0 is not proven and is based on heuristics. Given our experiments and the current understanding of elliptic curve isogenies, there is no evidence to the contrary. The other statements follow from the way the isogenies were constructed (see Section 3.3.4, or [Ren18]). Additional "degenerate" properties are given in the following list:

- P7 If the kernel point of the 3-isogeny is $\mathcal{O}$ or T, the image curve is degenerate.
- P8 On the degenerate curve, the tripling/image of both $\mathcal{O}$ and T is $[0 : 0]$.
- P9 If the kernel point of the 3-isogeny is T, the tripling/image of both $\mathcal{O}$ and T is $[0 : 0]$.

At this point the analysis of the isogeny computation boils down to analyzing the computations done on the public curve E_α recovered from PK_k^j (i.e., the first curve in the tree traversal). On this curve, the kernel point R is repeatedly tripled, and some intermediate results are saved as $R_i = [3^i]R$ for $i \in I$ where I is a set of indices determined by the strategy. These points are then evaluated with the isogeny of kernel $\langle [3^{e_3-1}]R \rangle$. The process is shown in Algorithm 6.2.

Algorithm 6.2 First vertical branch of the tree-traversal.

Input: Kernel point R, starting curve E_α, set of indices I.
Output: Image curve computed by the first 3-isogeny E'.
Output: Images of points evaluated by the first 3-isogeny $\{\phi_{[3^{e_3-1}]R}([3^i]R)\}_{i\in I}$.

```
for i = 0 to e_3 − 2 do
    if i ∈ I then
        R_i = R
    R = [3]R
E' = curve E_α/⟨R⟩
for i ∈ I do
    R_i = φ_R(R_i)
return (E', {R_i}_{i∈I})
```

Due to the fact that point R is of incompatible order, the image curve E' is an arbitrary, generally non-supersingular curve. From this point onward, the points and the curves are arbitrary. The only deterministic "structure" that the points can carry is that some of them may have projectively equivalent coordinates. There are three cases to consider.

(i) **There is a pair of saved points with equivalent coordinates.** Assume that R_a and R_b have projectively equivalent coordinates and $a < b \in I$. As the points are equivalent, their images through consecutive 3-isogenies will stay equivalent until we compute the isogeny generated by some image of R_b. This isogeny will send R_a to $\mathcal{O}$ (Property P3). The point $\mathcal{O}$ is fixed by point tripling and isogenies (Property P5). At a certain point an isogeny of kernel $\langle\mathcal{O}\rangle$ is computed, whose image curve is the degenerate curve (Property P7). The images of $\mathcal{O}$ are $\mathcal{O}$ (Property P5). The first next tripling will be a tripling of $\mathcal{O}$ on the degenerate curve whose output is the undefined point $[0:0]$ (Property P8). From this point onward all values will be 0, and the final j-invariant will be computed as $0/0$. An example of such computation is given in Figure 6.1.

(ii) **There is a pair of saved points with *flipped* coordinates.** Assume $R_a = [x:z]$ and $R_b = [z:x]$ and $a < b \in I$. The property of R_a, R_b having *flipped* coordinates is preserved (Property P2) until the image of R_b is used to compute an isogeny. This isogeny will send the image of R_a to T (Property P4). The point T is fixed by point tripling and isogenies (Property P6). Eventually an isogeny of kernel $\langle T\rangle$ is computed, whose image curve is the degenerate curve (Property P7). The first next image of T or $\mathcal{O}$ under the isogeny of kernel $\langle T\rangle$ is the undefined point $[0:0]$ (Property P9). From that point onward all values will be 0, and the final j-invariant will be computed as $0/0$.

(iii) **There are no points with equivalent nor *flipped* coordinates.** The points and curves became arbitrary after computing the first 3-isogeny (Property P0). From this point onward we have different arbitrary values which propagate. The final curve and its j-invariant are random.

Note that Case (i) is equivalent to the existence of $a < b \in I$ such that $[3^a]R = \pm[3^b]R$, while Case (ii) is equivalent to $[3^a]R = \pm[3^b]R + T$. In Case (iii), such $a < b \in I$ simply do not exist. These properties are characterized by the order of point R as shown in the following lemma.

Lemma 2. *For each set of SIKE parameters, let R be a point in the 2^{e_2}-torsion. Furthermore assume that $T \notin \langle R \rangle$. Then there is an integer $o > 0$ such that:*

1. $\text{ord}(R) \mid 2^{o-1}$ *if and only if Case (i) applies,*
2. $2^o \mid \text{ord}(R)$ *if and only if Case (iii) applies.*

If $T \in \langle R \rangle$, then the following is true for the same exponent o:

1. $\text{ord}(R) \mid 2^o$ *if and only if Case (i) or Case (ii) apply,*
2. $2^{o+1} \mid \text{ord}(R)$ *if and only if Case (iii) applies.*

Proof. The statement from Case (i) can alternatively be expressed as: "there are $a < b \in I$ such that $R_a = \pm R_b$, that is, $[3^a]R = \pm[3^b]R$". This reduces to a modular equivalence, more precisely $3^a \mp 3^b \equiv 0 \pmod{2^r}$, where we define r to be such that $\text{ord}(R) = 2^r$.

This equation is certainly satisfied for $r = 0$, for all $a, b \in I$. Furthermore, an equality modulo 2^r for some a, b reduces to an equality modulo $2^{r'}$ for all $r' \leq r$. Therefore it is only left to prove that the equation is not satisfied for some $r \leq e$, for all $a, b \in I$. This is proven by observing SIKE parameters, in particular by observing the strategies. We call the smallest such exponent o the *break-point exponent* (see Table 6.1).

The statement from Case (ii) can alternatively be expressed as: "there exist values $a < b \in I$ such that $R_a = \pm R_b + T$". If R and T are independent, this cannot happen. Therefore, the first part of the lemma is proven.

If R and T are dependent, we must have $[2^{r-1}]R = T$. Thus, Case (ii) reduces to $3^a \mp 3^b - 2^{r-1} \equiv 0 \pmod{2^r}$. This equation is equivalent to the following two equations:

$$\begin{cases} 3^a \mp 3^b \equiv 0 \pmod{2^{r-1}}, \\ 3^a \mp 3^b \not\equiv 0 \pmod{2^r}. \end{cases}$$

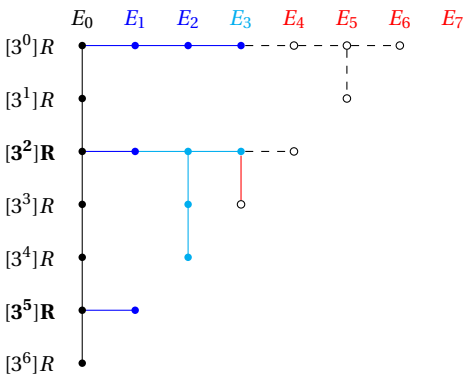
By the definition of the break-point exponent, $3^a \mp 3^b \equiv 0 \pmod{2^{o-1}}$ and $3^a \mp 3^b \not\equiv 0 \pmod{2^o}$, therefore $3^a \mp 3^b - 2^o \equiv 0 \pmod{2^{o+1}}$ for some a and b. On the other hand, $3^a \mp 3^b \not\equiv 0 \pmod{2^o}$ for all a and b implies $3^a \mp 3^b - 2^o \not\equiv 0 \pmod{2^{o+1}}$ for all a and b. □

A visual explanation of the isogeny attack. In Figure 6.1, we can see a 3^7-isogeny computation with a kernel of incompatible order.

- With black we denote regular points and supersingular elliptic curves.
- With blue we denote arbitrary points, isogenies whose image is random, and arbitrary (non-supersingular) elliptic curves.
- With cyan we denote the point $\mathcal{O}$, isogenies whose image is $\mathcal{O}$, triplings of $\mathcal{O}$ and degenerate elliptic curves.

- With red we denote the isogeny (tripling) which first creates the undefined point $[0:0]$, and undefined elliptic curves.
- With open circles we denote undefined points $[0:0]$.
- With dashed lines we denote isogenies which send points to the undefined point.

Figure 6.1: An example of a 3^7-isogeny computation with a kernel of wrong order.



Assume that $[3^2]R$ and $[3^5]R$ are equivalent. On the first curve E_0 the point R is tripled 6 times, and $[3^0]R$, $[3^2]R$ and $[3^5]R$ are saved. A 3-isogeny is computed from $[3^6]R$, and the saved points are evaluated. The images of $[3^2]R$ and $[3^5]R$ are still equivalent. Another 3-isogeny is computed from the image of the scalar-multiplication function $[3^5]$ which sends the image of $[3^2]R$ to $\mathcal{O}$, which is then tripled on the next curve. The next isogeny (of kernel $\langle\mathcal{O}\rangle$) has for codomain the degenerate curve. The first tripling of $\mathcal{O}$ on the degenerate curve outputs the undefined point $[0:0]$. From this point onward all the outputs are $[0:0]$.

6.1.1.3 Attack sketch

Using Lemma 2, two different outcomes of the isogeny computation can be forced depending on the value of a secret bit: the party either computes only zero values from a certain point in the tree traversal and onward, or completely random values. When zero values can be distinguished from random ones with a side channel, such a behavior enables an adaptive bit-by-bit key recovery.

We propose to perform the zero-value distinction on the subroutine responsible of the subfield inversion within the j-invariant computation. This is because the j-invariant computation occurs at one of the last steps of the key exchange, making it a conveniently identifiable target, and because the subfield inversion is usually computed as $a^{-1} = a^{p-2}$; a noticeable sequence of ≥ 200 similar field operations. This scenario is illustrated in Algorithm 6.3.

6.1.1.4 Other SIKE instances

The attack was analyzed in the case of the target computing an isogeny of degree 3^{e_3}. In the general case, the isogeny is of degree $\ell_I^{e_I}$ and the cardinality of the curve is of degree $\ell_I^{2e_I}\ell_J^{2e_J}$. The private key is extracted in base-ℓ_J digits per turn, $sk = sk_0\ell_J^0 + sk_1\ell_J^1 + \ldots$ and the point R has an order of a power of ℓ_J. The exponent o can be found with the same procedure as in Lemma 2, where $(3, e_3, 2, e_2)$ is swapped with $(\ell_I, e_I, \ell_J, e_J)$. In particular, the

Algorithm 6.3 Attack scenario relating to the isogeny computation.

Input: o – The break point as found in Lemma 2.
Output: The private key SK.

for $k = 0$ **to** $e_2 - o$ **do**
  Assume we know $\text{SK}_{<k} = \sum_{i=0}^{k-1} \text{SK}_i 2^i$.
  Generate PK_k^j with $(k, sk_{<k}, o)$ as in Algorithm 6.1.
  Send PK_k^j to the target.
  Detect exponentiation with side-channel analysis:
  if computation of 0^{p-2} is detected **then**
    $\text{SK}_k = 1$,
  else
    $\text{SK}_k = 0$.
Perform an exhaustive search on the remaining bits of the private key.
return SK

case with T can never happen unless $\ell_J = 2$. We report the values for o for both parties and all parameter sets in Table 6.1. Attacks on all SIKE parameter sets are provided in [Feo+22], and also included on our GitHub page: https://github.com/nKolja/SIKE-zero-value-attacks. The work by Kaluđerović [Kal22] also provides illustrations.

Table 6.1: Break-point exponents o for all parameter sizes.

Instances	SIKEp434	SIKEp503	SIKEp610	SIKEp751
2^{e_2}-isogeny	3	4	2	5
3^{e_3}-isogeny	9	7	7	8

6.2 Experimental verifications

In this section, we verify the correctness of the attack in practice.

6.2.1 Setup

Hardware. The experimental evaluation of the attack was performed on an STM32F3 as the DUT, using the ChipWhisperer framework as described in Section 2.1.1. The clock speed of the DUT was set to 44 MHz, and the oscilloscope was programmed with:

- A bandwidth of 200 MHz.
- A sampling rate of 250 samples per μs.
- A resolution of 10 bits per samples.
- A memory of 25,000 samples per acquisition.

Moreoever, we measure the power consumption through a 20dB Low-Noise Amplifier (LNA).

Software. The attacked software calls the functions of the recommended implementation of SIKEp434 for 32-bit Cortex-M4 microcontrollers with input ciphertexts received from the computer. This software enables the acquisition of power consumption of specific operations during the execution of the SIKE key decapsulation.

In the target software code the host computer sends a ciphertext to the target, and the target computes the shared secret with the decapsulation procedure using a static private key. The target device runs a custom version of ChipWhisperer's simpleserial library, like the one in Section 4.3.

Moreover, the scalar multiplication of the library is protected with coordinate randomization. As the original library does not offer such a countermeasure, coordinates are randomized after computing the coefficient of the received curve, and before the Montgomery three-point ladder, as in Section 5.3. A random representation of the points $(Q, P, Q-P)$ is generated from the received affine coordinates (x_Q, x_P, x_{Q-P}). This countermeasure consumes $6 \times \log_2(p)$ random bits to generate three random Z coordinates and requires three $\mathbb{F}_{p^2}$ multiplications.

Finally, the code is further modified to allow a GPIO to trigger the side-channel trace collection of the oscilloscope. When toggled, the GPIO notifies the oscilloscope to start the capture of the power consumption of the DUT. The purpose of this modification is to make the collection of traces more convenient. Note, however, that the attack is still applicable without a trigger and that the synchronization of traces can be performed using, e.g., cross-correlation techniques [Dug+16].

6.2.2 Experiment

The following experiment describes a proof of concept for the attack on the j-invariant computation as described in Section 6.1.1 using power analysis.

6.2.2.1 Traces collection

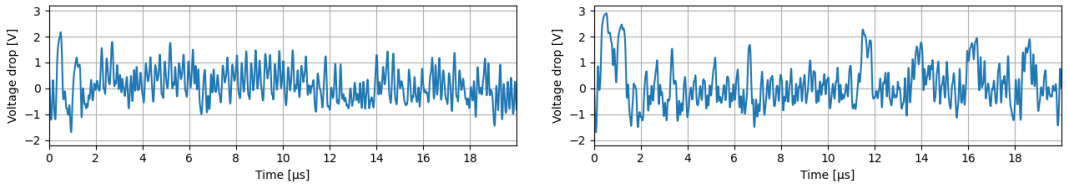
In the experiment, only the power consumption of the first field multiplication (i.e., `fpmul_mont` in the source code, shown in Listing A.10) from the modular inversion involved in the computation of the j-invariant is measured. As a myriad of zero-valued operations are executed when the j-invariant is undefined (i.e., $[0:0]$), it would be superfluous to capture the entire computation of j and compare every operation involved. Still, this specific multiplication was selected because the same function is called a total of 93 times during the modular inversion (with all zeros when the j-invariant is undefined). Accordingly, in case the leakage of one field multiplication alone is not enough to correctly detect the presence of zero values, a single trace including all the calls to the field multiplication can be segmented into multiple sub-power traces to boost the accuracy of the comparison (even though doing so turned out to be unnecessary in our experiment).

6.2.2.2 ZVP procedure

The experiment followed the approach with the two baselines as described in Section 2.1.3.3. Let E_α be a random curve, Q_2, P_2 generators of $E_\alpha[2^{e_2}]$ with $[2^{e_2-1}]Q \neq T$, and let n correspond to the number of bits in a private key (i.e., $n = 218$ for SIKEp434) and o to the order corresponding to the breaking point between zero or nonzero j-invariants (i.e., $o = 9$ for SIKEp434). The steps taken by each single experiment are the following:

1. Set up a fixed random private key SK on the DUT.
2. Capture the baselines $\mathrm{B}^{(0)}$, $\mathrm{B}^{(*)}$ for the two categories—zero and random:
   (a) Send $Q^{(0)} = [2^{e_2-(o-1)}]Q_2$, $P^{(0)} = [2^{e_2-(o-1)}]P_2$, $P^{(0)} - Q^{(0)}$ to capture $\mathrm{B}^{(0)} \in \mathbb{R}^m$.
   (b) Send $Q^{(*)} = [2^{e_2-(o+1)}]Q_2$, $P^{(*)} = [2^{e_2-(o+1)}]P_2$, $P^{(*)} - Q^{(*)}$ to capture $\mathrm{B}^{(*)} \in \mathbb{R}^m$.
3. For all recoverable bits $0 \leq i < n - o$ (starting with $\mathrm{SK}' = 0$):
   (a) Send $Q_i = [2^{e_2-(i+o)}]Q_2$, $P_i = [2^{e_2-(o-1)}]P_2 - [\mathrm{SK}']Q_i$, $P_i - Q_i$.
   (b) Capture $\mathrm{Tr}_i \in \mathbb{R}^m$.
   (c) If $\rho(\mathrm{Tr}_i, \mathrm{B}^{(*)}) > \rho(\mathrm{Tr}_i, \mathrm{B}^{(0)})$ then $\mathrm{SK}' = \mathrm{SK}' + 2^i$.
4. Return SK'.

An example for the two baselines which both consist of $m = 4{,}960$ samples is shown in Figure 6.2.



(a) Zero-valued baseline $\mathrm{B}^{(0)}$.

(b) Random-valued baseline $\mathrm{B}^{(*)}$.

Figure 6.2: Examples of baseline traces corresponding to a single $\mathbb{F}_{p^2}$ multiplication processing zero values in one case, and random (nonzero) values in the other.

6.2.2.3 Results

Across $N = 1{,}000$ experiments, the first $n - o = 208$ private-key bits were always successfully extracted through collision power analysis with baselines. Table 6.2 shows the average correlation coefficients when a target trace is compared against the two baselines. This outcome shows that the recommended implementation of SIKE for Cortex-M4 is vulnerable to the zero-value attack on the j-invariant computation as described in Section 6.1.1.

Table 6.2: Average PCCs between baselines and target traces ($N = 1,000$).

	Target	
Baseline	$j = [0:0]$	$j \neq [0:0]$
$j = [0:0]$	**0.9975**	0.3915
$j \neq [0:0]$	0.3916	**0.9909**

6.2.2.4 Discussion

Given the significant correlations for a single field multiplication, the results give strong evidence that zero values can be easily detected by comparing the power consumption of an operation with a baseline.

6.3 Countermeasures

Scalar randomization is a classic countermeasure against zero-value attacks in ECC [Cor99]. It could be adapted to protect the three-point ladder in SIKE, but would be useless against the isogeny computation attack. We propose instead a countermeasure that protects SIKE against both.

6.3.1 CLN test

Our attack relies on ciphertexts containing maliciously generated point triplets (Q, P, R) which are not the legitimate images $(\phi(Q_3), \phi(P_3), \phi(Q_3 - P_3))$ of the public 3^{e_3}-torsion basis under an isogeny of degree 2^{e_2}. As we already mentioned, before the attack by Castryck and Decru [CD22] was discovered, validating SIKE ciphertexts was a problem believed to be as hard as breaking SIKE itself, thus we could not hope to completely rule out side-channel attacks using malicious ciphertexts. Nevertheless our malicious ciphertexts deviate from the legitimate format in a detectable way, letting us design an effective countermeasure.

Indeed, the attack on the isogeny computation uses points of order 2^n instead of 3^{e_3}. To counter it, it is enough to check that E_α is a supersingular curve, that P and Q are both of order 3^{e_3}, and that they generate $E_\alpha[3^{e_3}]$, i.e., that[1] $[3^{e_3-1}]P \neq [\pm 3^{e_3-1}]Q$. Note that by construction we automatically have that $R = Q - P$. We shall name this the *CLN test*, after the names of its first proponents [CLN16].

The original test in [CLN16] did not verify that E_α is supersingular, however this can be done at little extra cost. First, we check that α defines an elliptic curve by excluding the undefined, the degenerate, and the singular cases described in the Preliminaries. Then, we check that Q and P generate $E_\alpha[3^{e_3}]$, which proves that $\#E_\alpha(\mathbb{F}_{p^2}) = 3^{2e_3}D$ for some integer D. This nearly implies

[1] Equivalent conditions are that R is also of order 3^{e_3}, or that the Weil pairing $e_{3^{e_3}}(Q, P)$ has multiplicative order 3^{e_3}.

E_α is supersingular; by Hasse's bound, $(p-1)^2 \leq 3^{2e_3}D \leq (p+1)^2$ hence $2^{2e_2} - 4p/3^{2e_3} \leq D \leq 2^{2e_2}$. Because $3^{2e_3} \approx p$, only a few choices are possible for D, the largest one corresponding to a supersingular curve. It is then enough to find some power $2^d \mid D$ such that $2^d > 4p/3^{2e_3}$, then, dividing all sides by 2^d, we conclude that the curve is supersingular.

For all SIKE proposed parameters, except the NIST IV parameter SIKEp610, it turns out that $4p/3^{2e_3} < 2$. But any Montgomery curve has order divisible by 4 (see [CS18]), thus we are done. For SIKEp610, $2(p-2)/3^{2e_3} < 8$; checking that $A+2$ and $A-2$ are both squares in $\mathbb{F}_{p^2}$ ensures that $8 \mid D$ (see [CS18, Table 1]), proving that E_α is supersingular.

Remark 1. Swapping the roles of 2 and 3, analogous checks would also work for verifying public keys. However, $4p/2^{2e_2}$ tends to be quite large for SIKE instances: as much as ≈ 447.6 for SIKEp751. It is thus not realistic to look for simple algebraic conditions that would guarantee the existence of points of small order 3^n. Instead, one may take random points on E_α, and multiply them by a cofactor until a point of sufficiently large order 3^n is found. Alternatively, one may be just content with testing that $2^{2e_2} | \#E_\alpha(\mathbb{F}_{p^2})$: although this does not guarantee that the curve is supersingular, it is believed to be computationally hard to find ordinary curves with such a large fixed factor in their order.

This also applies to compressed SIKE, where the roles of the 2^{e_2}- and 3^{e_3}-torsion are swapped. The *entangled basis generation* procedure of [Zan+18] guarantees[2] that $2^{2e_2} \mid \#E_\alpha(\mathbb{F}_{p^2})$, and ciphertext decompression ensures that (Q,P) generates $E_\alpha[2^{e_2}]$. Implementations may then choose between relying on a computational assumption ensuring that E_α is supersingular, or doing a little extra work to find a point of appropriate order 3^n on E_α. At any rate, our attack does not apply to compressed SIKE because of this.

We added the countermeasure to the SIKEp434 implementation described in Section 6.2.1 and tested it on an STM32F4-DISCOVERY clocked at 168 MHz. Without the CLN countermeasure, 95,899 k-cycles are needed for the decapsulation, and with it, 108,273 k-cycles are needed. There is thus a performance hit of around 12.9%.

6.4 Conclusion

We described a zero-value attack against SIKE on the isogeny computation. The attack is based on special-point inputs that enable an adaptive bit-by-bit key recovery. We analyzed it in theory, but also verified it experimentally on the recommended SIKE implementation for Cortex-M4 with power analysis using collision power analysis. At last, we argued that the Costello–Longa–Naehrig test which verifies the order of the points is sufficient to stop ZVP attacks.

Even though our attack has been outdated by the recent key recovery due to Castryck and Decru [CD22], we have exhibited the first power analysis of the isogeny computation, which is

[2] The proof therein requires that $A^2 - 4$ is a square in $\mathbb{F}_{p^2}$, thus an implementation of compressed SIKE expecting malicious ciphertexts should check this condition before generating the basis.

expected to lead to countless similar attacks, such as the one by Campos et al. in [Cam+22] that is still applicable to CSIDH. Moreover, we explored a new scenario in which collision power analysis with baselines is used to mount a zero-valued power analysis, which could have applications beyond this particular attack.

Hash-based cryptography Part II

7 SPHINCS and SPHINCS+

Hash-based cryptography defines cryptographic schemes whose security is solely based on the properties of (cryptographic) hash functions. Informally speaking, a hash function transforms a bitstring input of any size into a fixed-size bitstring output, known as a *hash*, in such a way that computing the hash from any input is easy but recovering the original input from the hash is hard. This property gives rise to a variety of hash-based cryptosystems that notably includes digital signatures and zero-knowledge proofs.

Hash-based digital signature schemes are of particular interest because they provide a unique security assumption to public-key cryptography which is often deemed conservative, as hash functions have been thoroughly analyzed over the past three decades and are therefore considered well understood. In such schemes, the signer generates a public key by hashing secret values with a public hash function, and then signs a message by revealing only a subset of the secret values depending on the message being signed. A verifier checks the correctness of the signature by hashing the revealed values and verifying that the resulting hashes correspond to the public key.

There exist three main categories of hash-based digital signature schemes:

- One-Time Signature (OTS) schemes, where any signing key should be used at most *once*,
- Multiple-Time Signature (MTS) schemes, which combine multiple OTS key pairs to provide a *limited number* of signatures (after which the scheme simply stops working),
- Few-Time Signature (FTS) schemes, in which the security of one key pair slowly deteriorates with the number of uses (and gets compromised if used too many times).

Moreover, a hash-based digital signature scheme is considered *stateful* if the signer is required to keep track of the signatures used, and *stateless* otherwise.

History. The first hash-based digital signature scheme dates back to 1979 where Leslie Lamport developed the Lamport–Diffie One-Time Signature (LD-OTS) scheme [Lam79] in

which the signer commits to two secret values per bit to be signed, and reveals only one of them for each bit of the message. In 1989, Ralph Merkle creates the Merkle Signature Scheme (MSS) [Mer90] which combines multiple key pairs of one-time signatures with a binary hash tree to create a stateful MTS. In the same publication, Merkle introduces the Winternitz One-Time Signature (W-OTS) scheme which improves the LD-OTS by enabling multiple bits to be signed at once with a single value, which was later improved into W-OTS$^+$ by Hülsing in [Hül13] to provide shorter signatures with the same security guarantees. Later, Adrian Perrig introduces the Bins-and-Balls (BiBa) one-time signatures [Per01] which involves finding two secret values from a list of secret values that hash into a same value. This idea was further developed by Leonid and Nathan Reyzin in their Hash to Obtain a Random Subset (HORS) scheme [RR02]. HORS is the first FTS scheme and works as follows: the signer reveals per signature a small number of secret values from a large list of pre-hashed secret values, allowing thus more than one message to be signed without significantly compromising security.

MSS has known many improvements to make the scheme practical for real-world applications, including notably the eXtended Merkle Signature Scheme (XMSS), introduced by Buchmann, Dahmen, and Hülsing in [BDH11], that provably enables forward security with only a pseudo-random function, and a second-preimage resistant function. Two years later, XMSS was itself improved into Multi Tree XMSS (abbreviated XMSS^{MT}) by Hülsing, Rausch, and Buchmann to enable a virtually unlimited number of signatures by creating a hypertree of XMSS; an idea borrowed from the Coronado MSS (CMSS) by Coronado García et al. [Buc+06]. In 2015, Bernstein et al. develop SPHINCS [Ber+15]; the first practical stateless hash-based digital signatures built upon XMSS^{MT}. The scheme involves multiple improvements over classical MSS, including Oded Goldreich's approach [Gol01] that makes MSS stateless by replacing each node in the hash tree by key pairs of one-time signatures, an enhanced MSS that requires only a hash function resistant to second-preimage attacks, and an extra layer of HORST—an improved version of HORS.

In 2017, Hülsing et al. submit SPHINCS$^+$ [Hül+20] as an improvement over SPHINCS to the NIST post-quantum standardization process. Compared to SPHINCS, SPHINCS$^+$ features stronger security guarantees (especially against multi-target attacks), lower memory requirements, faster signing and verification, and more flexibility in terms of public key and signature sizes, thanks to the following changes:

- The replacement of HORST by a novel FTS scheme named Forest Of Random Subset (FORS) which, overall, performs better than HORST in the context of SPHINCS.
- The introduction of a verifiable index that changes the way a path is chosen in the hypertree.
- The use of keyed hash functions that the authors refer to as tweakable hash functions.
- The replacement of the W-OTS$^+$ public-key compression with a tree-less hash function.
- Two new instantiations: robust and simple, which offer a trade-off between security models and speed.

7.1 Background

7.1.1 Notation

Table 7.1 summarizes the different notations used throughout the following chapters.

Table 7.1: Notations for Chapters 7—9.

Expression	Meaning
$\mathbb{B}$	The set of bytes (i.e., $\mathbb{B} \cong \{0,1,\ldots,255\}$).
$x \| \| y$	The concatenation of two bitstrings x and y.
$x \oplus y$	The bitwise eXclusive OR (XOR) of two bitstrings[1] x and y.
$x \lll n$	The bitwise left rotation of n bits in the bitstring x.
$\mathcal{U}(S)$	A uniform random variable defined on set S.

7.1.2 Functions

Hash-based digital signatures rely on functions which look like random. Pseudorandomness is an essential concept in cryptography that has many applications. For instance, a *pseudorandom number generator* is a deterministic algorithm that, when fed a (secret) input seed, outputs a sequence of generated numbers that is indistinguishable[2] from a sequence of truly random numbers. Similarly, a *pseudorandom function* is a mapping of byte strings, depending on a key, that is indistinguishable[2] from a truly random mapping. These notions are formally defined in [Gol04]. As the analyses in the current thesis do not rely on these formal notions, we will call pseudorandom function a function which makes "random-looking" outputs.

A fundamental property of pseudorandom functions is their *one-way property*, that is, there does not exist any probabilistic algorithm that can be expected to recover the input of a (public) pseudorandom function in polynomial time given their output (see [GGM84] for proof). This property enables the commitment of secret values to be secure.

With this interpretation in mind, we define the following pseudorandom functions:

- $F : \mathbb{B}^n \to \mathbb{B}^n$ refers to a *size-preserving pseudorandom function*,
- $H : \mathbb{B}^{2n} \to \mathbb{B}^n$ refers to a *compression pseudorandom function*, and
- $T_l : \mathbb{B}^{ln} \to \mathbb{B}^n$ refers to an *l-compression pseudorandom function* (where $l > 2$).
- $G_l : \mathbb{B}^n \to \mathbb{B}^{ln}$ refers to a *pseudorandom number generator* (where $l > 0$).

Definition 7.1.1 (Chaining pseudorandom function)**.** Given a size-preserving pseudorandom function $F : \mathbb{B}^n \to \mathbb{B}^n$, a *chaining pseudorandom function* $C^i : \mathbb{N} \times \mathbb{B}^n \to \mathbb{B}^n$ consists of

[1] The bitstrings are supposed to have the same length. If not, zero bits are usually appended to the shortest bitstring to match the length of the longest bitstring.

[2] Two objects are said to be indistinguishable from each other if there does not exist any probabilistic algorithm that can be expected to tell them apart (see [Gol04, §3.2] for details).

recursively applying F a specified number of times $i \geq 0$ starting with an initial input $x \in \mathbb{B}^n$:

$$C^i(x) = \begin{cases} C^{i-1}(F(x)) & \text{if } i > 0, \\ x & \text{else } (i = 0). \end{cases}$$

Given a chain-length $W > 0$, the sequence $(C^k(x))_{0 \leq k < W}$ is referred to as a *pseudorandom chain* (or, simply, a *chain*). The *position* of an element $y \in (C^k(x))_{0 \leq k < W}$ is the number $i \geq 0$ such that $C^i(x) = y$.

Figure 7.1 illustrates a pseudorandom chain of length $W = 8$. In the figure, the position of the rightmost element y is 7, as $y = C^7(x)$.

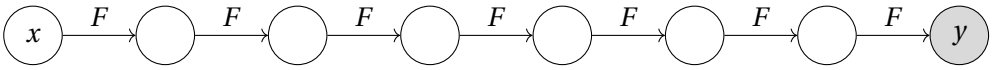


Figure 7.1: Illustration of the chaining pseudorandom function.

Remark 1. All chaining pseudorandom functions satisfy $C^{i+j}(x) = C^j(C^i(x)) = C^i(C^j(x))$.

Definition 7.1.2 (Cryptographic hash function [MvV97])**.** A *cryptographic hash function* is an easy-to-compute function $\mathscr{H} : \{0,1\}^* \to \mathbb{B}^n$ which maps bit strings of arbitrary finite length to byte strings of fixed length n and fulfills the following properties:

1. *first preimage resistance*: for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that $\mathscr{H}(x') = y$ when given any y for which a corresponding input is not known.
2. *second preimage resistance*: it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage $x' \neq x$ such that $\mathscr{H}(x) = \mathscr{H}(x')$.
3. *collision resistance*: it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that $\mathscr{H}(x) = \mathscr{H}(x')$.

7.1.3 Treehash

A *binary hash tree* (also known as a *Merkle tree*) is a structure of hash nodes in which an initial number of hash leaves are combined two at a time using a pseudorandom compression function. This process continues until a single value—referred to as the *tree root*—is reached. The hash nodes are organized in such a way that the root is on the top and the leaves at the bottom. A *level* refers to the distance of a row of nodes from the deepest leaves. A binary hash tree with a power-of-two number of hash leaves $m > 0$ will therefore have a total of $\log_2(m)$ levels, known as the *height* of the tree. Figure 7.2 illustrates the terminology.

The process of combining nodes two at a time in a binary hash tree is called the *treehash process*. Algorithm 7.1 provides the specific steps for calculating the tree root for an initial

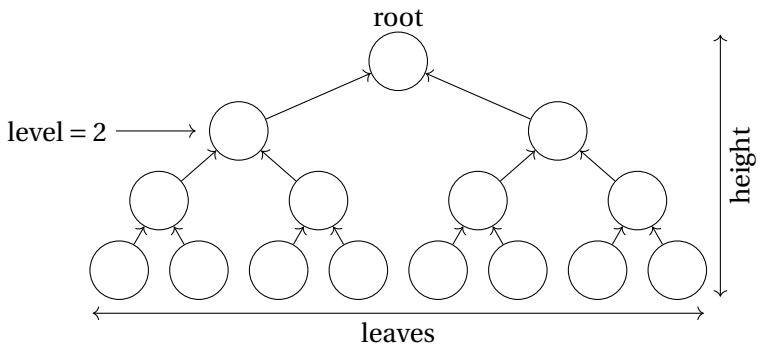


Figure 7.2: Illustration of a Merkle tree.

number of leaves that may not necessarily be a power of two. In this regard, if the number of nodes at a certain level is odd, the right-most node does not have a neighbor to be compressed with and must be moved to the next level. This algorithm is memory-efficient because the nodes are computed in place.

