On the Security of Lattice-Based Signature Schemes in a Post-Quantum World

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Nina Laura Bindel, M.Sc.

geboren in Friedrichroda.

Referenten:	Prof. Dr. Johannes Buchmann Prof. Dr. Douglas Stebila
Tag der Einreichung:	08.08.2018
Tag der mündlichen Prüfung:	20.09.2018
Hochschulkennziffer:	D 17

Darmstadt 2018

On the Security of Lattice-Based Signature Schemes in a Post-Quantum World

Genehmigte Dissertation von Nina Bindel, M.Sc. aus Friedrichroda

Technische Universität Darmstadt, Darmstadt, Germany

Tag der Einreichung: 08.08.2018
Tag der mündlichen Prüfung: 20.09.2018
Jahr der Veröffentlichung: 2018

List of Publications

[B1] Sedat Akleylek, Nina Bindel, Johannes Buchmann, Juliane Krämer, and Giorgia Azzurra Marson: An Efficient Lattice-Based Signature Scheme with Provably Secure Instantiation. *Progress in Cryptology – 8th International Conference on Cryptology in Africa (AFRICACRYPT 2016)*, pages 44–60, Springer, 2016. [**Part of Chapter 3**].

[B2] Erdem Alkim, Nina Bindel, Johannes Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, Filip Palewa: Revisiting TESLA in the quantum random oracle model. *Post-Quantum Cryptography – 8th International Workshop (PQCrypto 2017)*, pages 143–162, Springer, 2017. [**Part of Chapter 3**].

[B3] Nina Bindel: Ein deutsches digitales Signaturverfahren auf dem Weg zum internationalen kryptographischen Standard. *Tagungsband zum 15. Deutscher IT-Sicherheitskongress*, pages 11–21, SecuMedia Verlag, 2017.

[B4] Nina Bindel, Sedat Akleylek, Erdem Alkim, Paulo Barreto, Johannes Buchmann, Edward Eaton, Gus Gutoski, Juliane Krämer, Patrick Longa, Harun Polat, Jefferson Ricardini, Gustavo Zanon: Submission to NIST's post-quantum project: lattice-based digital signature scheme qTESLA. *Submission to the NIST Post-Quantum Cryptography Standardization, Round 1*, 2017. [**Part of Chapter 3**].

[B5] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Gonzales, Douglas Stebila: Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. Submitted to *Advances in Cryptology – 24th International Conference on the Theory and Applications of Cryptology and Information Security (ASIACRYPT 2018)*. [**Part of Chapter 5**].

[B6] Nina Bindel, Johannes Buchmann, Florian Göpfert, and Markus Schmidt: Estimation of the Hardness of the Learning with Errors Problem with a Restricted Number of Samples. *Journal of Mathematical Cryptology*, 2018.

[B7] Nina Bindel, Johannes Buchmann, and Juliane Krämer: Lattice-Based Signature Schemes and their Sensitivity to Fault Attacks. *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2016)*, pages 63–77, IEEE Computer Society, 2016. [**Part of Chapter 4**].

[B8] Nina Bindel, Johannes Buchmann, Juliane Krämer, Heiko Mantel, Johannes Schickel, and Alexandra Weber: Verification of Lattice-Based Signature Schemes against Cache Side Channels. *Foundations and Practice of Security - 10th International Symposium (FPS 2017*), pages 225–241, Springer, 2017. [**Part of Chapter 4**].

[B9] Nina Bindel, Johannes Buchmann, and Susanne Rieß: Comparing Apples with Apples: Performance Analysis of Lattice-Based Authenticated Key Exchange Protocols. *International Journal of Information Security* 17(6), pages 701-718, 2017.

[B10] Nina Bindel, Udyani Herath, Matthew McKague, and Douglas Stebila: Transitioning to a Quantum-Resistant Public Key Infrastructure. *Post-Quantum Cryptography – 8th International Workshop (PQCrypto 2017), pages 384–405, Springer, 2017.* [**Part of Chapter 5**].

[B11] Nina Bindel, Juliane Krämer, and Johannes Schreiber: Special Session: Hampering fault attacks against lattice-based signature schemes– countermeasures and their efficiency. *Proceedings of the 12th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis Companion (CODES+ISSS 2017) – Special Session*, pages 8:1–8:3, ACM, 2017. [**Part of Chapter 4**].

[B12] Johannes Buchmann and Nina Bindel: *Verschlüsselung und die Grenzen der Geheimhaltung. Beitrag für den Band zum Jahresthema "Leibniz: Vision als Aufgabe"*, pages 147-159, Berlin-Brandenburgische Akademie der Wissenschaften, 2016.

Life is and will ever remain an equation incapable of solution, but it contains certain known factors.

— Nikola Tesla *(1856-1943)*

Acknowlegdement

I would like to first and foremost, express my deep gratitude to my supervisor Johannes Buchmann. He taught me many life lessons that extended well beyond research, contributing with his experience and guidance to my academic as well as personal development. I greatly appreciated the freedom I had to pursue research that I found important and interesting. At the same time I also knew that Johannes Buchmann's door would always be open and I, my research, and any upcoming issue would be taken seriously by him.

I am also especially grateful to Douglas Stebila. I thank him for agreeing to co-review my thesis, for being a fantastic host when I visited McMaster University in Hamilton in June 2016, and lastly for his support and advice since then.

Additionally, I would like to extend my thanks to Reiner Hähnle, Christian Reuter, and Guido Salvaneschi for joining my defense committee. An additional thanks to Anshika Suri as well for proofreading my thesis.

Special thanks also to my co-authors. I benefited greatly from exchanging ideas on varied (research) topics. In particular, I would like to express the deepest appreciation to the qTESLA team: Thanks for walking this sometimes rocky road with me and for believing in qTESLA. Above all, I am truly grateful to Sedat Akleylek and Erdem Alkim for their unwavering support.

I also thank all my colleagues at TU Darmstadt. I really appreciated our discussions during the hikes in the CROSSING events or while guessing solutions during pub-quizzes. A special thanks to Thomas Wunderer whose maddening attention to detail and strong opinions made me reflect deeply on my statements in my dissertation until the end. In addition, I am thankful to Rachid El Bansarkhani and Florian Göpfert who I shared my office with and who never seemed to get annoyed by my questions. I also owe a great debt to Özgür Dagdelen: Although we worked together for a short while only, these first months have influenced my work greatly and became the corner stone of this thesis.

Last but not least, I thank my family and friends for their support, encouragement, patient listening, and never-ceasing interest in my research and well-being.

Nina Bindel
Darmstadt, August 2018

Abstract

Digital signatures are indispensable for security on the Internet, because they guarantee authenticity, integrity, and non-repudiation, of namely e-mails, software updates, and in the Transport Layer Security (TLS) protocol which is used for secure data transfer, for example. Most signature schemes that are currently in use such as the RSA signature scheme, are considered secure as long as the integer factorization problem or the discrete logarithm (DL) problem are computationally hard. At present, no algorithms have yet been found to solve these problems on conventional computers in polynomial time. However, in 1997, Shor published a polynomial-time algorithm that uses *quantum computation* to solve the integer factorization and the DL problem. In particular, this means that RSA signatures are considered broken as soon as large-scale quantum computers exist. Due to significant advances in the area of quantum computing, it is reasonable to assume that within 20 years, quantum computers that are able to break the RSA scheme, could exist. In order to maintain authenticity, integrity, and non-repudiation of data, cryptographic schemes that cannot be broken by quantum attacks are required. In addition, these so-called *post-quantum* secure schemes should be sufficiently efficient to be suitable for all established applications. Furthermore, solutions enabling a timely and secure transition from classical to post-quantum schemes are needed.

This thesis contributes to the above-mentioned transition. In this thesis, we present the two lattice-based digital signature schemes TESLA and qTESLA, whereby lattice-based cryptography is one of five approaches to construct post-quantum secure schemes. Furthermore, we prove that our signature schemes are secure as long as the so-called Learning With Errors (LWE) problem is computationally hard to solve. It is presumed that even quantum computers cannot solve the LWE problem in polynomial time. The security of our schemes is proven using security reductions. Since our reductions are *tight* and *explicit*, efficient instantiations are possible that provably guarantee a selected security level, as long as the corresponding LWE instance provides a certain hardness level. Since both our reductions (as proven in the quantum random oracle model) and instantiations, take into account quantum attackers, TESLA and qTESLA are considered post-quantum secure. Concurrently, the run-times for generating and verifying sig-

natures of qTESLA are similar (or faster) than those of the RSA scheme. However, key and signature sizes of RSA are smaller than those of qTESLA.

In order to protect both the theoretical signature schemes and their implementations against attacks, we analyze possible vulnerabilities against implementation attacks. In particular, cache-side-channel attacks resulting from observing the cache behavior and fault attacks, which recover secret information by actively disrupting the execution of an algorithm are focused. We present effective countermeasures for each implementation attack we found. Our analyses and countermeasures also influence the design and implementation of qTESLA.

Although our schemes are considered (post-quantum) secure according to state-of-the-art LWE attacks, cryptanalysis of lattice-based schemes is still a relatively new field of research in comparison to RSA schemes. Hence, there is a lack of confidence in the concrete instantiations and their promised security levels. However, due to developments within the field of quantum computers, a transition to post-quantum secure solutions seems to be more urgently required than ever. To solve this dilemma, we present an approach to combine two schemes, e.g., qTESLA and the RSA signature scheme, so that the combination is secure as long as one of the two combined schemes is secure. We present several of such combiners to construct hybrid signature schemes and hybrid key encapsulation mechanisms to ensure both authenticity and confidentiality in our Public-Key Infrastructure (PKI). Lastly, we also demonstrate how to apply the resulting *hybrid schemes* in standards such as X.509 or TLS.

To summarize, this work presents post-quantum secure candidates which can, using our hybrid schemes, add post-quantum security to the current classical security in our PKI.

Contents

Glossary

BDD Bounded Distance Decoding (problem).
BKW Blum-Kalai-Wassermann (algorithm).
BKZ Blockwise Korkine-Zolotarev (algorithm).

CA Certificate Authority.
CMS Cryptographic Message Syntax.
CPU Central Processing Unit.
CVP Closest Vector Problem.

DCK Decisional Compact Knapsack (problem).
DL Discrete Logarithm (problem).
DPA Differential Power Analysis.

EUF-CMA Existential Unforgeability under Chosen-Message Attack.

FO Fujisaki-Okamoto (transform).
FPGA Field-Programmable Gate Array.

GapSVP decisional approximate Shortest Vector Problem.
GPG Gnu Privacy Guard.

HKDF (HMAC)-based Key Derivation Function.
HMAC Hashed Message Authentication Code.

IKE Internet Key Exchange Protocol.
IND-CCA INDistinguishability against Chosen-Ciphertext Attacks.
IND-CPA INDistinguishability against Chosen-Plaintext Attacks.

KEM Key Encapsulation Mechanism.

LRU Least Recently Used.
LWE (decisional) Learning With Errors (problem).
LWR Learning With Rounding (problem).
LWT Learning With Truncation (problem).

M-LWE Matrix decisional Learning With Errors (problem).
MAC Message Authentication Code.

NIST National Institute of Standards and Technology (United States).
NTT Number Theoretic Transform.

OT-sEUF One-Time strong Existential Unforgeability.
OW-CCA One-Way security against Chosen-Ciphertext Attacks.
OW-CPA One-Way security against Chosen-Plaintext Attacks.

PKE Public-Key Encryption (scheme).
PKI Public-Key Infrastructure.
PQ Post-Quantum.
PRF Pseudo-Random Function.

QROM Quantum Random Oracle Model.

R-LWE Ring Learning With Errors (problem).
R-SIS Ring Shortest Vector Solution (problem).
RAM Random-Access Memory.
RISC Reduced Instruction Set Computing.
RNG Random Number Generator.
ROM Random Oracle Model.

S/MIME Secure/Multipurpose Internet Mail Extensions.
SEMA Simple Electro-Magnetic Analysis.
sEUF-CMA strong Existential Unforgeability under Chosen-Message Attack.
SIS Shortest Vector Solution (problem).
SIVP approximate Shortest Independent Vector Problem.
SPA Simple Power Analysis.
SVP Shortest Vector Problem.

TLS Transport Layer Security (protocol).

uSVP Unique Shortest Vector Problem.

XOF eXtendable Output Function.

List of Algorithms

List of Figures

List of Tables

1 Introduction

When the idea of digital signatures was first proposed in 1976 by Diffie and Hellman, its objective stated that it was to "discover a digital phenomenon with the same properties as a written signature" [75]. In particular, everyone should be able to check for themselves the authenticity of a message using its digital signature. Specifically, the existence of the digital signature should somehow demonstrate that the person who created the signature of a particular message is the sole author of the message. As it turns out, digital signatures are much more powerful than their analog counterpart. They ensure authenticity, integrity, and non-repudiation of messages or documents in the Internet. This makes them essential, e.g., for shopping on the Internet, where digital signatures can be used to ensure security in a multitude of ways: By verifying a digital signature, customers can be convinced of the authenticity of the online-shop's server. Moreover, since a signature of a shopping order ensures non-repudiation, the signed shopping order cannot be denied by the customer. Finally, the integrity of the financial transaction to pay for the ordered items can be guaranteed using signatures, e.g., the payment of the correct amount can be checked using its signature. This is exemplified through the case of Amazon Prime, who shipped more than five billion items in 2017 alone [175], showing the significance of online-shopping and hence, the importance of digital signatures.

As indicated by the examples above, digital signature schemes can be considered as a stand-alone primitive, e.g., to sign a transaction, or as a building block within our PKI. Example for the latter are certificates that prove a subject's ownership of a public key using digital signatures. This is implemented, e.g., in the Transport Layer Security (TLS) protocol which is used to securely connect to more than 50% of the webpages loaded by the popular web browser *Firefox* [127].

The first construction of a digital signature scheme, called the RSA signature scheme from now on in this thesis, was presented in 1978 by Rivest, Shamir, and Adleman [196]. Subsequently, many different signature schemes were proposed, e.g., by Lamport [149], Merkle [159], Rabin [188], and ElGamal [89]. However, until today, modern variants of the RSA scheme are still the most widely used signatures [86]. The RSA signature scheme is secure as long as the so-called RSA

problem [158, Section 3.3] is computationally hard to solve. One approach to solve the RSA problem is to factor large integers into their prime factors. Since it was presumed that no algorithm existed which could solve the integer factorization problem in polynomial time, the RSA scheme was considered secure. This assumption holds true till today, at least for algorithms that run on conventional or also called classical computers. However, in 1997, Shor presented an algorithm to solve the integer factorization problem in polynomial time using *quantum* computation [206]. Operations of a quantum computer use the principles of quantum physics: Information is stored, processed, and transmitted in quantum physical states, the so-called qubits. The existence of Shor's algorithm implies that all cryptographic schemes based upon the hardness of the integer factorization (or the discrete logarithm) problem are considered broken as soon as large-scale quantum computers are invented. In particular, RSA signatures must be regarded insecure.

Small-scale quantum computers already exist [125, 136] and at present, both research and industry are taking significant efforts to construct large-scale quantum computers for a good reason: Quantum computers seem to offer huge advantages compared to conventional computers. To exemplify, some important applications of quantum computers [128] are molecular modeling, financial modeling, and weather forecasting with each of them generating a high economic interest. In the beginning of 2018, the *Google Quantum AI lab* announced its new–currently record-breaking–quantum processor which works with 72 qubits [136]. To break for example RSA-2048, a number of 4096 qubits is necessary [185][1]. Some researchers and government bodies have estimated that quantum computers capable of breaking currently used cryptographic schemes could exist by the 2030s [71, 160].

Therefore, in order to maintain uninterrupted security on the Internet, it is important to prepare a secure transition from our current, classical PKI to cryptographic solutions that are secure even in the presence of a quantum attacker. Schemes that are secure against quantum attackers are called *post-quantum* secure from now on in this thesis.

A key initial step towards this goal is to provide post-quantum secure and practical signature schemes. Thus, it is desirable to construct signature schemes that are *provably secure* against quantum adversaries and come with *secure instantiations* chosen against quantum attacks. To ensure security when theoretically sound schemes are brought into praxis, it is preferable if the signature scheme's implementation additionally *resists physical attacks* such as fault and side-channel attacks. In order to maintain almost all the current available applications, signature schemes should be efficient in run-time and space, e.g., their key and signature sizes. In

[1] The numbers of qubits to break RSA instances reported in [185, Section 6.3], are the numbers of *logical* qubits. To perform computations on large-scale quantum computers, techniques such as error correction are necessary that increase the number of bits required drastically.

particular, since a vast majority of certificates are generated using RSA signatures, post-quantum signature schemes that are candidates to substitute classical schemes, must be as efficient as RSA schemes.

However, even if post-quantum schemes are instantiated against state-of-the-art classical and quantum attacks, researchers, industry, and standardization bodies might hesitate to deploy them for the following reason. There is not sufficient confidence yet in the instantiations of most post-quantum schemes actually providing the promised security levels because the security has not been studied as long as the security of RSA schemes. For example, Albrecht et al. [8] summarize and compare different hardness estimations of post-quantum instances and highlight that depending on the estimation method the differences of bit hardness are up to several hundred bits. At the same time, the demand to transition to post-quantum secure schemes has gotten stronger because of the recent progress in quantum computing. Therefore, solutions that circumvent this dilemma, i.e., solutions that guarantee at least the classical security and provide post-quantum security in addition, are in high demand for a secure and smooth transition.

The following branches of cryptography offer to provide post-quantum secure solutions: multivariate, hash-, code-, isogeny-, and lattice-based cryptography [39, 93]. All five approaches rely on hardness assumptions that are presumed unbreakable by quantum computers in polynomial time. In addition, lattice-based constructions allow for versatile applications, simple and fast arithmetic operations leading to high efficiency, and strong security guarantees. Consequently, they promise to be a particularly valuable approach to solve the above-mentioned challenge of a post-quantum transition, which is targeted in this thesis.

This thesis in particular presents two lattice-based signature schemes called TESLA and qTESLA. Both schemes are proven to be secure against quantum adversaries as long as the so-called Learning With Errors (LWE) problem is computationally hard even for quantum algorithms. To show the respective security guarantees we use so-called *security reductions*. Moreover, our provided security reductions are tight in the sense that an attacker $\mathcal{A}$ that breaks one of our signature schemes can be used to construct an algorithm $\mathcal{B}$ that solves LWE in about the same run-time and with the same success probability as $\mathcal{A}$. A tight security reduction is desirable because when instantiating the scheme according to its security reductions, tight reductions lead to smaller parameters and hence, to better performance [60]. The parameters of TESLA and qTESLA are chosen according to their security reduction and to withstand state-of-the-art quantum LWE solvers. While TESLA is tailored for high-security applications, qTESLA is optimized for efficiency, e.g., qTESLA signatures are generated faster than RSA signatures. The design of TESLA is based on the original signature scheme by Bai and Galbraith [24], includes the improvements suggested by Dagdelen et al. [69],

and improves upon them further.

qTESLA is based on its predecessor ring-TESLA which we constructed as a variant of TESLA and which is also described briefly in this thesis. Essentially, qTESLA assembles the advantages acquired in TESLA and ring-TESLA, resulting in a submission of a signature scheme to the post-quantum cryptography standardization process initiated by the National Institute of Standards and Technology (NIST) [164].

Secondly, we analyze the vulnerability of lattice-based signature schemes against implementation attacks such as fault and side-channel attacks. As implementing lattice-based signature schemes without cache side channels seems particularly challenging [110, 176, 177, 199], we analyze ring-TESLA taking cache side channels in regard and eventually present an implementation free of cache side channels. Moreover, we present an in-depth analysis of three efficient lattice-based signature schemes against fault attacks, since these schemes appear to be particularly vulnerable. The reason for their perceived vulnerability is explained as follows: The most efficient lattice-based signature schemes, such as BLISS [81], GLP [114], and ring-TESLA, are based on Fiat-Shamir identification [94] which was targeted by one of the very first fault attacks found [47]. In our fault attack analysis we consider zeroing, randomizing, and skipping faults against BLISS, GLP, and ring-TESLA, and find that all three schemes are vulnerable. We then propose effective countermeasures to mitigate each found vulnerability. For completeness, we also analyze the occurrence of power, electromagnetic, and timing side channels in lattice-based signature schemes as these are (in addition to cache side channels) the most commonly exploited side channels. Our analyses and countermeasures can also be applied to qTESLA.

Lastly, we present an approach to transition from the classical to post-quantum secure PKI using so-called *hybrid schemes*. We build different general combiners to securely combine two signature schemes or key encapsulation mechanisms (KEMs) such that the resulting combined scheme is at least as secure as one of the single schemes. Signature schemes and KEMs are two of the most important building blocks in protocols such as TLS. Other important primitives such as Public-Key Encryption schemes (PKE), passively secure key exchange, or authenticated key exchanges can be built from signatures, KEMs, and symmetric primitives such as hash or Pseudo-Random Functions (PRFs) [123, 145]. Since a fast and secure transition is particularly important for currently used standards and protocols, we also in addition provide examples of how to use our combiners in such protocols.

Contribution and Outline

The following paragraphs will now summarize our contributions and elaborate the structure of this thesis.

Background (Chapter 2). Firstly, we explain the relevant background. This includes the definition of lattices and relevant lattice-based assumptions. Furthermore, we briefly introduce quantum computation and summarize existing quantum scenarios. Subsequently, we provide the definition of digital signature schemes, KEMs, and their security notions in respective quantum models. Finally, we present a short introduction to side-channel and fault attacks.

The Signature Schemes TESLA and qTESLA (Chapter 3). This chapter presents the signature schemes TESLA and qTESLA.

We start this chapter by describing the signature schemes TESLA and qTESLA. In addition, we discuss the advantages and disadvantages of both compared to each other and briefly explain qTESLA's predecessor ring-TESLA. Furthermore, we prove the tight security reductions from LWE to TESLA and Ring-LWE (R-LWE) to qTESLA in the Quantum Random Oracle Model (QROM) [45]. Since our security reductions are explicit, i.e., they explicitly relate an instantiation of one of our signature schemes with an (R-)LWE instance, we are able to select parameters according to the reductions. In order to describe the parameter selection, we first explain how to estimate the hardness of the (R-)LWE problem and the correspondence between the bit security of our signature schemes and the bit hardness of (R-)LWE. We then present instantiations targeting different (quantum) security levels. Due to the tightness of the reduction, our parameters are also efficient as we demonstrate by presenting experimental results. Our benchmarks are obtained from the C implementations of TESLA and qTESLA compiled for and run on different platforms such as desktop computers, microcontrollers, and Field-Programmable Gate Arrays (FPGAs). Lastly, we compare our schemes with other signature schemes from the literature and discuss the extent of our results.

Implementation Security of Lattice-Based Signature Schemes (Chapter 4). This chapter is divided into three parts: cache-side-channel analysis in Section 4.1, fault-attack analysis in Section 4.2, and analysis regarding other side channels in Section 4.3.

Starting with Section 4.1, we develop an implementation of ring-TESLA that is resistant to four types of cache-side-channel attacks. As the first step towards this goal, we inspect the C code of ring-TESLA manually, identify vulnerable subroutines, and argue for the cache-side-channel resistance of the other subroutines. Finally, we implement mitigations and argue for their effectiveness. Although our analysis targets ring-TESLA as an example, our findings can also be transferred to other lattice-based primitives since the potential vulnerabilities are found in common building blocks of lattice-based cryptography. In particular, our findings are also used to secure the implementation of qTESLA.

Moving to Section 4.2, we present a careful analysis of three different signature schemes and their implementations, and scrutinize whether certain schemes or instantiations thereof are more vulnerable than others. Before we turn to our analysis, we present a hybrid approach that combines fault attacks and lattice analyses to reduce the number of necessary faults. The section then explains our analyses of vulnerabilities of the signature schemes BLISS, ring-TESLA, and GLP and their implementations with respect to zeroing, randomizing, and skipping faults. Our research demonstrates that the examined signature schemes and their implementations are vulnerable to all the three kinds of considered fault attacks, based on the effective attacks we present. We then present countermeasures against all our successful attacks. Furthermore, we demonstrate how to combine and implement our countermeasures by again using ring-TESLA as an example.

Finally, in Section 4.3, we summarize power-, electromagnetic-, and timing-side-channel attacks against lattice-based schemes and discuss if and how these attacks can be applied to lattice-based signature schemes, particularly to qTESLA.

Hybrid Signature and KEMs (Chapter 5). In this chapter, we construct hybrid signature schemes and KEMs for a secure transition from classical to post-quantum cryptography.