7.1.4 Paths

A *path* in a hash tree refers to the nodes encountered while traversing the tree from a specific leaf to the root. Concretely, given a leaf index $0 \leq \lambda < m$, the nodes that consitute the path starting from λ are the nodes $(N_0, \ldots, N_{\lceil \log_2(m) \rceil - 1})$ such that $N_j = L_{\lfloor \lambda / 2^j \rfloor}$ at every level $0 \leq j < \lceil \log_2(m) \rceil$ of the hash tree (i.e., at every iteration of the while-loop in Algorithm 7.1).

Algorithm 7.1 The `treehash` compression algorithm.

Input: $(L_0, \ldots, L_{m-1}) \in (\mathbb{B}^n)^m$ – A number $m > 1$ of leaves.
Input: $H : \mathbb{B}^{2n} \to \mathbb{B}^n$ – A compression pseudorandom function.
Output: The root of the binary tree $r \in \mathbb{B}^n$.

while $m > 1$ **do**
  for $i = 0$ **to** $\lfloor m/2 \rfloor - 1$ **do**
    $L_i \leftarrow H(L_{2i}, L_{2i+1})$.
  if $m \bmod 2 = 1$ **then** ▷ *If there is a node without a neighbor.*
    $L_{\lfloor m/2 \rfloor} \leftarrow L_{m-1}$.
  $m \leftarrow \lceil m/2 \rceil$.
return L_0.

Given a path in a hash tree starting from $0 \leq \lambda < m$, an *authentication path* refers to the list of neighboring nodes in the given path that are involved in the computation of the root. Authentication paths are usually derived while treehashing (i.e., during Algorithm 7.1). Concretely, given a leaf index $0 \leq \lambda < m$, the nodes that constitute the authentication path from λ are the nodes $(A_0, \ldots, A_{\lceil \log_2(m) \rceil - 1})$ such that $A_j = L_{\lfloor \lambda / 2^j \rfloor \oplus 1}$ at every level $0 \leq j < \lceil \log_2(m) \rceil$ of the hash tree (i.e., at every iteration of the while-loop in Algorithm 7.1).

Along with a given leaf at known index, an authentication path enables the recomputation of the tree root with minimal information. Such a recomputation is achieved by recursively compressing the leaf with the next node in the authentication path. The binary representation of the leaf index dictates whether the nodes are on the left or right in the path to the root. Algorithm 7.2 describes the steps to recompute a root from a leaf and its authentication path.

Algorithm 7.2 The root recomputation with authentication path algorithm.

Input: $L \in \mathbb{B}^n$ – A leaf in a binary tree of index $0 \leq \lambda < 2^h$.
Input: $(A_0, \ldots, A_{h-1}) \in (\mathbb{B}^n)^h$ – An authentication path.
Input: $H : \mathbb{B}^{2n} \to \mathbb{B}^n$ – A compression pseudorandom function.
Output: The root of the binary tree $r \in \mathbb{B}^n$.

1: **for** $i = 0$ **to** $h - 1$ **do**
2: **if** $\lfloor \lambda / 2^i \rfloor \bmod 2 = 0$ **then** ▷ *If the current node is on the left of its neighbor.*
3: $L \leftarrow H(L, A_i)$.
4: **else**
5: $L \leftarrow H(A_i, L)$.
6: **return** L.

An example of a path and authentication path is shown on Figure 7.3. Notice that r can be recomputed using only v and (A_0, A_1, A_2).

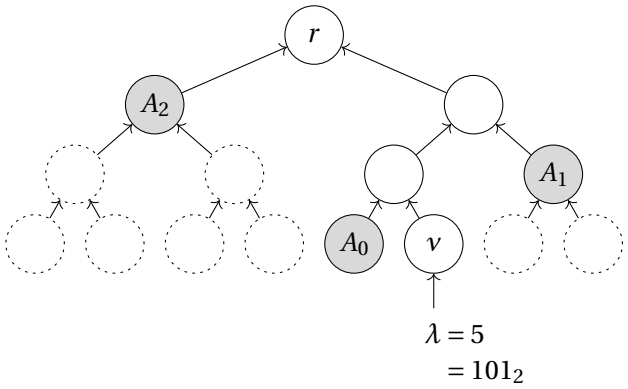


Figure 7.3: Binary hash tree where the *path* from v (the leaf indexed $\lambda = 5$) to r (the root) consists of the nodes in solid lines, whereas its corresponding *authentication path* consists of the nodes highlighted in gray.

7.1.5 Digital signature scheme

Definition 7.1.3 (Digital signature scheme)**.** A *digital signature scheme* is a triple of probabilistic algorithms $\texttt{KeyGen}(n)$, $\texttt{Sign}(M, \text{SK})$, $\texttt{Verify}(M, \sigma, \text{PK})$ that achieves *integrity*, *authenticity*, and *non-repudiation*.

The algorithms of a digital signature scheme must fulfill the following properties:

- `KeyGen`(n) generates a random *key pair* (SK, PK) of security parameter n where:
  - SK is a *signing key*; a private key used to sign messages.
  - PK is the *public key* corresponding to SK used to verify signatures.
- `Sign`(M, SK) signs a message M with the signing key SK to produce a digital signature σ that corresponds to M.
- `Verify`(M, σ, PK) verifies that the digital signature σ corresponds to the message M under the public key PK.

The security parameter n ensures that there is no probabilistic algorithm that compromises any of the security guarantees of the digital signature scheme (integrity, authenticity, and non-repudiation) in time polynomial nor subexponential in n.

7.2 One-time signatures

Hash-based One-Time Signatures (OTS) schemes describe digital signature schemes in which the security guarantees only hold when the signing key is used only *one* time; i.e., to sign a single message. Such schemes rely on pseudorandom functions to commit to secret values which are revealed as part of the signature depending on the bit values of the message to sign.

7.2.1 W-OTS+

W-OTS$^+$ (Winternitz One-Time Signatures) [Hül13] is an OTS scheme which commits to secret elements by using a chaining pseudorandom function. A W-OTS$^+$ signature consists of chain elements at positions that depend on the bit values of the message to sign. Since, in a chain, the elements at lower positions lead to the elements at higher positions, the scheme involves signing an inverted checksum of the message to prevent the forgery of a valid signature using the elements at higher positions derived from the signature elements. Figure 7.4 illustrates a W-OTS$^+$ signature in a W-OTS$^+$ structure. In SPHINCS and SPHINCS$^+$, W-OTS$^+$ is used to authenticate public keys of various hash-based schemes.

Parameters. A W-OTS$^+$ instance is parameterized with:

- n : number of bytes of security.
- η : number of bits in a message digest.
- ω : a (short) window of bits signed at a time.
- $C^i : \mathbb{N} \times \mathbb{B}^n \to \mathbb{B}^n$: a chaining pseudorandom function.
- $G_l : \mathbb{B}^n \to \mathbb{B}^{ln}$: a pseudorandom number generator.
- $\mathscr{H} : \{0,1\}^* \to \{0,1\}^\eta$: a cryptographic hash function.

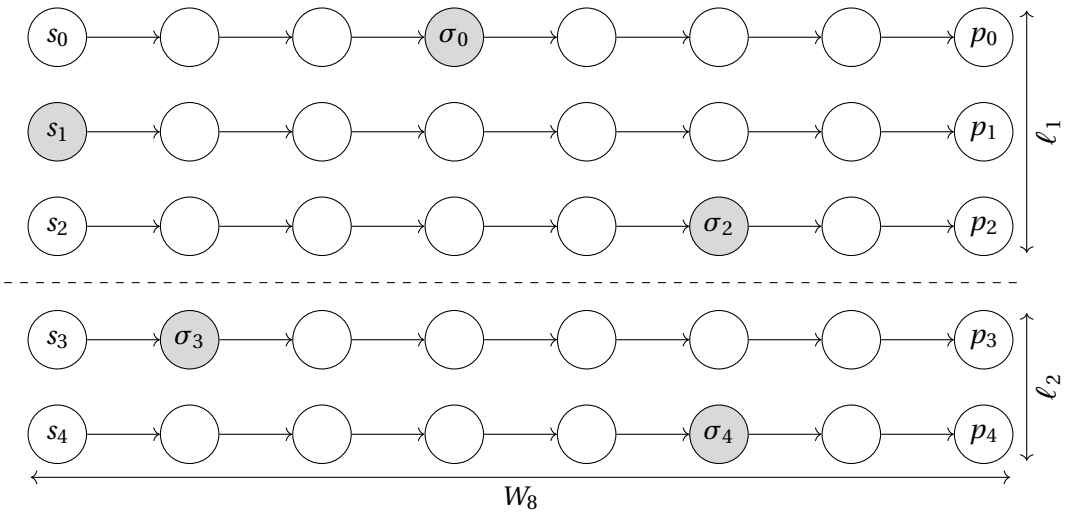


Figure 7.4: Illustration of a W-OTS$^+$ structure with $\eta = 9$, $\omega = 3$. The highlighted nodes correspond to the signature for $\mathscr{H}(M) = 011\,000\,101$ ($c = 001\,101$).

Furthermore, given the above parameters, let the following quantities be defined:

- $W = 2^\omega$: the length of the pseudorandom chains.
- $\ell_1 = \lceil \eta/\omega \rceil$: the number of chunks in a message digest (see below).
- $\ell_2 = \left\lfloor \log_2((W-1)\ell_1)/\omega \right\rfloor + 1$: the number of chunks in a checksum (see below).
- $\ell = \ell_1 + \ell_2$: the total number of elements in a W-OTS$^+$ signature.

Key generation. Given a secret seed $\mathrm{SK} \in \mathbb{B}^n$, a W-OTS$^+$ key pair is generated as follows:

$$\begin{aligned} \mathrm{SK}^W \leftarrow G_\ell(\mathrm{SK}) \quad &= \quad (s_0, \ldots, s_{\ell-1}), \\ \mathrm{PK}^W \leftarrow (p_0, \ldots, p_{\ell-1}), \quad &\quad \text{where } p_i = C^{W-1}(s_i) \quad (0 \le i < \ell). \end{aligned}$$

Signing procedure. Given a W-OTS$^+$ signing key SK^W, the scheme signs a message $M \in \{0,1\}^*$ with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{\ell_1-1})$ of ω bits.
2. $c \leftarrow \sum_{i=0}^{\ell_1-1} (W-1-b_i)$.
3. Split c into chunks $(b_{\ell_1}, \ldots, b_{\ell_1+\ell_2-1})$ of ω bits.
4. $\sigma_i \leftarrow C^{b_i}(s_i)$ for $i = 0$ to $\ell - 1$.
5. Return $\sigma^W = (\sigma_0, \ldots, \sigma_{\ell-1})$.

Public key extraction. Given a W-OTS$^+$ signature σ^W corresponding to a known message $M \in \{0,1\}^*$, the W-OTS$^+$ public key PK^W can be extracted with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{\ell_1-1})$ of ω bits.
2. $c \leftarrow \sum_{i=0}^{\ell_1-1} (W-1-b_i)$.

3. Split c into chunks $(b_{\ell_1}, \ldots, b_{\ell_1+\ell_2-1})$ of ω bits.
4. $p_i \leftarrow C^{W-b_i-1}(\sigma_i)$ for $i = 0$ to $\ell - 1$.
5. Return $\text{PK}' = (p_0, \ldots, p_{\ell-1})$.

Verification procedure. Given a W-OTS$^+$ public key PK^W, a W-OTS$^+$ signature σ^W, and a message $M \in \{0,1\}^*$, the signature can be verified to correspond to the message with the following steps:

1. Extract the public key PK' using M and σ^W.
2. Return `True` if $\text{PK}' = \text{PK}^W$, `False` otherwise.

7.3 Few-time signatures schemes

Hash-based Few-Time Signatures (FTS) schemes describe digital signature schemes in which the security guarantees slowly decline with the number of times the signing key is used; i.e., to sign only *a few* message. This is achieved in a similar fashion as with OTS schemes, but usually by committing to more secret values than with an OTS, so that revealing many of them does not immediately break the system.

7.3.1 HORST

HORST (Hash to Obtain Random Subset with Trees) [Ber+15] is an FTS scheme that involves 2^τ secret values (one per τ-bit chunk) that are compressed with a hash tree. A HORST signature then consists of the secret values at indexes that correspond to the value of the bit chunks in the messages to sign, along with their authentication paths. In SPHINCS, HORST is used to sign the digest of the message. Figure 7.5 illustrates a HORST signature in a HORST structure.

Because the probability is high that the authentication paths contain several duplicate nodes, all nodes at a fixed level $x > 0$ are given; the choice for x depends on implementation efficiency considerations.

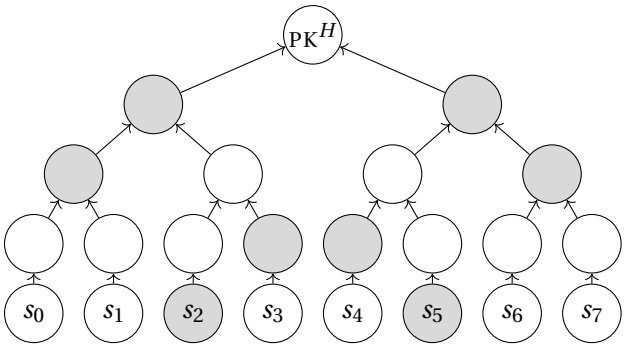


Figure 7.5: Illustration of a HORST structure with $\eta = 6$, $k = 2$, $x = 2$. The highlighted nodes correspond to the signature for $\mathscr{H}(M) = 010\,101$.

Parameters. A HORST instance is parameterized with:

- n : number of bytes of security.
- η : number of bits in a message digest.
- k : the number of blocks in a message (see below).
- x : cut-off height for full authentication path derivation.
- $F : \mathbb{B}^n \to \mathbb{B}^n$: a size-preserving pseudorandom function.
- $G_l : \mathbb{B}^n \to \mathbb{B}^{ln}$: a pseudorandom number generator.
- $H : \mathbb{B}^{2n} \to \mathbb{B}^n$: a compression pseudorandom function.
- $\mathscr{H} : \{0,1\}^* \to \{0,1\}^\eta$: a cryptographic hash function.

Furthermore, given the above parameters, let the following quantities be defined:

- $\tau = \lceil \eta / k \rceil$: number of bits in one block.
- $t = 2^\tau$: the total number of secret leaves.

Key generation. Given a secret seed $\text{SK} \in \mathbb{B}^n$, a HORST key pair is generated as follows:

$$\begin{aligned} \text{SK}^H &\leftarrow G_t(\text{SK}) = (s_0, \ldots, s_{t-1}), \\ \text{PK}^H &\leftarrow \texttt{treehash}(F(s_0), \ldots, F(s_{t-1})). \end{aligned}$$

Signing procedure. Given a HORST signing key SK^H, the scheme signs a message $M \in \{0,1\}^*$ with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{k-1})$ of τ bits.
2. For all chunk indices i ranging from 0 to $k-1$:
   (a) Let $\sigma_i = s_{b_i}$.
   (b) Let $\text{auth}_i = (A^i_0, \ldots, A^i_{x-1})$ be the authentication path from $F(s_{b_i})$ until level $x-1$.
3. Let N_i be all the tree nodes at level x (i.e., for $0 \le i < 2^{\tau - x}$).
4. Return $\sigma^H = ((\sigma_0, \text{auth}_0), \ldots, (\sigma_{k-1}, \text{auth}_{k-1}), (N_0, \ldots, N_{2^{\tau-x}-1}))$.

Public key extraction. Given a HORST signature $\sigma^H = ((\sigma_0, \text{auth}_0), \ldots, (\sigma_{k-1}, \text{auth}_{k-1}), (N_0, \ldots, N_{2^{\tau-x}-1}))$, the HORST public key PK^H can be extracted with the following steps:

1. Let $\text{PK}' = \texttt{treehash}(N_0, \ldots, N_{2^{\tau-x}})$.
2. Return PK'.

Verification procedure. Given a HORST public key PK^H, a HORST signature σ^H, and a message $M \in \{0,1\}^*$, the signature can be verified to correspond to the message with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{k-1})$ of τ bits.

2. For all chunk indices $i = 0$ to $k-1$:
   (a) Recompute $N'_{s_i/2^x}$ from $F(s_i)$ and auth_i.
   (b) Ensure that $N'_{s_i/2^x} = N_{s_i/2^x}$ or return `False` otherwise.
3. Extract the public key PK' using σ^H.
4. Return `True` if $\text{PK}' = \text{PK}^H$, `False` otherwise.

7.3.2 FORS

FORS (Forest of Random Subsets) is a few-time signature scheme introduced in SPHINCS$^+$ as an enhancement over HORST. Rather than all bit chunks in a message sharing a same tree, FORS features unique trees per chunk index; making thus a forest. In SPHINCS$^+$, FORS is used to sign the digest of the message. Figure 7.6 illustrates a FORS signature in a FORS structure.

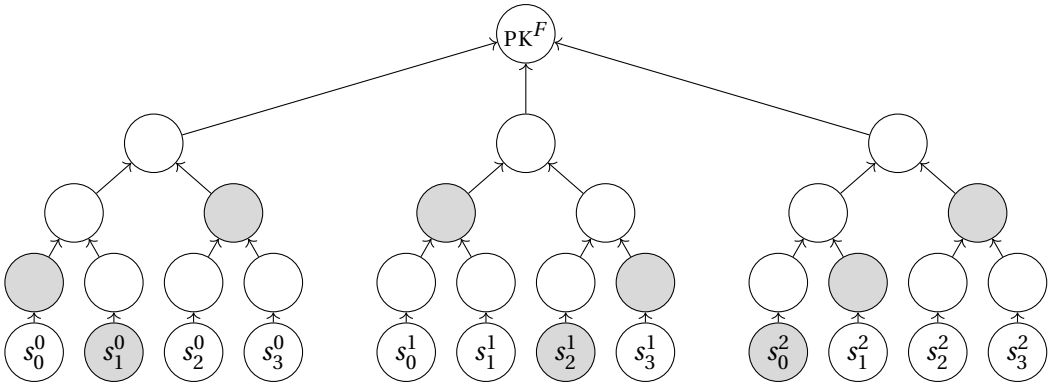


Figure 7.6: Illustration of a FORS structure with $\eta = 6$, $k = 2$, $t = 4$. The highlighted nodes correspond to the signature for $\mathscr{H}(M) = 01\ 10\ 00$.

Parameters. A FORS instance requires the following parameters:

- n : number of bytes of security.
- η : number of bits in a message digest.
- k : number of trees.
- $t = 2^a$: number of leaves in a tree (of height a).
- $F : \mathbb{B}^n \to \mathbb{B}^n$: a size-preserving pseudorandom function.
- $G_l : \mathbb{B}^n \to \mathbb{B}^{ln}$: a pseudorandom number generator.
- $H : \mathbb{B}^{2n} \to \mathbb{B}^n$: a compression pseudorandom function.
- $T_k : \mathbb{B}^{kn} \to \mathbb{B}^n$: a k-compression pseudorandom function.
- $\mathscr{H} : \{0,1\}^* \to \{0,1\}^\eta$: a cryptographic hash function.

Note that $\eta = ka$.

Key generation. A FORS key pair consists of the k sub-key pairs $(\text{SK}_i^F, \text{PK}_i^F)$ for each FORS tree ($0 \le i < k$). Given a secret seed $\text{SK} \in \mathbb{B}^n$, the FORS key pair is therefore generated with the

following steps:

1. Let $\text{SK}^F \leftarrow G_{kt}(\text{SK}) = (s_0, \ldots, s_{kt-1})$.
2. Split SK^F into k subsets of t elements $\text{SK}_i^F = (s_0^i, \ldots, s_{t-1}^i)$ $(0 \le i < k)$.
3. Let $\text{PK}_i^F \leftarrow \texttt{treehash}(F(s_0^i), \ldots, F(s_{t-1}^i))$ for $0 \le i < k$.
4. Let $\text{PK}^F \leftarrow T_k(\text{PK}_0^F, \ldots, \text{PK}_{t-1}^F)$.
5. Return $(\text{SK}^F, \text{PK}^F)$.

Signing procedure. Given a FORS signing key $\text{SK}^F = (\text{SK}_0^F, \ldots, \text{SK}_{k-1}^F)$ where $\text{SK}_i^F = (s_0^i, \ldots, s_{t-1}^i)$, the scheme signs a message $M \in \{0,1\}^*$ with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{k-1})$ of a bits.
2. In each FORS tree $0 \le i < k$, compute auth_i as the authentication path starting from $F(s_{b_i}^i)$, the leaf indexed at b_i.
3. Return $\sigma^F = ((s_{b_0}^0, \text{auth}_0), \ldots, (s_{b_{k-1}}^{k-1}, \text{auth}_{k-1}))$.

Public key extraction. Given $\sigma^F = ((s_{b_0}^0, \text{auth}_0), \ldots, (s_{b_{k-1}}^{k-1}, \text{auth}_{k-1}))$ corresponding to a known message $M \in \{0,1\}^*$, the FORS public key PK^F can be extracted with the following steps:

1. Split $D = \mathscr{H}(M)$ into chunks $(b_0, \ldots, b_{k-1})$ of a bits.
2. In each FORS tree $0 \le i < k$, recompute the public keys PK_i^F from $F(s^i)$ and auth_i.
3. Return $\text{PK}' = T_k(\text{PK}_0^F, \ldots, \text{PK}_{k-1}^F)$.

Verification procedure. Given a FORS public key PK^F, a FORS signature σ^F, and a message $M \in \{0,1\}^*$, the signature can be verified to correspond to the message with the following steps:

1. Extract the public key PK' using M and σ^F.
2. Return `True` if $\text{PK}' = \text{PK}^F$, `False` otherwise.

7.4 Multiple-time signatures

Hash-based Multiple-Time Signatures (MTS) schemes describe digital signature schemes in which a single key pair is able to sign a fixed number of times only; i.e., to sign *multiple* (but not infinitely many) messages. A typical MTS will combine multiple OTS key pairs with a hash tree, so that it suffices to publish the root of the hash tree as a public key, while the signing key consists of all the signing keys for all the OTS key pairs.

7.4.1 XMSS

An XMSS (eXtended Merkle Signature Scheme) is a multiple-time signature scheme which combines multiple W-OTS$^+$s in a hash tree. As a standalone scheme, XMSS signs a message

with a W-OTS$^+$ key pair at the leaves of the tree that was not used before, making the scheme thus stateful. An XMSS signature consists of the W-OTS$^+$ signature of the message, along with the authentication path from the leaf used, as illustrated in Figure 7.7. SPHINCS and SPHINCS$^+$ use a large tree of XMSSs to authenticate the layer of FTS (as well as an unpredictable path in the tree to avoid statefulness).

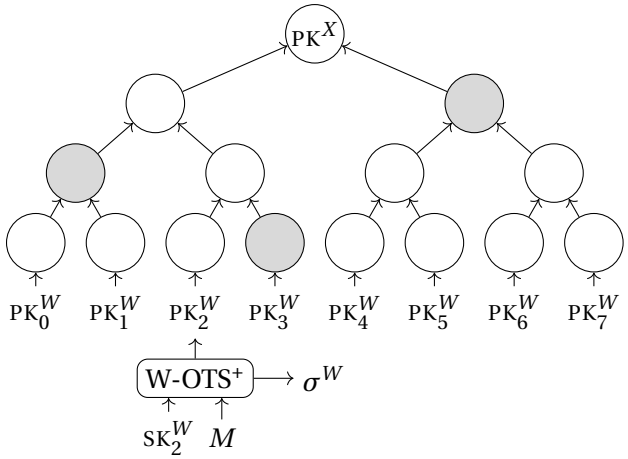


Figure 7.7: Illustration of an XMSS structure with $h' = 3$. The signature using the leaf at $\lambda = 2$ consists of σ^W along with the highlighted nodes (i.e., the authentication path).

Parameters. An XMSS instance requires the following parameters:

- n : number of bytes of security.
- h' : height of the tree.
- $H : \mathbb{B}^{2n} \to \mathbb{B}^n$: a compression pseudorandom function.
- $T_\ell : \mathbb{B}^{\ell n} \to \mathbb{B}^n$: an ℓ-compression pseudorandom function (ℓ as in Section 7.2.1).
- $\eta, \omega, C^i, G_l, \mathscr{H}$: W-OTS$^+$ parameters (see Section 7.2.1).

Key generation. Given a secret seed $\mathrm{SK} \in \mathbb{B}^n$, the key generation starts by generating $2^{h'}$ W-OTS$^+$ key pairs $(\mathrm{SK}_i^W, \mathrm{PK}_i^W)$ (for $0 \le i < 2^{h'}$). Thus, an overall XMSS key pair consists of:

$$\begin{aligned} \mathrm{SK}^X &\leftarrow (\mathrm{SK}_0^W, \ldots, \mathrm{SK}_{2^{h'}-1}^W), \\ \mathrm{PK}^X &\leftarrow \texttt{treehash}(T_\ell(\mathrm{PK}_1^W), \ldots, T_\ell(\mathrm{PK}_{2^{h'}-1}^W)). \end{aligned}$$

Signing procedure. Given an XMSS signing key SK^X, the scheme signs a message $M \in \{0,1\}^*$ at leaf index $0 \le \lambda < 2^{h'}$ with the following steps:

1. Use SK_λ^W to produce σ_λ^W; the W-OTS$^+$ signature of M.
2. Compute the authentication path auth_λ from the leaf index λ.
3. Return $\sigma^X = (\sigma_\lambda^W, \mathrm{auth}_\lambda)$.

Note that each leaf index can be used to sign at most one message (thus stateful).

Public key extraction. Given $\sigma^X = (\sigma^W_\lambda, \text{auth}_\lambda)$ bound to a known message $M \in \{0,1\}^*$ at leaf index $0 \leq \lambda \leq 2^{h'} - 1$, the XMSS public key can be extracted with the following steps:

1. Extract the W-OTS$^+$ public key PK^W_λ from σ^W_λ using M.
2. Recompute the XMSS public key PK^X from $T_\ell(\text{PK}^W_\lambda)$ and auth_λ.
3. Return PK^X.

Verification procedure. Given an XMSS public key PK^X, an XMSS signature σ^X, and a message M, the signature can be verified to correspond to the message with the following steps:

1. Extract the public key PK' using M and σ^X.
2. Return `True` if $\text{PK}' = \text{PK}^X$, `False` otherwise.

7.4.2 Hypertree

In the context of SPHINCS or SPHINCS$^+$, a hypertree (also known as CMSS or XMSS^{MT} (Multi Tree XMSS)) consists of a tree of XMSS key pairs in which the XMSSs higher in the tree sign the lower XMSSs. As a standalone scheme, a hypertree signs a message with a W-OTS$^+$ key pair that was not used before in one of the XMSS at the leaves of the hypertree, making the scheme thus stateful. A hypertree signature therefore consists of all the XMSS signatures in the path from the leaf used to the top of the hypertree, as illustrated in Figure 7.8. In SPHINCS and SPHINCS$^+$, a hypertree is used to authenticate the layer of FTS (as well as an unpredictable path in the tree to avoid statefulness).

Parameters. A hypertree instance requires the following parameters:

- n : number of bytes of security.
- h : total height of the hypertree.
- d : number of layers in the hypertree.
- H, T_ℓ : XMSS parameters (see Section 7.4.1).
- $\eta, \omega, C^i, G_l, \mathscr{H}$: W-OTS$^+$ parameters (see Section 7.2.1).

Note that the hypertree instantiates XMSS with $h' = h/d$.

Addressing scheme. The hypertree requires an *addressing scheme* to distinguish the XMSSs in the hypertree so that their signing key can be generated from a single secret seed. Concretely, all the XMSSs in the hypertree have a unique address that consists of the index of the layer $0 \leq l < d$, and the index of the tree $0 \leq \tau_l < 2^{(d-1-l)h/d}$ depending on the position of the XMSS

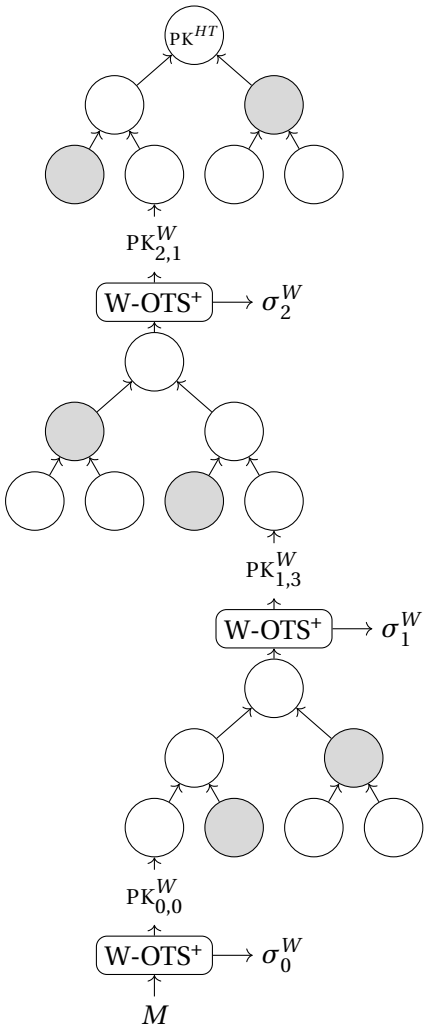


Figure 7.8: Illustration of a hypertree structure with $h = 6$ and $d = 3$. The signature using the hyperleaf at $\lambda = 29$ consists of $(\sigma_0^W, \sigma_1^W, \sigma_2^W)$ along with the highlighted nodes (i.e., the authentication paths) in each tree.

in the hypertree.

Due to their structure, the addresses of the XMSSs higher in the hypertree can be entirely derived from the address of the subtree lower in the hypertree. Let τ_l be the tree index of an XMSS at layer $0 \leq l < d-1$. Such an XMSS is signed with the XMSS at tree index τ_{l+1} and leaf index λ_{l+1} derived as follows:

$$\begin{cases} \tau_{l+1} & = & \text{the } h - h'(l+1) \text{ most significant bits of } \tau_l, \\ \lambda_{l+1} & = & \text{the } h' \text{ least significant bits of } \tau_l. \end{cases}$$

All addresses involved in the above XMSSs are reconstructed from these indices.

Key generation. Given a secret seed $\text{SK} \in \mathbb{B}^n$, the public key of the hypertree consists of the public key of the top-most XMSS PK^X_{d-1} at layer $d-1$ and tree index $\tau_{d-1} = 0$:

$$\begin{aligned} \text{SK}^{HT} &\leftarrow \text{SK}, \\ \text{PK}^{HT} &\leftarrow \text{PK}^X_{d-1}. \end{aligned}$$

Signing procedure. Given a hypertree signing key SK^{HT}, the scheme signs a message $M \in \{0,1\}^*$ at hyperleaf index $0 \leq \lambda < 2^h$ with the following procedure:

1. For all layer indices i ranging from 0 to d:
   (a) Derive τ_i, λ_i from τ_{i-1} (starting with $\tau_{-1} = \lambda$).
   (b) Generate the XMSS key pair $(\text{SK}^X_i, \text{PK}^X_i)$ with SK^{HT} at the address corresponding to τ_i.
   (c) Sign M with SK^X_i using λ_i as leaf index to produce σ_i and update M with PK^X_i.
2. Return $\sigma^{HT} = (\sigma_0, \ldots, \sigma_{d-1})$.

Note that each hyperleaf index should be used to sign at most one message (thus stateful).

Public key extraction. Given a hypertree signature $\sigma^{HT} = (\sigma^X_0, \ldots, \sigma^X_{d-1})$ which corresponds to a known message $M \in \{0,1\}^*$ at hyperleaf index $0 \leq \lambda < 2^h$, the hypertree public key can be extracted with the following steps:

1. For all layer indices i ranging from 0 to d:
   (a) Derive τ_i, λ_i from τ_{i-1} (starting with $\tau_{-1} = \lambda$).
   (b) Extract the XMSS public key PK^X_i from σ^X_i using M at the address corresponding to τ_i and using the λ_i as leaf index.
   (c) Update M with PK^X_i.
2. Return the last M computed, i.e, PK^X_{d-1}.

Verification procedure. Given a hypertree public key PK^{HT}, a hypertree signature σ^{HT}, and a message $M \in \{0,1\}^*$, the signature can be verified to correspond to the message with the following steps:

1. Extract the public key PK' using M and σ^{HT}.
2. Return `True` if $\text{PK}' = \text{PK}^{HT}$, `False` otherwise.

7.5 SPHINCS-256

SPHINCS-256 is the standard instantiation of the SPHINCS signature scheme. The scheme combines a hypertree with a bottom layer of HORST key pairs.

Hash functions. Table 7.2 lists all the hash functions involved in SPHINCS-256. The scheme relies on the cryptographic hash functions BLAKE-256 and BLAKE-512 [Aum+14], as well as $\text{ChaCha12}_\kappa(x)_i$ (i.e., the stream cipher ChaCha12 [Ber08a] called $i > 0$ times with $\kappa \in \mathbb{B}^n$ as the key, and $x \in \{0,1\}^{64}$ as the nonce). The function π_{ChaCha} refers to the permutation involved in ChaCha12, while $\texttt{Chop}(x,i)$ truncates a bitstring x down to the bit length of $i > 0$. The constant $\texttt{C}$ corresponds to the string `"expand 32-byte state to 64-byte state!"` encoded in ASCII. Note that in SPHINCS-256, the outputs of most hash function calls are XORed with a public bitmask to obtain a unique hash function family per user and, thus, mitigate multi-target attacks.