Accounting for security against quantum adversaries, we first present a family of security notions for signatures and KEMs that unify and extend existing quantum security models such as [45, 48, 49]. To do this, we adapt the traditional security experiments to enable a fine-grained distinction between the following different quantum scenarios: *fully classical* (all computation and oracle interaction is classical), *future-quantum* (the adversary is classical today but gains quantum power later), *post-quantum* (the adversary has quantum power but still interacts classically with the decapsulation or sign oracle), and *fully quantum* (all computation and interaction can be quantum). Furthermore, we show that these different security notions form a strict hierarchy. Subsequently, we present three different signature and three different KEM combiners. Lastly, we present examples to demonstrate the applicability of our combiners in not just protocols such as TLS but also in X.509 for certificates [65] and in Cryptographic Message Syntax (CMS) [124], as part of Secure/Multipurpose Internet Mail Extensions (S/MIME) [189] for secure e-mail. Through these examples, we discuss the possibility of instantiating the hybrid signature scheme with qTESLA or TESLA.

Conclusion (Chapter 6). This chapter concludes this thesis with a discussion on possible directions of future research.

2 Background

In this chapter, we provide the foundations of the different topics covered in this thesis. We start with an introduction to lattices in Section 2.1. Moreover, we define relevant lattice-based assumptions such as the Short Integer Solution (SIS) problem and the LWE problem in Section 2.2. In Section 2.3, we give a brief overview about quantum computation. We define digital signature schemes, KEMs, and their security notions in Section 2.4. Finally, we give a short introduction to fault and side-channel attacks in Section 2.5.

2.1 Lattices

In the following paragraphs, we give a short introduction to lattices. We start with the definition of notations that are used throughout the thesis.

Vectors with entries in $\mathbb{R}$ are viewed as column vectors and denoted with bold lowercase letters, e.g., $\mathbf{y}, \mathbf{z}, \mathbf{w}$. Moreover, matrices are denoted with bold uppercase letters, e.g., $\mathbf{A}, \mathbf{S}, \mathbf{E}$. The transpose of a vector or a matrix is denoted by $\mathbf{v}^T$ or $\mathbf{M}^T$, respectively. Additionally, we denote the *inner product* of two column vectors $\mathbf{v}, \mathbf{w} \in \mathbb{R}^n$ by $\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^T \cdot \mathbf{w}$.

After introducing some notations, we now start with the definition of a lattice. Let $m, n, k \in \mathbb{Z}_{>0}$ with $m \geq k$. An m-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^m$ containing all integer linear combinations of k linearly independent vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$. That is $\Lambda = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^k\}$ with matrix $\mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_k) \in \mathbb{R}^{m \times k}$. We call $\mathbf{B}$ the basis of Λ and k the rank of the lattice. If it holds that $k = m$ then Λ is called a full-rank lattice. In this thesis, only full-rank latticed are considered. The basis is not unique for a lattice. The determinant of a lattice, however, is an invariant of the lattice and defined as follows. The determinant of a lattice Λ with basis $\mathbf{B}$ is defined by $\det(\Lambda) = \sqrt{|\det\{(\mathbf{B}^\top \mathbf{B})\}|}$. If Λ is a full-ranked lattice then it holds that $\det(\Lambda) = |\det(\mathbf{B})|$. The Hermite factor of a lattice basis describes the quality of the basis, which, for example, may be the output of a basis reduction algorithm such as the Blockwise Korkine-Zolotarev

(BKZ) algorithm [20, 64, 100]. The Hermite factor of a basis $\mathbf{B}$ is given as

$$\delta_0^k = \frac{\|\mathbf{v}\|_2}{\det(\Lambda)^{\frac{1}{k}}},$$

where $\mathbf{v}$ is the shortest non-zero vector in the basis [11]. Throughout this thesis, $\|\mathbf{v}\|_2$ denotes the Euclidean norm of a vector $\mathbf{v}$ and $\|\mathbf{v}\|_\infty$ denotes its infinity norm.

Additionally, the distance between a lattice Λ and a vector $\mathbf{v} \in \mathbb{R}^m$ is defined as $\text{dist}(\mathbf{v}, \Lambda) = \min\{\|\mathbf{v} - \mathbf{x}\|_2 \mid \mathbf{x} \in \Lambda\}$. Furthermore, the i-th successive minimum $\lambda_i(\Lambda)$ of the lattice Λ is defined as the smallest radius r, such that there are i linearly independent vectors of norm at most r in the lattice.

Throughout this thesis we are mostly concerned with q-ary lattices that we define next. A lattice Λ is called q-ary if $q\mathbb{Z}^m \subset \Lambda \subset \mathbb{Z}^m$ for some $m \in \mathbb{Z}_{>0}$. Let $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ and let $\mathbf{A}$ be sampled uniformly random over $\mathbb{Z}_q^{m \times n}$. We define the corresponding q-ary lattices

$$\Lambda_q(\mathbf{A}) = \{\mathbf{x} \in \mathbb{Z}^m \mid \exists \mathbf{y} \in \mathbb{Z}^n : \mathbf{x} = \mathbf{A}\mathbf{y} \mod q\} \text{ and}$$
$$\Lambda_q^\perp(\mathbf{A}) = \left\{\mathbf{x} \in \mathbb{Z}^m \mid \mathbf{A}^T\mathbf{x} = 0 \mod q\right\}.$$

If q is prime, the determinant for a matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$ with $m \geq n$ is given by $\det(\Lambda_q(\mathbf{A})) = q^{m-n}$ and for a matrix with $m < n$ by $\det\left(\Lambda_q^\perp(\mathbf{A})\right) = q^m$ with high probability [39].

2.2 Lattice-Based Security Assumptions

After explaining the basics about lattices, we now define hardness assumptions that are related to lattices and used in this thesis. After explaining some further notations, we first define the shortest and closest vector problem. Afterwards, we turn to the LWE problem. Finally, we define the SIS problem.

We first introduce some notation that is used throughout this thesis. We define for an algorithm $\mathcal{A}$ that the value $y \leftarrow \mathcal{A}(x)$ denotes the output of $\mathcal{A}$ on input x. If $\mathcal{A}$ uses randomness then $\mathcal{A}(x)$ is a random variable. Moreover, for an oracle $\mathcal{O}$ we write $\mathcal{A}^\mathcal{O}$ to indicate that $\mathcal{A}$ has access to that oracle. Similarly, for a probability distribution χ over some set S, $\mathcal{A}^\chi$ denotes that $\mathcal{A}$ can request samples from the probability distribution χ and $x \leftarrow \chi$ or $x \leftarrow_\chi S$ means to sample an element x with probability distribution χ. For convenience we write simply *distribution* instead of *probability distribution*. For a vector $\mathbf{x} \in S^n$, we write $\mathbf{x} \leftarrow \chi^n$ or $\mathbf{x} \leftarrow_\chi S^n$ to denote that every vector entry is sampled with distribution χ. Analogously, we denote sampling each coefficient of a polynomial x in some ring R with distribution χ by $x \leftarrow_\chi R$. Lastly, for a finite set S, we denote sampling the element x uniformly random from S with $x \leftarrow \mathcal{U}(S)$ or simply $x \leftarrow_\$ S$.

2.2.1 The Shortest and Closest Vector Problem

We start by defining the shortest and closest vector problem.

Definition 2.1 (Shortest Vector Problem)**.** *Given a lattice Λ, the Shortest Vector Problem (SVP) is to find a non-zero vector $\mathbf{v} \in \Lambda$ with $\|\mathbf{v}\|_2 = \lambda_1(\Lambda)$.*

There exist many variants of SVP such as the Unique Shortest Vector Problem (uSVP), the decisional approximate SVP (GapSVP), or the approximate Shortest Independent Vector Problem (SIVP). For a formal definition of these variants we refer to [173, 191] or the references therein. Similarly, there exists several variants of the Closest Vector Problem (CVP) which we define next.

Definition 2.2 (Closest Vector Problem)**.** *Given a lattice $\Lambda \subset \mathbb{R}^m$ and $\mathbf{t} \in \mathbb{R}^m$, the CVP is to find a lattice vector $\mathbf{v} \in \Lambda$ such that $\|\mathbf{v} - \mathbf{t}\|_2 \leq dist(\mathbf{t}, \Lambda)$.*

While the mathematical problems above are well-studied, modern cryptography bases their security not directly on these problems. Instead the security of lattice-based primitives relies on the hardness of the LWE or the SIS problem. Part of the allure of cryptographic schemes based on LWE or SIS is that these assumptions enjoy worst-case to average-case reductions from fundamental lattice problems such as SIVP or GapSVP [173]. These reductions suggest that the ability to solve LWE or SIS on randomly chosen instances implies the ability to solve SIVP or GapSVP even on the hardest instances. In the following subsections we define LWE and SIS formally.

2.2.2 The Learning with Errors Problem

Next we recall different variants of the LWE problem that are used in this thesis.

Definition 2.3 (Learning with Errors Problem)**.** *Let $n, m, q > 0$ be integers and χ be a distribution over $\mathbb{Z}$. For $\mathbf{s} \leftarrow_\chi \mathbb{Z}_q^n$ define $\mathcal{D}_{\mathbf{s},\chi}$ to be the distribution that samples $\mathbf{a} \leftarrow_\$ \mathbb{Z}_q^n$ and $e \leftarrow_\chi \mathbb{Z}$ and then returns $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*

- *Given n, $q \geq 2$, and m independent samples from the distribution $\mathcal{D}_{\mathbf{s},\chi}$, the search LWE problem is to find $\mathbf{s}$.*
- *Given n, $q \geq 2$, and m independent samples $(\mathbf{a}_i, b_i)$, the decisional LWE problem is to distinguish for $i = 1, ..., m$ whether $(\mathbf{a}_i, b_i) \leftarrow_\$ (\mathbb{Z}_q^n \times \mathbb{Z}_q)$ or $(\mathbf{a}_i, b_i) \leftarrow \mathcal{D}_{\mathbf{s},\chi}$.*
  In particular, we say that the decisional LWE problem $LWE_{n,m,q,\chi}$ is (t, ϵ)-hard if for any algorithm $\mathcal{A}$, running in time t, it holds that

$$\left| \Pr\left[\mathcal{A}^{\mathcal{D}_{\mathbf{s},\chi}}(\cdot) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{U}(\mathbb{Z}_q^n \times \mathbb{Z}_q)}(\cdot) = 1\right] \right| \leq \epsilon \, .$$

Typically, we refer to the vector $\mathbf{s}$ as the secret and to the e as the error of the LWE instance. We can also write m LWE instances to a secret $\mathbf{s} \in \mathbb{Z}^n$–also called LWE samples–as $(\mathbf{A}, \mathbf{As} + \mathbf{e} \mod q)$ with $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{m \times n}$ and $\mathbf{e} \leftarrow_\chi \mathbb{Z}^m$.

Additionally, the matrix version of the decisional LWE is defined as follows.

Definition 2.4 (Matrix-LWE)**.** *Let $n, n', m, q > 0$ be integers and χ be a distribution over $\mathbb{Z}$. Define $\mathcal{D}_{\mathbf{S},\chi}$ to be the distribution that, for $\mathbf{S} = (\mathbf{s_1}, ..., \mathbf{s_{n'}})$ with $\mathbf{s_1}, ..., \mathbf{s_{n'}} \leftarrow_\chi \mathbb{Z}^n$, samples $\mathbf{a} \leftarrow_\$ \mathbb{Z}_q^n$ and $e_1, ..., e_{n'} \leftarrow_\chi \mathbb{Z}$ and then returns $\left(\mathbf{a}^T, \mathbf{a}^T\mathbf{S} + (e_1, ..., e_{n'})\right) \in \mathbb{Z}_q^n \times \mathbb{Z}_q^{n'}$.*

Given n, $q \geq 2$, and m independent samples $(\mathbf{a_i}, \mathbf{b_i})$, the Matrix decisional LWE (M-LWE) problem is to distinguish for $i = 1, ..., m$ whether $(\mathbf{a_i}, \mathbf{b_i}) \leftarrow_\$ (\mathbb{Z}_q^n \times \mathbb{Z}_q^{n'})$ or $(\mathbf{a_i}, \mathbf{b_i}) \leftarrow \mathcal{D}_{\mathbf{S},\chi}$.

In particular, we say that the M-LWE$_{n,n',m,q,\chi}$ problem is (t, ϵ)-hard if for any algorithm $\mathcal{D}$, running in time t and making at most m queries to the distribution $\mathcal{D}_{\mathbf{S},\chi}$, it holds that

$$\left| \Pr\left[\mathcal{A}^{\mathcal{D}_{\mathbf{s},\chi}}(\cdot) = 1\right] - \Pr\left[\mathcal{A}^{\mathcal{U}\left(\mathbb{Z}_q^n \times \mathbb{Z}_q^{n'}\right)}(\cdot) = 1\right] \right| \leq \epsilon \, .$$

As before, m M-LWE samples to the secret matrix $\mathbf{S} = (\mathbf{s}_1, ..., \mathbf{s}_{n'}) \in \mathbb{Z}^{n \times n'}$ can be written as $(\mathbf{A}, \mathbf{AS} + \mathbf{E}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n'}$ with $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{m \times n}$ and $\mathbf{E} \leftarrow_\chi \mathbb{Z}^{m \times n'}$. We call $(\mathbf{A}, \mathbf{B}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n'}$ a *yes-instance* if there exists an $\mathbf{S} = (\mathbf{s_1}, ..., \mathbf{s_{n'}})$ with $\mathbf{s_1}, ..., \mathbf{s_{n'}} \in \mathbb{Z}_q^n$ and $(\mathbf{A}, \mathbf{B})$ are m M-LWE samples from the distribution $\mathcal{D}_{\mathbf{S},\chi}$. Otherwise, i.e., when $(\mathbf{A}, \mathbf{B}) \leftarrow_\$ \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n'}$, we call $(\mathbf{A}, \mathbf{B})$ a *no-instance*.

The following theorem relates M-LWE and LWE.

Theorem 2.5. *If LWE$_{n,m,q,\chi}$ is $(\epsilon/n', t)$-hard then M-LWE$_{n,n',m,q,\chi}$ is (ϵ, t)-hard.*

Intuitively, the reduction loss exists since an adversary that can solve LWE has n' possibilities to solve M-LWE (see also [24, 52, 174]). The proof follows similar arguments as given in [174].

After defining LWE over so-called standard or unstructured lattices, we now define a variant of LWE over rings, called R-LWE. To state the definition we fix some notation first. Let $\mathcal{R} = \mathbb{Z}[x]/\langle x^n + 1\rangle$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ in the remainder of this thesis if not stated differently. Furthermore, let invertible elements in this ring be represented by $\mathcal{R}^\times$. Additionally, we use $\mathcal{R}_{q,[I]} = \{f \in \mathcal{R}_q \mid f = \sum_{i=0}^{n-1} f_i x^i,\ f_i \in [-I, I]\}$ throughout the thesis. For any polynomial $f = \sum_{i=0}^{n-1} f_i x^i \in \mathcal{R}$ we define the reduction of f modulo q to be $(f \bmod q) = \sum_{i=0}^{n-1} (f_i \bmod q) x^i \in \mathcal{R}_q$. Via the coefficient embedding, we identify a polynomial $f = f_0 + f_1 x + ... + f_{n-1} x^{n-1} \in \mathcal{R}_q$ with its coefficient vector $\mathbf{f} = (f_0, \ldots, f_{n-1})$. Without further mentioning, we denote the coefficient vector of a polynomial $f \in \mathcal{R}_q$ by $\mathbf{f}$.

We define the ring-LWE Problem next.

Definition 2.6 (Ring LWE Problem)**.** *Let $n, q > 0$ be integers and χ be a distribution over $\mathcal{R}$. For $s \leftarrow_\chi \mathcal{R}$ define $\mathcal{D}_{s,\chi}$ to be the distribution that samples $a \leftarrow_\$ \mathcal{R}_q$ and $e \leftarrow_\chi \mathcal{R}$ and then returns $(a, as + e) \in \mathcal{R}_q \times \mathcal{R}_q$.*

- *Given n, $q \geq 2$, and k independent samples from the distribution $\mathcal{D}_{s,\chi}$, the search ring-LWE problem is to find s.*
- *Given n, $q \geq 2$, and k independent tuples (a_i, b_i) the decisional ring-LWE problem is to distinguish for all $i = 1, ..., k$ whether $(a_i, b_i) \leftarrow_\$ (\mathcal{R}_q \times \mathcal{R}_q)$ or $(a_i, b_i) \leftarrow \mathcal{D}_{s,\chi}$. We say that the R-LWE$_{n,k,q,\chi}$ problem is (t, ϵ)-hard if for any algorithm $\mathcal{A}$, running in time t, it holds that*

$$\left| \Pr\left[\mathcal{A}^{\mathcal{D}_{s,\chi}}(\cdot) = 1\right] - \Pr\left[\mathcal{D}^{\mathcal{U}(\mathcal{R}_q \times \mathcal{R}_q)}(\cdot) = 1\right] \right| \leq \epsilon.$$

While we define LWE and R-LWE over an arbitrary distribution χ, the most important instantiation of χ is the centered discrete Gaussian distribution which we define in the following paragraph. The centered discrete Gaussian distribution for $x \in \mathbb{Z}$ with standard deviation σ is given by its probability density function $\mathcal{D}_\sigma = \rho_\sigma(x)/\rho_\sigma(\mathbb{Z})$, where $\sigma > 0$, $\rho_\sigma(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right)$, and $\rho_\sigma(\mathbb{Z}) = 1 + 2\sum_{x=1}^{\infty} \rho_\sigma(x)$. We write $c \leftarrow_\sigma \mathbb{Z}$ to denote sampling of a value c with distribution $\mathcal{D}_\sigma$. For a polynomial $c \in \mathcal{R}$, we write $c \leftarrow_\sigma \mathcal{R}$ to denote sampling each coefficient of c with distribution $\mathcal{D}_\sigma$. Moreover, we denote sampling each coordinate of a matrix $\mathbf{A} \in \mathbb{Z}^{m \times n}$ with distribution $\mathcal{D}_\sigma$ by $\mathbf{A} \leftarrow_\sigma \mathbb{Z}^{m \times n}$ with $m, n \in \mathbb{Z}_{>0}$.

To achieve more compact constructions, the security of cryptographic primitives is sometimes based on sparse variants of the (R-)LWE problem, such as the Decisional Compact Knapsack (DCK) problem which is used as the hardness assumption of the signature scheme by Güneysu et al. [114]. The DCK problem is essentially an R-LWE problem with ternary instantiation.

Definition 2.7 (Decisional Compact Knapsack Problem)**.** *Let $n, q > 0$ be integers with $n = 2^k$ for some $k \in \mathbb{N}_{>0}$, let ψ be the uniform distribution over $\mathcal{R}_{q,[1]}$, and let $s \leftarrow_\psi \mathcal{R}_{q,[1]}$. We define by $\mathcal{D}_{s,\psi}$ the distribution which outputs $(a, as + e) \in \mathcal{R}_q \times \mathcal{R}_q$, where $a \leftarrow_\$ \mathcal{R}_q$ and $e \leftarrow_\psi \mathcal{R}_{q,[1]}$.*

Given n, q, and k independent tuples (a_i, b_i) the DCK problem is to distinguish for $i = 1, ..., k$ whether $(a_i, b_i) \leftarrow_\$ (\mathcal{R}_q \times \mathcal{R}_q)$ or $(a_i, b_i) \leftarrow \mathcal{D}_{s,\psi}$.

2.2.3 The Short Integer Solution Problem

After defining variants of the LWE problem, we now define the SIS problem. Prominent examples of signature schemes whose security is based on (R-)SIS are BLISS [81] or the Bai-Galbraith signature [24].

Definition 2.8 (Short Integer Solution Problem)**.** *Given a modulus q, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ with $m \geq n$, and a real constant $\nu \in \mathbb{R}_{\geq 0}$, the SIS problem is to find a non-zero vector $\mathbf{u} \in \mathbb{Z}^m$ such that $\mathbf{Au} = 0 \mod q$ and $\|\mathbf{u}\|_2 \leq \nu$.*

As for LWE, there also exists a ring-SIS problem which is defined as follows.

Definition 2.9 (Ring SIS Problem)**.** *Let $n, q > 0$ be integers and $n = 2^l$ for some $l \in \mathbb{N}_{>0}$ and χ be a distribution over $\mathcal{R}$. Given $a_1, ..., a_k \leftarrow_\$ \mathcal{R}_q$, a modulus q, and a real constant $\nu \in \mathbb{R}_{\geq 0}$ then the ring-SIS problem (R-SIS) is to find a polynomial $s \in \mathcal{R}_q$ such that $a_1 s + ... + a_k s = 0 \mod q$ and $\|s\|_2 \leq \nu \in \mathbb{R}$.*

As in the case of LWE, sometimes special instantiations are used to gain more efficient constructions (in run-time and size). For example, the signature scheme BLISS is instantiated with an version of SIS defined over so-called NTRU-lattices as defined next.

Definition 2.10. *(NTRU SIS Problem) Let $n = 2^l$, $l > 0$, $q \in \mathbb{Z}_{>0}$. Moreover, let χ_f and χ_g be distributions over R_q. Furthermore, let $f \leftarrow_{\chi_f} R_q^\times$ and $g \leftarrow_{\chi_g} R_q$. Given $h = gf^{-1} \in R_q$, a modulus q, and a real constant $\nu \in \mathbb{R}_{\geq 0}$ then the NTRU SIS problem is to find $v_1, v_2 \in R_q$ such that $av_1 + v_2 = 0 \mod q$ and $\|(\mathbf{v}_1, \mathbf{v}_2)^T\|_2 \leq \nu$.*

2.3 Quantum Computation and Scenarios

Next, we introduce quantum computation knowledge and elaborate on our categorization of quantum scenarios used in this thesis. A standard text for a more complete explanation has been given for example by Nielsen and Chuang [167].

Quantum States and Quantization of Classical Algorithms. Let $\mathcal{H}$ be a complex Hilbert space with inner product $\langle y|x\rangle$ of vectors $|x\rangle, |y\rangle \in \mathcal{H}$. A quantum state is an element in $\mathcal{H}$ of norm 1. Let $\{|x\rangle\}_x$ be an orthonormal basis for $\mathcal{H}$, then we can represent any (pure) quantum state $|y\rangle$ as $|y\rangle = \sum_x \psi_x |x\rangle$ where ψ_x are complex numbers, also called *amplitudes*, such that $|y\rangle$ has norm 1, i.e., $\sum_x |\psi_x|^2 = 1$. We say that $|y\rangle$ is a quantum superposition of basis states $|x\rangle$.

Operations on elements of $\mathcal{H}$, i.e., quantum operations, are represented by unitary transformations $\mathbf{U}$. Hence, quantum operations (prior to measurement) are reversible. This impacts the quantization of classical operations as follows. Let A be a classical algorithm with input $x \in \{0,1\}^a$ and output $y \in \{0,1\}^b$. Moreover, let $\{0,1\}^a \times \{0,1\}^b \to \{0,1\}^a \times \{0,1\}^b : (x,t) \mapsto (x, t \oplus A(x))$ be a classical reversible mapping. The corresponding unitary transformation $\mathbf{A}$ acting linearly on quantum states is given by $\mathbf{A} : \sum_{x,t} \psi_{x,t} |x,t\rangle \mapsto \sum_{x,t} \psi_{x,t} |x, t \oplus A(x)\rangle$, where we use the usual notation $|x,y\rangle = |x\rangle \otimes |y\rangle$ with tensor product $\otimes$. We may add a workspace register to the input and the output registers to allow for more generality. Thus, the quantized classical algorithm is given as

$$\mathbf{A} : \sum_{x,t,z} \psi_{x,t,z} |x,t,z\rangle \mapsto \sum_{x,t,z} \psi_{x,t,z} |x, t \oplus A(x), z\rangle .$$

Examples for such a quantization of classical algorithms are quantum oracles such as the quantum random oracle or the quantum signing oracle defined in Section 2.4.

Mixed States, Channels, Density Matrices, and the Trace Norm. Until now we only defined pure quantum states. There exists, however, a more general notation called the *density matrix formalism* which covers also *mixed* states which cannot be represented as pure states. Instead, they are represented by their density matrix ρ and can be written as a probability distribution over sets of pure states $|x\rangle$, i.e., $\rho = \sum_x p_x |x\rangle \langle x|$ with $p_x \in \mathbb{R}$ and $\sum_x |p_x|^2 = 1$.

After introducing mixed states, we can now explain unitary and classical channels. A channel is a physically realizable map on mixed quantum states–the quantum analogue of a stochastic map applied to a classical probability distribution. Now, unitary channels preserve purity of input states and act on classical basis states in *superposition*, while classical channels measure input states in the standard basis and output a probabilistic mixture of classical basis states. In this thesis, hash oracles are often viewed as unitary channels while signing oracles are viewed as classical channels (except for the signing oracles in Section 5).

Additionally, in this thesis we use the trace norm $\|\cdot\|_{\mathrm{tr}}$ (also known as the Schatten-1 norm) explained next. It is a matrix norm generalizing the concept of statistical distance from probability distributions to quantum states. For example, the quantity $\|\rho - \rho'\|_{\mathrm{tr}}$ captures the physically observable difference between two quantum states represented by their density matrices ρ, ρ'. Formally, the trace norm of a matrix ρ is defined as $\|\rho\|_{\mathrm{tr}} = \mathrm{tr}\left(\sqrt{\rho\rho^\dagger}\right)$ where $\rho^\dagger$ is the *adjoint* of ρ and $\mathrm{tr}\,()$ is the *trace* of ρ, i.e., the sum of the diagonal elements of ρ.