Function	Input	Implementation	Output
F	$M_1 \in \mathbb{B}^n$	$\texttt{Chop}(\pi_{\text{ChaCha}}(M_1 \Vert \texttt{C}), 256)$	$y \in \mathbb{B}^n$
$\mathcal{F}_\alpha$	$(A,K) \in \mathbb{B}^\alpha \times \mathbb{B}^n$	BLAKE-256$(K \Vert A)$	$y \in \mathbb{B}^n$
$\mathcal{F}$	$(M,K) \in \mathbb{B}^* \times \mathbb{B}^n$	BLAKE-512$(K \Vert M)$	$y \in \mathbb{B}^n$
H	$(M_1,M_2) \in \mathbb{B}^n \times \mathbb{B}^n$	$\texttt{Chop}(\pi_{\text{ChaCha}}(\pi_{\text{ChaCha}}(M_1 \Vert \texttt{C}) \oplus \pi_{\text{ChaCha12}}(M_2 \Vert 0^{256})), 256)$	$y \in \mathbb{B}^n$
G_l	$\text{SEED} \in \mathbb{B}^n$	$\text{ChaCha12}_{\text{SEED}}(0)_l$	$(s_0,\ldots,s_{l-1}) \in \mathbb{B}^{ln}$
$\mathscr{H}$	$(R,M) \in \mathbb{B}^n \times \mathbb{B}^*$	BLAKE-256$(R \Vert M)$	$D \in \mathbb{B}^m$

Table 7.2: Hash functions involved in SPHINCS-256.

SPHINCS-256 implements the ℓ-compression function T_ℓ in W-OTS$^+$ by treehashing[3] the ℓ input values, while the chaining pseudorandom function C^i is simply obtained by consecutively applying F.

The addressing scheme of SPHINCS-256 is simply implemented by concatenating the layer index with the tree index and the leaf index, where these indices are represented in binary:

$$A(l,\tau,\lambda) = (l \Vert \tau \Vert \lambda),$$

with $l \in \{0,1\}^4$, $\tau \in \{0,1\}^{55}$, and $\lambda \in \{0,1\}^5$, resulting thus in an address of $\alpha = 8$ bytes.

Parameters. SPHINCS-256 is parameterized with the following:

- $n = 32$: number of bytes of security.
- $m = 64$: number of bytes of the digest.
- $\alpha = 8$: number of bytes in an address.
- $k = 32$: number of elements in a HORST signature.
- $t = 2^{16}$: number of leaves in a HORST tree.
- $W = 2^\omega = 2^4$: W-OTS$^+$ parameters.
- $h' = h/d = 60/12$: hypertree parameters.

[3] This process is referred to as an L-tree compression in the original submission.

Key generation. The SPHINCS-256 signing key consists of two secret seeds $\mathrm{SK}_1 \sim \mathcal{U}(\mathbb{B}^n)$ and $\mathrm{SK}_2 \sim \mathcal{U}(\mathbb{B}^n)$:

- SK_1 is used to derive the key pairs of all the hash-based instances involved in the scheme.
- SK_2 is used to deterministically choose a starting HORST in an unpredictable way.

The SPHINCS-256 public key PK_1 consists of the public key of the hypertree, as well as a public bitmask PK_2 that is uniformly generated to make all the hash function calls unique per user.

Signing procedure. Given a SPHINCS-256 signing key $(\mathrm{SK}_1, \mathrm{SK}_2)$, the scheme signs a message $M \in \{0,1\}^*$ with the following procedure:

1. Generate $(R_1, R_2) = F(M, \mathrm{SK}_2)$.
2. Compute $D = \mathscr{H}(M, R_1)$.
3. Let τ be the first $(d-1)h'$ bits of R_2, and λ the next h', so that $A = (d||\tau||\lambda)$.
4. Generate the HORST signing key $\mathrm{SK}^H = F_\alpha(A, \mathrm{SK}_1)$ and let PK^H be its public key.
5. Sign D with SK^H to produce $\sigma^H = ((\sigma_0, \mathrm{auth}_0), \ldots, (\sigma_{k-1}, \mathrm{auth}_{k-1}), (N_0, \ldots, N_{2^{\tau-x}-1}))$.
6. Sign PK^H with the hypertree starting at tree index τ and leaf index λ to produce $\sigma^{HT} = (\sigma^X_0, \ldots, \sigma^X_{d-1})$.
7. Return $\Sigma = (\tau, \lambda, R_1, \sigma^H, \sigma^{HT})$.

Verification procedure. Given a SPHINCS-256 public key PK_1 and the public bitmasks, the scheme verifies that a SPHINCS-256 signature $\Sigma = (\tau, \lambda, R_1, \sigma^H, \sigma^{HT})$ corresponds to the message $M \in \{0,1\}^*$ with the following procedure:

1. Compute $D = \mathscr{H}(M, R_1)$.
2. Extract PK^H, the public key of the HORST at $A = (d||\tau||\lambda)$, from D and σ^H.
3. Extract PK^{HT}, the public key of the hypertree at tree index τ and leaf index λ, from PK^H and $\sigma^{HT} = (\sigma^X_0, \ldots, \sigma^X_{d-1})$.
4. Return `True` if $\mathrm{PK}^{HT} = \mathrm{PK}_1$, `False` otherwise.

7.6 SPHINCS+

SPHINCS$^+$ is a stateless signature scheme which combines FORSs with a hypertree. SPHINCS$^+$ signs the digest of a message with a FORS chosen at random for which the key pair is authenticated by the hypertree. Figure 7.9 illustrates the SPHINCS$^+$ structure.

Hash functions. In the original submission of SPHINCS$^+$, different[4] hash functions are used depending on the operation. Table 7.3 summarizes all the hash functions and pseudorandom

[4] Even though the functions are listed separately, all the hash functions involved in SPHINCS$^+$ are instantiable from a single cryptographic hash function such as SHA2-256.

functions involved in SPHINCS$^+$. Given a SPHINCS$^+$ signing key SK_1, SK_2, and public key PK_1, PK_2, the parameters column includes public and secret seeds (resp., PK_1, PK_2, and SK_2) that make hash function calls unique per key pair, and contextual information (i.e., $\mathtt{ADRS}$, R, opt) that makes hash function calls unique per use. These are sometimes considered implicitly given in the notation.

Function	Parameters	Input	Output
$\mathbf{T}_l$	$(\mathrm{PK}_2, \mathtt{ADRS}) \in \mathbb{B}^n \times \mathbb{B}^\alpha$	$(x_1, \ldots, x_l) \in \mathbb{B}^{ln}$	$y \in \mathbb{B}^n$
$\mathbf{F}$	$(\mathrm{PK}_2, \mathtt{ADRS}) \in \mathbb{B}^n \times \mathbb{B}^\alpha$	$x \in \mathbb{B}^n$	$y \in \mathbb{B}^n$
$\mathbf{H}$	$(\mathrm{PK}_2, \mathtt{ADRS}) \in \mathbb{B}^n \times \mathbb{B}^\alpha$	$(x_\mathrm{L}, x_\mathrm{R}) \in \mathbb{B}^{2n}$	$y \in \mathbb{B}^n$
$\mathbf{PRF}$	$(\mathrm{PK}_2, \mathtt{ADRS}) \in \mathbb{B}^n \times \mathbb{B}^\alpha$	$\mathrm{SK}_1 \in \mathbb{B}^n$	$s \in \mathbb{B}^n$
$\mathbf{PRF_{msg}}$	$(\mathrm{SK}_2, \mathrm{opt}) \in \mathbb{B}^{2n}$	$M \in \mathbb{B}^*$	$R \in \mathbb{B}^n$
$\mathbf{H_{msg}}$	$(\mathrm{PK}_1, R) \in \mathbb{B}^{2n}$	$M \in \mathbb{B}^*$	$(\mathtt{md}, \mathtt{ADRS}) \in \mathbb{B}^m$

Table 7.3: Hash functions involved in SPHINCS$^+$.

The addressing scheme of SPHINCS$^+$ is extended to uniquely address *all* hash function calls (rather than only XMSSs). Concretely, an address in SPHINCS$^+$ represents each hash structure by a unique bytestring (of size $\alpha = 32$) which is composed of different fields, including notably the *tree index* that is used to uniquely address every tree involved in the scheme (FORS or XMSS), the *leaf index* to uniquely address the leaves, and also the *hash index* which, in the context of W-OTS$^+$, refers to the position of the element in the chain.

In SPHINCS$^+$, the sequences of pseudorandomly generated numbers are obtained with **PRF**:

$$\mathbf{G}_l(\mathrm{PK}_2, \mathtt{ADRS})(x) := (\mathbf{PRF}(\mathrm{PK}_2, \mathtt{ADRS}^{(0)})(x), \ldots, \mathbf{PRF}(\mathrm{PK}_2, \mathtt{ADRS}^{(l-1)})(x)),$$

where $\mathtt{ADRS}^{(i)}$ is the initial address of the hash structure where the field corresponding to the index of the secret value has been updated to i.

Finally, the chaining pseudorandom function in SPHINCS$^+$ is defined as follows:

$$\mathbf{C}^i(\mathrm{PK}_2, \mathtt{ADRS})(x) := (F(\mathrm{PK}_2, \mathtt{ADRS}_{i-1}) \circ \cdots \circ F(\mathrm{PK}_2, \mathtt{ADRS}_0))(x),$$

where $\mathtt{ADRS}_i$ is the initial address of the hash structure where the field corresponding to the hash index has been updated to i.

Parameters. SPHINCS$^+$ is parameterized with the following:

- n : number of bytes of security.
- k, $t = 2^a$: FORS parameters.
- $W = 2^\omega$: W-OTS$^+$ parameters.
- $h' = h/d$: hypertree parameters.

The number of bytes in the digest is $m = \lfloor (ka+7)/8 \rfloor + \lfloor (h-h'+7)/8 \rfloor + \lfloor (h'+7)/8 \rfloor$.

Instance	n	k	t	W	h	d	h'	m
SPHINCS$^+$-128s	16	10	2^{15}	2^4	64	8	8	27
SPHINCS$^+$-128f	16	30	2^9	2^4	60	20	3	43
SPHINCS$^+$-192s	24	14	2^{16}	2^4	64	8	8	36
SPHINCS$^+$-192f	24	33	2^8	2^4	66	22	3	42
SPHINCS$^+$-256s	32	22	2^{14}	2^4	64	8	8	47
SPHINCS$^+$-256f	32	30	2^{10}	2^4	68	17	7	47

Table 7.4: Standard SPHINCS$^+$ parameters sets, as submitted in the third round of the NIST post-quantum standardization process [Hül+20].

Key generation. The SPHINCS$^+$ signing key consists of two secret seeds $\mathrm{SK}_1 \sim \mathcal{U}(\mathbb{B}^n)$ and $\mathrm{SK}_2 \sim \mathcal{U}(\mathbb{B}^n)$:

- SK_1 is used to derive the key pairs of all the hash-based instances involved in the scheme.
- SK_2 is used to deterministically choose a starting FORS in an unpredictable way.

The SPHINCS$^+$ public key PK_1 consists of the public key of the hypertree as well as a public seed PK_2 that makes hash function calls unique per user.

Signing procedure. Given a SPHINCS$^+$ signing key $(\mathrm{SK}_1, \mathrm{SK}_2)$, the scheme signs a message $M \in \{0,1\}^*$ with the following procedure:

1. Generate $R = \mathbf{PRF_{msg}}(\mathrm{SK}_2, \mathrm{opt})(M)$ where $\mathrm{opt} \in \mathbb{B}^n$ is uniformly drawn at random to enable *randomized signing*.
2. Compute $(\mathtt{md}, \mathtt{ADRS}) = \mathbf{H_{msg}}(\mathrm{PK}_1, R)(M)$.
3. Sign $\mathtt{md}$ with the FORS at $\mathtt{ADRS}$ using SK_1 to produce σ^F, and let PK^F be the FORS public key.
4. Sign PK^F with the hypertree to produce $\sigma^{HT} = (\sigma^X_0, \ldots, \sigma^X_{d-1})$.
5. Return $\Sigma = (R, \sigma^F, \sigma^{HT})$.

In the above, R is not simply replaced by a uniform number to mitigate bad randomness [Hül+20].

Verification procedure. Given a SPHINCS$^+$ public key PK_1, the scheme verifies that a SPHINCS$^+$ signature $\Sigma = (R, \sigma^F, \sigma^{HT})$ corresponds to the message $M \in \{0,1\}^*$ with the following procedure:

1. Compute $(\mathtt{md}, \mathtt{ADRS}) = \mathbf{H_{msg}}(\mathrm{PK}_1, R)(M)$.
2. Extract PK^F, the public key of the FORS at $\mathtt{ADRS}$, from $\mathtt{md}$ and σ^F.
3. Extract PK^{HT}, the public key of the hypertree at the leaf index given by $\mathtt{ADRS}$, from PK^F and $\sigma^{HT} = (\sigma^X_0, \ldots, \sigma^X_{d-1})$.
4. Return true if $\mathrm{PK}^{HT} = \mathrm{PK}_1$, false otherwise.

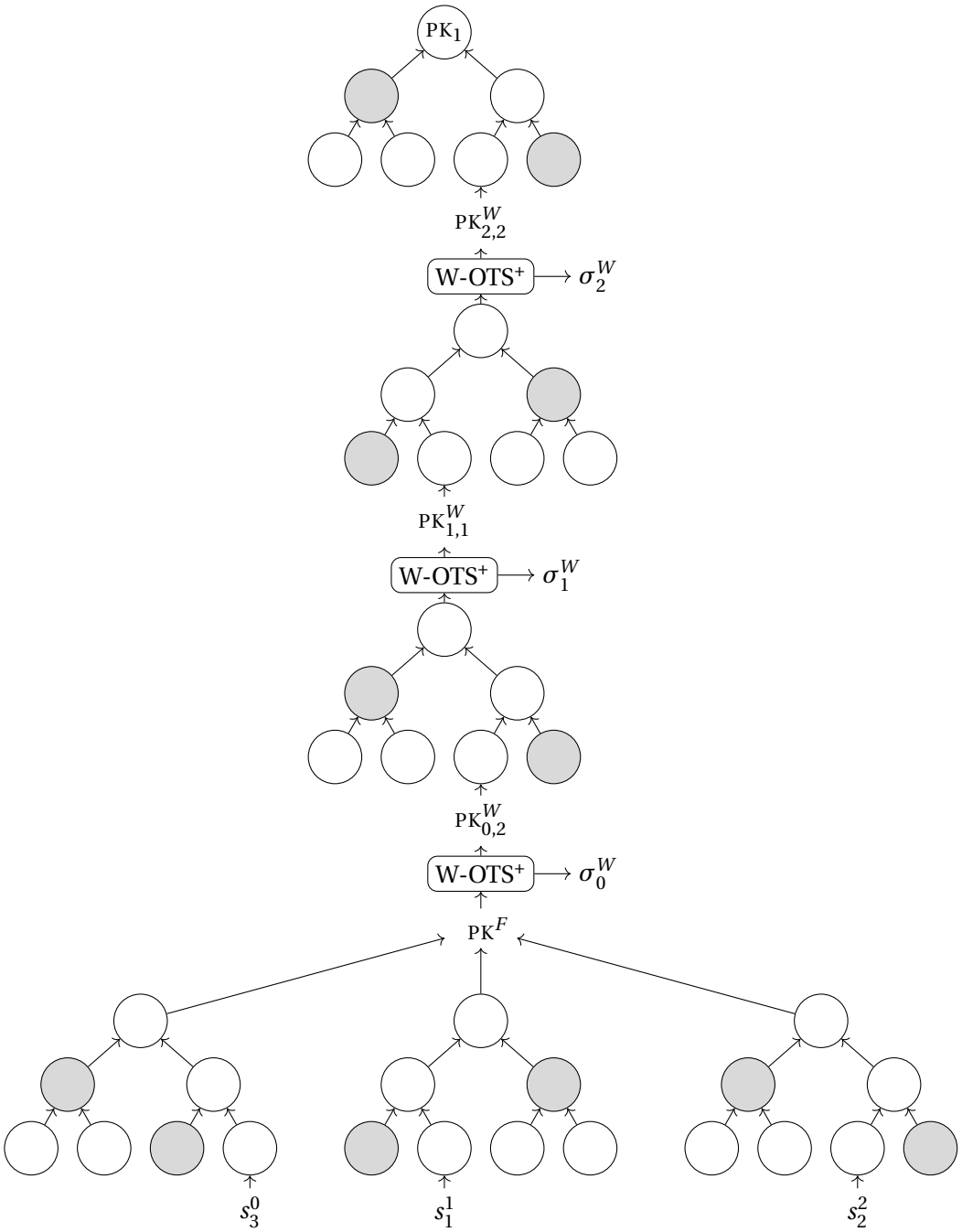


Figure 7.9: Illustration of a SPHINCS$^+$ structure.

8 Differential power analysis of SPHINCS-256

The content of this chapter is based on the work:

[Kan+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. "Differential Power Analysis of XMSS and SPHINCS". in: *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Apr. 2018, pp. 168–188. DOI: 10.1007/978-3-319-89641-0_10

Context. During the second half of the 2010s, the IETF (Internet Engineering Task Force) devoted significant efforts towards the standardization of the stateful XMSS, resulting in the publication of RFC 8391 (Request For Comments) in [Hue+18]. In the RFC, hash-based digital signatures are claimed to be "naturally resistant to most kinds of side-channel attacks" [Hue+18, §1] as an argument in favor of their adoption. While the claim especially refers to timing attacks, no systematic study of the side-channel resistance of hash-based cryptography was conducted at the time. Consequently, a deeper look into the resistance of side-channel weaknesses of hash-based cryptography was desirable.

In 2017, Kannwischer addressed the conjectured resistance of XMSS against side-channel attacks in [Kan17] and, in particular, describes a theoretical DPA of an XMSS implementation based on a SHA2-256 pseudorandom number generator. The attack would enable the recovery of the signing key, resulting thus in a total security break, but does not apply to the parameters described in the RFC. Independently, in [Gen17], the present author uncovered a similar vulnerability in one of the pseudorandom function of SPHINCS-256 (based on BLAKE-256) but did not manage to provide any practical attack.

The current chapter is built upon the two above studies and extends the power analysis of [Gen17] with the methodology of [Kan17] to mount a full-fledged side-channel attack against SPHINCS-256 that is applicable within the standard parameters.

Results. In this chapter, we analyze the side-channel resistance of SPHINCS-256 with a focus on DPA resistance. We present a novel DPA vulnerability of the BLAKE-256-based pseudorandom function used within SPHINCS-256 (see Section 7.5). The attack is practical for the actual parameters of SPHINCS-256.

Outline. We describe in Section 8.1 a DPA on a BLAKE-256-based pseudorandom number generator which applies to SPHINCS-256, analyze its impact, and discuss implications for implementers. We present a countermeasure in Section 8.3, then conclude in Section 8.4.

8.1 Attack description

As detailed in Section 7.5, SPHINCS-256 relies on XMSS^{MT}, HORST, and a stateless way of addressing hash-based instances within the scheme. Since the HORST hash tree construction does not leak anything about its secret key, we can assume this component to be side-channel resistant. Moreover, XMSS^{MT} can also be assumed secure by the analysis by [Kan+18]. This leaves us only with the stateless way of computing the pseudorandom number generator seeds, which we now analyze.

8.1.1 SPHINCS-256 pseudorandom function analysis

In SPHINCS-256, the W-OTS$^+$ and HORST secret seeds are generated with BLAKE-256($\mathrm{SK}_1 \,||\, A$) where $\mathrm{SK}_1 \in \{0,1\}^{256}$ is the SPHINCS-256 secret key, $A \in \{0,1\}^{64}$ the address of the key pair, and BLAKE-256 the hash function [Aum+14]. Recovering SK_1 would therefore result in a universal forgery as an adversary with access to SK_1 could follow the signing procedure with an arbitrary message and uniform (R_1, R_2), and still produce a valid signature. We now present a 6-DPA attack (as described in Section 2.1.3.1) on the BLAKE hash function in the context of SPHINCS-256 that recovers one 32-bit chunk of the secret key SK_1.

DPA. The BLAKE-256 compression procedure takes 12 similar rounds during which the input is mixed. The goal is to subsequently recover intermediate values at certain points in the procedure, to eventually recover one secret chunk. As these values are mixed with variable values early in the procedure, the DPA focuses on the first two rounds. Within SPHINCS-256, the first round is summarized in Algorithm 8.1. Here, the values v_i for $0 \leq i < 15$ are initialized with known constant values. A general mixing subroutine `Mix` involved in these steps is shown in Algorithm 8.2. Here, $M_i \in \{0,1\}^{32}$ for $0 \leq i < 15$ is a chunk of the input padded with a constant and known padding. The function $\sigma_z(i)$ is a permutation that depends on the round $0 \leq z < 12$. Again, the values of C_i for $0 \leq i < 15$ are given constants.

Algorithm 8.1 Round $z = 0$ of BLAKE-256 compression algorithm [Aum+14].

Input: $(s_0, \ldots, s_7)$ — secret key SK_1 split into 8 chunks of 32 bits each
Input: (a_0, a_1) — address A split into two chunks of 32 bits each

1: $\texttt{Mix}(v_0, v_4, v_8, v_{12}; s_0, s_1)$
2: $\texttt{Mix}(v_1, v_5, v_9, v_{13}; s_2, s_3)$
3: $\texttt{Mix}(v_2, v_6, v_{10}, v_{14}; s_4, s_5)$
4: $\texttt{Mix}(v_3, v_7, v_{11}, v_{15}; s_6, s_7)$
5: $\texttt{Mix}(v_0, v_5, v_{10}, v_{15}; a_0, a_1)$
6: $\texttt{Mix}(v_1, v_6, v_{11}, v_{12}; \texttt{0x80000000}, \texttt{0x00000000})$
7: $\texttt{Mix}(v_2, v_7, v_8, v_{13}; \texttt{0x00000000}, \texttt{0x00000001})$
8: $\texttt{Mix}(v_3, v_4, v_9, v_{14}; \texttt{0x00000000}, \texttt{0x00000140})$

Algorithm 8.2 `Mix` procedure involved in Algorithm 8.1.

Input: (v_a, v_b, v_c, v_d) — intermediate values of 32 bits each
Input: $(M_{\sigma_z(e)}, M_{\sigma_z(e+1)})$ — hash function input chunks of 32 bits each

1: $v_a \leftarrow (v_a + v_b) + (M_{\sigma_z(e)} \oplus C_{\sigma_z(e+1)})$
2: $v_d \leftarrow (v_d \oplus v_a) \lll 16$
3: $v_c \leftarrow v_c + v_d$
4: $v_b \leftarrow (v_b \oplus v_c) \lll 12$
5: $v_a \leftarrow (v_a + v_b) + (M_{\sigma_z(e+1)} \oplus C_{\sigma_z(e)})$
6: $v_d \leftarrow (v_d \oplus v_a) \lll 8$
7: $v_c \leftarrow v_c + v_d$
8: $v_b \leftarrow (v_b \oplus v_c) \lll 7$

In Algorithm 8.1, line 5 involves v_0, v_5, v_{10}, and v_{15}, which all respectively depend on two constant chunks of SK_1, and the address. When the `Mix` procedure is unrolled, the operation $v_0 \leftarrow (v_0 + v_5) + (a_0 \oplus C_9)$ at line 1 involves $(v_0 + v_5)$, and $(a_0 \oplus C_9)$: the first half of the address A masked with a constant. By targeting this addition, we can recover $(v_0 + v_5)$ with a first DPA. Once recovered, the following values for v_5, v_{10}, and v_{15} can be consecutively recovered with additional DPA. Since the rest of the `Mix` procedure does not involve any other unknown value, and since these values are not mixed again during round 0, they are, therefore, all known at the beginning of round 1.

In round 1 of the BLAKE-256 compression algorithm, $\texttt{Mix}(v_1, v_5, v_9, v_{13}; s_4, s_5)$ is called. Line 1 in Algorithm 8.2 for this call involves v_5 which has been recovered from before, and v_1 which can be recovered with a fifth DPA. Finally, a sixth DPA on $(v_1 + v_5) + (s_4 \oplus C_5)$ can recover s_4, which consists of one chunk of 32 bits of the secret key SK_1.

8.2 Experimental verification

In this section, we mount the attack described in Section 8.1 on a custom microcontroller.

8.2.1 Setup

The poewr analysis was performed on the Arduino Due framework detailed in Section 2.1.1 through electromagnetic emanations. The attack considers the BLAKE-256 reference implementation [Aum+14] with an additional assumption: the addition of $(v_a + v_b)$ at lines 1 and 4 in Algorithm 8.2 is performed before the rest. This makes the recovery of v_a or v_b alone harder, but should not affect our results. We provide the code that was used for evaluating the attack at [Kan+17].

8.2.2 Experiment

To confirm the practicality of the attack, we performed the first two DPA of our attack on real traces.

Traces collection. We collected $t = 1000$ different traces of the two targeted operations, where the secret key SK_1 was fixed and the addresses A were drawn uniformly at random. This many traces can be obtained by signing around 175 different messages, as BLAKE-256 is called on 7 different layers with a different a_0 for each signature. Figure 8.1 shows the average electromagnetic emanation of the two targeted operations next to each other. We use the HW leakage model.

Results. We evaluated the relation between the power traces and the guesses on, first, $(v_0 + v_5)$, and, then, on v_{15}, using Pearson's correlation coefficient with a partial DPA on 16 bits, as explained in Section 8.1. Results for both the addition and the XOR operation are shown in Figure 8.1. The upper plots show the PCC of 2^{16} guesses on the most significant bits of the targeted value, while the lower plots show the power traces average. It was observed that the correct values were those with the highest correlation coefficient in *absolute value* with $v_0 + v_5$ (for the addition) and with v_{15} (for the XOR), respectively. Similar results were found with the 16 least significant bits, which confirms that the overall attack can be successfully mounted, as the other DPAs target the same kind of operations.

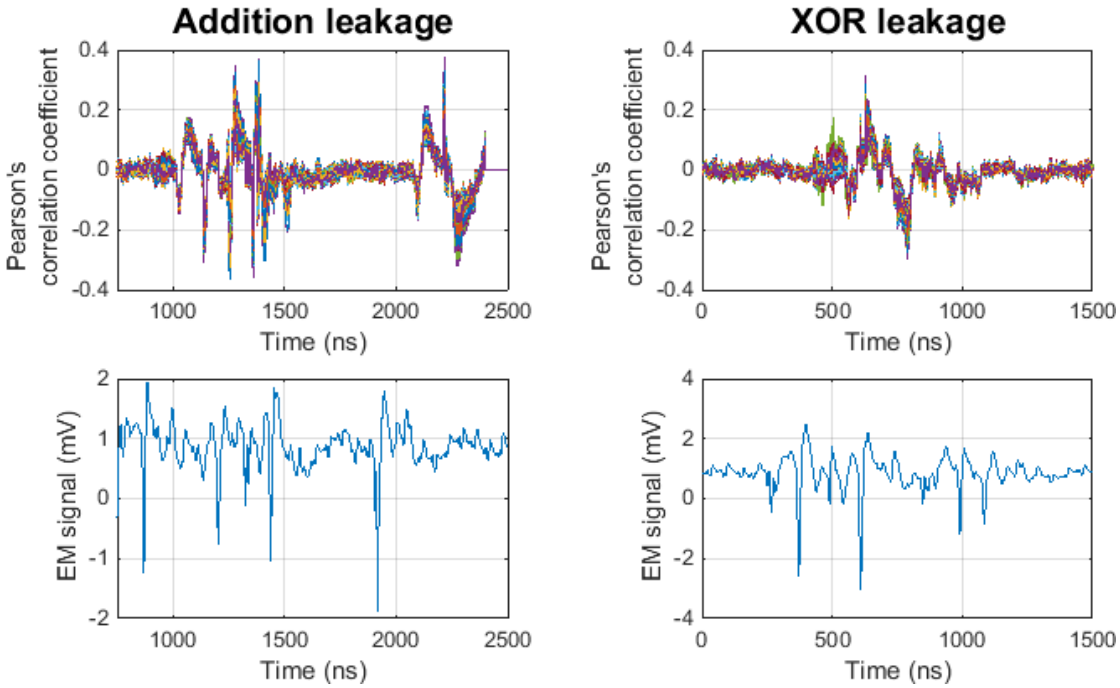


Figure 8.1: Power traces average and PCC on 16 bits of the targeted values for the addition and XOR operations ($t = 1000$).

Conclusion. The described attack recovers s_4, the fifth 32-bit chunk of SK_1. This proof of concept demonstrates that the stateless construction of SPHINCS-256 is vulnerable to DPA. Recovering this chunk is expected to lead to the recovery of other chunks. Still, a full-fledged attack that recovers the entire signing key SK_1 needs yet to be mounted.

8.3 Countermeasures

In order to mitigate the effect of this attack, we suggest *hiding* the order of the `Mix` procedures. During a BLAKE-256 round, the first four calls—as well as the next four—do not depend on each other. Their order can thus be rearranged randomly. This forces an attacker to synchronize the collected traces, making the DPA more complex.

8.4 Conclusion

In this chapter, we analyzed the side-channel resistance of SPHINCS-256, with a focus on DPA resistance. We presented a novel DPA attack on the BLAKE-256-based pseudorandom function used within SPHINCS-256 which is shown to be practical for the actual parameters of SPHINCS-256. Future work could examine the efficiency of the proposed countermeasure.

9 Fault analysis of SPHINCS+

The content of this chapter is based on the work:

[Gen23] Aymeric Genêt. "On Protecting SPHINCS+ Against Fault Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.2 (2023). https://tches.iacr.org/index.php/TCHES/article/view/10278, pp. 80–114. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i2.80-114

Context. In the previous chapter, SPHINCS has been demonstrated to be vulnerable to a DPA that recovers the signing key by analyzing the power consumption of the operations in the BLAKE-256 function. While this result may extend to SPHINCS$^+$, the attack is specific to the actual implementation of the underlying hash function and requires a path of attack as the one studied in the previous chapter. As a result, the attack cannot be applied if, say, the hash function used is unknown or if no such path can be identified in the implementation of the function.

In this chapter, we present a side-channel attack against SPHINCS$^+$ that works regardless of the hash function used. The attack was originally described against SPHINCS in 2018 by Castelnovi, Martinelli, and Prest in [CMP18] and experimentally verified by Genêt et al. in [Gen+18] and works by faulting (as described in Section 2.2) the construction of any non-top subtree. The attack enables the forgery of a valid signature for *any* chosen message once both a valid and a faulty signature of the same arbitrary message were collected. In 2020, Amiet et al. mounted the same attack on a custom hardware implementation of SPHINCS$^+$ in [Ami+20].

Even though the attack critically impacts the security of the scheme, an effective countermeasure has not been discovered yet. In the work of [CMP18] by Castelnovi, Martinelli, and Prest, the authors failed to find a specific countermeasure and recommend classical redundancy instead. The work by Mozaffari Kermani, Azarderakhsh, and Aghaie in [KAA17] proposes specific error-detection mechanisms in hash function implementations which therefore do

not entirely cover the SPHINCS$^+$ signing procedure, as well as a generic countermeasure based on recomputing hash trees with swapped nodes (i.e., also redundancy). In [Gen+18], Genêt et al. show that caching the one-time signatures of the hash trees in stateful hash-based signature schemes effectively protects against similar fault-based forgeries. This countermeasure prevents the recomputation of one-time signatures by storing the signatures of the hash trees that can still provide new signatures. However, the authors assert that the same countermeasure applied to stateless schemes is ineffective but do not provide any evidence for the claim. As a result, the extent to which SPHINCS$^+$ can be protected against fault attacks with this technique is unclear.

This chapter works in the direction of finding an effective countermeasure by analyzing the current algorithms that aim to prevent fault-based forgeries. Currently, the official specifications of SPHINCS$^+$ [Hül+20] present two mechanisms that address fault attacks: randomizing the signing procedure, and including the public key in the signing procedure so the resulting signatures can be verified. The chapter therefore analyzes fault injections against SPHINCS$^+$ in presence of these two mechanisms.

Results. Specifically, the contributions are the following:

- First, the paper starts by expanding the universal forgery with fault injections of Castelnovi, Martinelli, and Prest from [CMP18] to SPHINCS$^+$ when any types of faulty signatures are obtained. While the core of the attack is identical, the paper particularly shows that the attack is still applicable even if the adversary has collected two non-verifiable faulty signatures of the same W-OTS$^+$ keypair.
- Considering the extension, the paper presents a deep analysis of the universal forgery with a particular attention to the faulty signature collection. The analysis shows that for all parameter sets the number of queries required to circumvent the first mechanism is on average within the limit of 2^{64} signatures established by NIST, and that the probability is very high that a random faulty signature is still verifiable, defeating thus the second mechanism.
- The paper then revisits the countermeasures based on caching the W-OTS$^+$ signatures in between the intermediate subtrees as suggested in previous work (see [AE17; Gen+18]) and shows that such countermeasures are ineffective, as an active adversary can always work around the caching system with a tolerable query complexity, and as a random fault still leads to an exploitable faulty signature with a marginally lower probability than without the countermeasure. This analysis is then experimentally verified on the SPHINCS$^+$ reference implementation using the ChipWhisperer framework.

As a consequence of the above points, the paper concludes that SPHINCS$^+$ is extremely sensitive to any kinds of faults, and that no other current solution apart from redundancy effectively protects SPHINCS$^+$ against fault attack. Therefore, all real-world deployments of SPHINCS$^+$ are recommended to implement redundancy checks to mitigate the risk. Lastly,

all source code used to derive each result in the paper is made available at https://www.github.com/AymericGenet/SPHINCSplus-FA. The repository notably features a SPHINCS$^+$ implementation entirely developed in Python, as well as tools to mount the fault attack in practice.

Outline. The chapter is structured as follows: Section 9.1 describes the fault attack on SPHINCS$^+$ which is analyzed in Section 9.2. Countermeasures are discussed and analyzed in Section 9.3. Finally, the chapter reports experimental results of the countermeasures analyses in Section 9.4, and concludes with a discussion in Section 9.5.

9.1 Attack description

In their original attack in [CMP18], Castelnovi, Martinelli, and Prest present a fault attack that forces a W-OTS$^+$ key pair to sign a corrupted message by injecting a fault during the construction of any non-top subtree. Along with the valid (i.e., non-faulted) signature of the subtree, the resulting W-OTS$^+$ faulty signature is used to compromise the corresponding W-OTS$^+$ key pair under a two-message attack and provide a valid signature for another subtree for which the secrets are known. This process—similar to a *tree grafting*—enables the forgery of an overall signature for any message.

This section conducts a fault analysis (as described in Section 2.2) of SPHINCS$^+$ and expands the attack from [CMP18] to any combination of valid and faulty signatures obtained.

Attack preliminaries

Target. In the following, we consider a target device which runs any instance of SPHINCS$^+$ with a fixed and unknown signing key, but a known public key. Furthermore, such instance is supposed hardened with randomized signing (as described in Section 7.6) using a source of true randomness.

Adversarial model. The threat model considers an adversary who has access to a number of valid and faulty signatures (along with their messages) produced by the target device. The goal of the adversary is to forge a SPHINCS$^+$ signature that verifies any chosen message under the target device's public key.

Fault characteristics. The faulty signatures consist of outputs from the target device when a single unconstrained corruption of one-to-many bits occurs in any value involved in the entire SPHINCS$^+$ signing procedure. Such an outcome can happen due to the accidental or intentional effect of, e.g., the target device overheating [Bar+12], voltage disturbances [Bar+12], or row-hammer [Kim+14]. The typical use cases where the fault model is relevant include all scenarios in which a large number of signatures may be queried, such as with embedded devices, or TLS.

Due to their significant cost compared to other instructions, the fault is further assumed to occur in a hash function call. Moreover, such a fault is supposed to cause the output of the hash function to completely deviate from its intended value and be uniformly drawn at random in the co-domain of the hash function. This is aligned with the avalanche property of cryptographic hash functions in which a single bit flip early in the procedure causes an extremely different output. Besides, even if a bit flip occurs in the output of a hash function, such a bit flip will propagate in subsequent hash function calls and eventually cause uniform outputs (unless, of course, the fault hits the output of the very last hash function call of the hash structure).

9.1.1 Signatures collection

In a first phase, the adversary requires to collect both valid and faulty SPHINCS$^+$ signatures from the target device.

Verifiability. Distinguishing between valid and faulty signatures is not straightforward, as faulty signatures can still verify their message under the right public key. Instead, we differentiate two types of signatures:

1. *Verifiable signatures*: signatures that still verify their associated message under the public key of the device.

   These signatures generally correspond to valid signatures, but can also correspond to faulty signatures for which a fault occurred during the derivation of any node in an authentication path. This property enables the correct rederivation of all the subtree roots that were involved in the signature, as well as a necessarily valid top part.

2. *Non-verifiable signatures*: signatures that do not verify their associated message under the public key of the device.

   While these signatures are necessarily faulty, there are two further distinctions of non-verifiable signatures that can be made:

   - *Non-verifiable but correct*: all W-OTS$^+$ signatures still correspond to actual W-OTS$^+$ values at correct addresses.

     This type of signatures is obtained when a fault occurs on the path from the leaf to the root of a subtree. No subtree root can be recovered for sure from this kind of signature (unless the layer index at which the fault occurred is known).

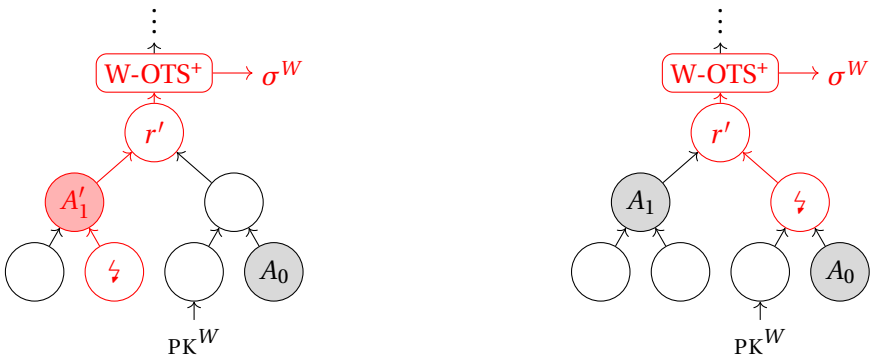
   - *Non-verifiable and incorrect*: the W-OTS$^+$ signatures do not correspond to W-OTS$^+$ values.

     This type of signatures is typically obtained when the entire output is corrupted; an outcome that commonly occurs when faulting a device. These signatures do not divulge any information and need to be discarded.

Figure 9.1 shows two examples of a faulty XMSS signature: a verifiable one, and a non-verifiable but correct one:

- In the left subfigure, the fault (ϟ) hits the computation of a node *in the authentication path*, leading thus to a faulty but known authentication path node A'_1.
- In the right subfigure, the fault (ϟ) hits the computation of a node *in the path* from the leaf used to the root, leading thus to a faulty but unknown path node.

In both case, the computation of the root has deviated from its intended value. However, in the case of the verifiable signature, the root can be recomputed from the (faulty) authentication path and the W-OTS$^+$ public key, while the root is unrecoverable in the case of the non-verifiable but correct signature, as the faulty computation was performed internally (and, thus, not exposed through the signature).



(a) Verifiable signature.

(b) Non-verifiable signature.

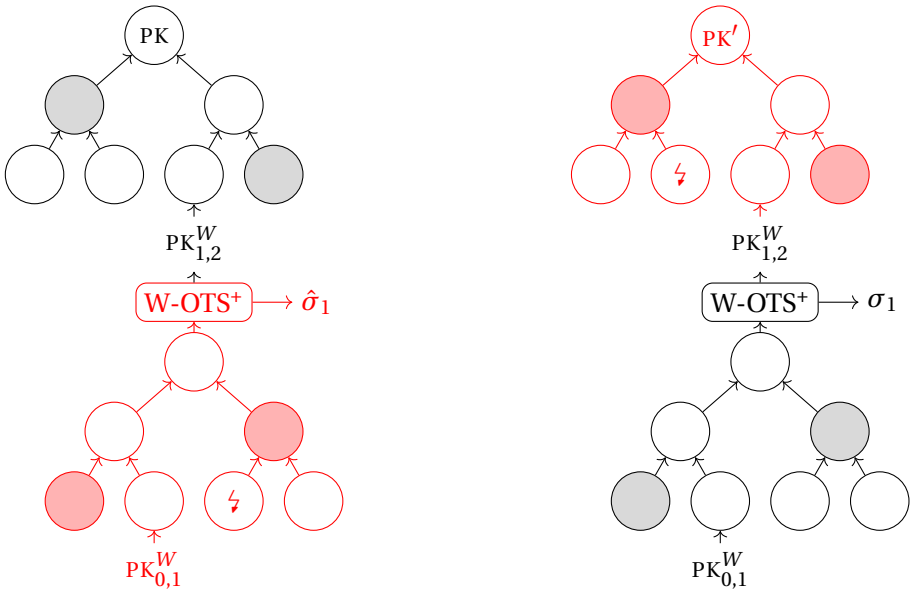
Figure 9.1: Examples of a verifiable and non-verifiable but correct faulty XMSS signatures.

Fault exploitability. In addition to the above nomenclature, a faulty signature is said to be *exploitable* when the resulting signature contains a faulty W-OTS$^+$ signature which discloses unintentional secret values of the associated W-OTS$^+$ key pair. Such an outcome occurs only when a fault hits any non-top subtree (including the ones in FORS).

Figure 9.2 shows two examples of a faulty XMSS signature: an exploitable one, and a non-exploitable one:

- In the left subfigure, the fault (ϟ) hits an XMSS for which the root is signed by a W-OTS$^+$ in another XMSS.
- In the right subfigure, the fault (ϟ) hits the top XMSS layer for which the root is not signed.

In the case of the exploitable signature, the W-OTS$^+$ key pair is used to sign a faulty message (i.e., the faulty root of the XMSS). However, in the case of the non-exploitable signature, even though the fault has hit an XMSS, no W-OTS$^+$ key pair is used to sign the faulty result.



(a) Exploitable signature.

(b) Non-exploitable signature.

Figure 9.2: Examples of an exploitable and non-exploitable faulty XMSS signatures.

An exploitable signature alone is not sufficient to compromise a W-OTS$^+$ key pair. At least one more signature of the same W-OTS$^+$ (such as the valid one) is needed as well. In this case, the secret values in both signatures can be used to forge the signature for a counterfeit XMSS (or FORS). As a result, the next step of the attack aims to determine the compromised W-OTS$^+$s by identifying the different signatures that correspond to a same key pair.

Compromised W-OTS$^+$ identification. Once valid and faulty SPHINCS$^+$ signatures $\{\Sigma_i : 0 \leq i < N\}$ have been collected, the W-OTS$^+$ signatures in the SPHINCS$^+$ signatures need to be arranged by layer and address:

1. Derive the ADRS of each W-OTS$^+$ signature in all Σ_i $(0 \leq i < N)$ from the hypertree leaf index obtained in $(\_, \texttt{ADRS}) = \mathbf{H_{msg}}(R)(\texttt{msg})$ (see Section 7.4.2).
2. Map all the W-OTS$^+$ signatures in Σ_i to their respective layer and ADRS.

If two or more different W-OTS$^+$ signatures are mapped to a same ADRS at the end of the arrangement, then the corresponding W-OTS$^+$ key pair is said to be *compromised*. In this case, such collection of W-OTS$^+$ signatures is referred to as the *faulty W-OTS$^+$ signatures* and are denoted by $(\hat{\sigma}^{(i)})_{i=0}^{M}$, while their respective full SPHINCS$^+$ signatures are denoted by $(\hat{\Sigma}^{(i)} : \hat{\sigma}^{(i)} \in \hat{\Sigma}^{(i)})_{i=0}^{M}$. We denote their layer index[1] by $l^* \in \{0, \ldots, d\}$, and denote their address by $\texttt{ADRS}^*$. Finally, we refer to all layers below (resp. above) the faulted layer as the *bottom part* (resp. as the *top part*) of the hypertree, as illustrated in Figure 9.3.

[1] Note that $l^* = 0$ means that the fault has hit the FORS layer, and that $l^* = d$ means that the fault has hit the top XMSS which does not lead to an exploitable signature.

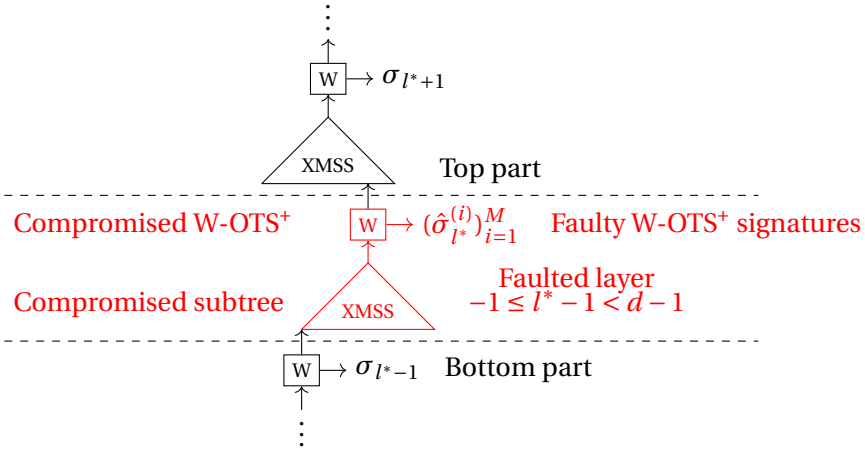


Figure 9.3: Terminology used throughout the description of the attack.

9.1.2 Faulty signatures processing

The next step processes the faulty SPHINCS$^+$ signatures identified in the previous section to extract the information that enables the universal forgery.

Secret values identification. As the elements in a W-OTS$^+$ signature correspond to secret values associated to chunks of ω bits (see Section 7.2.1), the following process aims to identify the value of the chunks that are associated to each element.

Such a process depends on the types of signatures obtained:

- **Case 1**: At least one verifiable signature is available.

  Given a verifiable signature, the correct public key of the compromised W-OTS$^+$ can be extracted from the SPHINCS$^+$ signature (see Section 7.2.1). Note that the integrity of the extracted public key must be preserved even when its corresponding subtree was faulted, as the signature verifies the extracted key under the correct SPHINCS$^+$ public key.

  The extracted W-OTS$^+$ public key can then be used to identify all the secret values in the other signatures, including the non-verifiable (but valid) ones. Strictly speaking, given the W-OTS$^+$ public key $\mathrm{PK}^W = (p_1, \ldots, p_\ell)$ and any type of W-OTS$^+$ signature $\hat{\sigma}^W = (\hat{\sigma}_1, \ldots, \hat{\sigma}_\ell)$, the secret values are identified with the following exhaustive search:

  1. Create the next ω-bit chunk b_i corresponding to $\hat{\sigma}_i$ $(1 \leq i \leq \ell)$.
  2. Check that the value is correct with $\mathbf{C}_{W-1-b_i}(\mathrm{PK}_2, \mathtt{ADRS}^{*(b_i)})(\hat{\sigma}_i) = p_i$.

  If no value leads to the W-OTS$^+$ public key element, then the $\hat{\sigma}^W$ is *incorrect*.

  Complexity. Extracting the public key of the compromised W-OTS$^+$ is equivalent to running a truncated SPHINCS$^+$ verification procedure with l^* layers (see Section 7.6), which therefore amounts to an average number of hash function calls of:

$$2 + k(a+1) + l^*(\ell(W-1)/2 + 1 + h').$$

Now, suppose that the ω-bit chunks $(\hat{b}_1^{(i)}, \ldots, \hat{b}_\ell^{(i)})$ that correspond to the W-OTS$^+$ signature $(\hat{\sigma}_1^{(i)}, \ldots, \hat{\sigma}_\ell^{(i)})$ are uniformly distributed. For $1 \leq j \leq \ell$, finding the value of the chunk $\hat{b}_j^{(i)}$ that corresponds to $\hat{\sigma}_j$ requires $W-1-x$ applications of **F** for a hypothesized initial position $0 \leq x \leq W-1$ until the resulting value equals p_i. As each value occurs with probability $1/W$, the average number of hash function calls is:

$$\sum_{x=0}^{W-1} \left(\frac{1}{W}\right)(W-1-x) = (W-1)/2.$$

As there are ℓ blocks in each $\hat{\sigma}^W$, the overall number of hash function calls for this case is $\ell(W-1)/2$.

- **Case 2**: Only non-verifiable signatures are available.

  Since none of the subtree roots can be recovered for sure, the adversary cannot extract the compromised W-OTS$^+$ public key from the non-verifiable signatures. However, the adversary can determine the positions of each W-OTS$^+$ value by using one value as a reference for the other.

  In other words, given a pair of different W-OTS$^+$ signatures, i.e., $(\hat{\sigma}^{(0)}, \hat{\sigma}^{(1)})$ where $\hat{\sigma}^{(0)} = (\hat{\sigma}_1^{(0)}, \ldots, \hat{\sigma}_\ell^{(0)})$ and $\hat{\sigma}^{(1)} = (\hat{\sigma}_1^{(1)}, \ldots, \hat{\sigma}_\ell^{(1)})$, consider the two values $\hat{\sigma}_i^{(0)}, \hat{\sigma}_i^{(1)}$ at a same index $1 \leq i \leq \ell$. There are two possibilities:

  – $\hat{\sigma}_i^{(0)} \neq \hat{\sigma}_i^{(1)}$: in this case, if both signatures are *correct*, then there must exist positions $0 \leq u < v < W$ such that

  $$\mathbf{C}_v(\mathrm{PK}_2, \mathtt{ADRS}^{*(u)})(\hat{\sigma}_i^{(0)}) = \hat{\sigma}_i^{(1)}, \text{ or } \mathbf{C}_v(\mathrm{PK}_2, \mathtt{ADRS}^{*(u)})(\hat{\sigma}_i^{(1)}) = \hat{\sigma}_i^{(0)}.$$

  The above property enables confirming guesses on u and v, which directly leads to the i^{th} ω-bit chunk of both roots, since the hash applications use different addresses at each step of the chaining pseudorandom function. As a result, the values are extracted as follows:

  1. Create the next ω-bit chunks u, v respectively corresponding to $\hat{\sigma}_i^{(0)}, \hat{\sigma}_i^{(1)}$ $(1 \leq i \leq \ell)$.
  2. Check that the values are correct with $\mathbf{C}_v(\mathrm{PK}_2, \mathtt{ADRS}^{*(u)})(\hat{\sigma}_i^{(0)}) = \hat{\sigma}_i^{(1)}$ or $\mathbf{C}_v(\mathrm{PK}_2, \mathtt{ADRS}^{*(u)})(\hat{\sigma}_i^{(1)}) = \hat{\sigma}_i^{(0)}$.

  If no such u and v exist, then at least one of the signatures is *incorrect*.

  Complexity. Supposing that all chunks are uniformly distributed, the probability that $\hat{b}_j^{(0)}$ and $\hat{b}_j^{(1)}$ take different values is $1/(W(W-1))$. Since, for a fixed u, the exhaustive search on v can apply **F** on the previous hash result, the average number of hash calls is:

  $$\sum_{x=1}^{W(W-1)} \left(\frac{1}{W(W-1)}\right) x = \frac{W(W-1)+1}{2}.$$

– $\hat{\sigma}_i^{(0)} = \hat{\sigma}_i^{(1)}$: in this case, if both signatures are *correct*, then the two values must correspond to the same ω-bit chunk, but of unknown position in the chaining pseudorandom function. Another signature with a different value at index i is required to identify the value of the chunks.

If there are still chunks of unknown positions at the end of the secret values identification, such positions can be retrieved while extracting the top part of the SPHINCS$^+$ signature (see below).

An illustration for the two possibilities for $\hat{\sigma}_i^{(0)}$, $\hat{\sigma}_i^{(1)}$ in a chaining pseudorandom function is shown in Figure 9.4.

(a) Different positions in **C**.

(b) Same (unknown) position in **C**.

Figure 9.4: Identifying W-OTS$^+$ values within non-verifiable signatures.

The above process is applied to all faulty W-OTS$^+$ signatures in order to retrieve as many secret values as possible to forge a signature for a variety of different ω-bit chunks.

Note that since the values at lower positions in the chaining function enable the recomputation of values at higher positions, it suffices to keep track of the values at the lowest positions only. As a result, in the following, we refer to the lowest positions learnt by the secret extraction as the *most secret* elements which are denoted by $(\tilde{\theta}_1, \ldots, \tilde{\theta}_\ell)$ and are respectively associated to the ω-bit chunks of $(\tilde{b}_1, \ldots, \tilde{b}_\ell)$.

Top part extraction. Extracting a valid top part is required so that the verification of the forged SPHINCS$^+$ signature leads to the target device's public key.

The extraction considers the case in which multiple top parts are available due to our fault model. Under these circumstances, all the top parts available need to be tried out starting from the compromised W-OTS$^+$ public key until one leads to the target device's public key.

Let $(\sigma^{X^{(i)}}_{l^*+1}, \ldots, \sigma^{X^{(i)}}_{d-1})$ be the top part of the SPHINCS$^+$ signature $\hat{\Sigma}_i$ $(0 \leq i < M)$, and PK_1 be the SPHINCS$^+$ public key, and suppose that the compromised W-OTS$^+$ public key $\mathrm{PK}^W = (p_1, \ldots, p_\ell)$ has been successfully extracted (see above):

1. Extract the hypertree public key PK^{HT} from $\mathbf{T}_\ell(\mathrm{PK}_2, \mathtt{ADRS}^*)(\mathrm{PK}^W)$ and the XMSS signatures $(\sigma^{X^{(i)}}_{l^*+1}, \ldots, \sigma^{X^{(i)}}_{d-1})$ (see Section 7.4.2).
2. Check that $\mathrm{PK}^{HT} = \mathrm{PK}_1$.

In case there were still ω-bit chunks of unknown values at the end of the secrets extraction, such chunks may be identified during this part by guessing all of the unknown chunks at once,

deriving the corresponding W-OTS$^+$ public key, and trying this public key with the above steps. Such a process both confirms the values of the unknown chunks, and extracts a valid top part of the signature.

Complexity. Verifying that a selected top part is valid requires a truncated SPHINCS$^+$ verification procedure starting from the compromised W-OTS$^+$. Along with all the top parts available, there may be chunks that need to be exhaustively searched in case no verifiable signature was available (see above). Supposing that the blocks are uniformly distributed, the probability that, for a fixed index $1 \leq j \leq \ell$, all W-OTS$^+$ elements $\hat{\sigma}_j^{(i)}$ are the same is $1/W^{M-1}$. Therefore, on average, the number of chunks of unknown value is:

$$\mathbb{E}(\text{Non-id. chunks}) = \ell / W^M.$$

Given $\text{PK}^W = (p_1, \ldots, p_\ell)$, each trial requires one application of $\mathbf{T}_\ell$ and the recomputation of the root of the XMSS right above the faulted layer, as well as the full public key extraction of $(d-1) - l^*$ XMSS public keys. Therefore, the average number of hash function calls is:

$$1 + h' + (d - l^* - 1)(\ell(W-1)/2 + 1 + h').$$

9.1.3 Tree grafting

Once the most secret values of a compromised W-OTS$^+$ key pair were successfully extracted from the faulty signatures, the adversary aims to *graft* a subtree (or a forest) to the extracted top part, i.e., find another XMSS (or FORS) for which a valid W-OTS$^+$ signature can be forged in order to spoof the compromised instance at its own address.

During this step, the adversary attempts to sign the root of a forged FORS or XMSS with the W-OTS$^+$ secret values at disposal. Let $(\tilde{\theta}_1, \ldots, \tilde{\theta}_\ell)$ be the most secret W-OTS$^+$ values extracted which correspond to the ω-bit chunks $(\tilde{b}_1, \ldots, \tilde{b}_\ell)$, the grafting procedure repeats the following until successful:

1. Draw $\text{SK}' \in \mathbb{B}^n$ uniformly at random.
2. If $l^* = 0$, create a FORS of public key r' with SK' at $\texttt{ADRS}^*$ (see Section 7.3.2),
   else, create an XMSS of public key r' with SK' at $\texttt{ADRS}^*$ (see Section 7.4.1).
3. Split r' and its checksum into chunks $(r'_1, \ldots, r'_\ell)$ of ω bits.
4. Check that $r'_i \leq b_i$ for all $1 \leq i \leq \ell$.

Once found, the secret key of the grafted subtree is SK' and its signature is:

$$\sigma'^X_{l^*} = (\mathbf{C}_{r'_1 - \tilde{b}_1}(\text{PK}_2, \texttt{ADRS}^{*(\tilde{b}_1)})(\tilde{\theta}_1), \ldots, \mathbf{C}_{r'_\ell - \tilde{b}_\ell}(\text{PK}_2, \texttt{ADRS}^{*(\tilde{b}_\ell)})(\tilde{\theta}_\ell)).$$

Complexity. The tree grafting depends on the layer hit:

- In case a FORS needs to be forged ($l^* = 0$), the public key derivation amounts to k generations of FORS trees, each of them requiring a treehash procedure of height $\log_2(t) = a$, in addition to a final application of $\mathbf{T}_k$ with all the FORS tree roots:

$$k\left(t + \sum_{i=0}^{a} 2^{i-1}\right) + 1 = k(3t-1) + 1.$$

- In case an XMSS needs to be forged ($1 \leq l^* \leq d-1$), the public key derivation amounts to $2^{h'}$ W-OTS$^+$ public keys generations and a treehash procedure of height h':

$$2^{h'}(\ell + \ell(W-1) + 1) + \sum_{i=1}^{h'} 2^{i-1} = 2^{h'}(\ell W + 2) - 1.$$

The probability that one attempt is successful is given by an extension of the work of Bruinderink and Hülsing in [BH17]. Given $M > 1$ different W-OTS$^+$ signatures, supposing that the chunks $(b_1, \ldots, b_\ell)$ are uniformly[2] distributed, each chunk x occurs with probability $1/W$ and enables the forgery of all chunks from x to $W-1$. Thus, the overall probability that the root of a forged XMSS can be signed is:

$$\begin{aligned}
\mathbb{P}(\text{Grafting}) &\leq \frac{1}{W^\ell}\left(\sum_{x=0}^{W-1}\left(1 - \left(\frac{W-1-x}{W}\right)^M\right)\right)^\ell \\
&\leq \frac{1}{W^\ell}\left(W - \sum_{x=0}^{W-1}\frac{x^M}{W^M}\right)^\ell \\
&\leq \frac{1}{W^\ell}\left(W - \frac{1}{W^M}\left(\frac{(W-1)^{M+1}}{M+1} + \mathcal{O}\left(W^M\right)\right)\right)^\ell \\
&\leq \frac{1}{W^\ell}\left(\left(\frac{M}{M+1}\right)W + \mathcal{O}(1)\right)^\ell \\
&\leq \left(1 - \left(\frac{1}{M+1}\right)W\right)^\ell + \mathcal{O}(1) \quad \approx \quad e^{-\ell/(M+1)}.
\end{aligned}$$

9.1.4 Path seeking

The SPHINCS$^+$ signing procedure follows a path in the hypertree depending on the message and a value R. As a result, the adversary requires to find an adequate value R that makes the forged signature visit the compromised subtree.

[2] We call attention to the fact that the uniform hypothesis of the blocks $(b_{\ell_1+1}, \ldots, b_{\ell_1+\ell_2})$ is not rigorous as these blocks are actually sums of uniform random variables. However, simulations in [CMP18; Gen+18] show that such a discrepancy is tolerable for our use cases.

Straightforwardly, given the message `msg'` to be maliciously signed, the value R is brute-forced until the corresponding tree index at layer l^* is the same as the grafted subtree:

1. Draw $R' \in \mathbb{B}^n$ uniformly at random.
2. Check that the hypertree leaf index in $(\_, \texttt{ADRS}) = \mathbf{H}_{\texttt{msg}}(\texttt{PK}_1, R')(\texttt{msg'})$ leads to the tree index of the grafted subtree (see Section 7.4.2).

While a single grafted subtree allows the adversary to forge valid SPHINCS$^+$ signatures for as many messages as desired, note that path seeking depends on the message and therefore needs to be repeated for each new message.

Complexity. Finding R' is equivalent to an exhaustive search of n bytes such that the $h - h'l^*$ most significant bits of the tree index give the index of the grafted subtree (see Section 7.4.2). Each trial requires only a single hash function application, and its probability of success is simply $2^{-(h-h'l^*)}$. Consequently, the adversary requires $2^{h-h'l^*}$ hash function calls on average to find an appropriate value for R'.

9.1.5 Universal forgery

Piecing everything together, the adversary uses the grafted subtree and the value R' to forge a bottom part of the signature, then plugs the extracted top part onto the forged part to craft a valid signature for the malicious message (selected in Section 9.1.4). The procedure goes as follows:

1. Generate arbitrary key pairs to forge $(\sigma'^F, \sigma'^X_0, \ldots, \sigma'^X_{l^*-1})$, i.e., all the signatures in the layers below the grafted subtree (see Section 7.3.2 and Section 7.4.2).
2. Sign $\sigma'^X_{l^*-1}$ with the grafted XMSS at address $\texttt{ADRS}^*$ using SK' (see Section 7.4.1).
3. Copy the top part for the rest of the signatures.

The final signature that verifies $\texttt{msg}'$ under the device's public key is therefore:

$$\Sigma' = (\underbrace{R'}_{\substack{\uparrow \\ \text{sought} \\ \text{Section 9.1.4}}}, \underbrace{\sigma'^F, \sigma'^X_0, \ldots, \sigma'^X_{l^*-1}}_{\substack{\text{forged} \\ \text{Section 9.1.5}}}, \underbrace{\sigma'^X_{l^*}}_{\substack{\uparrow \\ \text{grafted} \\ \text{Section 9.1.3}}}, \underbrace{\sigma^X_{l^*+1}, \ldots, \sigma^X_{d-1}}_{\substack{\text{extracted} \\ \text{Section 9.1.2}}}).$$

The average computational complexity of each step in the universal forgery is shown in Table 9.1 for all SPHINCS$^+$ parameters sets. These numbers suggest that the fault attack is feasible in all scenarios, although the number of required hashes varies significantly depending on the specific layer targeted by the attack. However, even though the reported numbers seem high, the overall number of required hashes can still be attainable in practice[3]. This result is especially important as the fault attack can therefore be successful even if the fault is uncontrolled. The latter will be analyzed in the next section.

[3] For reference, for SHA2-256, an Nvidia RTX 3090 is reported to perform $2^{36.95}$ hashes per second (see [Onl22]). The actual performance may vary since, in our use cases, the results of previous hash function calls need to be used as inputs for the next ones.

	Processing (Section 9.1.2)						Path seeking (Section 9.1.4)				
	Case 1 (hashes)	Case 2 (hashes)	$\mathbb{E}$(Non-id. chunks)				(hashes)				
			$M=$	2	3	4	$l^*=$	0	1	…	$d-1$
128s	$2^{8.04}$	$2^{12.04}$		2.12	0.14	0.01		2^{64}	2^{56}	…	2^{8}
128f	$2^{8.04}$	$2^{12.04}$		2.12	0.14	0.01		2^{60}	2^{57}	…	2^{3}
192s	$2^{8.58}$	$2^{12.59}$		3.19	0.20	0.01		2^{64}	2^{56}	…	2^{8}
192f	$2^{8.58}$	$2^{12.59}$		3.19	0.20	0.01		2^{66}	2^{63}	…	2^{3}
256s	$2^{8.97}$	$2^{12.98}$		4.19	0.26	0.01		2^{64}	2^{56}	…	2^{8}
256f	$2^{8.97}$	$2^{12.98}$		4.19	0.26	0.01		2^{68}	2^{64}	…	2^{4}

	Grafting (Section 9.1.3)									
	FORS (hashes)					XMSS (hashes)				
$M=$	2	4	8	16	32	2	4	8	16	32
128s	$2^{38.11}$	$2^{29.32}$	$2^{24.25}$	$2^{21.59}$	$2^{20.36}$	$2^{35.34}$	$2^{26.55}$	$2^{21.48}$	$2^{18.81}$	$2^{17.58}$
128f	$2^{33.70}$	$2^{24.90}$	$2^{19.84}$	$2^{17.17}$	$2^{15.94}$	$2^{30.34}$	$2^{21.55}$	$2^{16.48}$	$2^{13.81}$	$2^{12.58}$
192s	$2^{47.92}$	$2^{35.11}$	$2^{27.72}$	$2^{23.84}$	$2^{22.05}$	$2^{44.21}$	$2^{31.39}$	$2^{24.01}$	$2^{20.12}$	$2^{18.33}$
192f	$2^{41.16}$	$2^{28.34}$	$2^{20.96}$	$2^{17.07}$	$2^{15.28}$	$2^{39.21}$	$2^{26.39}$	$2^{19.01}$	$2^{15.12}$	$2^{13.33}$
256s	$2^{54.90}$	$2^{38.06}$	$2^{28.36}$	$2^{23.26}$	$2^{20.91}$	$2^{52.92}$	$2^{36.09}$	$2^{26.39}$	$2^{21.28}$	$2^{18.93}$
256f	$2^{51.35}$	$2^{34.51}$	$2^{24.81}$	$2^{19.71}$	$2^{17.35}$	$2^{48.92}$	$2^{32.09}$	$2^{22.39}$	$2^{17.28}$	$2^{14.93}$

Table 9.1: Average complexity of each step of the universal forgery for all SPHINCS$^+$ parameters (the 'f' instances stand for "fast", while the 's' instances stand for "small").

9.2 Attack analysis

This section analyzes the fault attack described in Section 9.1.

9.2.1 Fault analysis

Since our fault model considers that faults only affect the results of hash functions, the following counts the number of hash function calls in the entire SPHINCS$^+$ signing procedure to determine the proportion of calls that, when faulted, lead to an exploitable or a verifiable faulty signature.