Categorization of Quantum Scenarios. Moving further, we define quantum scenarios that exist in the literature. The security of lattice-based primitives is proven against adversaries with different quantum capabilities. For example, the schemes presented in [26, 69, 81, 105, 222] have been proven secure against fully classical adversaries that have no access to quantum computers. Other constructions such as [45, 51, 52, 123, 139] have been proven to be secure against adversaries that have local quantum computing power, i.e., the adversary can perform all computations over public information such as the encryption of a message, the verification of a signature, or the evaluation of a hash function on a quantum computer. To account for the additional quantum power in case random oracles are used, the QROM has been introduced [45] where adversaries can query the (quantum) random oracle in superposition. Additionally, some constructions such as [48, 49, 99, 220] are proven to be secure against fully quantum adversaries that are able to make all computations on quantum computers and access all

oracles in superposition. In this thesis we refer to the different quantum scenarios as follows.

Fully classical: All parties at all times use only a classical computer, and access all oracles classically.

Post-quantum: The adversary at all times can use a quantum computer. However, the honest participants in the system only ever use their secret keys on classical computers, so oracles such as the signing or decryption oracle are always accessed classically.

Fully quantum: The adversary at all times can use a quantum computer and has quantum access to all oracles.

Security Reductions in the QROM. As explicated by Boneh et al. [45], a variety of techniques used to prove security in the Random Oracle Model (ROM) [34] are generally not known to be applicable in the QROM. Such techniques are for example adaptive programmability of the random oracle or rewinding. However, if a reduction in the ROM is so-called *history-free* [45], it *lifts* to the QROM. For example, Boneh et al. show that the (tight) security reduction of the lattice-based signature scheme by Gentry, Peikert, and Vaikuntanathan [104], also called the GPV scheme, that is given in the ROM, also holds in the QROM. Recently, the framework by Boneh et al. was applied to Fiat-Shamir signature schemes by Kiltz, Lyubashevsky, and Schaffner [139]. An application of their framework has been, for example, the security reduction of the lattice-based signature scheme Dilithium [82]. Another framework used to show post-quantum security has been proposed by Song [208]. Moreover, security reductions of signature schemes or KEMs that are given in the *standard model* should also hold in the QROM, since these reductions, such as [53], do not rely on any assumptions about a random oracle.

After recalling existing quantum scenarios, we define cryptographic primitives and their respective security notions in the different quantum scenarios next.

2.4 Cryptographic Primitives and Their Security Definitions

In this section we recall the definition of digital signature schemes and KEMs. Additionally, we recall their respective security notions.

2.4.1 Digital Signature Schemes

We review the definition of signature schemes and existential unforgeability against classical and quantum adversaries next.

Definition 2.11 (Signature Scheme)**.** *A digital signature scheme $\mathcal{S}$ defined over the message space $\mathcal{M}$, the public key space $\mathcal{PK}$, the secret key space $\mathcal{SK}$, and the signature space $\mathcal{S}$, is a tuple $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ of algorithms that are described as follows.*

- $\mathsf{KeyGen}() \to (\mathsf{sk}, \mathsf{pk})$ *is a probabilistic algorithm that returns a secret or signing key* $\mathsf{sk} \in \mathcal{SK}$ *and a public or verification key* $\mathsf{pk} \in \mathcal{PK}$*.*
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \mathsf{s}$ *is a probabilistic algorithm which takes as input a secret key* $\mathsf{sk} \in \mathcal{SK}$ *and a message* $\mathsf{m} \in \mathcal{M}$*, and outputs a signature* $\mathsf{s} \in \mathcal{S}$*.*
- $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \mathsf{s}) \to 0$ *or* -1 *is a deterministic algorithm which takes as input a public key* pk*, a message* m*, and a signature* s*, and returns a bit* $b \in \{0, -1\}$*. If* $b = 0$*, we say that the algorithm accepts, otherwise we say that it rejects the signature* s *for message* m*.*

If a proof for $\mathcal{S}$ is being given in the ROM, we use $\mathcal{H}$ to denote the space of functions from which the random hash function is randomly sampled.

Moreover, we say $\mathcal{S}$ is ε-correct if, for every message m in the message space, we have that

$$\Pr\left[\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \mathsf{s}) = 0 : (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(), \mathsf{s} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \text{ for } \mathsf{m} \in \mathcal{M}\right] \geq 1 - \varepsilon,$$

where the probability is taken over the randomness of the probabilistic algorithms. We say that $\mathcal{S}$ is correct if $\varepsilon = 0$.

The standard security requirement for signature schemes, namely Existential Unforgeability under Chosen-Message Attack (EUF-CMA), dates back to Goldwasser, Micali, and Rivest [107]: The adversary can obtain q_S signatures via signing oracle queries, and must output one valid signature on a message not queried to the oracle. The corresponding experiment involving an adversary $\mathcal{A}$ against a signature scheme $\mathcal{S}$ is given in Figure 2.1. Our depicted EUF-CMA experiment grants $\mathcal{A}$ access to a random oracle $\mathcal{O}_H$, i.e., we consider the security game in the ROM. In the ROM, a random function H is sampled uniformly from the space of all such functions $\mathcal{H}$ at the start of the experiment. We define the random oracle in Figure 2.2. In the standard model the adversaries is not allowed to query the random oracle. We define EUF-CMA security formally in the following definition.

Definition 2.12 (EUF-CMA Security)**.** *Given the experiment in Figure 2.1, we say that a signature scheme $\mathcal{S}$ is (t, q_S, q_H, ϵ)-*$\mathsf{EUF\text{-}CMA}$ *secure (in the fully classical setting) if for every classical adversary $\mathcal{A}$ which runs in time t and poses at most q_s (classical) queries to the signing oracle and q_h (classical) queries to the random oracle the advantage is*

$$Adv_{\mathcal{S}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}_{\mathcal{S}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A}) = 0\right] \leq \epsilon.$$

$\underline{\text{Expt}_{\mathcal{S}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A})}$:
1: $H \leftarrow_{\$} \mathcal{H}$
2: $q_H \leftarrow 0$, $q_S \leftarrow 0$
3: $Q_S \leftarrow \{\}$
4: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
5: $(\mathsf{m}^*, \mathsf{s}^*) \leftarrow \mathcal{A}^{\mathcal{O}_S, \mathcal{O}_H}(\mathsf{pk})$
6: if $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}^*, \mathsf{s}^*) = 0 \wedge (\mathsf{m}^*, *) \notin Q_S$:
7: return 1
8: else
9: return 0

$\underline{\text{Classical signing oracle } \mathcal{O}_S(\mathsf{m})}$:
1: $q_S \leftarrow q_S + 1$
2: $\mathsf{s} \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m})$
3: $Q_S \leftarrow Q_S \cup (\mathsf{m}, \mathsf{s})$
4: return s

Figure 2.1: EUF-CMA security experiment in the ROM where the adversary $\mathcal{A}$ returns one valid message-signature pair

$\underline{\text{Classical random oracle } \mathcal{O}_H(x)}$:	$\underline{\text{Quantum random oracle } \mathcal{O}_H(\sum_{x,t,z} \psi_{x,t,z} \lvert x,t,z\rangle)}$:
1: $q_H \leftarrow q_H + 1$	1: $q_H \leftarrow q_H + 1$
2: return $H(x)$	2: return $\sum_{x,t,z} \psi_{x,t,z} \lvert x, t \oplus H(x), z\rangle$

Figure 2.2: Definition of the classical and quantum random oracle

The security notion of strong Existential Unforgeability under Chosen-Message Attack (sEUF-CMA) is defined similarly to EUF-CMA security except for the following difference. The security game against an sEUF-CMA attacker $\mathcal{A}$ returns 1 even if $\mathcal{A}$ forges a second signature to a message it queried to the signing oracle. In particular that means instead of line 6 in Figure 2.1 it is enough that $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}^*, \mathsf{s}^*) = 0 \wedge (\mathsf{m}^*, \mathsf{s}^*) \notin Q_S$.

We emphasize that the above security definition of EUF-CMA is well-defined in the fully classical scenario and the post-quantum scenario which allows adversaries that have local quantum power as defined in Section 2.3. The quantum analogue to the classical random oracle is called quantum random oracle [45]. We depict the quantized random oracle in Figure 2.2. Standard formalism for a quantum oracle for a hash function $\mathsf{H} : X \to T$ specifies that the user has access to a unitary channel that applies the linear map $|x, t, z\rangle \mapsto |x, t \oplus \mathsf{H}(x), z\rangle$ on standard basis states. It is understood that the range T of H has a $\oplus$ operation defined on it with the property that $t_1 \oplus t_2 \oplus t_2 = t_1$ for all $t_1, t_2 \in T$, so that the unitary channel is its own inverse. Typically, one simply imagines that elements of T are written as binary strings, and the $\oplus$ operation is the bitwise exclusive-OR.

In the post-quantum setting, EUF-CMA security is defined as in Definition 2.12

except that the above classical adversary $\mathcal{A}$ is changed to a quantum adversary $\mathcal{A}$. Moreover, the adversary is allowed to ask at most q_h (quantum) queries to the quantum random oracle.

In this thesis we are mostly concerned with post-quantum security in the QROM. In Chapter 5, however, we also allow more powerful quantum adversaries that might query the signing oracle in superposition as in the fully quantum scenario. Therefore, we describe a quantized EUF-CMA experiment next. The aforementioned, traditional definition of the EUF-CMA experiment given in Figure 2.1 cannot be directly quantized: If the adversary is allowed to query oracles in superposition, we cannot restrain the adversary's forgery to be on a new message since the experiment cannot keep a copy of messages queried to the signing oracle for later checking. An equivalent formulation of the security experiment in the classical setting demands that the adversary outputs $q_S + 1$ valid signatures on distinct messages. Boneh and Zhandry [49] used this formulation to give a quantum analogue of EUF-CMA which involves a fully quantum adversary. We depict the corresponding security experiment and the quantized sign oracle in Figure 2.3. The definition of EUF-CMA in the fully quantum setting follows easily from this.

$\underline{\mathrm{Expt}_{\mathcal{S}}^{\mathsf{EUF\text{-}CMA}}(\mathcal{A})}$:

1: $H \leftarrow_{\$} \mathcal{H}$
2: $q_H \leftarrow 0,\ q_S \leftarrow 0$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow_{\$} \mathsf{KeyGen}()$
4: $((\mathsf{m}_1^*, \mathsf{s}_1^*), \ldots, (\mathsf{m}_{q_S+1}^*, \mathsf{s}_{q_S+1}^*)) \leftarrow \mathcal{A}^{\mathcal{O}_S, \mathcal{O}_H}(\mathsf{pk})$
5: if $\forall\ i, j \in [1, q_S + 1], i \neq j$: $(\mathsf{Verify}(\mathsf{pk}, \mathsf{m}_i^*, \mathsf{s}_i^*) = 0) \wedge \left(\mathsf{m}_i^* \neq \mathsf{m}_j^*\ \right)$:
6: return 1
7: else
8: return 0

Quantum signing oracle $\mathcal{O}_S(\sum_{\mathsf{m},t,z} \psi_{\mathsf{m},t,z} \left|\mathsf{m}, t, z\right\rangle)$:

1: $q_S \leftarrow q_S + 1$
2: $r \leftarrow_{\$} \mathcal{R}$
3: return $\sum_{\mathsf{m},t,z} \psi_{\mathsf{m},t,z} \left|\mathsf{m}, t \oplus \mathsf{Sign}(\mathsf{sk}, \mathsf{m}; r), z\right\rangle$

Figure 2.3: EUF-CMA security experiment in the (Q)ROM where the adversary $\mathcal{A}$ returns $q_S + 1$ valid signatures

It is important to note that we do not define the security of a scheme in terms of a security parameter λ. Instead we view λ as a system parameter of the scheme that defines the targeted security. In particular, we assume that the computation of 2^λ is infeasible and accordingly that a probability of $2^{-\lambda}$ is *negligible*.

2.4.2 Key Encapsulation Mechanisms

Moving forward, we now elaborate on the definition of KEMs and their security notions.