1. **Path derivation:** $R = \mathbf{PRF_{msg}}(\mathtt{SK}_2, \mathrm{opt})(\mathtt{msg})$.
   - *Total hash function calls:* 1.
   - *Fault exploitability:* No.
   - *Signature verifiability:* The resulting signature is *verifiable* (even valid), as R is anyway included in the signature and the result of a random selection.
2. **Digest and initial address:** $(\mathtt{md}, \mathtt{ADRS}) = \mathbf{H_{msg}}(\mathtt{PK}_1, R)(\mathtt{msg})$.
   - *Total hash function calls:* 1.
   - *Fault exploitability:* No.
   - *Signature verifiability:* The resulting signature is *non-verifiable and incorrect*, as an improper FORS is used to sign an improper digest.
3. **FORS signature** (i.e., $l^* = 0$).
   - *Total hash function calls:* $\#\mathrm{Total}^F = k(3t-1)+1$.
   - *Fault exploitability:* Yes.
   - *Signature verifiability:* The verifiability of the resulting signature depends on the location of the fault in the subtrees:
     - A *verifiable signature* is obtained when a fault hits any value involved in an authentication path of a FORS tree. Each authentication path requires the derivation of $t-1$ secret values, as well as $2^{a-i}-1$ nodes in level $0 \leq i \leq a$ of a tree. As there are k trees, this amounts to a total number of verifiable signatures of:

$$\#\mathrm{Verif}^F = k\left((t-1) + \sum_{i=0}^{a}(2^{a-i}-1)\right) = k(3t-a-3).$$

     - A *non-verifiable but correct* signature is obtained when a fault hits any value on the path from a leaf to the root of a FORS tree. The values in a path consist of the secret leaf derivation, in addition to a single node in all levels of a tree, and the computation of the FORS public key. As there are k trees of $t = 2^a$ leaves,

this amounts to a total number of non-verifiable but correct signatures of:

$$\#\text{Non-verif}^F = k\left(1+\sum_{i=0}^{a} 1\right)+1 = k(a+2)+1.$$

4. **XMSS signature** at a non-top layer (i.e., $1 \leq l^* < d$).
   - *Total hash function calls:* $\#\text{Total}^X = 2^{h'}(\ell W+2)-1$.
   - *Fault exploitability:* Yes.
   - *Signature verifiability:* The verifiability of the resulting signature depends on the location of the fault in the subtree:
     - A *verifiable signature* is obtained when a fault hits any value involved in the authentication path of a non-top XMSS. Such an authentication path starts with the derivation of $2^{h'}-1$ W-OTS$^+$ public keys, as well as the computation of $2^{h'-i}-1$ nodes at each level $1 \leq i \leq h'$ of the subtree. Every W-OTS$^+$ public key requires the derivation of ℓ secret values; each of them chained $W-1$ times with the chaining pseudorandom function, so that all the results can be compressed with $\mathbf{T}_\ell$. This amounts to a total number of verifiable signatures of:
     $$\begin{aligned}\#\text{Verif}^X &= (2^{h'}-1)(\ell+\ell(W-1)+1)+\textstyle\sum_{i=1}^{h'}(2^{h'-i}-1)\\ &= (2^{h'}-1)(\ell W+1)+2^{h'}-h'-1.\end{aligned}$$
     - A *non-verifiable but correct* signature is obtained when a fault hits any value on the path from a leaf to the root of a non-top XMSS. The values in a path consist of a single W-OTS$^+$ public key, in addition to a single node in all levels of a tree. As above, the W-OTS$^+$ public key requires the derivation of ℓ secret values; each of them chained $W-1$ times with the chaining pseudorandom function, so that all the results can be compressed with $\mathbf{T}_\ell$. Since there are h' levels, this amounts to a total number of non-verifiable but correct signatures of:
     $$\#\text{Non-verif}^X = \ell W+1+\sum_{i=1}^{h'} 1 = \ell W+1+h'.$$
5. **XMSS signature** at the top layer (i.e., $l^* = d$).
   - *Total hash function calls:* $\#\text{Total}^X = 2^{h'}(\ell W+2)-1$.
   - *Fault exploitability:* No.
   - *Signature verifiability:* The resulting signature is *non-verifiable but correct*, as the reconstruction of this XMSS does not lead to the SPHINCS$^+$ public key. All the W-OTS$^+$ signatures involved are valid, however no valid top part can be extracted.

Summing up the hash function calls of all the components above, the grand total of hash function calls in a single SPHINCS$^+$ signature is therefore given by:

$$\#\text{Total} = 1+1+\#\text{Total}^F + d\cdot\#\text{Total}^X = 3+k(3t-1)+d(2^{h'}(\ell W+2)-1).$$

Table 9.2 computes the total numbers of possible verifiable and non-verifiable faulty signatures for both FORS and non-top XMSS in all SPHINCS$^+$ parameters sets. This table shows that a random fault is much likelier to give a verifiable signature rather than a non-verifiable one, and so that verifying the signature is not effective in detecting faulty signatures.

	FORS ($l^* = 0$)				XMSS ($1 \leq l^* < d$)			
	Verif.		Non-verif.		Verif.		Non-verif.	
	Total	Ratio	Total	Ratio	Total	Ratio	Total	Ratio
128s	982,860	0.9998	171	0.0002	143,302	0.9960	569	0.0040
128f	45,720	0.9928	331	0.0072	3,931	0.8745	564	0.1255
192s	2,752,246	0.9999	253	0.0001	208,582	0.9961	825	0.0039
192f	24,981	0.9869	331	0.0131	5,723	0.8747	820	0.1253
256s	1,080,970	0.9997	353	0.0003	273,862	0.9961	1,081	0.0039
256f	91,770	0.9961	361	0.0039	16,106	0.9373	1,077	0.0627

Table 9.2: Proportion of verifiable vs. non-verifiable signatures for faulty FORS and (non-top) XMSS for all SPHINCS$^+$ parameters sets.

Suppose that a fault can hit any hash function call uniformly at random. The above enumerations lead to the following probabilities:

Fault exploitability. The probability that the faulty signature is exploitable is given by the proportion of faulty signature outcomes that leads to an exploitable signature:

$$\mathbb{P}(\text{Expl.}) = \frac{\#\text{Total}^F + (d-1)\cdot\#\text{Total}^X}{\#\text{Total}} = \frac{k(3t-1)+1+(d-1)(2^{h'}(\ell W+2)-1)}{3+k(3t-1)+d(2^{h'}(\ell W+2)-1)}.$$

Fault verifiability. Similarly, the probability that the faulty signature is verifiable is given by the proportion of the faulty signature outcomes that leads to a verifiable signature:

$$\begin{aligned}\mathbb{P}(\text{Verif.}) &= \frac{1+\#\text{Verif}^F+(d-1)\cdot\#\text{Verif}^X}{\#\text{Total}} \\ &= \frac{1+k(3t-a-3)+(d-1)((2^{h'}-1)(\ell W+1)+2^{h'}-h'-1)}{3+k(3t-1)+d(2^{h'}(\ell W+2)-1)}.\end{aligned}$$

Layer hit. Let $L = l^*$ denote the event that a fault has affected $\sigma^X_{l^*}$ (i.e., that a hash function call in the layer l^*-1 is hit by a fault). The probability that $L = l^*$ is therefore given by the total number of hash function calls at layer l^*-1:

$$\mathbb{P}(L = l^*) = \begin{cases} \dfrac{\#\text{Total}^F}{\#\text{Total}} = \dfrac{k(3t-1)+1}{3+k(3t-1)+d(2^{h'}(\ell W+2)-1)} & \text{if } l^* = 0, \\ \dfrac{\#\text{Total}^X}{\#\text{Total}} = \dfrac{2^{h'}(\ell W+2)-1}{3+k(3t-1)+d(2^{h'}(\ell W+2)-1)} & \text{if } 1 \le l^* \le d. \end{cases}$$

Table 9.3 computes the above probabilities given all SPHINCS$^+$ parameters sets. This table shows that the probability that a random fault leads to a signature that is both exploitable and verifiable is high.

	$\mathbb{P}$(Expl.)	$\mathbb{P}$(Verif.)	$\mathbb{P}(L = l^*)$				
			$l^* =$ 0	1	…	$d-1$	d
128s	0.9326	0.9306	0.4607	0.0674	…	0.0674	0.0674
128f	0.9669	0.8857	0.3387	0.0331	…	0.0331	0.0331
192s	0.9527	0.9513	0.6216	0.0473	…	0.0473	0.0473
192f	0.9613	0.8576	0.1495	0.0387	…	0.0387	0.0387
256s	0.9162	0.9138	0.3296	0.0838	…	0.0838	0.0838
256f	0.9553	0.9095	0.2398	0.0447	…	0.0447	0.0447

Table 9.3: Fault analysis results for all SPHINCS$^+$ parameters.

9.2.2 Universal forgery analysis: one-fault model

This section analyzes the use case where the adversary has access to many valid signatures (i.e., $M_v > 1$) but only a single faulty one (i.e., $M_f = 1$) which is supposed exploitable and which corresponds to layer $0 \le l^* < d$. Let $N = 2^{h-h'l^*}$ be the total number of W-OTS$^+$ key pairs on layer l^*.

Collecting the corresponding valid signature. The probability that the valid signature corresponding to the same key pair as the faulty signature is included in the collected M_v signatures is simply given by:

$$\mathbb{P}(\text{W-OTS}^+ \text{ break}) = 1 - \left(1 - \frac{1}{N}\right)^{M_v}.$$

Alternatively, the expected number of valid queries to obtain the corresponding valid signature is given by a geometric random variable with probability $1/N$:

$$\mathbb{E}(M_v) = N.$$

Table 9.4 computes the expected numbers of valid queries to obtain in order to mount the universal forgery for each SPHINCS$^+$ parameters set. The average number of queries required to mount the forgery is in most cases lower than NIST's security definition for digital signatures

which limits the number of queries to 2^{64} (see [NIS16]).

		$\mathbb{E}(M_v)$				
	$l^* =$	0	1	…	$d-1$	d
128s		2^{64}	2^{56}	…	2^8	–
128f		2^{60}	2^{57}	…	2^3	–
192s		2^{64}	2^{56}	…	2^8	–
192f		2^{66}	2^{63}	…	2^3	–
256s		2^{64}	2^{56}	…	2^8	–
256f		2^{68}	2^{64}	…	2^4	–

Table 9.4: Average numbers of valid signatures to collect the valid signature corresponding to a single faulty signature for all SPHINCS$^+$ parameters.

9.2.3 Universal forgery analysis: multiple-fault model

This section analyzes the use case where the adversary has access to multiple valid and faulty signatures (i.e., $M_v > 1$, $M_f > 1$) which are all supposed to be exploitable, different, and which all correspond to the same layer $0 \leq l^* < d$. Let $N = 2^{h-h'l^*}$ be the total number of W-OTS$^+$ key pairs on layer l^*. Also, let $\left\{ {a \atop b} \right\}$ denote the Stirling number of the second kind which counts the number of ways to partition a objects into b non-empty subsets.

Faulty signatures collision. As the universal forgery can be mounted with only faulty signatures, the probability that two faulty signatures correspond to the same W-OTS$^+$ key pair is an instance of the birthday paradox [FGT92]:

$$\mathbb{P}(\text{W-OTS}^+ \text{ break}) = 1 - \frac{N!}{N^{M_f}(N-M_f)!}.$$

Pair of valid and faulty signatures. Combining the faulty signatures with the valid ones, the probability that a faulty signature corresponds to the same W-OTS$^+$ key pair as a valid signature is an instance of the occupancy problem with two types of balls [NS88]:

$$\mathbb{P}(\text{W-OTS}^+ \text{ break}) = 1 - \frac{1}{N^{M_v+M_f}} \sum_{t_v=1}^{M_v} \sum_{t_f=1}^{M_f} \left\{ {M_v \atop t_v} \right\} \left\{ {M_f \atop t_f} \right\} \frac{N!}{(N-t_v-t_f)!}.$$

Table 9.5 computes the above probabilities with $N = 256$ (i.e., when $l^* = d-1$ for the 128s, 192s, and 256s parameters sets of SPHINCS$^+$, or $l^* = d-2$ for SPHINCS$^+$-256f). This table shows that the randomness plays in the favor of the adversary, as only very few faulty queries are required to break a W-OTS$^+$. This number drops even lower when combined with very few valid queries.

$M_f \backslash M_v$	0	4	8	16	32	64
4	0.0233	0.0607	0.1177	0.2215	0.3939	0.6325
8	0.1046		0.2215	0.3939	0.6325	0.8647
16	0.3803			0.6325	0.8647	0.9815
32	0.8676				0.9815	0.9996
64	0.9997					1.0000

Table 9.5: Probability of collision with either only faulty queries (under $M_v = 0$) or with M_f faulty and M_v valid queries ($N = 256$). Symmetrical values were omitted.

Increasing the numbers of faulty signatures. While a single pair of different W-OTS$^+$ signatures corresponding to a same key pair is enough to mount the universal forgery, the grafting step becomes easier the more faulty W-OTS$^+$ signatures are obtained for a same key pair (see Section 9.1.3). In order to study this, notice that collecting M_f faulty signatures from N key pairs can be modeled as a multinomial distribution with uniform probabilities (i.e., $p_k = 1/N$ for $1 \leq k \leq N$).

The probability that at least one W-OTS$^+$ key pair has been reused c times is an instance of the maximal frequency in a multinomial distribution. Let s_k determine the accumulated number of W-OTS$^+$ signatures counting from the first W-OTS$^+$ key pair to the k^{th} key pair (so $s_0 = 0$ and $s_N = M_f$). Then, from the analysis by Corrado in [Cor11], the transition probability from s_{k-1} to s_k is given by:

$$\mathbb{P}(s_k \mid s_{k-1}) = \left(\begin{array}{c} M_f - s_{k-1} \\ s_k - s_{k-1} \end{array} \right) {\pi_k}^{s_k - s_{k-1}} (1 - \pi_k)^{M_f - s_k},$$

where $\pi_k = p_k / (\sum_{i=k}^{N} p_k) = (1/N)/(\sum_{i=k}^{N} 1/N) = 1/(N-k+1)$.

Given the above probabilities, the stochastic matrix that determines the transitions from s_{k-1} to s_k is defined as follows:

$$\mathbf{Q}_k = \left(\begin{array}{cccc} \mathbb{P}(0 \mid 0) & \mathbb{P}(1 \mid 0) & \cdots & \mathbb{P}(M_f \mid 0) \\ 0 & \mathbb{P}(1 \mid 1) & \cdots & \mathbb{P}(M_f \mid 1) \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & 1 \end{array} \right) \quad \text{for } 1 \leq k \leq N-1,$$

$$\mathbf{Q}_N = \left(\begin{array}{cccc} 1 & 1 & \cdots & 1 \end{array} \right)^{\top}.$$

Let $\bar{\mathbf{Q}}_k$ be the result of culling the transition probabilities that assign more than c signatures to a key pair (i.e., by setting $\mathbb{P}(s_k - s_{k-1} > C) = 0$ for the relevant s_k, s_{k-1}) and let $\bar{\mathbf{Q}}_1^{(1)}$ be the first row of $\bar{\mathbf{Q}}_1$.

The probability that the maximum load is no more than c is given by the transition from s_1 to s_n which is determined by the following product of stochastic matrices:

$$\mathbb{P}(\text{Max. load} \leq c \mid M_f) = \bar{\mathbf{Q}}_1^{(1)} \times \bar{\mathbf{Q}}_2 \times \cdots \times \bar{\mathbf{Q}}_N.$$

Alternatively, the expected maximum load given M_f faulty signatures is:

$$\mathbb{E}(\text{Max. load} \mid M_f) = \sum_{c=0}^{M_f - 1} \mathbb{P}(\text{Max. load} > c \mid M_f).$$

Combining this result with the valid signatures, notice that the maximum load is increased by one by collecting the valid signature of the W-OTS$^+$ for which the maximum load is reached. As such an event can be modeled as a Bernoulli random variable with probability $1-(1-1/N)^{M_v}$, we ultimately have:

$$\mathbb{E}(\text{Max. load} \mid M_v, M_f) = \mathbb{E}(\text{Max. load} \mid M_f) + \left(1 - \left(\frac{N-1}{N}\right)^{M_v}\right).$$

Table 9.6 computes the maximum load averages with M_f signatures for the N that correspond to the few first top layers of the SPHINCS$^+$ parameters sets, as increasing the number of signatures is especially relevant when targeting such layers.

$N \backslash M_f$	64	128	256	512	1,024
2^3	12.23	21.90	40.26	75.60	144.31
2^4	7.88	13.35	23.43	42.36	78.50
2^6	3.96	5.97	9.37	15.33	26.10
2^8	2.46	3.38	4.77	6.99	10.69
2^9	2.12	2.74	3.68	5.16	7.48

Table 9.6: Maximum load averages with various numbers of faulty signatures M_f in different layers of N signatures.

Layer coverage. The probability that the collected valid signatures cover the entire layer—in which case, all valid signatures are known—is an instance of the coupon collector's problem [FGT92]:

$$\mathbb{P}(\text{Layer is covered}) = \frac{N!}{N^{M_v}} \left\{ {M_v - 1 \atop N - 1} \right\}.$$

Alternatively, the expected number of valid queries to cover the entire layer is given by:

$$\mathbb{E}(M_v \text{ to cover layer}) = N \sum_{i=1}^{N} \frac{1}{i}, \quad \text{which is } \Theta(N \log N).$$

Table 9.7 computes the expected numbers of queries required to cover the few first top layers of the SPHINCS$^+$ parameters sets, as obtaining all valid signatures is especially relevant when targeting such layers.

N	2^3	2^4	2^6	2^8	2^9	2^{12}	2^{16}
$\mathbb{E}(M_v)$	$2^{4.44}$	$2^{5.76}$	$2^{8.25}$	$2^{10.61}$	$2^{11.77}$	$2^{15.15}$	$2^{19.54}$

Table 9.7: Average numbers of valid signatures to cover various layers of N signatures.

9.3 Countermeasure analysis

In order to prevent faulty signatures from being collected, the W-OTS$^+$ signatures computed throughout a SPHINCS$^+$ signing procedure can be cached (i.e., stored in memory, sometimes temporarily, and then retransmitted without recomputation when requested). Such a process not only prevents accidental faulty recomputations of a W-OTS$^+$ signature, but also improves the performances of the SPHINCS$^+$ signature generation. Notice also that the valid W-OTS$^+$ signatures are leakage-agnostic, so the cache can therefore be shared with verifiers (in a read-only fashion).

In this section, we consider two different strategies of caching W-OTS$^+$s: caching layers and caching branches.

9.3.1 Caching layers

This strategy, originally proposed in Gravity-SPHINCS [AE17], consists of caching all the W-OTS$^+$ within one or more layers (starting from the top layer). Since the cache is not updated with new signature requests, the cache is static and can therefore be added to the public key.

Algorithms. Let c be the number of layers for which all W-OTS$^+$ signatures and public keys are cached. The countermeasure consists of replacing the key generation algorithm and the signing procedure algorithm of the hypertree and XMSS by the following:

- The new *key generation* procedure consists of discovering all the XMSSs from layers $d-1-c$ to $d-1$ and storing all the W-OTS$^+$ signatures and public keys on the way to the top XMSS. The secret and public keys are the same.

  This strategy increases the complexity of the key generation by a factor of $\sum_{i=0}^{c} 2^{h'i} = (2^{ch'+h'}-1)/(2^{h'}-1)$.

- The new *signing procedure* derives the XMSS signatures for the cached layers by using the W-OTS$^+$ signatures and public keys from the cache. An n-byte digest `msg` at hyperleaf index $1 \leq \lambda \leq 2^h$ is therefore signed as follows:

1. For $0 \leq i < d - c$:
   (a) Derive τ_i, λ_i from τ_{i-1} (starting with $\tau_{-1} = \lambda$).
   (b) Generate the XMSS key pair $(\text{SK}_i^X, \text{PK}_i^X)$ at the address corresponding to τ_i.
   (c) Sign r with SK^X using λ_i as leaf index to produce σ_i and update r with PK_i^X.
2. For $d - c \leq i < d$:
   (a) Derive τ_i, λ_i from τ_{i-1} (starting with $\tau_{-1} = \lambda$).
   (b) Read σ_i^W from the cache at tree index τ_i and leaf index λ_i.
   (c) Compute the XMSS authentication path auth_i starting from the leaf λ_i and using, as leaves, the cached W-OTS$^+$ public keys at tree index τ_i.
   (d) Let $\sigma_i = (\sigma_i^W, \text{auth}_i)$.
3. Return $\sigma^{HT} = (\sigma_0, \ldots, \sigma_{d-1})$.

This strategy saves a total of $c \times 2^{h'}(\ell W + 1)$ hash function calls in the signing procedure.

Analysis. While the algorithm prevents faulting c W-OTS$^+$ signatures, the new algorithm features a reduced total number of hash function calls which therefore impacts the proportion of vulnerable hash function calls, hence the chance that a random fault produces an exploitable faulty signature.

In a cached XMSS, the total number of hash function calls is: $\#\text{Total}^{\tilde{X}} = 2^{h'-1} - 1$. This leads to a new grand total of hash function calls in the SPHINCS$^+$ signing procedure:

$$\begin{aligned} \#\text{Total} &= 2 + \#\text{Total}^F + (d-c) \cdot \#\text{Total}^X + c \cdot \#\text{Total}^{\tilde{X}} \\ &= 3 + k(3t-1) + (d-c)(2^{h'}(\ell W + 2) - 1) + c(2^{h'} - 1). \end{aligned}$$

As a result, since a fault in an XMSS below a cached layer is not exploitable anymore, the proportion of hash function calls that lead to an exploitable faulty signature is:

$$\begin{aligned} \mathbb{P}(\text{Expl.}) &= \frac{\#\text{Total}^F + (d-c-1) \cdot \#\text{Total}^X}{\#\text{Total}} \\ &= \frac{1 + k(3t-1) + (d-c-1)(2^{h'}(\ell W + 2) - 1)}{3 + k(3t-1) + (d-c)(2^{h'}(\ell W + 2) - 1) + c(2^{h'} - 1)} \end{aligned}$$

where $0 < c < d$ ($\mathbb{P}(\text{Expl.}) = 0$ if $c = d$).

Table 9.8 shows how the probability that a single random fault is exploitable decreases with c for all SPHINCS$^+$ parameter sets. This table shows that the probability that a random fault gives an exploitable faulty signature stays fairly high, especially for the fast variants of SPHINCS$^+$.

In terms of memory, let C denote the total number of W-OTS$^+$ signatures cached. We therefore obtain:

$$C = \sum_{i=1}^{c} 2^{h'i} = 2^{h'}(2^{ch'} - 1)/(2^{h'} - 1).$$

		$\mathbb{P}$(Expl.)						
	$c=$	1	2	3	4	…	$d-1$	d
128s		0.8972	0.8591	0.8179	0.7733	…	0.6141	0.0000
128f		0.9505	0.9335	0.9158	0.8975	…	0.5076	0.0000
192s		0.9287	0.9034	0.8767	0.8486	…	0.7539	0.0000
192f		0.9420	0.9218	0.9007	0.8787	…	0.2625	0.0000
256s		0.8711	0.8216	0.7670	0.7066	…	0.4784	0.0000
256f		0.9327	0.9090	0.8840	0.8578	…	0.3864	0.0000

Table 9.8: Analysis of the layer caching countermeasure for all SPHINCS$^+$ parameter sets.

As a W-OTS$^+$ signature consists of ℓ elements of n bytes and since a W-OTS$^+$ public key consists of a single element of n bytes, caching c layers requires $C(\ell+1)n$ bytes in total. Table 9.9 shows how the cost of caching layers evolves with c for all SPHINCS$^+$ parameter sets. This table demonstrates that the memory requirements for this countermeasure blow up very early and that only the first few top layers can be cached in practice.

		Memory (bytes)					
	$c=$	1	2	3	4	…	d
128s		1.43×10^5	3.68×10^7	9.43×10^9	2.41×10^{12}	…	1.04×10^{22}
128f		4.48×10^3	4.03×10^4	3.27×10^5	2.62×10^6	…	7.38×10^{20}
192s		3.13×10^5	8.05×10^7	2.06×10^{10}	5.28×10^{12}	…	2.27×10^{22}
192f		9.79×10^3	8.81×10^4	7.15×10^5	5.73×10^6	…	1.03×10^{23}
256s		5.49×10^5	1.41×10^8	3.61×10^{10}	9.24×10^{12}	…	3.97×10^{22}
256f		3.43×10^4	5.83×10^5	9.36×10^6	1.50×10^8	…	6.75×10^{23}

Table 9.9: Analysis of the layer caching countermeasure for all SPHINCS$^+$ parameter sets.

9.3.2 Caching branches

This strategy consists of caching all the W-OTS$^+$ signatures and public keys in a path during a signing procedure. The cache is dynamic and may require to be updated for each new signature.

As reported in [Gen+18], this strategy completely prevents similar fault-based universal forgeries in stateful hash-based signature schemes (such as XMSS^{MT} [HRB13]). This is because the subtrees involved in stateful schemes provide only a limited number of signatures whose availability is remembered by the signer. Thus, once computed, the signature of a subtree can be retained as long as the subtree is involved in new signatures, at which point it is replaced by the next subtree in line. This prevents faulty recomputations of the signatures by caching only one W-OTS$^+$ per layer. This section shows that applying the same idea to SPHINCS$^+$ is ineffective, even when multiple W-OTS$^+$s per layer are cached.

Algorithms. The countermeasure consists of adding a cache of size C_l to each layer $0 \leq l < d$ of XMSSs, where $C_l \leq 2^{h'l}$ denotes the number of W-OTS$^+$ signatures and public keys that can be stored in the cache at layer l. The new XMSS signing procedure therefore signs an n-byte digest `msg` at leaf index $1 \leq \lambda \leq 2^{h'}$ as follows:

1. Check if the W-OTS$^+$ signature at leaf index λ is in the cache:
   - On *cache hit*, read the signature σ_λ^W and PK_λ^W from the cache.
   - On *cache miss*:
     (a) If the cache is full, evict the least recent signature.
     (b) Use SK_λ^X to produce PK_λ^W and σ_λ^W; the W-OTS$^+$ signature of `msg`.
     (c) Put σ_λ^W and PK_λ^W in the cache.
2. Compute the authentication path auth_λ starting from PK_λ^W (using cached W-OTS$^+$ public keys when accessible).
3. Return $\sigma^X = (\sigma_\lambda^W, \mathrm{auth}_\lambda)$.

When all caches are filled, this strategy saves an average of $\sum_{l=0}^{d-1} 2^{h'}(\ell W + 1)(C_l/2^{h-h'l})$ hash function calls in the XMSS signing procedure. We assume that all caches are empty at the device startup.

Analysis. As not all branches of the hypertree can realistically be cached, in order for the countermeasure to be effective, we suppose that we cache only a significant ratio of a layer. We furthermore focus on the significantly cached layer, as the layers above are necessarily all cached while the layers below are only marginally covered.

A faulty signature is exploitable if the fault hits a layer for which the corresponding W-OTS$^+$ signature is uncached. As the cache is dynamically filled, the probability of a cache miss depends on the number of distinct signatures visited after M queries to the signing procedure.

Let D_l denote the number of distinct visited W-OTS$^+$ signatures in layer $0 \leq l < d$ after M queries, and $N = 2^{h-h'l}$ the total number of W-OTS$^+$ signatures in such layer. Then, the distribution of D_l is an instance of the occupancy problem [Fel68]:

$$\mathbb{P}(D_l = i) = \frac{N!\alpha_{i,M}}{(N-i)!N^M}, \quad \text{where } \alpha_{i,M} = \frac{1}{i!}\sum_{k=1}^{i}(-1)^{i-k}\binom{i}{k}k^M \quad (1 \leq i \leq N).$$

Now, suppose that a total of $D_l \leq 2^{h-h'l}$ signatures are cached at each layer $0 \leq l < d$ (after a certain number M of queries). Then, as before, the probability that a fault leads to an exploitable faulty W-OTS$^+$ signature is derived by counting the number of vulnerable hash function calls in the procedure. However, in this case, the totals of hash function calls in all layers behave as random variables which depend on the cache status of each XMSS. So,

instead of deriving the exact totals, we evaluate the following heuristic:

$$\mathbb{P}(\text{Expl.}) = \frac{\mathbb{E}(\#\text{Expl.})}{\mathbb{E}(\#\text{Total})}$$

where $\mathbb{E}(\#\text{Expl.})$ denotes the average number of hash function calls that lead to an exploitable faulty signature when faulted, and $\mathbb{E}(\#\text{Total})$ the average total number of hash function calls in a SPHINCS$^+$ signing procedure.

Starting with the average total of hash function calls, notice that only the XMSS signing procedure was changed. Supposing that the cache is uniformly filled, we obtain:

$$\begin{aligned}
\mathbb{E}(\#\text{Total}) &= 2 + \#\text{Total}^F + \sum_{l=0}^{d-1} \mathbb{E}\left(\#\text{Total}^{\tilde{X}_l}\right) \\
&= 2 + \#\text{Total}^F + \sum_{l=0}^{d-1} \left(2^{h'} - 1 + \sum_{i=1}^{2^{h'}} \mathbb{P}(\text{Cache miss at layer } l)(\ell W + 1)\right) \\
&= 2 + \#\text{Total}^F + \sum_{l=0}^{d-1} \left(2^{h'} - 1 + 2^{h'}\left(1 - \frac{D_l}{2^{h-h'l}}\right)(\ell W + 1)\right).
\end{aligned}$$

The average total of vulnerable hash function calls is determined by the average total number of hash function calls in each layer of the SPHINCS$^+$ structure. At each layer, such a number now depends on the number of W-OTS$^+$ cached on the layer, as well as the number of W-OTS$^+$ cached on the layer above. Again, supposing that the cache is uniformly distributed, we obtain:

$$\begin{aligned}
\mathbb{E}(\#\text{Expl.}) &= \mathbb{E}\left(\#\text{Expl.}^{\tilde{F}}\right) + \sum_{l=0}^{d-2} \mathbb{E}\left(\#\text{Expl.}^{\tilde{X}}\right) \\
&= \mathbb{P}(\text{Cache miss at layer } 0) \cdot \#\text{Total}^F + \\
&\qquad \sum_{l=0}^{d-2} \mathbb{P}(\text{Cache miss at layer } l+1) \cdot \mathbb{E}\left(\#\text{Total}^{\tilde{X}}\right) \\
&= \left(1 - \frac{D_{2^h}}{2^h}\right)(3 + k(3t-1)) + \\
&\qquad \sum_{l=0}^{d-2} \left(1 - \frac{D_{l+1}}{2^{h-h'(l+1)}}\right)\left(2^{h'} - 1 + 2^{h'}\left(1 - \frac{D_l}{2^{h-h'l}}\right)(\ell W + 1)\right).
\end{aligned}$$

Table 9.10 shows how the probability that a random fault is exploitable decreases with b for all SPHINCS$^+$ parameter sets supposing that all the caches are filled to capacity (i.e., $D_l = \min(b, 2^{h-h'l})$, so after a sufficiently large number of queries M were made). As with caching layers, since the total number of hash function calls in the entire signing procedure decreases with the number of vulnerable hash function calls, the proportion of exploitable faulty signatures stays fairly high, especially for the fast variants of SPHINCS$^+$. Note however that such a countermeasure still leaves fewer hash function calls vulnerable than an unprotected SPHINCS$^+$.

				ℙ(Expl.)			
	$b=$	$(2/3)2^{h'}$	$(2/3)2^{2h'}$	$(2/3)2^{3h'}$	$(2/3)2^{4h'}$	…	$(2/3)2^{dh'}$
128s		0.9292	0.9238	0.9174	0.9098	…	0.3172
128f		0.9647	0.9634	0.9620	0.9605	…	0.3219
192s		0.9511	0.9485	0.9457	0.9425	…	0.3249
192f		0.9585	0.9568	0.9549	0.9528	…	0.3052
256s		0.9111	0.9023	0.8917	0.8785	…	0.3068
256f		0.9530	0.9507	0.9481	0.9453	…	0.3130

Table 9.10: Analysis of the branch caching countermeasure for all SPHINCS$^+$ parameter sets. The numbers b are rounded up to the next integer.

Since the universal forgery requires at least two recomputations of the same W-OTS$^+$ signature, we study the number of queries before a W-OTS$^+$ signature needs to be recomputed (i.e., two cache misses for a same W-OTS$^+$). We solve this problem with a Markov chain (see, e.g., [GS97] for a reference on the methodology) as shown in Figure 9.5. The corresponding transition matrix $\mathbf{P} = \left(p_{i,j}\right)$ is defined as follows (for $0 \le i, j \le N+2$):

$$p_{i,j} = \begin{cases} \min(i, C_l)/N & \text{if } j = i \neq N+1, \\ (N-i)/N & \text{if } j = i+1, \\ (i-C_l)/N & \text{if } j = N+1 \text{ and } i > C_l, \\ 1 & \text{if } j = i = N+1, \\ 0 & \text{otherwise.} \end{cases}$$

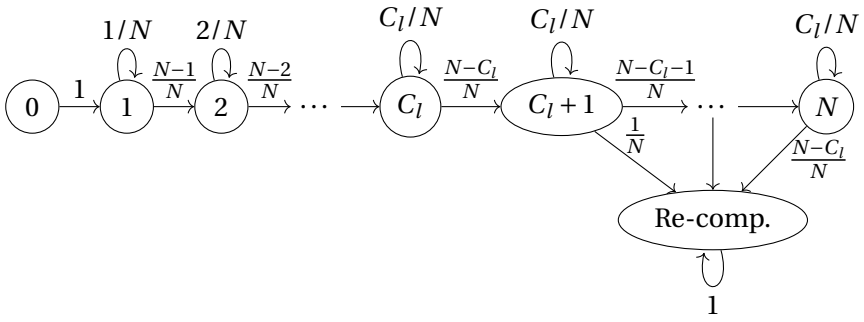


Figure 9.5: Markov chain representing the transitions from the cache being empty to any W-OTS$^+$ being recomputed. The states (others than "Recomp.") count the number of cache misses without recomputation.

The fundamental matrix that counts the average number of discrete steps spent in each state is computed as follows:

$$\mathbf{N} = (\mathbf{I} - \mathbf{Q})^{-1},$$

where $\mathbf{I}$ is the $(N+1) \times (N+1)$ identity matrix, and $\mathbf{Q}$ is the $(N+1) \times (N+1)$ submatrix of $\mathbf{P}$ without the last column and row. As we start with the cache being empty, the expected

number of queries M before a W-OTS$^+$ is recomputed is given by summing the first row of the fundamental matrix:

$$\mathbb{E}(M \text{ to recomp.}) = \sum_{j=0}^{N} \mathbf{N}_{0,j}.$$

Table 9.11 computes the expected numbers of queries required so that any W-OTS$^+$ gets recomputed for few first layers of all the SPHINCS$^+$ parameter sets, given various cache sizes. This table shows that the recomputation of a W-OTS$^+$ signature can be triggered with very few queries.

$N \backslash C_l$	$(1/2)N$	$(2/3)N$	$(3/4)N$	$N-1$
2^3	$2^{3.53}$	$2^{4.30}$	$2^{4.30}$	$2^{4.89}$
2^4	$2^{4.26}$	$2^{4.85}$	$2^{5.07}$	$2^{6.13}$
2^6	$2^{5.89}$	$2^{6.49}$	$2^{6.78}$	$2^{8.52}$
2^8	$2^{7.69}$	$2^{8.31}$	$2^{8.63}$	$2^{10.83}$
2^9	$2^{8.63}$	$2^{9.26}$	$2^{9.58}$	$2^{11.97}$

Table 9.11: Average number of queries such that a W-OTS$^+$ is recomputed for various cache sizes C_l and different layers of N signatures.

In terms of memory, let C denote the total number of W-OTS$^+$ signatures cached when b branches are fully cached. As $C_l \leq 2^{h'l}$, we have that:

$$C = \sum_{l=0}^{d-1} \min(b, 2^{h-h'l}).$$

As with caching layers, a W-OTS$^+$ signature consists of ℓ elements of n bytes and a W-OTS$^+$ public key consists of a single element of n bytes, so caching b layers requires $C(\ell+1)n$ bytes in total. Table 9.12 shows the cost of caching various numbers of branches for all SPHINCS$^+$ parameter sets. The memory requirements for this countermeasure blow up very early, so only the first few top layers are expected to be covered in practice.

		Memory (bytes)					
	$b =$	$(2/3)2^{h'}$	$(2/3)2^{2h'}$	$(2/3)2^{3h'}$	$(2/3)2^{4h'}$	…	$(2/3)2^{dh'}$
128s		8.14×10^5	1.82×10^8	4.00×10^{10}	8.53×10^{12}	…	7.36×10^{21}
128f		7.14×10^4	4.91×10^5	3.71×10^6	2.80×10^7	…	5.55×10^{20}
192s		1.74×10^6	3.90×10^8	8.56×10^{10}	1.83×10^{13}	…	1.58×10^{22}
192f		1.68×10^5	1.16×10^6	8.81×10^6	6.69×10^7	…	7.62×10^{22}
256s		3.02×10^6	6.77×10^8	1.49×10^{11}	3.17×10^{13}	…	2.74×10^{22}
256f		4.13×10^5	6.08×10^6	9.12×10^7	1.36×10^9	…	4.79×10^{23}

Table 9.12: Analysis of the branch caching countermeasure for all SPHINCS$^+$ parameter sets. The numbers b are rounded up to the next integer.

9.4 Experimental verifications

The following section aims to experimentally verify the fault attack as described in Section 9.1, the analysis of the fault attack from Section 9.2, and the analysis of the caching countermeasures from Section 9.3.

9.4.1 Setup

Hardware. As the fault attack does not require sophisticated glitching technology, our proof of concept uses the ChipWhisperer framework to perform experiments, as described in Section 2.2. In our analysis, the DUT (STM32F4) is configured to run at its maximal clock frequency (i.e., 180 MHz).

Software. We attack the reference implementation of SPHINCS$^+$ from [Flu+22] which was slightly adapted to run on the Cortex-M4 of the DUT. The instance attacked, which is claimed to achieve the maximal theoretical security guarantees, is `sphincs-shake-256s-robust`. The hash function SHAKE was instantiated with a portable software implementation.

For practicality purposes, the software was further modified to limit the signing procedure to the computation of a single layer. As a result, the software would use the W-OTS$^+$ keypair of an XMSS at a fixed layer $0 < l^* < d$ to sign the XMSS root at layer $l^* - 1$ addressed by a given index. The output signature consists of the W-OTS$^+$ signature along with the authentication path in the XMSS of layer $l^* - 1$.

The laptop communicates with the DUT through UART and the protocol is implemented using ChipWhisperer's simpleserial library. The DUT can be commanded to:

- Program the SPHINCS$^+$ secret and public seeds SK_1 and PK_2.
- Given an address, compute the W-OTS$^+$ signature and the authentication path of the XMSS at layer $l^* - 1$.
- Retrieve the bytes of the last W-OTS$^+$ signature and authentication path computed.

See https://www.github.com/AymericGenet/SPHINCSplus-FA for the source code.

Fault injection. To collect faulty signatures, the ChipWhisperer is used to inject a glitch in the system clock of the DUT. We do not synchronize the glitch injection with a trigger signal as we do not require to hit a precise instruction to collect exploitable signatures. Instead, the glitch is manually injected after a (progressive) delay that follows the communication with the DUT.

The glitch characteristics were explored experimentally to favor faulty signatures. Using a width of 20 samples and a clock offset of -4 samples, we report $\approx 1/3$ of output signatures to be faulty (so $\approx 2/3$ of valid outputs).

9.4.2 Experiment 1: randomized + cached layer

In the first experiment, we simulate the layer caching countermeasure (see Section 9.3) by pretending that all the W-OTS$^+$ signatures on the last layer are cached. In practice, such a cache would amount to 0.55 MB of ROM. The experiment therefore aims to show the feasibility of an attack on the second last layer (i.e., $l^* = d - 2 = 6$).

The experiment protocol to query a signature goes as follows:

1. The laptop sends to the DUT three bytes that correspond to the XMSS address at layer $l^* - 1$, i.e.:
   - τ_{l^*-1} = the first two bytes sent.
   - λ_{l^*-1} = the last byte sent.
2. The DUT computes:
   (a) The authentication path of the XMSS at layer $l^* - 1$ and tree index τ_{l^*-1}, starting from the leaf index λ_{l^*-1}.
   (b) The root r of the XMSS at layer $l^* - 1$ and tree index τ_{l^*-1}.
   (c) The W-OTS$^+$ signature of r, using the W-OTS$^+$ key pair from the XMSS at layer l^*, tree index τ_{l^*}, and leaf index λ_{l^*}, where:
      - τ_{l^*} = the first byte of τ_{l^*-1}.
      - λ_{l^*} = the last byte of τ_{l^*-1}.
3. The laptop then retrieves the W-OTS$^+$ signature and authentication path.

The DUT takes around 79 seconds to compute a single XMSS authentication path and W-OTS$^+$ signature, during which the clock glitch is blindly injected. We conduct $N = 5$ trials where a single trial consists of repeating the above with a fixed SPHINCS$^+$ secret seed to collect 1,024 potentially faulty signatures.

Results. The faulty signature collection is successful across all trials, as a W-OTS$^+$ is always found to be compromised at the end of the collection. Table 9.13 reports the types of signatures collected during the trials which were identified by recomputing the correct W-OTS$^+$ signature and authentication path from the programmed secret seed.

Table 9.14 reports the results related to the universal forgery. Given the analysis from Section 9.2 and using $M_f \approx (1/3)1024$ and $M_v \approx (2/3)1024$, we have that a W-OTS$^+$ signature is compromised with a probability of 0.5877 using only faulty signatures, and of 0.9714 using both valid and faulty signatures. The maximum load is expected to be $1.59 + 0.01$. On average, the probability that the grafting step is successful with two different W-OTS$^+$ signatures is $2^{-34.85}$. All these numbers correspond to the ones obtained in practice.

	Signatures		Faulty signatures		Non-verif. signatures	
	Valid	Faulty	Verif.	Non-verif.	Correct	Incorrect
Mean	660.4	363.6	269.6	94	1.8	92.2
SD	14.8762	14.8762	14.8257	11.2694	1.3038	10.1094
Min.	639	346	257	83	1	82
Max.	678	385	295	113	4	109

Table 9.13: Analysis of the collected signatures in $N = 5$ fault attacks against SPHINCS$^+$-shake-256s-robust at layer $l^* = 6$.

	Compromised W-OTS$^+$s	Maximum load	Best $\mathbb{P}$(grafting)
Mean	2.2	2	$2^{-35.7388}$
SD	1.7889	0	$2^{-35.5089}$
Min.	1	2	$2^{-47.0379}$
Max.	5	2	$2^{-34.2432}$

Table 9.14: Analysis of the universal forgery in $N = 5$ fault attacks against SPHINCS$^+$-shake-256s-robust at layer $l^* = 6$.

Conclusion. The experiment has demonstrated that despite the fact that the layer presents 2^{16} signatures, as few as 2^{10} signature queries with a fault probability of $\approx 1/3$ are enough to compromise at least one W-OTS$^+$ and, therefore, mount a SPHINCS$^+$ universal forgery.

9.4.3 Experiment 2: randomized + cached branches

In the second experiment, we simulate the branch caching countermeasure (see Section 9.3) by implementing an internal cache of C addresses for which we pretend that the corresponding W-OTS$^+$ are transmitted without recomputation. When requesting a W-OTS$^+$ at a certain address, the computation is triggered only if the given address was not previously cached.

The experiment aims to show that an attack is possible even when a significant portion of the layer is cached. For practicality purposes, we target the last layer (i.e., $l^* = d - 1 = 7$) and use a cache of size $C = 171$ to cover two thirds of the $2^{h'} = 256$ possible addresses. In theory, such a cache would amount to 2.93 MB of RAM.

At the beginning of the experiment, the DUT's cache is empty. The experiment protocol to query a signature goes as follows:

1. The laptop sends to the DUT two bytes that correspond to the XMSS address at layer $l^* - 1$, i.e.:
   - $\tau_{l^*-1} =$ the first byte sent.
   - $\lambda_{l^*-1} =$ the last byte sent.

2. If τ_{l^*-1} is in the DUT's cache, then the DUT computes nothing and the protocol stops here.
3. Else, if τ_{l^*-1} is not cached, then the DUT saves τ_{l^*-1} in the cache (after evicting the least recent address cached if the cache is full), and computes:
   (a) The authentication path of the XMSS at layer l^*-1 at tree index τ_{l^*-1}, starting from the leaf index λ_{l^*-1}.
   (b) The root r of the XMSS at layer l^*-1 and tree index τ_{l^*-1}.
   (c) The W-OTS$^+$ signature of r, using the W-OTS$^+$ key pair from the XMSS at layer l^*, tree index τ_{l^*}, and leaf index λ_{l^*}, where:
   - $\tau_{l^*} = 0$.
   - $\lambda_{l^*} = \tau_{l^*-1}$.
4. The laptop then retrieves the W-OTS$^+$ signature and authentication path.

On a cache miss, the DUT takes around 79 seconds to compute a single XMSS authentication path and W-OTS$^+$ signature, during which the clock glitch is blindly injected. The glitch is not injected on a cache hit. We conduct a total of $N = 10$ trials where a single trial consists of repeating the above with a fixed SPHINCS$^+$ secret seed to collect 512 potentially faulty signatures.

Results. The faulty signature collection is successful across all trials, as a W-OTS$^+$ is always found to be compromised at the end of the collection. Table 9.15 reports the types of signatures collected during the trials which were identified by recomputing the correct W-OTS$^+$ signature and authentication path using the programmed secret seed.

	Signatures		Faulty signatures		Non-verif. signatures	
	Valid	Faulty	Verif.	Non-verif.	Correct	Incorrect
Mean	419.3	92.7	76.4	16.3	0.2	16.1
SD	7.4841	7.4841	5.1251	4.7854	0.42	4.7714
Min.	409	81	67	9	0	9
Max.	431	103	84	25	1	25

Table 9.15: Analysis of the collected signatures in $N = 10$ fault attacks against SPHINCS$^+$-shake-256s-robust at layer $l^* = 7$ when 171 branches are cached.

Table 9.16 reports the results related to the universal forgery. Given the analysis from Section 9.3, the number of queries before a W-OTS$^+$ is recomputed is 318.09. Using a probability of successful fault injection of 1/3, a W-OTS$^+$ is successfully compromised upon recomputation with a probability of $1-(1-1/3)^2 = 0.5555$. This number corresponds to the ones obtained in practice.

	Queries before first recomp.	Compromised W-OTS$^+$s	Maximum load	Best $\mathbb{P}$(grafting)
Mean	318.6	14.1	2	$2^{-30.4274}$
SD	25.3693	3.7253	0	$2^{-30.3094}$
Min.	284	7	2	$2^{-35.7972}$
Max.	374	20	2	$2^{-28.8953}$

Table 9.16: Analysis of the universal forgery in $N = 10$ fault attacks at layer $l^* = 7$ against SPHINCS$^+$-shake-256s-robust when 171 branches are cached.

Conclusion. The experiment has demonstrated that despite the fact that two thirds of the attacked layer are cached, as few as 2^9 signature queries with a fault probability of $\approx 1/3$ are enough to compromise at least one W-OTS$^+$ and, thus, mount a SPHINCS$^+$ universal forgery.

9.5 Conclusion

In this chapter, a refined fault attack against SPHINCS$^+$ that is less restrictive than the original attack from Castelnovi, Martinelli, and Prest in [CMP18] has been presented. The complexity of the attack in terms of required queries, hashes, and success probability has also been scrupulously analyzed. Finally, the effectiveness of countermeasures based on caching both layers and branches has been shown to be underwhelming; a result which was experimentally verified.

The main takeaway of the current analysis is that SPHINCS$^+$ is extremely fragile against faults. As Section 9.2 shows, a *single* unconstrained corruption of *almost any* computation has a catastrophic impact on the security guarantees of *all* SPHINCS$^+$ parameters sets. This amounts to *millions* of hash function calls that need to be carried out faultlessly in order to sign a single message; a number that is not considering other subroutines (such as, e.g., the checksum in W-OTS$^+$) which are at least equally vulnerable.

While the other post-quantum signature algorithms selected by NIST in 2022 are also susceptible to fault attacks, this vulnerability makes SPHINCS$^+$ the most sensitive candidate to faults. For example, Bruinderink and Pessl have demonstrated in [BP18] that the lattice-based signature scheme CRYSTALS-Dilithium is also vulnerable to a universal forgery using an equivalent fault model. However, the attack on CRYSTALS-Dilithium can only be mounted when an adversary obtains the valid and faulty signatures of the same message, while SPHINCS$^+$ is vulnerable even when the device signs different and uncontrolled messages. Additionally, while the authors of [BP18] suggest that verifying signatures or randomization can serve as effective countermeasures against differential fault attacks on CRYSTALS-Dilithium, both of these approaches have been shown to be ineffective when applied to SPHINCS$^+$. The current attacks against FALCON—another lattice-based signature scheme chosen by NIST—only

work when these faults result in an early abort and zeroing of values, which requires a higher precision and more capabilities than the fault model considered in this paper (see [McC+19]).

Such a fragility needs to be taken seriously, as faults are reported to naturally happen in conventional hardware such as, e.g., in DRAM. For instance, Schroeder, Pinheiro, and Weber have reported 25,000 to 70,000 errors in DRAM per billion device hours per MBit in Google's 2009 fleet [SPW11]. As a result, with long enough deployments of SPHINCS$^+$ on standard computers, the fault attack is eventually going to affect real-world users.

As ordinary hardware cannot be fully trusted to protect against faults, and since faults can also be maliciously injected, a proper countermeasure that entirely prevents the fault attack is preferable. However, the problem is not obvious to solve, as the universal forgery exploits the fact that the signing procedure recomputes one-time signatures; a core feature of the SPHINCS family that makes the scheme practical and stateless. Yet, as long as one-time signatures are being recomputed on the fly, the risk of reusing a one-time key pair to sign an unexpected message will always be present (which, in practice, is accomplished with a fault injection). While this problem is solved in stateful schemes such as XMSS^{MT} by caching the relevant W-OTS$^+$s, Section 9.3 shows that the same countermeasure fails to properly protect SPHINCS$^+$.

Since the threat of a fault can never be completely eliminated, the current best solution to protect the signature scheme against accidental and intentional faults is through *redundancy*; an observation that is shared by others (see [CMP18; Ami+20]). Redundancy consists of recomputing a same signature multiple times (ideally, with different implementations (but the same random realizations)) and abort the procedure in case a mismatch in the signatures is detected. Even though parallelizable, this solution at least doubles the signing time which strikes a huge blow to the performance of the scheme which was already lacking in the original submission. Specially protected implementations on the hardware level, as recommended in the SPHINCS$^+$ specifications [Hül+20], may also offer an adequate protection against faults but would require fault-protection mechanisms not only in the hash function implementation, but also in the other subroutines of the scheme, as well as in the device memory.

In conclusion, the results of this paper urge all real-world deployments of SPHINCS$^+$ to come with redundancy checks, even if the use case is not prone to faults (such as, e.g., with firmware updates). Unless an adversary can query the signature for any message, randomized signing may be disabled as such measure is not a reliable way to prevent the fault attack. Verification, on the other hand, is still recommended as non-verifiability (even though unlikely) implies the occurrence of a fault.

Future work. The results of this chapter call for novel countermeasures that make SPHINCS$^+$ inherently resistant to fault attacks. As argued above, such a solution should avoid the accidental or intentional recomputation of one-time signatures which will likely necessitate a new way of performing hash-based signatures. For instance, an ambitious reader might come up with a solution that changes the one-time signatures in SPHINCS$^+$ by one-*message*

signatures which, if such a primitive makes sense, might even lead to an entirely new scheme. Other solutions that, for instance, make faulty signatures always non-verifiable would also be a desirable step forward, so a signing device could at least block bad signatures by running the verification procedure on the produced signatures.

Aside from researching countermeasures that make the scheme resistant to faults, investigating countermeasures that make the scheme *resilient* to faults could be of equal interest. A fault-resilient countermeasure does not prevent faulty signatures from being collected but from being exploited by hindering at least one step of the universal forgery. While preventing secret extraction or tree grafting would be difficult to achieve without significantly impacting the signing procedure performance (e.g., by replacing the one-time signatures by few-time signatures), a countermeasure that makes path seeking hard to find may reveal to be effective. Such a direction is left as an open problem.

At last, regarding the offensive side of the attack, as the current work is limited to faulting the hash functions, deriving similar attacks by faulting other subroutines of the scheme may lead to equally critical forgeries. Also, tampering with the control flow of a SPHINCS$^+$ software to force one-time signatures to sign unexpected messages would be an interesting direction to consider. Finally, differential fault attacks to recover secret values is yet another breach to explore.

10 Conclusion

In this thesis, we investigated the side-channel vulnerabilities of the cryptosystem SIKE, as well as the SPHINCS family. Specifically, we explored the susceptibility of SIKE to power analysis and identified a horizontal DPA and a clustering power analysis targeting the scalar multiplication involved in the scheme, in addition to a ZVP attack against the isogeny computation. All attacks were successfully executed on the official implementation of SIKE for the Cortex-M4 microcontroller. Additionally, we derived a vertical DPA of the PRNG in SPHINCS-256 which was experimentally verified on a custom Cortex-M3 implementation of the scheme, and analyzed the impact of a random fault injection against SPHINCS$^+$ regardless of the implementation.

Although SIKE is no longer considered secure as a result of the attack by Castryck and Decru in [CD22], our findings lay the groundwork for the development of future isogeny-based standards. In particular, our ZVP attack has made a significant contribution to the side-channel analysis of isogeny computations, as the attack has been shown to extend to the surviving isogeny-based key encapsulation mechanism CSIDH in [Cam+22]. Additionally, as NIST plans to standardize SPHINCS$^+$, our analyses of SPHINCS-256 and SPHINCS$^+$ have direct implications for real-world implementations of these schemes, particularly in embedded environments. Most notably, our fault analysis of SPHINCS$^+$ has highlighted the importance of implementing redundancy checks to prevent the derivation of faulty signatures, as generating any kind of faulty signatures has been shown to critically impact the security of the scheme.

While the attacks presented in this thesis exploit side-channel leakages to be executed, thus requiring special capabilities to be mounted in practice, side-channel attacks are widely acknowledged for posing existing threats to real-world devices. These attacks are especially relevant in the current stage of NIST's post-quantum standardization process, as the finalists are soon expected to be implemented for actual security purposes. Besides, we have demonstrated the applicability of our attacks to actual devices on an experimental setup based on the ChipWhisperer. Although our setup can be considered unrealistic in certain cases (due to, e.g., the addition of trigger signals, or software-induced delays), our verification represents a crucial initial step towards mounting actual attacks on real-world devices. As attacks only

get better, our findings will directly impact users of these cryptosystems, particularly when these schemes will transition to official standards, as is the case with SPHINCS$^+$. Also, note that even though certain attacks were tailored to exploit specific implementation details (such as in our clustering power analysis), others are applicable regardless of the implementation (such as the fault analysis), and our methodology can always be extended to similar targets.

While our work has provided a comprehensive exploration of side-channel analysis against isogeny-based and hash-based cryptosystems, there remain open questions which offer avenues for future research. Firstly, it would be interesting to explore whether the countermeasures proposed in our work can be defeated by new attacks. Procedures such as the elliptic curve scalar multiplication are challenging to fully protect, given the large surface for side-channel leakage that these operations offer. Consequently, new attacks continue to be discovered despite state-of-the-art countermeasures (see, e.g., the work by Perin et al. in [Per+21]), highlighting the constant relevance of their study. Secondly, it would be valuable to explore alternative attack vectors against secret isogeny derivations and examine the feasibility of input validation to prevent all forms of side-channel chosen-text attacks in isogeny-based schemes. Last but not least, the question of whether the SPHINCS family can ever be effectively protected against fault attacks remains unsolved. Considering the impact of a single fault on the security of the scheme, it would be preferable to revise the scheme to mitigate the potential of such attacks, making this direction an ideal focus for future research.

In the end, this thesis has explored a mere fraction of the vulnerabilities within the expansive realm of side-channel analysis against post-quantum cryptosystems, opening the door to countless future explorations. As the world becomes increasingly interconnected, the importance of security and privacy becomes progressively more apparent, which demonstrates the significance of research and development in cryptanalysis. As the landscape of cryptography continues to evolve to address new threats, such as the ones posed by quantum computers, unique vectors of attack emerge and require attention. This never-ending pursuit of knowledge is therefore an essential process to ensure the security of digital systems to protect individuals, organizations, and societies from emerging threats and, in consequence, to contribute to a better future for humanity.

A Cortex-M4 implementation of SIKE

The following appendix shows the attacked source code of the library by [Seo+20].

```
static void LADDER3PT(point_proj_t R0, point_proj_t R, point_proj_t R2,
    const digit_t* m, const f2elm_t A24)
{
    digit_t mask;
    int i, bit, swap, prevbit = 0;

    // Main loop
    for (i = 0; i < NBITS; i++) {
        bit = (m[i >> LOG2RADIX] >> (i & (RADIX-1))) & 1;
        swap = bit ^ prevbit;
        prevbit = bit;
        mask = 0 - (digit_t)swap;

        swap_points(R, R2, mask);
        xDBLADD(R0, R2, R->X, R->Z, A24);
    }
    swap = 0 ^ prevbit;
    mask = 0 - (digit_t)swap;
    swap_points(R, R2, mask);
}
```

Listing A.1: Source code of the LADDER3PT function (from [Seo+20]).

```
void xDBLADD(point_proj_t Q, point_proj_t P, point_proj_t QP, const f2elm_t A24) {
   f2elm_t t0, t1, t2;

   fp2add(Q->X, Q->Z, t0);
   fp2sub(Q->X, Q->Z, t1);
   fp2sqr_mont(t0, Q->X);
   fp2sub(P->X, P->Z, t2);
   fp2correction(t2);
   fp2add(P->X, P->Z, P->X);
   fp2mul_mont(t0, t2, t0);
```

```
    fp2sqr_mont(t1, Q->Z);
    fp2mul_mont(t1, P->X, t1);
    fp2sub(Q->X, Q->Z, t2);
    fp2mul_mont(Q->X, Q->Z, Q->X);
    fp2mul_mont(t2, A24, P->X);
    fp2sub(t0, t1, P->Z);
    fp2add(P->X, Q->Z, Q->Z);
    fp2add(t0, t1, P->X);
    fp2mul_mont(Q->Z, t2, Q->Z);
    fp2sqr_mont(P->Z, P->Z);
    fp2sqr_mont(P->X, P->X);
    fp2mul_mont(P->Z, QP->X, P->Z);
    fp2mul_mont(P->X, QP->Z, P->X);
}
```

Listing A.2: Source code of the xDBLADD function (from [Seo+20]).

```
void fp2mul_mont(const f2elm_t a, const f2elm_t b, f2elm_t c) {
    felm_t t1, t2;
    dfelm_t tt1, tt2, tt3;
    digit_t mask;
    unsigned int i;

    mp_addfast(a[0], a[1], t1);
    mp_addfast(b[0], b[1], t2);

    fpmul_mont(a[0], b[0], c[0]);
    fpmul_mont(a[1], b[1], tt2);
    fpmul_mont(t1, t2, c[1]);

    fpsub(c[1],c[0],c[1]);
    fpsub(c[1],tt2,c[1]);

    fpsub(c[0],tt2,c[0]);
}
```

Listing A.3: Source code of the fp2sqr_mont function (from [Seo+20]).

```
void fp2sqr_mont(const f2elm_t a, f2elm_t c) {
    felm_t t1, t2, t3;

    mp_addfast(a[0], a[1], t1);
    fpsub(a[0], a[1], t2);
    mp_addfast(a[0], a[0], t3);
    fpmul_mont(t1, t2, c[0]);
    fpmul_mont(t3, a[1], c[1]);
}
```

Listing A.4: Source code of the fp2mul_mont function (from [Seo+20]).

```
void __attribute__ ((noinline, naked))  mp_addfast(const digit_t* a, const digit_t* b,
  digit_t* c) {
  asm(
      "push  {r4-r9,lr}       \n\t"
      "mov r14, r2            \n\t"

      "ldmia r0!, {r2-r5}     \n\t"
      "ldmia r1!, {r6-r9}     \n\t"

      "adds r2, r2, r6        \n\t"
      "adcs r3, r3, r7        \n\t"
      "adcs r4, r4, r8        \n\t"
      "adcs r5, r5, r9        \n\t"

      "stmia r14!, {r2-r5}    \n\t"

      "ldmia r0!, {r2-r5}     \n\t"
      "ldmia r1!, {r6-r9}     \n\t"

      "adcs r2, r2, r6        \n\t"
      "adcs r3, r3, r7        \n\t"
      "adcs r4, r4, r8        \n\t"
      "adcs r5, r5, r9        \n\t"

      "stmia r14!, {r2-r5}    \n\t"

      "ldmia r0!, {r2-r5}     \n\t"
      "ldmia r1!, {r6-r9}     \n\t"

      "adcs r2, r2, r6        \n\t"
      "adcs r3, r3, r7        \n\t"
      "adcs r4, r4, r8        \n\t"
      "adcs r5, r5, r9        \n\t"

      "stmia r14!, {r2-r5}    \n\t"

      "ldmia r0!, {r2-r3}     \n\t"
      "ldmia r1!, {r6-r7}     \n\t"

      "adcs r2, r2, r6        \n\t"
      "adcs r3, r3, r7        \n\t"

      "stmia r14!, {r2-r3}    \n\t"

      "pop  {r4-r9,pc}        \n\t"
  : : :
  );
}
```

Listing A.5: Source code of the `mp_addfast` function (from [Seo+20]).

```
static void swap_points(point_proj_t P, point_proj_t Q, const digit_t mask) {
    digit_t temp;
    unsigned int i;

    for (i = 0; i < NWORDS_FIELD; i++) {
        temp = mask & (P->X[0][i] ^ Q->X[0][i]);
        P->X[0][i] = temp ^ P->X[0][i];
        Q->X[0][i] = temp ^ Q->X[0][i];
        temp = mask & (P->X[1][i] ^ Q->X[1][i]);
        P->X[1][i] = temp ^ P->X[1][i];
        Q->X[1][i] = temp ^ Q->X[1][i];
        temp = mask & (P->Z[0][i] ^ Q->Z[0][i]);
        P->Z[0][i] = temp ^ P->Z[0][i];
        Q->Z[0][i] = temp ^ Q->Z[0][i];
        temp = mask & (P->Z[1][i] ^ Q->Z[1][i]);
        P->Z[1][i] = temp ^ P->Z[1][i];
        Q->Z[1][i] = temp ^ Q->Z[1][i];
    }
}
```

Listing A.6: Source code of the `swap_points` function (from [Seo+20]).

```
rsb   r8, r6, #0    /* mask = (0 - swap) */
add.w r2, r4, #92   /* P->X  */
add.w r3, r4, #540  /* Q->X */
mov.w ip, #0        /* i = 0 */
<loop>:
ldr   r7, [r2, #0]
ldr   r1, [r3, #0]
eor.w r0, r7, r1    /* mask & (P->X[0][i] ^ Q->X[0][i]) */
and.w r0, r0, r8
eors  r7, r0        /* P->X[0][i] = temp ^ P->X[0][i] */
eors  r1, r0        /* Q->X[0][i] = temp ^ Q->X[0][i] */
str.w r7, [r2], #4
str.w r1, [r3], #4

...                 /* repeat above (with different offsets) */

add.w ip, ip, #1    /* i++ */
cmp.w ip, #14       /* i < NWORDS_FIELD */
bne.n <loop>
```

Listing A.7: Assembly instructions of the compiled `swap_points` function (i.e., Listing A.6).

```
and.w %[u1], %[u1], #0xFFFFFFFD  /* u1 = randombytes(4) & 0xFFFFFFFD */
and.w %[m1], %[u2], #0xFFFFFFFE  /* m1 = randombytes(4) & 0xFFFFFFFE */
add.w %[u2], %[u1], %[swap]      /* u2 = u1 + swap */
add.w %[m2], %[m1], %[swap]      /* r  = m1 + swap */
add.w %[u1], %[u1], #1           /* u1 = u1 + 1 */
mul.w %[u1], %[u1], %[m2]        /* u1 = u1*r */
add.w %[u2], %[u2], %[swap]      /* u2 = u2 + swap */
```

```
mul.w %[u2], %[u2], %[m2]        /* u2 = u2*r */
sub.w %[m2], %[u1], %[u2]        /* m2 = u1 - u2 */
```

Listing A.8: Assembly instructions of the secure masks generation.

```
ldr.w %[a], [%[R]]           /* a = R[i]  */
ldr.w %[b], [%[R2]]          /* b = R2[i] */
eor.w %[tmp1], %[a], %[b]    /* tmp1 = a ^ b */
and.w %[tmp1], %[m1]         /* tmp1 = tmp1 & m1 */
eor.w %[b], %[b], %[tmp1]    /* a = a ^ tmp1 */
eor.w %[a], %[a], %[tmp1]    /* b = b ^ tmp1 */
eor.w %[tmp2], %[a], %[b]    /* tmp2 = a ^ b */
str.w %[b], [%[R2]]          /* R2[i] = b */
and.w %[tmp2], %[m2]         /* tmp2 = tmp2 & m2 */
str.w %[a], [%[R]]           /* R[i]  = a */
eor.w %[b], %[b], %[tmp2]    /* b = b ^ tmp2 */
eor.w %[a], %[a], %[tmp2]    /* a = a ^ tmp2 */
str.w %[a], [%[R]], #4       /* R[i]  = a */
str.w %[b], [%[R2]], #4      /* R2[i] = b */

...                          /* repeat above */
```

Listing A.9: Assembly instructions of the secure swapping.

```
void __attribute__ ((naked)) fpmul_mont(const felm_t ma, const felm_t mb, felm_t mc)
{ // Multiprecision multiplication, c = a*b mod p.
    //dfelm_t temp = {0};
asm volatile(\
STRFY(P_MUL_PROLOG)
"SUB SP, #4*28          \n\t"

//ROUND#1
STRFY(P_LOAD2(R0, P_OP_A0, P_OP_A1, 12))
STRFY(P_LOAD(R1, P_OP_B0, P_OP_B1, P_OP_B2, P_OP_B3, 0))
STRFY(P_MUL_TOP(SP, 12))

//ROUND#2
STRFY(P_LOAD(R0, P_OP_A0, P_OP_A1, P_OP_A2, P_OP_A3, 8))
STRFY(P_MUL_FRONT(SP, 8))
"LDR R0, [SP, #4 * 29] \n\t"
STRFY(P_MUL_MID_OP_B_SHORT(SP, 12, R0, 4))
"LDR R0, [SP, #4 * 28] \n\t"
STRFY(P_MUL_MID_OP_A_SHORT(SP, 14, R0, 12))
STRFY(P_MUL_BACK2(SP, 16))

//ROUND#3
"LDR R0, [SP, #4 * 28] \n\t"// OP_A
"LDR R1, [SP, #4 * 29] \n\t"// OP_B
STRFY(P_LOAD(R0, P_OP_A0, P_OP_A1, P_OP_A2, P_OP_A3, 4))
STRFY(P_LOAD2(R1, P_OP_B0, P_OP_B1, 0))
STRFY(P_MUL_FRONT(SP, 4))
"LDR R0, [SP, #4 * 29] \n\t"
```

```
29 STRFY(P_MUL_MID_OP_B(SP, 8, R0, 4))
30 STRFY(P_MUL_MID_OP_B_SHORT(SP, 12, R0, 8))
31 "LDR R0, [SP, #4 * 28] \n\t"
32 STRFY(P_MUL_MID_OP_A_SHORT(SP, 14, R0, 8))
33 STRFY(P_MUL_MID_OP_A2(SP, 16, R0, 10))
34 STRFY(P_MUL_BACK2(SP, 20))
35
36 //ROUND#4
37 "LDR R0, [SP, #4 * 28] \n\t"// OP_A
38 "LDR R1, [SP, #4 * 29] \n\t"// OP_B
39 STRFY(P_LOAD(R0, P_OP_A0, P_OP_A1, P_OP_A2, P_OP_A3, 0))
40 STRFY(P_LOAD(R1, P_OP_B0, P_OP_B1, P_OP_B2, P_OP_B3, 0))
41 STRFY(P_MUL_FRONT(SP, 0))
42 "LDR R0, [SP, #4 * 29] \n\t"
43 STRFY(P_MUL_MID_OP_B(SP, 4, R0, 4))
44 STRFY(P_MUL_MID_OP_B(SP, 8, R0, 8))
45 STRFY(P_MUL_MID_OP_B_SHORT(SP, 12, R0, 12))
46 "LDR R0, [SP, #4 * 28] \n\t"
47 STRFY(P_MUL_MID_OP_A_SHORT(SP, 14, R0, 4))
48 STRFY(P_MUL_MID_OP_A2(SP, 16, R0, 6))
49 STRFY(P_MUL_MID_OP_A2(SP, 20, R0, 10))
50 STRFY(P_MUL_BACK2(SP, 24))
51
52 //TEST
53 "MOV R1, #0            \n\t"//CARRY
54 "ADDS R1, R1, R1    \n\t"
55 "LDR R0, [SP, #4 * 30] \n\t"//RESULT POINTER
56
57 //ROUND#1
58 STRFY(P_LOAD_M)
59 STRFY(P_LOAD_Q(SP, P_OP_Q0, P_OP_Q1, P_OP_Q2, P_OP_Q3, 0))
60 STRFY(P_RED_FRONT(SP, SP, 6, 6))
61 STRFY(P_RED_MID(SP, 10))
62
63 //ROUND#2
64 STRFY(P_LOAD_M)
65 STRFY(P_LOAD_Q(SP, P_OP_Q0, P_OP_Q1, P_OP_Q2, P_OP_Q3, 4))
66 STRFY(P_RED_FRONT(SP, SP, 10, 10))
67 STRFY(P_RED_MID(SP, 14))
68
69 //ROUND#3
70 STRFY(P_LOAD_M)
71 STRFY(P_LOAD_Q(SP, P_OP_Q0, P_OP_Q1, P_OP_Q2, P_OP_Q3, 8))
72 "LDR R0, [SP, #4 * 30] \n\t"// RESULT
73 STRFY(P_RED_FRONT(R0, SP, 0, 14))
74 //STRFY(RED_MID2(R0, SP, 4, 18))
75 STRFY(P_RED_MID(SP, 18))
76
77 //ROUND#4
78 STRFY(P_LOAD_M2)
79 STRFY(P_LOAD_Q2(SP, P_OP_Q0, P_OP_Q1, 12))
80 "LDR R0, [SP, #4 * 30] \n\t"// RESULT
```

```
81 STRFY(P_RED_FRONT2(R0, SP, 4, 18))
82 STRFY(P_RED_MID3(R0, SP, 6,  20))
83
84 "ADD SP, #4*31          \n\t"
85 STRFY(P_MUL_EPILOG)
86   :
87   :
88   : "cc", "memory"
89 );
90 }
```

Listing A.10: Source code of the `fpmul_mont` function (from [Seo+20]).