Definition 2.13 (Key Encapsulation Mechanism)**.** *A KEM $\mathcal{K}$ defined over the message space $\mathcal{M}$, the public key space $\mathcal{PK}$, the secret key space $\mathcal{SK}$, and the key space $\mathcal{K}$, is a triple of algorithms $\mathcal{K} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ defined as follows.*

- $\mathsf{KeyGen}() \to (\mathsf{sk}, \mathsf{pk})$ *is a probabilistic algorithm that returns a secret key* $\mathsf{sk} \in \mathcal{SK}$ *and a public key* $\mathsf{pk} \in \mathcal{PK}$*.*
- $\mathsf{Encaps}(\mathsf{pk}) \to (c, \kappa)$ *is a probabilistic algorithm that takes as input a public key* pk *and outputs a ciphertext c as well as a key $\kappa \in K$.*
- $\mathsf{Decaps}(\mathsf{sk}, c) \to \kappa$ *or* $\perp$ *is a deterministic algorithm that takes as input a secret key* $\mathsf{sk} \in \mathcal{SK}$ *and a ciphertext c and returns a key $\kappa \in K$ or $\perp$, denoting failure.*

As before, we use $\mathcal{H}$ to denote the space of functions from which the random hash function is randomly sampled if a proof for $\mathcal{K}$ is being given in the ROM.

A KEM $\mathcal{K}$ is ε-correct if for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$ and $(c, \kappa) \leftarrow \mathsf{Encaps}(\mathsf{pk})$, it holds that $\Pr[\mathsf{Decaps}(\mathsf{sk}, c) \neq \kappa] \leq \varepsilon$. We say it is correct if $\varepsilon = 0$.

The security of KEMs can be defined in terms of the indistinguishability of the keys against chosen-plaintext (IND-CPA) and chosen-ciphertext (IND-CCA) adversaries. In the traditional IND-CPA experiment of KEMs, a challenger generates keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$, computes $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$, and samples κ_1^* uniformly at random from the key space K and a random bit b. The adversary $\mathcal{A}$ is given c^*, κ_b^*, and pk, and is asked to output a bit b', indicating whether it believes it received the key corresponding to c^* or a random value. The security experiment returns 1 if $b' = b$. Otherwise, 0 is returned. We depict the security experiment of IND-CPA in Figure 2.4 and define IND-CPA security formally next.

Definition 2.14 (IND-CPA Security)**.** *Given the experiment in Figure 2.4, we say that a KEM $\mathcal{K}$ is (t, q_H, ϵ)-*$\mathsf{IND\text{-}CPA}$ *secure in the fully classical (resp., in the post-quantum or the fully quantum setting) if for every classical (resp., quantum) adversary $\mathcal{A}$ with runtime at most t and asking at most q_H queries to the random oracle (resp., quantum random oracle) the advantage is*

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) = \left| \Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A}) = 1\right] - \frac{1}{2} \right| \leq \epsilon.$$

In the traditional IND-CCA experiment the adversary $\mathcal{A}$ additionally has access to a decapsulation oracle, which returns the decapsulation of any ciphertext not equal to the challenge ciphertext c^*. We depict the security experiment of IND-CCA in Figure 2.5. The definition of IND-CCA given next, is similar to Definition 2.14.

$\underline{\text{Expt}_{\mathcal{K}}^{\mathsf{IND\text{-}CPA}}(\mathcal{A})}$:
1: $H \leftarrow_{\$} \mathcal{H}$
2: $q_H \leftarrow 0$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
4: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
5: $\kappa_1^* \leftarrow_{\$} K$
6: $b \leftarrow_{\$} \{0,1\}$
7: $b' \leftarrow \mathcal{A}^{\mathcal{O}_H}(\mathsf{pk}, c^*, \kappa_b^*)$
8: return $[b = b']$

Figure 2.4: IND-CPA security experiment in the (Q)ROM

Definition 2.15 (IND-CCA Security)**.** *Given the experiment in Figure 2.5, we say that a KEM $\mathcal{K}$ is (t, q_H, q_D, ϵ)-*$\mathsf{IND\text{-}CCA}$ *secure in the fully classical (resp., post-quantum setting) if for every classical (resp., quantum) adversary $\mathcal{A}$, running in at most time t, asking at most q_H queries to the random oracle (resp., quantum random oracle), and at most q_D queries to the decapsulation oracle, the advantage is*

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) = \left| \Pr\left[\text{Expt}_{\mathcal{K}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A}) = 1\right] - \frac{1}{2} \right| \leq \epsilon.$$

In the fully quantum setting, the q_D queries to the decapsulation oracle can be posed in superposition.

$\underline{\text{Expt}_{\mathcal{K}}^{\mathsf{IND\text{-}CCA}}(\mathcal{A})}$:
1: $H \leftarrow_{\$} \mathcal{H}$
2: $q_D \leftarrow 0$, $q_H \leftarrow 0$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
4: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
5: $\kappa_1^* \leftarrow_{\$} K$
6: $b \leftarrow_{\$} \{0,1\}$
7: $b' \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_D}(\mathsf{pk}, c^*, \kappa_b^*)$
8: return $[b = b']$

$\underline{\mathsf{Decaps}^{\perp}(\mathsf{sk}, c, c^*)}$:
1: if $c = c^*$:
2: return $\perp$
3: else
4: return $\mathsf{Decaps}(\mathsf{sk}, c)$

$\underline{\mathcal{O}_D(c)}$:
1: $q_D \leftarrow q_D + 1$
2: return $\mathsf{Decaps}^{\perp}(\mathsf{sk}, c, c^*)$

$\underline{\mathcal{O}_D(\sum_{c,t,z} \psi_{c,t,z} \left|c, t, z\right\rangle)}$:
1: $q_D \leftarrow q_D + 1$
2: return $\sum_{c,t,z} \psi_{c,t,z} \left|c, t \oplus \mathsf{Decaps}^{\perp}(\mathsf{sk}, c, c^*), z\right\rangle$

Figure 2.5: IND-CCA security experiment in the (Q)ROM

The security of KEMs can also be defined in the notion of one-way security as

we explain next. During the one-way security experiment of KEMs, the adversary's task is to fully recover the session key, not just distinguish it from random as in the indistinguishability notions explained above. We can similarly consider the chosen-plaintext and chosen-ciphertext scenarios. Figure 2.6 shows the corresponding security experiments for One-Way Chosen-Plaintext Attacks (OW-CPA) and One-Way Chosen-Ciphertext Attacks (OW-CCA). We formally define OW-CPA first. Subsequently we give the definition of OW-CCA.

Definition 2.16 (OW-CPA Security)**.** *Given the experiment in Figure 2.6, we say that a KEM $\mathcal{K}$ is (t, q_H, ϵ)-$\mathsf{OW\text{-}CPA}$ secure in the fully classical (resp., in the post-quantum or the fully quantum setting) if for every classical (resp., quantum) adversary $\mathcal{A}$ with runtime at most t and asking at most q_H queries to the random oracle (resp., quantum random oracle) the advantage is*

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{OW\text{-}CPA}}(\mathcal{A}) = \left|\Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{OW\text{-}CPA}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right| \leq \epsilon.$$

$\underline{\mathrm{Expt}_{\mathcal{K}}^{\mathsf{OW\text{-}CPA}}(\mathcal{A})}$:
1: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{K}}$
2: $q_H \leftarrow 0$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
4: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
5: $\kappa' \leftarrow \mathcal{A}^{\mathcal{O}_H}(\mathsf{pk}, c^*)$
6: return $[\kappa^* = \kappa']$

$\underline{\mathrm{Expt}_{\mathcal{K}}^{\mathsf{OW\text{-}CCA}}(\mathcal{A})}$:
1: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{K}}$
2: $q_D \leftarrow 0$, $q_H \leftarrow 0$
3: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
4: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
5: $\kappa' \leftarrow \mathcal{A}^{\mathcal{O}_H, \mathcal{O}_D}(\mathsf{pk}, c^*)$
6: return $[\kappa^* = \kappa']$

Figure 2.6: OW-CPA (left) and OW-CCA (right) security experiment in the (Q)ROM

Definition 2.17 (OW-CCA Security)**.** *Given the experiment in Figure 2.6, we say that a KEM $\mathcal{K}$ is (t, q_H, q_D, ϵ)-$\mathsf{OW\text{-}CCA}$ secure in the fully classical (resp., post-quantum setting) if for every classical (resp., quantum) adversary $\mathcal{A}$, running in at most time t, asking at most q_H queries to the random oracle (resp., quantum random oracle), and at most q_D queries to the decapsulation oracle, the advantage is*

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{OW\text{-}CCA}}(\mathcal{A}) = \left|\Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{OW\text{-}CCA}}(\mathcal{A}) = 1\right] - \frac{1}{2}\right| \leq \epsilon.$$

In the fully quantum setting, the q_D queries to the decapsulation oracle can be posed in superposition.

For the corresponding definitions of IND-CPA, IND-CCA, OW-CPA, and OW-CCA security for PKE we refer to [123].

2.5 Implementation Attacks

The mathematical security of cryptographic algorithms is defined in Section 2.4. In the described mathematical security models a cryptographic algorithm is regarded as a black-box where only input, output, and the description of the algorithm are known. When an algorithm is executed in practice, however, additional information can be obtained since the algorithm is executed on a physical device. Attacks that take advantage of this additional information are called physical attacks. Whether and which information can be obtained depends mostly on the respective implementation of the cryptographic algorithms. Therefore, physical attacks are also called implementation attacks. In the following paragraphs we introduce the most important implementation attacks.

Implementation attacks can be categorized by their order as defined next. The order of an attack depends on the number of different exploited points of attack during the execution of the algorithm. First-order attacks exploit only one point of attack, while higher-order attacks combine several points of attack during the execution. We refer to, e.g., [77, 153] for more information.

Implementation attacks can also be categorized by the attacker's power into active (also known as fault attacks) or passive attacks (also known as side-channel attacks). We explain the difference between side-channel and fault attacks and describe known attacks of the respective categories next.

2.5.1 Side-Channel Attacks

Side-channel attacks exploit information that is obtained during the monitoring of executed algorithms. We describe side-channel attacks that are most relevant [144, 172, 209] for lattice-based signature schemes, namely cache, timing, power, and electromagnetic side channels. We refrain from describing side channel attacks such as acoustic, optical, or thermal attacks but refer to [85] for the respective descriptions. As to the best of our knowledge, no such attacks against lattice-based scheme have been presented so far. We refer to [209] for a systematic classification of side channels and for an extensive overview on existing literature.

2.5.1.1 Cache Side Channels

Software implementations that run on systems using memory caches might be particularly vulnerable to implementation attacks since processor caches are a rich source of side channels. A cache is a small piece of memory that stores selected entries from the main memory for quick access by the Central Processing Unit (CPU). Inside the cache, the memory entries are stored in so-called cache lines. The sequence of cache lines in a cache is partitioned into cache sets. In a k-way

set-associative cache, each cache set consists of k cache lines. A cache comes with a strategy for replacing entries if the cache is full. A popular strategy is to replace the Least Recently Used (LRU) entry. Variants of the LRU strategy are used, e.g., in Intel processors [2].

If the CPU accesses an entry in the memory, a cache hit (if the entry is stored in the cache) or a cache miss (if the entry is not stored in the cache) occurs. A cache-side-channel vulnerability exists if the interaction between a program and the cache depends on secret information, e.g., on a cryptographic key. In this case, an attacker, observing aspects of this interaction, might learn secret information. By monitoring the behavior of the cache, an attacker might be able to exploit run-time [142], memory consumption [129], or power consumption [143].

The idea of cache attacks was first mentioned by Kelsey et al. [138] and later on exploited to weaken the security of DES [73] by Page [170]. Groot Bruinderink et al. [110] presented the first attack against a lattice-based signature scheme and broke the scheme BLISS [81] using cache side channels of the Gaussian sampling during the signature generation.

2.5.1.2 Timing Side Channels

During a timing-side-channel attack, the execution time of (subroutines of) the targeted algorithms depending on different input is observed and analyzed. For example, if secret-dependent branching occurs in the algorithm and the time to compute the branches differs, an attacker might detect which branch was taken depending on the resulting run-time and, thus, recover the secret value.

The first timing side channel was presented by Kocher in 1996 [142]. Silverman and Whyte [207] presented the first timing-side-channel attack against lattice-based cryptography, namely against NTRUEncrypt [121].

2.5.1.3 Power Side Channels

During a power-side-channel attack the power consumption of (subroutines of) the targeted algorithms depending on different input is observed and analyzed. Power attacks are categorized depending on their analysis: Differential Power Analysis (DPA) [143] uses statistical analysis tailored specifically to the algorithm and Simple Power Analysis (SPA) [143] visually examines the graph of the power traces [156].

The first power analysis was presented by Kocher, Jaffe, and Jun in 1999 [143]. In 2010 the first power attack against the lattice-based scheme NTRUEncrypt was presented [150]. In 2017, two works analyzed lattice-based cryptography with regard to power side channels [91, 184].