Bibliography

[Adj+22] Gora Adj, Jesús-Javier Chi-Domínguez, Víctor Mateu, and Francisco Rodríguez-Henríquez. "Faulty isogenies: a new kind of leakage". In: *CoRR* abs/2202.04896 (2022). arXiv: 2202.04896. URL: https://arxiv.org/abs/2202.04896.

[AE17] Jean-Phillippe Aumasson and Guillaume Endignoux. *Gravity-SPHINCS*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. National Institute of Standards and Technology, 2017.

[Agr+03] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. "The EM Side-Channel(s)". In: *Cryptographic Hardware and Embedded Systems – CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Redwood Shores, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 29–45. DOI: 10.1007/3-540-36400-5_4.

[Agu+22] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. *HQC*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[Ala+21] Monjur Alam, Baki Yilmaz, Frank Werner, Niels Samwel, Alenka 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: *6th IEEE European Symposium on Security and Privacy, EuroS&P 2021, September 6-10, 2021*. IEEE, 2021. URL: https://cs.adelaide.edu.au/~yval/pdfs/AlamYWSZGYP21.pdf.

[Alb+22] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. *Classic McEliece*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[Ami+20] Dorian Amiet, Lukas Leuenberger, Andreas Curiger, and Paul Zbinden. "FPGA-based SPHINCS$^+$ Implementations: Mind the Glitch". In: *23rd Euromicro Conference on Digital System Design, DSD 2020, Kranj, Slovenia, August 26-28, 2020*. IEEE, 2020, pp. 229–237. DOI: 10.1109/DSD51259.2020.00046. URL: https://doi.org/10.1109/DSD51259.2020.00046.

[Apo20] Daniel Apon. *Passing the final checkpoint! NIST PQC 3rd round begins*. Aug. 2020. URL: https://www.scribd.com/document/474476570/PQC-Overview-Aug-2020-NIST (visited on 09/01/2022).

[APS19] Melissa Azouaoui, Romain Poussier, and François-Xavier Standaert. "Fast Side-Channel Security Evaluation of ECC Implementations - Shortcut Formulas for Horizontal Side-Channel Attacks Against ECSM with the Montgomery Ladder". In: *COSADE 2019: 10th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Ilia Polian and Marc Stöttinger. Vol. 11421. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, Apr. 2019, pp. 25–42. DOI: 10.1007/978-3-030-16350-1_3.

[Ara+22] Nicolas Aragon, Paulo Barreto, Slim Bettaieb, Loic Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Phillipe Gaborit, Shay Gueron, Tim Guneysu, Carlos Aguilar Melchor, Rafael Misoczki, Edoardo Persichetti, Nicolas Sendrier, Jean-Pierre Tillich, Gilles Zémor, Valentin Vasseur, Santosh Ghosh, and Jan Richter-Brokmann. *BIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[Ard05] Arduino. *Arduino - Home*. 2005. URL: https://www.arduino.cc/ (visited on 09/01/2022).

[Ard12] Arduino. *Due | Arduino Documentations*. 2012. URL: https://docs.arduino.cc/hardware/due (visited on 09/01/2022).

[ARM10] ARM. *Cortex-M4 Specifications*. 2010. URL: https://developer.arm.com/Processors/Cortex-M4 (visited on 09/01/2022).

[AT03] Toru Akishita and Tsuyoshi Takagi. "Zero-Value Point Attacks on Elliptic Curve Cryptosystem". In: *ISC 2003: 6th International Conference on Information Security*. Ed. by Colin Boyd and Wenbo Mao. Vol. 2851. Lecture Notes in Computer Science. Bristol, UK: Springer, Heidelberg, Germany, Oct. 2003, pp. 218–233.

[Aum+14] Jean-Philippe Aumasson, Willi Meier, Raphael C.-W. Phan, and Luca Henzen. *The Hash Function BLAKE*. Information Security and Cryptography. Springer, Heidelberg, Germany, 2014. ISBN: 978-3-662-44757-4. DOI: 10.1007/978-3-662-44757-4.

[Aza+16] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. "Key Compression for Isogeny-Based Cryptosystems". In: *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography, AsiaPKC@AsiaCCS, Xi'an, China, May 30 - June 03, 2016*. Ed. by Keita Emura, Goichiro

Hanaoka, and Rui Zhang. ACM, 2016, pp. 1–10. DOI: 10.1145/2898420.2898421. URL: https://doi.org/10.1145/2898420.2898421.

[Azo+22] Melissa Azouaoui, Olivier Bronchain, Clément Hoffmann, Yulia Kuzovkova, Tobias Schneider, and François-Xavier Standaert. "Systematic Study of Decryption and Re-encryption Leakage: The Case of Kyber". In: *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 236–256. DOI: 10.1007/978-3-030-99766-3_11. URL: https://doi.org/10.1007/978-3-030-99766-3_11.

[Bac+22] Linus Backlund, Kalle Ngo, Joel Gärtner, and Elena Dubrova. *Secret Key Recovery Attacks on Masked and Shuffled Implementations of CRYSTALS-Kyber and Saber*. Cryptology ePrint Archive, Paper 2022/1692. https://eprint.iacr.org/2022/1692. 2022.

[Bar+12] Alessandro Barenghi, Guido Marco Bertoni, Luca Breveglieri, Mauro Pellicioli, and Gerardo Pelosi. "Injection Technologies for Fault Attacks on Microprocessors". In: *Fault Analysis in Cryptography*. Ed. by Marc Joye and Michael Tunstall. Information Security and Cryptography. Springer, 2012, pp. 275–293. DOI: 10.1007/978-3-642-29656-7_16. URL: https://doi.org/10.1007/978-3-642-29656-7_16.

[Bat+19] Lejla Batina, Lukasz Chmielewski, Louiza Papachristodoulou, Peter Schwabe, and Michael Tunstall. "Online template attacks". In: *Journal of Cryptographic Engineering* 9.1 (Apr. 2019), pp. 21–36. DOI: 10.1007/s13389-017-0171-8.

[Bat+23] Lejla Batina, Łukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. "SoK: SCA-secure ECC in software - mission impossible?" In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.1 (2023). https://tches.iacr.org/index.php/TCHES/article/view/9701, pp. 557–589. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i1.557-589.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation Power Analysis with a Leakage Model". In: *Cryptographic Hardware and Embedded Systems – CHES 2004*. Ed. by Marc Joye and Jean-Jacques Quisquater. Vol. 3156. Lecture Notes in Computer Science. Cambridge, Massachusetts, USA: Springer, Heidelberg, Germany, Aug. 2004, pp. 16–29. DOI: 10.1007/978-3-540-28632-5_2.

[BDH11] Johannes A. Buchmann, Erik Dahmen, and Andreas Hülsing. "XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions". In: *Post-Quantum Cryptography - 4th International Workshop, PQCrypto 2011*. Ed. by Bo-Yin Yang. Tapei, Taiwan: Springer, Heidelberg, Germany, Nov. 2011, pp. 117–129. DOI: 10.1007/978-3-642-25405-5_8.

[BDL97] 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*. Ed. by Walter Fumy. Vol. 1233. Lecture Notes in Computer Science. Konstanz, Germany: Springer, Heidelberg, Germany, May 1997, pp. 37–51. DOI: 10.1007/3-540-69053-0_4.

[Bee+22] Piyush Beegala, Debapriya Basu Roy, Prasanna Ravi, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. "Efficient Loop Abort Fault Attacks on Supersingular Isogeny based Key Exchange (SIKE)". In: *IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2022, Austin, TX, USA, October 19-21, 2022*. Ed. by Luca Cassano, Sreejit Chakravarty, and Alberto Bosio. IEEE, 2022, pp. 1–6. DOI: 10.1109/DFT56152.2022.9962359. URL: https://doi.org/10.1109/DFT56152.2022.9962359.

[Bel+21] Davide Bellizia, Nadia El Mrabet, Apostolos P. Fournaris, Simon Pontié, Francesco Regazzoni, François-Xavier Standaert, Élise Tasso, and Emanuele Valea. "Post-Quantum Cryptography: Challenges and Opportunities for Robust and Secure HW Design". In: *36th IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2021, Athens, Greece, October 6-8, 2021*. Ed. by Luigi Dilillo, Luca Cassano, and Athanasios Papadimitriou. IEEE, 2021, pp. 1–6. DOI: 10.1109/DFT52944.2021.9568301. URL: https://doi.org/10.1109/DFT52944.2021.9568301.

[Ber08a] Daniel J. Bernstein. "ChaCha, a variant of Salsa20". In: (2008). URL: https://cr.yp.to/chacha/chacha-20080120.pdf.

[Ber08b] Daniel J. Bernstein. *The ChaCha family of stream ciphers*. 2008. URL: https://cr.yp.to/chacha.html (visited on 09/01/2022).

[Ber+15] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. "SPHINCS: Practical Stateless Hash-Based Signatures". In: *Advances in Cryptology – EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. Lecture Notes in Computer Science. Sofia, Bulgaria: Springer, Heidelberg, Germany, Apr. 2015, pp. 368–397. DOI: 10.1007/978-3-662-46800-5_15.

[BH17] Leon Groot Bruinderink and Andreas Hülsing. ""Oops, I Did It Again" - Security of One-Time Signatures Under Two-Message Attacks". In: *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. Lecture Notes in Computer Science. Ottawa, ON, Canada: Springer, Heidelberg, Germany, Aug. 2017, pp. 299–322. DOI: 10.1007/978-3-319-72565-9_15.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. "CSI-FiSh: Efficient Isogeny Based Signatures Through Class Group Computations". In: *Advances in Cryptology – ASIACRYPT 2019, Part I*. Ed. by Steven D. Galbraith and Shiho Moriai.

Vol. 11921. Lecture Notes in Computer Science. Kobe, Japan: Springer, Heidelberg, Germany, Dec. 2019, pp. 227–247. DOI: 10.1007/978-3-030-34578-5_9.

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. "Efficient Template Attacks Based on Probabilistic Multi-class Support Vector Machines". In: *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*. Ed. by Stefan Mangard. Vol. 7771. Lecture Notes in Computer Science. Springer, 2012, pp. 263–276. DOI: 10.1007/978-3-642-37288-9_18. URL: https://doi.org/10.1007/978-3-642-37288-9_18.

[BMR21] Luk Bettale, Simon Montoya, and Guénaël Renault. "Safe-Error Analysis of Post-Quantum Cryptography Mechanisms - Short Paper-". In: *18th Workshop on Fault Detection and Tolerance in Cryptography, FDTC 2021, Milan, Italy, September 17, 2021*. IEEE, 2021, pp. 39–44. DOI: 10.1109/FDTC53659.2021.00015. URL: https://doi.org/10.1109/FDTC53659.2021.00015.

[Boh+03] Lilian Bohy, Michael Neve, David Samyde, and Jean-Jacques Quisquater. "Principal and Independent Component Analysis for Crypto-systems with Hardware Unmasked Units". In: *Proceedings of e-Smart 2003*. 2003.

[Bos+19] Joppe W. Bos, Simon Friedberger, Marco Martinoli, Elisabeth Oswald, and Martijn Stam. "Assessing the Feasibility of Single Trace Power Analysis of Frodo". In: *SAC 2018: 25th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlos Cid and Michael J. Jacobson Jr: vol. 11349. Lecture Notes in Computer Science. Calgary, AB, Canada: Springer, Heidelberg, Germany, Aug. 2019, pp. 216–234. DOI: 10.1007/978-3-030-10970-7_10.

[BP18] Leon Groot Bruinderink and Peter Pessl. "Differential Fault Attacks on Deterministic Lattice Signatures". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2018.3 (2018). https://tches.iacr.org/index.php/TCHES/article/view/7267, pp. 21–43. ISSN: 2569-2925. DOI: 10.13154/tches.v2018.i3.21-43.

[Buc+06] Johannes Buchmann, Luis Carlos Coronado García, Erik Dahmen, Martin Döring, and Elena Klintsevich. "CMSS - An Improved Merkle Signature Scheme". In: *Progress in Cryptology - INDOCRYPT 2006: 7th International Conference in Cryptology in India*. Ed. by Rana Barua and Tanja Lange. Vol. 4329. Lecture Notes in Computer Science. Kolkata, India: Springer, Heidelberg, Germany, Dec. 2006, pp. 349–363.

[Cam+22] Fabio Campos, Michael Meyer, Krijn Reijnders, and Marc Stöttinger. *Patient Zero and Patient Six: Zero-Value and Correlation Attacks on CSIDH and SIKE*. Cryptology ePrint Archive, Report 2022/904. https://eprint.iacr.org/2022/904. 2022.

[Cas+18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. "CSIDH: An Efficient Post-Quantum Commutative Group Action". In: *Advances in Cryptology – ASIACRYPT 2018, Part III*. Ed. by Thomas Peyrin and Steven Galbraith. Vol. 11274. Lecture Notes in Computer Science. Brisbane, Queensland, Australia: Springer, Heidelberg, Germany, Dec. 2018, pp. 395–427. DOI: 10.1007/978-3-030-03332-3_15.

[Cay+21] Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Dragoi, Alexandre Menu, and Lilian Bossuet. "Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem". In: *Advances in Cryptology – EUROCRYPT 2021, Part II*. Ed. by Anne Canteaut and François-Xavier Standaert. Vol. 12697. Lecture Notes in Computer Science. Zagreb, Croatia: Springer, Heidelberg, Germany, Oct. 2021, pp. 438–467. DOI: 10.1007/978-3-030-77886-6_15.

[CCJ03] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye. *Low-Cost Solutions for Preventing Simple Side-Channel Analysis: Side-Channel Atomicity*. Cryptology ePrint Archive, Report 2003/237. https://eprint.iacr.org/2003/237. 2003.

[CD22] Wouter Castryck and Thomas Decru. *An efficient key recovery attack on SIDH (preliminary version)*. Cryptology ePrint Archive, Report 2022/975. https://eprint.iacr.org/2022/975. 2022.

[CGD17] Yann Le Corre, Johann Großschädl, and Daniel Dinu. *Micro-Architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors*. Cryptology ePrint Archive, Report 2017/1253. https://eprint.iacr.org/2017/1253. 2017.

[CGD18] Yann Le Corre, Johann Großschädl, and Daniel Dinu. "Micro-architectural Power Simulator for Leakage Assessment of Cryptographic Software on ARM Cortex-M3 Processors". In: *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Apr. 2018, pp. 82–98. DOI: 10.1007/978-3-319-89641-0_5.

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. "Constructing elliptic curve isogenies in quantum subexponential time". In: *Journal of Mathematical Cryptology* 8.1 (2014), pp. 1–29.

[CK20] Leonardo Colò and David Kohel. *Orienting supersingular isogeny graphs*. Cryptology ePrint Archive, Report 2020/985. https://eprint.iacr.org/2020/985. 2020.

[CKM21] Fabio Campos, Juliane Krämer, and Marcel Müller. "Safe-Error Attacks on SIKE and CSIDH". In: *Security, Privacy, and Applied Cryptography Engineering - 11th International Conference, SPACE 2021, Kolkata, India, December 10-13, 2021, Proceedings*. Ed. by Lejla Batina, Stjepan Picek, and Mainack Mondal. Vol. 13162. Lecture Notes in Computer Science. Springer, 2021, pp. 104–125. DOI: 10.1007/978-3-030-95085-9_6. URL: https://doi.org/10.1007/978-3-030-95085-9_6.

[Cla+10] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylene Roussellet, and Vincent Verneuil. *Horizontal Correlation Analysis on Exponentiation*. Cryptology ePrint Archive, Report 2010/394. https://eprint.iacr.org/2010/394. 2010.

[CLG09] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. "Cryptographic Hash Functions from Expander Graphs". In: *J. Cryptol.* 22.1 (2009), pp. 93–113. DOI: 10.1007/s00145-007-9002-x. URL: https://doi.org/10.1007/s00145-007-9002-x.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. "Efficient Algorithms for Supersingular Isogeny Diffie-Hellman". In: *Advances in Cryptology – CRYPTO 2016, Part I*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9814. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2016, pp. 572–601. DOI: 10.1007/978-3-662-53018-4_21.

[CMP18] Laurent Castelnovi, Ange Martinelli, and Thomas Prest. "Grafting Trees: A Fault Attack Against the SPHINCS Framework". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Fort Lauderdale, Florida, United States: Springer, Heidelberg, Germany, Apr. 2018, pp. 165–184. DOI: 10.1007/978-3-319-79063-3_8.

[Coh02] Paul Moritz Cohn. *Basic algebra: groups, rings and fields*. Springer Science & Business Media, 2002.

[Col+22] Brice Colombier, Vlad-Florin Drăgoi, Pierre-Louis Cayrel, and Vincent Grosso. "Profiled Side-Channel Attack on Cryptosystems Based on the Binary Syndrome Decoding Problem". In: *IEEE Transactions on Information Forensics and Security* 17 (2022), pp. 3407–3420. DOI: 10.1109/TIFS.2022.3198277.

[Col+23] Brice Colombier, Vincent Grosso, Pierre-Louis Cayrel, and Vlad-Florin Drăgoi. *Horizontal Correlation Attack on Classic McEliece*. Cryptology ePrint Archive, Paper 2023/546. https://eprint.iacr.org/2023/546. 2023. URL: https://eprint.iacr.org/2023/546.

[Cor11] Charles J. Corrado. "The exact distribution of the maximum, minimum and the range of Multinomial/Dirichlet and Multivariate Hypergeometric frequencies". In: *Stat. Comput.* 21.3 (2011), pp. 349–359. DOI: 10.1007/s11222-010-9174-3. URL: https://doi.org/10.1007/s11222-010-9174-3.

[Cor99] Jean-Sébastien Coron. "Resistance against Differential Power Analysis for Elliptic Curve Cryptosystems". In: *Cryptographic Hardware and Embedded Systems – CHES'99*. Ed. by Çetin Kaya Koç and Christof Paar. Vol. 1717. Lecture Notes in Computer Science. Worcester, Massachusetts, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 292–302. DOI: 10.1007/3-540-48059-5_25.

[Cos+17] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. "Efficient Compression of SIDH Public Keys". In: *Advances in Cryptology – EUROCRYPT 2017, Part I*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10210. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Apr. 2017, pp. 679–706. DOI: 10.1007/978-3-319-56620-7_24.

[Cos19] Craig Costello. "Supersingular Isogeny Key Exchange for Beginners". In: *SAC 2019: 26th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. Lecture Notes in Computer Science. Waterloo, ON, Canada: Springer, Heidelberg, Germany, Aug. 2019, pp. 21–50. DOI: 10.1007/978-3-030-38471-5_2.

[Cou06] Jean-Marc Couveignes. *Hard Homogeneous Spaces*. Cryptology ePrint Archive, Report 2006/291. https://eprint.iacr.org/2006/291. 2006.

[CP05] Xavier Charvet and Hervé Pelletier. "Improving the DPA attack using Wavelet transform". In: *NIST Physical Security Testing Workshop* 46 (2005).

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. "Template Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2002*. Ed. by Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar. Vol. 2523. Lecture Notes in Computer Science. Redwood Shores, CA, USA: Springer, Heidelberg, Germany, Aug. 2003, pp. 13–28. DOI: 10.1007/3-540-36400-5_3.

[CS18] Craig Costello and Benjamin Smith. "Montgomery curves and their arithmetic - The case of large characteristic fields". In: *Journal of Cryptographic Engineering* 8.3 (Sept. 2018), pp. 227–240. DOI: 10.1007/s13389-017-0157-6.

[D'A+19] Jan-Pieter D'Anvers, Marcel Tiepelt, Frederik Vercauteren, and Ingrid Verbauwhede. "Timing Attacks on Error Correcting Codes in Post-Quantum Schemes". In: *Proceedings of ACM Workshop on Theory of Implementation Security, TIS@CCS 2019, London, UK, November 11, 2019*. Ed. by Begül Bilgin, Svetla Petkova-Nikova, and Vincent Rijmen. ACM, 2019, pp. 2–9. DOI: 10.1145/3338467.3358948. URL: https://doi.org/10.1145/3338467.3358948.

[Dan+16] Jean-Luc Danger, Sylvain Guilley, Philippe Hoogvorst, Cédric Murdica, and David Naccache. "Improving the Big Mac Attack on Elliptic Curve Cryptography". In: *The New Codebreakers: Essays Dedicated to David Kahn on the Occasion of His 85th Birthday*. Ed. by Peter Y. A. Ryan, David Naccache, and Jean-Jacques Quisquater. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 374–386. ISBN: 978-3-662-49301-4. DOI: 10.1007/978-3-662-49301-4_23. URL: https://doi.org/10.1007/978-3-662-49301-4_23.

[DG19] Luca De Feo and Steven D. Galbraith. "SeaSign: Compact Isogeny Signatures from Class Group Actions". In: *Advances in Cryptology – EUROCRYPT 2019, Part III*. Ed. by Yuval Ishai and Vincent Rijmen. Vol. 11478. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, May 2019, pp. 759–789. DOI: 10.1007/978-3-030-17659-4_26.

[Din+17] 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. URL: https://doi.org/10.1007/978-3-319-75208-2_7.

[DJJ10] He Debiao, Chen Jianhua, and Hu Jin. *A Random Number Generator Based on Isogenies Operations*. Cryptology ePrint Archive, Report 2010/094. https://eprint.iacr.org/2010/094. 2010.

[DJP11] Luca De Feo, David Jao, and Jérôme Plût. *Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies*. Cryptology ePrint Archive, Report 2011/506. https://eprint.iacr.org/2011/506. 2011.

[DJP14] Luca De Feo, David Jao, and Jérôme Plût. "Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies". In: *J. Math. Cryptol.* 8.3 (2014), pp. 209–247. DOI: 10.1515/jmc-2012-0015. URL: https://doi.org/10.1515/jmc-2012-0015.

[DNG22] Elena Dubrova, Kalle Ngo, and Joel Gärtner. *Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste*. Cryptology ePrint Archive, Paper 2022/1713. https://eprint.iacr.org/2022/1713. 2022.

[DPV19] Thomas Decru, Lorenz Panny, and Frederik Vercauteren. "Faster SeaSign Signatures Through Improved Rejection Sampling". In: *Post-Quantum Cryptography - 10th International Conference, PQCrypto 2019*. Ed. by Jintai Ding and Rainer Steinwandt. Chongqing, China: Springer, Heidelberg, Germany, May 2019, pp. 271–285. DOI: 10.1007/978-3-030-25510-7_15.

[Dug+16] Margaux Dugardin, Louiza Papachristodoulou, Zakaria Najm, Lejla Batina, Jean-Luc Danger, and Sylvain Guilley. "Dismantling Real-World ECC with Horizontal and Vertical Template Attacks". In: *COSADE 2016: 7th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by François-Xavier Standaert and Elisabeth Oswald. Vol. 9689. Lecture Notes in Computer Science. Graz, Austria: Springer, Heidelberg, Germany, Apr. 2016, pp. 88–108. DOI: 10.1007/978-3-319-43283-0_6.

[Dun73] Joseph C. Dunn. "A Fuzzy Relative of the ISODATA Process and Its Use in Detecting Compact Well-Separated Clusters". In: *Journal of Cybernetics* 3.3 (1973), pp. 32–57. DOI: 10.1080/01969727308546046. eprint: https://doi.org/10.1080/01969727308546046. URL: https://doi.org/10.1080/01969727308546046.

[EMY14] Thomas Eisenbarth, Ingo von Maurich, and Xin Ye. "Faster Hash-Based Signatures with Bounded Leakage". In: *SAC 2013: 20th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Tanja Lange, Kristin Lauter, and Petr Lisonek. Vol. 8282. Lecture Notes in Computer Science. Burnaby, BC, Canada: Springer, Heidelberg, Germany, Aug. 2014, pp. 223–243. DOI: 10.1007/978-3-662-43414-7_12.

[Fan+10] 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. URL: https://doi.org/10.1109/HST.2010.5513110.

[Fel68] William Feller. *An introduction to probability theory and its applications*. 3rd. 1968.

[Feo17] Luca De Feo. "Mathematics of Isogeny Based Cryptography". In: *CoRR* abs/1711.04062 (2017). arXiv: 1711.04062. URL: http://arxiv.org/abs/1711.04062.

[Feo+20] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. "SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies". In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 2020, pp. 64–93. DOI: 10.1007/978-3-030-64837-4_3.

[Feo+22] Luca De Feo, Nadia El Mrabet, Aymeric Genêt, Novak Kaluđerović, Natacha Linard de Guertechin, Simon Pontié, and Élise Tasso. "SIKE Channels". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.3 (2022). https://tches.iacr.org/index.php/TCHES/article/view/9701, pp. 264–289. ISSN: 2569-2925. DOI: 10.46586/tches.v2022.i3.264-289.

[FGT92] Philippe Flajolet, Danièle Gardy, and Loÿs Thimonier. "Birthday Paradox, Coupon Collectors, Caching Algorithms and Self-Organizing Search". In: *Discret. Appl. Math.* 39.3 (1992), pp. 207–229. DOI: 10.1016/0166-218X(92)90177-C. URL: https://doi.org/10.1016/0166-218X(92)90177-C.

[Flu+22] Scott Fluhrer, Stefan Kölbl, Ruben Niederhagen, Joost Rijnevald, Peter Schwabe, Bas Westerbaan, and Thom Wiggers. *The SPHINCS+ reference code, accompanying the submission to NIST's Post-Quantum Cryptography project*. 2022. URL: https://github.com/sphincs/sphincsplus.git.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. "Secure Integration of Asymmetric and Symmetric Encryption Schemes". In: *Advances in Cryptology – CRYPTO'99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 537–554. DOI: 10.1007/3-540-48405-1_34.

[FV03] Pierre-Alain Fouque and Frédéric Valette. "The Doubling Attack - Why Upwards Is Better than Downwards". In: *Cryptographic Hardware and Embedded Systems – CHES 2003*. Ed. by Colin D. Walter, Çetin Kaya Koç, and Christof Paar. Vol. 2779. Lecture Notes in Computer Science. Cologne, Germany: Springer, Heidelberg, Germany, Sept. 2003, pp. 269–280. DOI: 10.1007/978-3-540-45238-6_22.

[Gal+16] Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. "On the Security of Supersingular Isogeny Cryptosystems". In: *Advances in Cryptology – ASIACRYPT 2016, Part I*. Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. Lecture Notes in Computer Science. Hanoi, Vietnam: Springer, Heidelberg, Germany, Dec. 2016, pp. 63–91. DOI: 10.1007/978-3-662-53887-6_3.

[GE21] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (Apr. 2021), p. 433. DOI: 10.22331/q-2021-04-15-433. URL: https://doi.org/10.22331%2Fq-2021-04-15-433.

[Gen17] Aymeric Genêt. "Hardware Attacks against Hash-based Cryptographic Algorithms". MA thesis. EPFL, 2017.

[Gen+18] Aymeric Genêt, Matthias J. Kannwischer, Hervé Pelletier, and Andrew McLauchlan. "Practical Fault Injection Attacks on SPHINCS". In: *Kangacrypt 2018, Australian Workshop on Offensive Cryptography* (2018).

[Gen23] Aymeric Genêt. "On Protecting SPHINCS+ Against Fault Attacks". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.2 (2023). https://tches.iacr.org/index.php/TCHES/article/view/10278, pp. 80–114. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i2.80-114.

[GGK21] Aymeric Genêt, Natacha Linard de Guertechin, and Novak Kaluđerović. "Full Key Recovery Side-Channel Attack Against Ephemeral SIKE on the Cortex-M4". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 228–254. URL: https://doi.org/10.1007/978-3-030-89915-8_11.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. "How to Construct Random Functions (Extended Abstract)". In: *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*. IEEE Computer Society, 1984, pp. 464–479. DOI: 10.1109/SFCS.1984.715949. URL: https://doi.org/10.1109/SFCS.1984.715949.

[GJ20] Qian Guo and Thomas Johansson. "A New Decryption Failure Attack Against HQC". In: *Advances in Cryptology – ASIACRYPT 2020, Part I*. Ed. by Shiho Moriai and Huaxiong Wang. Vol. 12491. Lecture Notes in Computer Science. Daejeon, South Korea: Springer, Heidelberg, Germany, Dec. 2020, pp. 353–382. DOI: 10.1007/978-3-030-64837-4_12.

[GJJ22] Qian Guo, Andreas Johansson, and Thomas Johansson. "A Key-Recovery Side-Channel Attack on Classic McEliece Implementations". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.4 (2022), pp. 800–827. DOI: 10.46586/tches.v2022.i4.800-827.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. "A Key-Recovery Timing Attack on Post-quantum Primitives Using the Fujisaki-Okamoto Transformation and Its Application on FrodoKEM". In: *Advances in Cryptology – CRYPTO 2020, Part II*. Ed. by Daniele Micciancio and Thomas Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2020, pp. 359–386. DOI: 10.1007/978-3-030-56880-1_13.

[GK22] Aymeric Genêt and Novak Kaluđerović. "Single-Trace Clustering Power Analysis of the Point-Swapping Procedure in the Three Point Ladder of Cortex-M4 SIKE". In: *Constructive Side-Channel Analysis and Secure Design - 13th International Workshop, COSADE 2022, Leuven, Belgium, April 11-12, 2022, Proceedings*. Ed. by Josep Balasch and Colin O'Flynn. Vol. 13211. Lecture Notes in Computer Science. Springer, 2022, pp. 164–192. DOI: 10.1007/978-3-030-99766-3_8. URL: https://doi.org/10.1007/978-3-030-99766-3_8.

[GLG22a] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. "A New Key Recovery Side-Channel Attack on HQC with Chosen Ciphertext". In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*. Ed. by Jung Hee Cheon and Thomas Johansson. Vol. 13512. Lecture Notes in Computer Science. Springer, 2022, pp. 353–371. DOI: 10.1007/978-3-031-17234-2_17. URL: https://doi.org/10.1007/978-3-031-17234-2_17.

[GLG22b] Guillaume Goy, Antoine Loiseau, and Philippe Gaborit. *Estimating the Strength of Horizontal Correlation Attacks in the Hamming Weight Leakage Model: A Side-Channel Analysis on HQC KEM*. https://www.wcc2022.uni-rostock.de/storages/uni-rostock/Tagungen/WCC2022/Papers/WCC_2022_paper_48.pdf. 2022.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*. Vol. 1. Cambridge, UK: Cambridge University Press, 2001, pp. xix + 372. ISBN: 0-521-79172-3 (hardback).

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*. Vol. 2. Cambridge, UK: Cambridge University Press, 2004. ISBN: ISBN 0-521-83084-2 (hardback).

[Gor+22] Alagic Gorjan, Apon Daniel, Cooper David, Dang Quynh, Dang Thinh, Kelsey John, Lichtinger Jacob, Miller Carl, Moody Dustin, Peralta Rene, Perlner Ray, Robinson Angela, and Smith-Tone Daniel. "Status report on the third round of the NIST post-quantum cryptography standardization process". In: *US Department of Commerce, NIST* (2022). DOI: 0.6028/NIST.IR.8413. URL: https://doi.org/10.6028/NIST.IR.8413.

[Gou01] Louis Goubin. "A Sound Method for Switching between Boolean and Arithmetic Masking". In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2001, pp. 3–15. DOI: 10.1007/3-540-44709-1_2.

[Gou03] Louis Goubin. “A Refined Power-Analysis Attack on Elliptic Curve Cryptosystems”. In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003, pp. 199–210. DOI: 10.1007/3-540-36288-6_15.

[GS97] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.

[HCH11] Debiao He, Jianhua Chen, and Jin Hu. “An authenticated key agreement protocol using isogenies between elliptic curves”. In: *International Journal of Computers Communications & Control* 6.2 (2011), pp. 258–265.

[Hey+13] Johann Heyszl, Andreas Ibing, Stefan Mangard, Fabrizio De Santis, and Georg Sigl. “Clustering Algorithms for Non-profiled Single-Execution Attacks on Exponentiations”. In: *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*. Ed. by Aurélien Francillon and Pankaj Rohatgi. Vol. 8419. Lecture Notes in Computer Science. Springer, 2013, pp. 79–93. URL: https://doi.org/10.1007/978-3-319-08302-5_6.

[HHK17] Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. “A Modular Analysis of the Fujisaki-Okamoto Transformation”. In: *TCC 2017: 15th Theory of Cryptography Conference, Part I*. Ed. by Yael Kalai and Leonid Reyzin. Vol. 10677. Lecture Notes in Computer Science. Baltimore, MD, USA: Springer, Heidelberg, Germany, Nov. 2017, pp. 341–371. DOI: 10.1007/978-3-319-70500-2_12.

[HPP21] Julius Hermelink, Peter Pessl, and Thomas Pöppelmann. “Fault-Enabled Chosen-Ciphertext Attacks on Kyber”. In: *Progress in Cryptology - INDOCRYPT 2021 - 22nd International Conference on Cryptology in India, Jaipur, India, December 12-15, 2021, Proceedings*. Ed. by Avishek Adhikari, Ralf Küsters, and Bart Preneel. Vol. 13143. Lecture Notes in Computer Science. Springer, 2021, pp. 311–334. DOI: 10.1007/978-3-030-92518-5_15. URL: https://doi.org/10.1007/978-3-030-92518-5_15.

[HRB13] Andreas Hülsing, Lea Rausch, and Johannes Buchmann. “Optimal Parameters for XMSS^{MT}”. In: *Security Engineering and Intelligence Informatics - CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Germany, September 2-6, 2013. Proceedings*. Ed. by Alfredo Cuzzocrea, Christian Kittl, Dimitris E. Simos, Edgar R. Weippl, and Lida Xu. Vol. 8128. Lecture Notes in Computer Science. Springer, 2013, pp. 194–208. DOI: 10.1007/978-3-642-40588-4_14. URL: https://doi.org/10.1007/978-3-642-40588-4_14.

[Hue+18] Andreas Huelsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. *XMSS: eXtended Merkle Signature Scheme*. RFC 8391. May 2018. DOI: 10.17487/RFC8391. URL: https://www.rfc-editor.org/info/rfc8391.

[Hül13] Andreas Hülsing. "W-OTS+ - Shorter Signatures for Hash-Based Signature Schemes". In: *AFRICACRYPT 13: 6th International Conference on Cryptology in Africa*. Ed. by Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien. Vol. 7918. Lecture Notes in Computer Science. Cairo, Egypt: Springer, Heidelberg, Germany, June 2013, pp. 173–188. DOI: 10.1007/978-3-642-38553-7_10.

[Hül+20] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. *SPHINCS+*. Tech. rep. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020.

[Hül+22] Andreas Hülsing, Daniel J. Bernstein, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Panos Kampanakis, Stefan Kolbl, Tanja Lange, Martin M Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, Jean-Philippe Aumasson, Bas Westerbaan, and Ward Beullens. *SPHINCS+*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[IBM22] IBM. *IBM Quantum Computing | Roadmap*. 2022. URL: https://www.ibm.com/quantum/roadmap (visited on 09/01/2022).

[Isl+22] Saad Islam, Koksal Mus, Richa Singh, Patrick Schaumont, and Berk Sunar. "Signature Correction Attack on Dilithium Signature Scheme". In: *7th IEEE European Symposium on Security and Privacy, EuroS&P 2022, Genoa, Italy, June 6-10, 2022*. IEEE, 2022, pp. 647–663. DOI: 10.1109/EuroSP53844.2022.00046. URL: https://doi.org/10.1109/EuroSP53844.2022.00046.

[IT03] Tetsuya Izu and Tsuyoshi Takagi. "Exceptional Procedure Attack on Elliptic Curve Cryptosystems". In: *PKC 2003: 6th International Workshop on Theory and Practice in Public Key Cryptography*. Ed. by Yvo Desmedt. Vol. 2567. Lecture Notes in Computer Science. Miami, FL, USA: Springer, Heidelberg, Germany, Jan. 2003, pp. 224–239. DOI: 10.1007/3-540-36288-6_17.

[Jac99] John David Jackson. *Classical electrodynamics*. 1999.

[Jao19] David Jao. *SIKE: Supersingular Isogeny Key Encapsulation*. Aug. 2019. URL: https://csrc.nist.gov/Presentations/2019/sike-round-2-presentation (visited on 09/01/2022).

[Jao+20] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. *SIKE*. Tech. rep. available at https://csrc.

nist.gov/projects/post-quantum-cryptography/round-3-submissions. National Institute of Standards and Technology, 2020.

[Jao+22] David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, David Urbanik, Geovandro Pereira, Koray Karabina, and Aaron Hutchinson. *SIKE*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions. National Institute of Standards and Technology, 2022.

[JT01] Marc Joye and Christophe Tymen. "Protections against Differential Analysis for Elliptic Curve Cryptography". In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2001, pp. 377–390. DOI: 10.1007/3-540-44709-1_31.

[KAA17] Mehran Mozaffari Kermani, Reza Azarderakhsh, and Anita Aghaie. "Fault Detection Architectures for Post-Quantum Cryptographic Stateless Hash-Based Secure Signatures Benchmarked on ASIC". In: *ACM Trans. Embed. Comput. Syst.* 16.2 (2017), 59:1–59:19. DOI: 10.1145/2930664. URL: https://doi.org/10.1145/2930664.

[KAJ17] Brian Koziel, Reza Azarderakhsh, and David Jao. "Side-Channel Attacks on Quantum-Resistant Supersingular Isogeny Diffie-Hellman". In: *SAC 2017: 24th Annual International Workshop on Selected Areas in Cryptography*. Ed. by Carlisle Adams and Jan Camenisch. Vol. 10719. Lecture Notes in Computer Science. Ottawa, ON, Canada: Springer, Heidelberg, Germany, Aug. 2017, pp. 64–81. DOI: 10.1007/978-3-319-72565-9_4.

[Kal22] Novak Kaluđerović. "Attacks On Some Post-quantum Cryptographic Protocols: The Case Of The Legendre PRF And SIKE". PhD thesis. EPFL, 2022.

[Kan17] Matthias J. Kannwischer. "Physical Attack Vulnerability of Hash-Based Signature Schemes". MA thesis. TU Darmstadt, 2017.

[Kan+17] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. *GitHub repositories for DPA code of SHA-256 PRNG and BLAKE-256 PRF*. 2017. URL: https://github.com/hbs-sca (visited on 09/01/2022).

[Kan+18] Matthias J. Kannwischer, Aymeric Genêt, Denis Butin, Juliane Krämer, and Johannes Buchmann. "Differential Power Analysis of XMSS and SPHINCS". In: *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Junfeng Fan and Benedikt Gierlichs. Vol. 10815. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, Apr. 2018, pp. 168–188. DOI: 10.1007/978-3-319-89641-0_10.

[Kan+19] Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen. "pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4". In: *Second PQC Standardization Conference, NIST* (2019). URL: https://csrc.nist.gov/CSRC/media/

Events / Second - PQC - Standardization - Conference / documents / accepted - papers/kannwischer-pqm4.pdf.

[Kan97] Ernst Kani. “The number of curves of genus two with elliptic differentials”. In: *Journal für die reine und angewandte Mathematik* 485 (1997), pp. 93–121. DOI: 10.1515/crll.1997.485.93. URL: https://doi.org/10.1515/crll.1997.485.93.

[Kim+14] 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. URL: https://doi.org/10.1109/ISCA.2014.6853210.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. “Differential Power Analysis”. In: *Advances in Cryptology – CRYPTO’99*. Ed. by Michael J. Wiener. Vol. 1666. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1999, pp. 388–397. DOI: 10.1007/3-540-48405-1_25.

[Lah+19] Norman Lahr, Ruben Niederhagen, Richard Petri, and Simona Samardjiska. *Side Channel Information Set Decoding*. Cryptology ePrint Archive, Report 2019/1459. https://eprint.iacr.org/2019/1459. 2019.

[Lam79] Leslie Lamport. *Constructing Digital Signatures from a One-way Function*. Technical Report SRI-CSL-98. SRI International Computer Science Laboratory, Oct. 1979.

[LO77] Jeffrey C Lagarias and Andrew M Odlyzko. “Effective versions of the Chebotarev density theorem”. In: *Algebraic number fields: L-functions and Galois properties (Proc. Sympos., Univ. Durham, Durham, 1975)*. Vol. 7. 1977, pp. 409–464.

[Lyu+22] Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. *CRYSTALS-DILITHIUM*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[Mac67] James MacQueen. “Some methods for classification and analysis of multivariate observations”. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[Mai+23] Luciano Maino, Chloe Martindale, Lorenz Panny, Giacomo Pope, and Benjamin Wesolowski. “A Direct Key Recovery Attack on SIDH”. In: *TBD*. Ed. by TBD. Vol. TBD. Lecture Notes in Computer Science. TBD: TBD, Apr. 2023, pp. 534–546. DOI: 10.1007/978-3-031-xxxxx-x.

[Mal08] Stéphane Mallat. *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. 3rd. USA: Academic Press, Inc., 2008. ISBN: 0123743702.

[McC+19] Sarah McCarthy, James Howe, Neil Smyth, Séamus Brannigan, and Máire O'Neill. "BEARZ Attack FALCON: Implementation Attacks with Countermeasures on the FALCON Signature Scheme". In: *Proceedings of the 16th International Joint Conference on e-Business and Telecommunications, ICETE 2019 - Volume 2: SECRYPT, Prague, Czech Republic, July 26-28, 2019*. Ed. by Mohammad S. Obaidat and Pierangela Samarati. SciTePress, 2019, pp. 61–71. DOI: 10.5220/0007834800610071. URL: https://doi.org/10.5220/0007834800610071.

[Mer90] Ralph C. Merkle. "A Certified Digital Signature". In: *Advances in Cryptology – CRYPTO'89*. Ed. by Gilles Brassard. Vol. 435. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 1990, pp. 218–238. DOI: 10.1007/0-387-34805-0_21.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. "Correlation-Enhanced Power Analysis Collision Attack". In: *Cryptographic Hardware and Embedded Systems – CHES 2010*. Ed. by Stefan Mangard and François-Xavier Standaert. Vol. 6225. Lecture Notes in Computer Science. Santa Barbara, CA, USA: Springer, Heidelberg, Germany, Aug. 2010, pp. 125–139. DOI: 10.1007/978-3-642-15031-9_9.

[Mon87] Peter Montgomery. "Speeding the Pollard and elliptic curve methods of factorization". In: *Mathematics of Computation* 48 (1987), pp. 243–264.

[Moo19a] Dustin Moody. *Round 2 of the NIST PQC "Competition" - What was NIST Thinking?* May 2019. URL: https://csrc.nist.gov/presentations/2019/round-2-of-the-nist-pqc-competition-what-was-nist (visited on 09/01/2022).

[Moo19b] Dustin Moody. *The 2nd Round of the NIST PQC Standardization Process-Opening Remarks at PQC 2019*. Aug. 2019. URL: https://csrc.nist.gov/Presentations/2019/the-2nd-round-of-the-nist-pqc-standardization-proc (visited on 09/01/2022).

[MvV97] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. The CRC Press series on discrete mathematics and its applications. 2000 N.W. Corporate Blvd., Boca Raton, FL 33431-9868, USA: CRC Press, 1997, pp. xxviii + 780. ISBN: 0-8493-8523-7.

[NC17] Erick Nascimento and Łukasz 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. URL: https://doi.org/10.1007/978-3-319-75208-2_13.

[New17] NewAE Technology Inc. *SimpleSerial - ChipWhisperer Wiki*. 2017. URL: https://wiki.newae.com/SimpleSerial (visited on 09/01/2022).

[New21a] NewAE Technology Inc. *CHIPWHIPERER | NewAE Technology*. 2021. URL: https://www.newae.com/chipwhisperer (visited on 09/01/2022).

[New21b] NewAE Technology Inc. *GitHub - newaetech/chipwhisperer: ChipWhisperer - the complete open-source toolchain for side-channel power analysis and glitching attacks*. 2021. URL: https://github.com/newaetech/chipwhisperer (visited on 09/01/2022).

[Ngo+21] Kalle Ngo, Elena Dubrova, Qian Guo, and Thomas Johansson. "A Side-Channel Attack on a Masked IND-CCA Secure Saber KEM Implementation". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.4 (2021). https://tches.iacr.org/index.php/TCHES/article/view/9079, pp. 676–707. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i4.676-707.

[NIS15] NIST. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*. Federal Information Processing Standards (FIPS) Publication 202. Aug. 2015. DOI: 10.6028/NIST.FIPS.202.

[NIS16] NIST. *Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process*. Dec. 2016. URL: https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf.

[NIS22] NIST. *Post-Quantum Cryptography: Digital Signature Schemes | CSRC*. Aug. 2022. URL: https://csrc.nist.gov/projects/pqc-dig-sig (visited on 09/01/2022).

[NS88] Kazuo Nishimura and Masaaki Sibuya. "Occupancy with two types of balls". In: *Annals of the Institute of Statistical Mathematics* 40 (1988), pp. 77–91.

[Onl22] Online Hash Crack. *Benchmark Hashcat RTX 3090*. 2022. URL: https://www.onlinehashcrack.com/tools-benchmark-hashcat-nvidia-rtx-3090.php (visited on 09/01/2022).

[PC15] Guilherme Perin and Łukasz Chmielewski. "A Semi-Parametric Approach for Side-Channel Attacks on Protected RSA Implementations". In: *Smart Card Research and Advanced Applications - 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*. Ed. by Naofumi Homma and Marcel Medwed. Vol. 9514. Lecture Notes in Computer Science. Springer, 2015, pp. 34–53. URL: https://doi.org/10.1007/978-3-319-31271-2_3.

[Per01] Adrian Perrig. "The BiBa One-Time Signature and Broadcast Authentication Protocol". In: *ACM CCS 2001: 8th Conference on Computer and Communications Security*. Ed. by Michael K. Reiter and Pierangela Samarati. Philadelphia, PA, USA: ACM Press, Nov. 2001, pp. 28–37. DOI: 10.1145/501983.501988.

[Per+14] Guilherme Perin, Laurent Imbert, Lionel Torres, and Philippe Maurine. "Attacking Randomized Exponentiations Using Unsupervised Learning". In: *COSADE 2014: 5th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Emmanuel Prouff. Vol. 8622. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, Apr. 2014, pp. 144–160. DOI: 10.1007/978-3-319-10175-0_11.

[Per+21] Guilherme Perin, Łukasz Chmielewski, Lejla Batina, and Stjepan Picek. "Keep it Unsupervised: Horizontal Attacks Meet Deep Learning". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (2021). https://tches.iacr.org/index.php/TCHES/article/view/8737, pp. 343–372. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i1.343-372.

[Pir+22] Sabine Pircher, Johannes Geier, Julian Danner, Daniel Mueller-Gritschneder, and Antonia Wachter-Zeh. "Key-Recovery Fault Injection Attack on the Classic McEliece KEM". In: *Code-Based Cryptography - 10th International Workshop, CBCrypto 2022, Trondheim, Norway, May 29-30, 2022, Revised Selected Papers*. Ed. by Jean-Christophe Deneuville. Vol. 13839. Lecture Notes in Computer Science. Springer, 2022, pp. 37–61. DOI: 10.1007/978-3-031-29689-5_3. URL: https://doi.org/10.1007/978-3-031-29689-5_3.

[PP21] Peter Pessl and Lukas Prokop. "Fault Attacks on CCA-secure Lattice KEMs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (2021). https://tches.iacr.org/index.php/TCHES/article/view/8787, pp. 37–60. ISSN: 2569-2925. DOI: 10.46586/tches.v2021.i2.37-60.

[Pre+22] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. *FALCON*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[PZS17] Romain Poussier, Yuanyuan Zhou, and François-Xavier Standaert. "A Systematic Approach to the Side-Channel Analysis of ECC Implementations with Worst-Case Horizontal Attacks". In: *Cryptographic Hardware and Embedded Systems – CHES 2017*. Ed. by Wieland Fischer and Naofumi Homma. Vol. 10529. Lecture Notes in Computer Science. Taipei, Taiwan: Springer, Heidelberg, Germany, Sept. 2017, pp. 534–554. DOI: 10.1007/978-3-319-66787-4_26.

[Raj+23] Gokulnath Rajendran, Prasanna Ravi, Jan-Pieter D'Anvers, Shivam Bhasin, and Anupam Chattopadhyay. "Pushing the Limits of Generic Side-Channel Attacks on LWE-based KEMs - Parallel PC Oracle Attacks on Kyber KEM and Beyond". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023.2 (2023). https://tches.iacr.org/index.php/TCHES/article/view/10289, pp. 418–446. ISSN: 2569-2925. DOI: 10.46586/tches.v2023.i2.418-446.

[Rav+19a] Prasanna Ravi, Mahabir Prasad Jhanwar, James Howe, Anupam Chattopadhyay, and Shivam Bhasin. "Exploiting Determinism in Lattice-based Signatures: Practical Fault Attacks on pqm4 Implementations of NIST Candidates". In: *ASIACCS 19: 14th ACM Symposium on Information, Computer and Communications Security*. Ed. by Steven D. Galbraith, Giovanni Russello, Willy Susilo, Dieter Gollmann, Engin Kirda, and Zhenkai Liang. Auckland, New Zealand: ACM Press, July 2019, pp. 427–440. DOI: 10.1145/3321705.3329821.

[Rav+19b] Prasanna Ravi, Debapriya Basu Roy, Shivam Bhasin, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. "Number "Not Used" Once - Practical Fault Attack on pqm4 Implementations of NIST Candidates". In: *COSADE 2019: 10th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Ilia Polian and Marc Stöttinger. Vol. 11421. Lecture Notes in Computer Science. Darmstadt, Germany: Springer, Heidelberg, Germany, Apr. 2019, pp. 232–250. DOI: 10.1007/978-3-030-16350-1_13.

[Rav+20] Prasanna Ravi, Sujoy Sinha Roy, Anupam Chattopadhyay, and Shivam Bhasin. "Generic Side-channel attacks on CCA-secure lattice-based PKE and KEMs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (2020). https://tches.iacr.org/index.php/TCHES/article/view/8592, pp. 307–335. ISSN: 2569-2925. DOI: 10.13154/tches.v2020.i3.307-335.

[Ren18] Joost Renes. "Computing Isogenies Between Montgomery Curves Using the Action of (0, 0)". In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Fort Lauderdale, Florida, United States: Springer, Heidelberg, Germany, Apr. 2018, pp. 229–247. DOI: 10.1007/978-3-319-79063-3_11.

[Rob22] Damien Robert. *Breaking SIDH in polynomial time*. Cryptology ePrint Archive, Report 2022/1038. https://eprint.iacr.org/2022/1038. 2022.

[RR02] Leonid Reyzin and Natan Reyzin. "Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying". In: *Information Security and Privacy, 7th Australian Conference, ACISP 2002, Melbourne, Australia, July 3-5, 2002, Proceedings*. Ed. by Lynn Margaret Batten and Jennifer Seberry. Vol. 2384. Lecture Notes in Computer Science. Springer, 2002, pp. 144–153. DOI: 10.1007/3-540-45450-0_11. URL: https://doi.org/10.1007/3-540-45450-0_11.

[RR21] Prasanna Ravi and Sujoy Sinha Roy. *Side-Channel Analysis of Lattice-based PQC Candidates*. Mar. 2021. URL: https://csrc.nist.gov/CSRC/media/Projects/post-quantum-cryptography/documents/round-3/seminars/mar-2021-ravi-sujoy-presentation.pdf (visited on 09/01/2022).

[RS06] Alexander Rostovtsev and Anton Stolbunov. *Public-Key Cryptosystem Based On Isogenies*. Cryptology ePrint Archive, Report 2006/145. https://eprint.iacr.org/2006/145. 2006.

[Sch+20] Thomas Schamberger, Julian Renner, Georg Sigl, and Antonia Wachter-Zeh. "A Power Side-Channel Attack on the CCA2-Secure HQC KEM". In: *Smart Card Research and Advanced Applications - 19th International Conference, CARDIS 2020, Virtual Event, November 18-19, 2020, Revised Selected Papers*. Ed. by Pierre-Yvan Liardet and Nele Mentens. Vol. 12609. Lecture Notes in Computer Science. Springer, 2020, pp. 119–134. DOI: 10.1007/978-3-030-68487-7_8. URL: https://doi.org/10.1007/978-3-030-68487-7_8.

[Sch+22a] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. "A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem". In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*. Ed. by Jung Hee Cheon and Thomas Johansson. Vol. 13512. Lecture Notes in Computer Science. Springer, 2022, pp. 327–352. DOI: 10.1007/978-3-031-17234-2_16. URL: https://doi.org/10.1007/978-3-031-17234-2_16.

[Sch+22b] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. *CRYSTALS-KYBER*. Tech. rep. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022. National Institute of Standards and Technology, 2022.

[Seo+20] Hwajeong Seo, Mila Anastasova, Amir Jalali, and Reza Azarderakhsh. *Supersingular Isogeny Key Encapsulation (SIKE) Round 2 on ARM Cortex-M4*. Cryptology ePrint Archive, Report 2020/410. https://eprint.iacr.org/2020/410. 2020.

[SH17] Bo-Yeon Sim and Dong-Guk Han. "Key Bit-Dependent Attack on Protected PKC Using a Single Trace". In: *Information Security Practice and Experience - 13th International Conference, ISPEC 2017, Melbourne, VIC, Australia, December 13-15, 2017, Proceedings*. Ed. by Joseph K. Liu and Pierangela Samarati. Vol. 10701. Lecture Notes in Computer Science. Springer, 2017, pp. 168–185. URL: https://doi.org/10.1007/978-3-319-72359-4_10.

[Shi+19] Fanyu Shi, Jizeng Wei, Dazhi Sun, and Guo Wei. "A Systematic Approach to Horizontal Clustering Analysis on Embedded RSA Implementation". In: *25th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2019, Tianjin, China, December 4-6, 2019*. IEEE, 2019, pp. 901–906. URL: https://doi.org/10.1109/ICPADS47876.2019.00132.

[Sho94] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *35th Annual Symposium on Foundations of Computer Science*. Santa Fe, NM, USA: IEEE Computer Society Press, Nov. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[Sil86] Joseph H. Silverman. *The arithmetic of elliptic curves*. Vol. 106. Graduate texts in mathematics. Department of Mathematics, Brown University, 151 Thayer St., Providence, RI 02912, USA: Springer, Heidelberg, Germany, 1986, pp. xvii+513. ISBN: 978-3-540-96203-8.

[SM11] Mutsuo Saito and Makoto Matsumoto. *Tiny Mersenne Twister pseudo-random number generator*. 2011. URL: https://github.com/MersenneTwister-Lab/TinyMT (visited on 09/01/2022).

[SM16] Tobias Schneider and Amir Moradi. "Leakage assessment methodology - Extended version". In: *Journal of Cryptographic Engineering* 6.2 (June 2016), pp. 85–99. DOI: 10.1007/s13389-016-0120-y.

[Sou+21] Youssef Souissi, M. Abdelaziz El Aabid, Nicolas Debande, Sylvain Guilley, and Jean-Luc Danger. "Novel Applications of Wavelet Transforms based Side-Channel Analysis". In: *Non-Invasive Attack Testing Workshop*. Nov. 2021.

[Spe+15] Robert Specht, Johann Heyszl, Martin Kleinsteuber, and Georg Sigl. "Improving Non-profiled Attacks on Exponentiations Based on Clustering and Extracting Leakage from Multi-channel High-Resolution EM Measurements". In: *COSADE 2015: 6th International Workshop on Constructive Side-Channel Analysis and Secure Design*. Ed. by Stefan Mangard and Axel Y. Poschmann: vol. 9064. Lecture Notes in Computer Science. Berlin, Germany: Springer, Heidelberg, Germany, Apr. 2015, pp. 3–19. DOI: 10.1007/978-3-319-21476-4_1.

[SPW11] Bianca Schroeder, Eduardo Pinheiro, and Wolf-Dietrich Weber. "DRAM errors in the wild: a large-scale field study". In: *Commun. ACM* 54.2 (2011), pp. 100–107. DOI: 10.1145/1897816.1897844. URL: https://doi.org/10.1145/1897816.1897844.

[Sto10] Anton Stolbunov. "Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves". In: *Adv. Math. Commun.* 4.2 (2010), pp. 215–235. DOI: 10.3934/amc.2010.4.215. URL: https://doi.org/10.3934/amc.2010.4.215.

[SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. "A New Class of Collision Attacks and Its Application to DES". In: *Fast Software Encryption – FSE 2003*. Ed. by Thomas Johansson. Vol. 2887. Lecture Notes in Computer Science. Lund, Sweden: Springer, Heidelberg, Germany, Feb. 2003, pp. 206–222. DOI: 10.1007/978-3-540-39887-5_16.

[Tan+22] Yutaro Tanaka, Rei Ueno, Keita Xagawa, Akira Ito, Junko Takahashi, and Naofumi Homma. *Multiple-Valued Plaintext-Checking Side-Channel Attacks on Post-Quantum KEMs*. Cryptology ePrint Archive, Report 2022/940. https://eprint.iacr.org/2022/940. 2022.

[Tas+21] Élise Tasso, Luca De Feo, Nadia El Mrabet, and Simon Pontié. "Resistance of Isogeny-Based Cryptographic Implementations to a Fault Attack". In: *Constructive Side-Channel Analysis and Secure Design - 12th International Workshop, COSADE 2021, Lugano, Switzerland, October 25-27, 2021, Proceedings*. Ed. by Shivam Bhasin and Fabrizio De Santis. Vol. 12910. Lecture Notes in Computer Science. Springer, 2021, pp. 255–276. DOI: 10.1007/978-3-030-89915-8_12. URL: https://doi.org/10.1007/978-3-030-89915-8_12.

[Uen+22] Rei Ueno, Keita Xagawa, Yutaro Tanaka, Akira Ito, Junko Takahashi, and Naofumi Homma. "Curse of Re-encryption: A Generic Power/EM Analysis on Post-Quantum KEMs". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2022.1 (2022), pp. 296–322. DOI: 10.46586/tches.v2022.i1.296-322.

[Vél71] Jacques Vélu. "Isogénies entre courbes elliptiques". In: *Comptes-Rendus de l'Académie des Sciences, Série I* 273 (July 1971), pp. 238–241.

[Wal01] Colin D. Walter. “Sliding Windows Succumbs to Big Mac Attack”. In: *Cryptographic Hardware and Embedded Systems – CHES 2001*. Ed. by Çetin Kaya Koç, David Naccache, and Christof Paar. Vol. 2162. Lecture Notes in Computer Science. Paris, France: Springer, Heidelberg, Germany, May 2001, pp. 286–299. DOI: 10.1007/3-540-44709-1_24.

[Wan+22] Yingchen Wang, Riccardo Paccagnella, Elizabeth He, Hovav Shacham, Christopher W. Fletcher, and David Kohlbrenner. “Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86”. In: *Proceedings of the USENIX Security Symposium (USENIX)*. 2022.

[Wes22] Benjamin Wesolowski. *Understanding and improving the Castryck-Decru attack on SIDH*. https://bweso.com/papers.php. 2022.

[Xag+21] Keita Xagawa, Akira Ito, Rei Ueno, Junko Takahashi, and Naofumi Homma. “Fault-Injection Attacks Against NIST’s Post-Quantum Cryptography Round 3 KEM Candidates”. In: *Advances in Cryptology – ASIACRYPT 2021, Part II*. Ed. by Mehdi Tibouchi and Huaxiong Wang. Vol. 13091. Lecture Notes in Computer Science. Singapore: Springer, Heidelberg, Germany, 2021, pp. 33–61. DOI: 10.1007/978-3-030-92075-3_2.

[Xu+22] Zhuang Xu, Owen Pemberton, Sujoy Sinha Roy, David F. Oswald, Wang Yao, and Zhiming Zheng. “Magnifying Side-Channel Leakage of Lattice-Based Cryptosystems With Chosen Ciphertexts: The Case Study of Kyber”. In: *IEEE Trans. Computers* 71.9 (2022), pp. 2163–2176. DOI: 10.1109/TC.2021.3122997. URL: https://doi.org/10.1109/TC.2021.3122997.

[Yoo+17] Youngho Yoo, Reza Azarderakhsh, Amir Jalali, David Jao, and Vladimir Soukharev. “A Post-quantum Digital Signature Scheme Based on Supersingular Isogenies”. In: *FC 2017: 21st International Conference on Financial Cryptography and Data Security*. Ed. by Aggelos Kiayias. Vol. 10322. Lecture Notes in Computer Science. Sliema, Malta: Springer, Heidelberg, Germany, Apr. 2017, pp. 163–181.

[Zan+18] Gustavo Zanon, Marcos A. Simplício Jr., Geovandro C. C. F. Pereira, Javad Doliskani, and Paulo S. L. M. Barreto. “Faster Isogeny-Based Compressed Key Agreement”. In: *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*. Ed. by Tanja Lange and Rainer Steinwandt. Fort Lauderdale, Florida, United States: Springer, Heidelberg, Germany, Apr. 2018, pp. 248–268. DOI: 10.1007/978-3-319-79063-3_12.

[Zha+20] Fan Zhang, Bolin Yang, Xiaofei Dong, Sylvain Guilley, Zhe Liu, Wei He, Fangguo Zhang, and Kui Ren. “Side-Channel Analysis and Countermeasure Design on ARM-Based Quantum-Resistant SIKE”. In: *IEEE Trans. Computers* 69.11 (2020), pp. 1681–1693. DOI: 10.1109/TC.2020.3020407. URL: https://doi.org/10.1109/TC.2020.3020407.

Aymeric Genêt

+41 78 840 47 16 | aymeric.genet@alumni.epfl.ch | aymericgenet | Lausanne, VD, CH

PhD in cryptography, expert in post-quantum cryptography and side-channel attacks, excellent programming skills

EDUCATION

École Polytechnique Fédérale de Lausanne (EPFL) 2017 – 2023
Doctor of Science, Computer and Communication Systems

Title: *Side-channel analysis of isogeny-based key encapsulation mechanisms and hash-based digital signatures.*

Supervised by both Prof. Arjen K. Lenstra in the Laboratory for Cryptologic Algorithms (LACAL) and Prof. Serge Vaudenay in the Security and Cryptography Laboratory (LASEC).

École Polytechnique Fédérale de Lausanne (EPFL) 2014 – 2017
Master's Degree, Communication Systems

École Polytechnique Fédérale de Lausanne (EPFL) 2011 – 2014
Bachelor's Degree, Communication Systems

EXPERIENCE

Nagra Kudelski Group Cheseaux-sur-Lausanne, VD, CH
Senior Security Engineer Oct. 2022 – Present

Security audit of embedded source code, blockchain architectures, cryptographic solutions in proprietary IP, and hardware implementations (AES, 3DES, ECC, and RSA).

Security Engineer Sep. 2017 – Oct. 2022

Security audit of embedded source code and hardware implementations (AES, 3DES, ECC, and RSA).

Master Thesis Feb. 2017 – Sep. 2017

Conducted research on side-channel and fault attacks against hash-based cryptography.

Student Internship Aug. 2016 – Feb. 2017

Developed a tool to mount lattice attacks against classical digital signature schemes.

École Polytechnique Fédérale de Lausanne (EPFL) Ecublens, VD, CH
Teaching Assistant Sep. 2017 – Feb. 2020

Provided leading assistance to the 1st-year class *Advanced Information, Computation, and Communication I*.

Summer Internship Jul. 2015 – Aug. 2015

Enhanced an implementation in C of the Schoof–Elkies–Atkin algorithm (point counting on elliptic curves).

Student Assistant Feb. 2013 – Feb. 2016

Provided assistance to 1st-year classes including *Introduction to OOP*, *IT Project*, and *Discrete Structures*.

NOTABLE PUBLICATIONS

On Protecting SPHINCS+ Against Fault Attacks CHES 2023

SIKE Channels - Zero-Value Side-Channel Attacks on SIKE CHES 2022

SKILLS

Languages : Python, Sagemath, C, Scala, Rust, Java, C#, C++, Bash, LaTeX, Matlab