2.5.1.4 Electromagnetic Side Channels

During an electromagnetic side-channel attack, the electromagnetic emanation of devices (or wires within the devices that carry current) during the execution of targeted algorithms depending on different input is measured and analyzed. As for power attacks, electromagnetic attacks are categorized in Simple Electromagnetic Attacks (SEMA) and differential electromagnetic attacks depending on the method used to analyze the obtained data.

Electromagnetic attacks were already discussed in classified documents, mentioned by Kocher et al. [143], and used to reconstruct content shown on a monitor by van Eck [211] in the last century (see [4, 197] for a historical overview). The first attack presented to the public was by Quisquater [186] during the rump session of EUROCRYPT 2000 and followed by [101, 187] in 2001. The first electromagnetic attacks against lattice-based cryptography have been given by Espiteau et al. [91] who have presented an attack against the signature scheme BLISS [81] and by Primas, Pessl, and Mangard [184] who have analyzed the Number Theoretic Transform (NTT) used for polynomial multiplication in the majority of ideal-lattice-based schemes.

2.5.2 Fault Attacks

After explaining passive implementation attacks, we turn to active implementation attacks that are also called fault attacks. Fault attacks disturb the execution of an algorithm to extract secret information or to force a behavior. The goal of most fault attacks is to recover the secret key used during the computation [144]. For example, recovering the secret signing key enables the attacker to generate valid signatures of any messages. In other scenarios, however, the attacker's goal might be to force the acceptance of a possibly invalid signature, e.g., to install malicious software updates [204]. To achieve this goal the attacker does not necessarily need to reveal the secret key but it is enough to force the verification algorithm to accept the invalid signature by fault injection.

A fault injection can be zeroing or randomizing a value, or skipping operations as defined by Rauzy and Guilley [190, Definition 1]:

Zeroing is a fault injection that results in setting a value or a part thereof to zero.

Randomizing is a fault injection that results in setting a value or a part thereof to a random value.

Skipping is a fault injection that results in skipping any number of consecutive instructions.

Another injection fault is to flip single bits. Bit flips are one of the first examples of recorded faults, e.g., May and Woods [157] reported about single-bit errors caused

by radio-active decay in 1978. However, bit-flipping attacks assume very powerful adversaries [212] and bit flips can be viewed as randomizing faults.

Fault attacks can be carried out using, e.g, glitching attacks such as clock glitching or power glitching, light attacks, or magnetic attacks [29].

The first theoretical fault attack was presented by Boneh, DeMillo, and Lipton [46]. The first fault attacks on lattice-based cryptography [12, 130] targeted NTRUEncrypt and NTRUSign [121]. Espiteau et al. [91] investigated fault attacks on the signature schemes GLP [114], BLISS [81], PassSign [122], and ring-TESLA. Concurrently, Bindel, Krämer, and Buchmann presented different fault attacks on ring-TESLA, GLP and BLISS [B7]. Moreover, in [111] a practical fault attack on deterministic signature schemes has been performed.

This chapter has thus focused on explaining the necessary background information for this thesis. We now present our lattice-based signature schemes in the next chapter.

3 The Signature Schemes TESLA and qTESLA

Digital signatures are essential for cybersecurity. For example, they provide proofs of authenticity for billions of software downloads daily. In modern cryptography, the security of signature schemes is guaranteed as long as a particular mathematical problem is computationally hard [134]. A very convincing approach to ensure this guarantee is to use security reductions–proofs that an attacker with predefined power cannot forge a signature as long as the problem is intractable. Such reductions can be asymptotic or explicit. Explicit reductions allow the relation of the scheme's parameters to a concrete problem instance. Another desirable property of reductions is tightness [60]: If parameters are selected taking into account the explicit reductions described above, tight reductions lead to smaller parameters and thus to more efficient instantiations [59, 60].

As progress towards constructing quantum computers capable of executing Shor's algorithm [206] continues, it has become desirable to construct an efficient signature scheme and to prove an explicit security reduction from a quantum-hard problem to the scheme which also holds in the presence of powerful quantum adversaries.

In this chapter, we present the post-quantum secure signature schemes TESLA and qTESLA. We prove tight and explicit security reductions from the (R-)LWE problem to TESLA and qTESLA in the QROM [45]. This allows us to choose efficient and quantum secure parameters according to our security reductions. We then report on experimental results to demonstrate the efficiency of our schemes and their respective instantiations.

The sections are divided as follows: Section 3.1 describes our schemes and all existing parameters. We then prove our novel quantum security reductions in Section 3.2. Based on the results, we select parameters of different (quantum) security levels in Section 3.3. Moving further, Section 3.4.2 presents experimental results based on implementations targeting different platforms. The last Section 3.5 finally discusses the extent of our solution and compares TESLA and qTESLA with other signature schemes from the literature.

This chapter is based on the publications [B2] (PQCrypto 2017) and [B1] (AFRICACRYPT 2016), and on the submission of qTESLA [B4] to NIST's post-quantum project [164]. The quantum security reduction of TESLA was proven in collaboration with Edward Eaton and Gus Gutoski. The benchmarks of TESLA and qTESLA are obtained from implementations by Akleylek, Alkim, Barreto, Deng, Longa, Paquin, Ricardini, Szefer, Tian, Szefer, Tian, and Zanon.

3.1 Description of the Signature Schemes

The signature schemes described in this section are the result of a long line of research. The first work in this line has been the signature scheme proposed by Bai and Galbraith [24] which has been based on the Fiat-Shamir construction of Lyubashevsky [154]. The scheme by Bai and Galbraith is constructed over standard lattices and comes with a (non-tight) security reduction from the LWE and the SIS problems in the ROM. Dagdelen et al. [69] presented improvements and the first implementation of the Bai-Galbraith scheme. The scheme was subsequently studied as a deterministic variant under the name TESLA [B2]. A variant of TESLA over ideal lattices was derived under the name ring-TESLA [B1]. Since then, there have appeared subsequent works aimed at improving the efficiency of the scheme [30, 113]. Most notably, a scheme called TESLA# [30] by Barreto, Longa, Naehrig, Ricardini, and Zanon included several implementation improvements. As a result of this line of research, a deterministic ideal-lattice-based signature scheme qTESLA was submitted to the NIST post-quantum standardization project [164] in 2017. The probabilistic qTESLA, as presented in this thesis, not only assembles the advantages acquired in TESLA, ring-TESLA, and TESLA#, but also takes latest implementation attacks, such as [111] into account. We summarize the history and relations of the different TESLA variants in Figure 3.1. In the Figure "scheme A $\rightarrow$ scheme B" means that scheme B is based on scheme A, i.e., B is a variant of A and/or B improves upon scheme A.

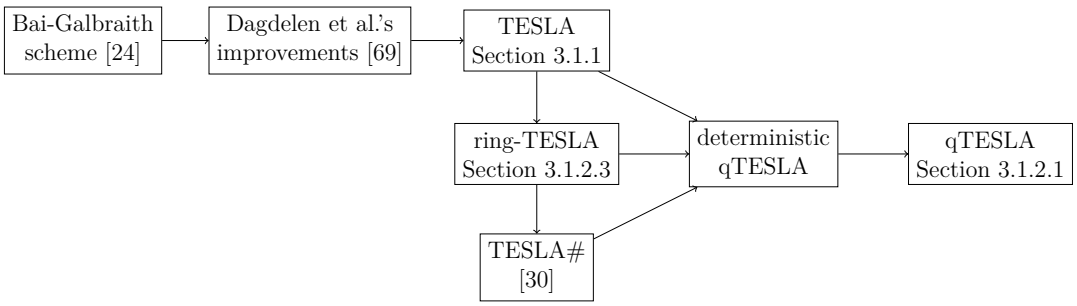


Figure 3.1: History of TESLA and qTESLA; inspired by [96, Figure 1.1]

In this section we describe the signature scheme TESLA that is defined over standard lattices and present its more efficient variant qTESLA defined over ideal latticed. Additionally, we also explain the differences between TESLA and the Bai-Galbraith signature schemes and the (dis-)advantages of different design choices made for TESLA and qTESLA.

Notation. We start be introducing notation that is used in the description of TESLA and qTESLA. If not stated otherwise, q is a prime integer. Moreover, for any positive integer m the set $\mathbb{Z}_m$ of integers modulo m is represented by $\{-\lfloor (m-1)/2 \rfloor, \ldots, \lfloor m/2 \rfloor\}$ and we define $c' = c \mod m$ as the unique element $-\lfloor m/2 \rfloor < c' \leq \lfloor m/2 \rfloor$ such that $c' = c \bmod m$. Furthermore, for a fixed positive integer d we define the functions $[\cdot]_M, [\cdot]_L : \mathbb{Z} \to \mathbb{Z}$ as follows. For any integer x let $[x]_L$ denote the unique integer in $(-2^{d-1}, 2^{d-1}] \cap \mathbb{Z}$, i.e., $x = [x]_L \bmod 2^d$. Moreover, let $[\cdot]_M$ be the function $[\cdot]_M : \mathbb{Z} \to \mathbb{Z}, c \mapsto (c \mod q - [c]_L)/2^d$. Informally, $[x]_L$ is viewed as the least significant bits of x and $[x]_M$ is viewed as the most significant bits of x. The definitions are easily extended to vectors or polynomials by applying the operators for each component or coefficient, respectively.

Finally, s-bit strings $r \in \{0,1\}^s$ and $r' \in \{-1,0,1\}^s$ are written as vectors over the sets $\{0,1\}$ and $\{-1,0,1\}$, respectively. Multiple instances of the same set are represented by appending an additional superscript. For example, $\{0,1\}^{s,t}$ corresponds to t s-bit strings each defined over the set $\{0,1\}$.

3.1.1 The Signature Scheme TESLA

Our signature scheme TESLA is parameterized by the matrix and vector dimensions n, n', m, the integer κ, the standard deviation of the Gaussian distribution σ, the modulus q, and the integers h, d, B, U, L_E, and L_S. We describe how these parameters are derived in Section 3.1.3. Moreover, TESLA is parameterized by the uniformly random sampled matrix $\mathbf{A} \leftarrow_{\$} \mathbb{Z}_q^{m \times n}$ which is publicly known as a global constant and can be shared among arbitrarily many signers.

Additionally, the description of TESLA includes several functions which are defined next. In order to obtain smaller signatures $(\mathbf{z}, c') \in \mathbb{Z}_q^n \times \{0,1\}^\kappa$, we break up the hashing into $\mathsf{Enc}(\mathsf{H}(\cdot))$, where the encoding function Enc takes the output of the hash function H and maps it to a vector with entries in $\{-1,0,1\}$ of length n' and weight h, i.e., it has h entries that are either 1 or -1. Let $\mathbb{H}$ denote the set of vectors $\mathbf{c} \in \{-1,0,1\}^{n'}$ with exactly h non-zero entries. In particular, let $\mathsf{H} : \mathbb{Z}_q^m \times \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function that takes as input a vector $\mathbf{v} \in \mathbb{Z}_q^m$ and computes $[\mathbf{v}]_M$. The result is then hashed together with a message m to a κ-bit string. We adopt SHA-3 [87] for the function H. Furthermore, let $\mathsf{Enc} : \{0,1\}^\kappa \to \mathbb{H}$ be the encoding function which takes the binary output of the hash function

and produces a vector in $\mathbb{H}$. Our encoding function has been based on [114]. Furthermore, we employ a PRF $\mathsf{PRF}_1 : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$ which maps the message to-be-signed and parts of the secret key to a pseudo-random value. The resulting pseudo-random value is then used to key $\mathsf{PRF}_2 : \{0,1\}^\kappa \times \mathbb{Z} \to [-B, B]^n$. Doing so, all randomness in the signature generation is deterministically derived. We adopt SHA-3 for PRF_1 and Chacha20 [38] for PRF_2.

Moreover, we define the functions checkE, introduced in [69, Section 3.2], as follows: For a matrix $\mathbf{E}$, define $\mathbf{E_i}$ to be the i-th row of $\mathbf{E}$. The function $\max_k$ returns the k-th largest absolute entry of a vector. The matrix $\mathbf{E}$ is rejected if for any row of $\mathbf{E}$ it holds that $\sum_{k=1}^{h} \max_k(\mathbf{E_i})$ is greater than some bound L_E. Hence, the routine checkE ensures that $\|\mathbf{E_i c}\|_\infty \leq L_E$ for all rows of $\mathbf{E}$. We apply a similar check checkS to $\mathbf{S}$: The matrix $\mathbf{S}$ is rejected if for any row of $\mathbf{S}$ it holds that $\sum_{k=1}^{h} \max_k(\mathbf{S_i})$ is greater than some bound L_S, i.e., the routine checkS ensures that $\|\mathbf{S_i c}\|_\infty \leq L_S$ for all rows of $\mathbf{S}$. The pseudo-codes of checkE and checkS are depicted in Algorithm 3.1 and 3.2, respectively.

Algorithm 3.1 checkE in TESLA

Require: $\mathbf{E} \in \mathbb{Z}_q^{m \times n'}$
Ensure: $\{0,1\}$ ▷ true, false

1: **for** $j = 1, ..., m$ **do**
2: **if** $\sum_{i=1}^{h} \max_i(\mathbf{E_j}) > L_E$ **then**
3: **return** 1
4: **return** 0

Algorithm 3.2 checkS in TESLA

Require: $\mathbf{S} \in \mathbb{Z}_q^{n \times n'}$
Ensure: $\{0,1\}$ ▷ true, false

1: **for** $j = 1, ..., n$ **do**
2: **if** $\sum_{i=1}^{h} \max_i(\mathbf{S_j}) > L_S$ **then**
3: **return** 1
4: **return** 0

Next we describe the scheme TESLA. The key generation, signature generation, and verification algorithms are summarized in Algorithm 3.3, 3.4, and 3.5, respectively.

Key Generation. At first the coefficients of the matrices $\mathbf{S} \in \mathbb{Z}^{n \times n'}$ and $\mathbf{E} \in \mathbb{Z}^{m \times n'}$ are sampled from the discrete Gaussian distribution. The matrix $\mathbf{E}$ has to satisfy certain constraints to ensure that the signatures are correct and short. These constraints are checked by the function checkE. We apply a similar check checkS to $\mathbf{S}$. If $\mathbf{S}$ or $\mathbf{E}$ do not fulfill the requirements, they are resampled. Furthermore, a secret value seed_y is sampled uniformly random from $\{0,1\}^\kappa$. Finally, the secret signing key $\mathsf{sk} \leftarrow (\mathbf{S}, \mathbf{E}, \mathsf{seed}_y)$ and the public verification key $\mathsf{pk} \leftarrow \mathbf{B} = \mathbf{AS} + \mathbf{E} \mod q$ are returned.

Signature Generation. During signing a message m, a secret seed $\mathsf{rand} \leftarrow \mathsf{PRF}_1(\mathsf{seed}_y, \mathsf{m})$ is generated and a counter is initialized with 0 first. Afterwards, a pseudo-random vector $\mathbf{y}$ is obtained by computing $\mathbf{y} \leftarrow \mathsf{PRF}_2(\mathsf{rand}, \mathsf{counter}) \in [-B, B]^n$ and multiplied by $\mathbf{A}$ in $\mathbb{Z}_q$. Then the higher order bits of $\mathbf{v} \leftarrow \mathbf{Ay}$

Algorithm 3.3 Key generation of TESLA

Require: Publicly available matrix $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$
Ensure: Secret key $\mathsf{sk} = (\mathbf{S}, \mathbf{E}, \mathsf{seed}_y)$ and public key $\mathsf{pk} = \mathbf{B}$

1: $\mathbf{S} \leftarrow_\sigma \mathbb{Z}_q^{n \times n'}$
2: **if** $\mathsf{checkS}(\mathbf{S}) \neq 0$ **then**
3: Restart in line 1
4: $\mathbf{E} \leftarrow_\sigma \mathbb{Z}_q^{m \times n'}$
5: **if** $\mathsf{checkE}(\mathbf{E}) \neq 0$ **then**
6: restart in line 4
7: $\mathsf{seed}_y \leftarrow_\$ \{0,1\}^\kappa$
8: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \mod q$
9: $\mathsf{sk} \leftarrow (\mathbf{S}, \mathbf{E}, \mathsf{seed}_y)$, $\mathsf{pk} \leftarrow \mathbf{B}$
10: **return** $(\mathsf{sk}, \mathsf{pk})$

mod q and the message m are hashed, yielding the hash value c'. Applying the encoding function to c', we obtain the vector $\mathbf{c} \leftarrow \mathsf{Enc}(c')$. Further on, we compute $\mathbf{z} \leftarrow \mathbf{Sc} + \mathbf{y}$. Now, rejection sampling is applied to make sure that the signature does not leak any information about the secret $\mathbf{S}$ and that the signature verifies for the applied compression. That is, if either $\|[\mathbf{w}]_L\|_\infty \geq 2^{d-1} - L_E$, $\|\mathbf{w}\|_\infty \geq \lfloor q/2 \rfloor - L_E$, or $\|\mathbf{z}\|_\infty > B - U$, with $\mathbf{w} = \mathbf{v} - \mathbf{Ec} \mod q$, then the signing algorithm discards $(\mathbf{z}, c')$ and repeats all steps. Finally, it returns the signature $(\mathbf{z}, c')$ on m.

Algorithm 3.4 Signature generation of TESLA

Require: Message m, secret key $\mathsf{sk} = (\mathbf{S}, \mathbf{E}, \mathsf{seed}_y)$, and matrix $\mathbf{A}$
Ensure: Signature $\mathsf{s} = (\mathbf{z}, c')$

1: $\mathsf{counter} \leftarrow 0$
2: $\mathsf{rand} \leftarrow \mathsf{PRF}_1(\mathsf{seed}_y, \mathsf{m})$
3: $\mathbf{y} \leftarrow \mathsf{PRF}_2(\mathsf{rand}, \mathsf{counter})$
4: $\mathbf{v} \leftarrow \mathbf{Ay} \mod q$
5: $c' \leftarrow \mathsf{H}([\mathbf{v}]_M, \mathsf{m})$
6: $\mathbf{c} \leftarrow \mathsf{Enc}(c')$
7: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$
8: $\mathbf{w} \leftarrow \mathbf{v} - \mathbf{Ec} \mod q$
9: **if** $\|[\mathbf{w}]_L\|_\infty \geq 2^{d-1} - L_E \vee \|\mathbf{w}\|_\infty \geq \lfloor q/2 \rfloor - L_E \vee \|\mathbf{z}\|_\infty > B - U$ **then**
10: $\mathsf{counter}{+}{+}$
11: restart in line 3
12: **return** $(\mathbf{z}, c')$

Verification. The algorithm, upon input of a message m and a signature $(\mathbf{z}, c')$, first computes $\mathbf{c} \leftarrow \mathsf{Enc}(c')$ to then obtain $\mathbf{w}' \leftarrow \mathbf{Az} - \mathbf{Bc} \mod q$, and returns 0 if $c = \mathsf{H}(\lfloor \mathbf{w}' \rceil_M, \mathsf{m})$ and $\|\mathbf{z}\|_\infty \leq B - U$ are satisfied; otherwise, it returns -1.

Algorithm 3.5 Verification of TESLA

Require: Message m, signature $\mathsf{s} = (c', \mathbf{z})$, public key $\mathsf{pk} = \mathbf{B}$, and matrix $\mathbf{A}$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $\mathbf{c} \leftarrow \mathsf{Enc}(c')$
2: $\mathbf{w}' \leftarrow \mathbf{Az} - \mathbf{Bc} \mod q$
3: **if** $c' = \mathsf{H}(\lfloor \mathbf{w}' \rceil_M, \mathsf{m}) \wedge \|\mathbf{z}\|_\infty \leq B - U$ **then**
4: **return** 0
5: **return** -1

After describing TESLA we now move on to prove its the correctness.

Correctness of TESLA. To establish the correctness of TESLA we need to prove that the nonce input to the hash function H at signing (line 5 of Algorithm 3.4) is the same as the nonce input to the hash function H at verification (line 3 of Algorithm 3.5). That is, we need to prove that, for genuine signatures it holds that

$$\begin{aligned} \lfloor \mathbf{Ay} \mod q \rceil_M &= \lfloor \mathbf{w}' \rceil_M \\ &= \lfloor \mathbf{Az} - \mathbf{Bc} \mod q \rceil_M \\ &= \lfloor \mathbf{A}(\mathbf{y} + \mathbf{Sc}) - (\mathbf{AS} + \mathbf{E})\mathbf{c} \mod q \rceil_M \\ &= \lfloor \mathbf{Ay} + \mathbf{ASc} - \mathbf{ASc} - \mathbf{Ec} \mod q \rceil_M \\ &= \lfloor \mathbf{Ay} - \mathbf{Ec} \mod q \rceil_M. \end{aligned}$$

From the definition of $\lfloor \cdot \rceil_M$, this means proving that

$$(\mathbf{Ay} \mod q - \lfloor \mathbf{Ay} \mod q \rceil_L)/2^d = (\mathbf{Ay} - \mathbf{Ec} \mod q - \lfloor \mathbf{Ay} - \mathbf{Ec} \mod q \rceil_L)/2^d,$$

or simply

$$\lfloor \mathbf{Ay} \mod q \rceil_L = \mathbf{Ec} + \lfloor \mathbf{Ay} - \mathbf{Ec} \mod q \rceil_L. \tag{3.1}$$

Equation (3.1) must hold component-wise, so let us prove the corresponding property for individual integers. Assume that for integers α and ε it holds that $|\lfloor \alpha - \varepsilon \mod q \rceil_L| < 2^{d-1} - L_E$, $|\varepsilon| \leq L_E < \lfloor q/2 \rfloor$, $|\alpha - \varepsilon \mod q| < \lfloor q/2 \rfloor - L_E$, and $-\lfloor q/2 \rfloor < \alpha \leq \lfloor q/2 \rfloor$ (i.e., $\alpha \mod q = \alpha$). Then, we need to prove that

$$\lfloor \alpha \rceil_L = \varepsilon + \lfloor \alpha - \varepsilon \mod q \rceil_L. \tag{3.2}$$

Proof. To prove Equation (3.2), we start by noticing that $|\varepsilon| \leq L_E < 2^{d-1}$ implies $[\varepsilon]_L = \varepsilon$. Thus, from $-2^{d-1} + L_E < [\alpha - \varepsilon \mod q]_L < 2^{d-1} - L_E$ and $-L_E \leq [\varepsilon]_L \leq L_E$ it follows that

$$-2^{d-1} = -2^{d-1} + L_E - L_E < [\varepsilon]_L + [\alpha - \varepsilon \mod q]_L < 2^{d-1} - L_E + L_E = 2^{d-1},$$

and therefore

$$[[\varepsilon]_L + [\alpha - \varepsilon \mod q]_L]_L = [\varepsilon]_L + [\alpha - \varepsilon \mod q]_L = \varepsilon + [\alpha - \varepsilon \mod q]_L. \quad (3.3)$$

Next we prove that

$$[[\varepsilon]_L + [\alpha - \varepsilon \mod q]_L]_L = [\alpha]_L. \quad (3.4)$$

It is important to note that since $|\varepsilon| \leq L_E < \lfloor q/2 \rfloor$ it holds that $[\varepsilon]_L = [\varepsilon \mod q]_L$. It holds further that

$$\begin{aligned} & [[\varepsilon \mod q]_L + [\alpha - \varepsilon \mod q]_L]_L && (3.5)\\ = \; & \Big((\varepsilon \mod q) \mod 2^d + (\alpha - \varepsilon \mod q) \mod 2^d\Big) \mod 2^d && (3.6)\\ & \text{by the definition of } [\cdot]_L \\ = \; & (\varepsilon \mod q + (\alpha - \varepsilon \mod q)) \mod 2^d. && (3.7) \end{aligned}$$

Since $|\varepsilon| \leq L_E$ and $|\alpha - \varepsilon \mod q| < \lfloor q/2 \rfloor - L_E$, it holds that $|\alpha - \varepsilon| + |\varepsilon| < (\lfloor q/2 \rfloor - L_E) + L_E = \lfloor q/2 \rfloor$. Hence, Equation (3.7) is the same as

$$\begin{aligned} = \; & (\varepsilon + \alpha - \varepsilon \mod q) \mod 2^d = (\alpha \mod q) \mod 2^d = \alpha \mod 2^d \\ = \; & [\alpha]_L. \end{aligned}$$

Combining Equation (3.3) and (3.4), we deduce that $[\alpha]_L = \varepsilon + [\alpha - \varepsilon \mod q]_L$, which is the equation we needed to prove. □

Now we define $\alpha = (\mathbf{Ay})_i$ and $\varepsilon = (\mathbf{Ec})_i$. From line 9 of Algorithm 3.4, we know that $\|[\mathbf{Ay} - \mathbf{Ec}]_L\|_\infty < 2^{d-1} - L_E$ and $\|\mathbf{Ay} - \mathbf{Ec}\|_\infty < \lfloor q/2 \rfloor - L_E$ for a valid signature, and that Algorithm 3.3 (line 5) guarantees $\|\mathbf{Ec}\|_\infty \leq L_E$. Likewise, by definition it holds that $L_E < \lfloor q/2 \rfloor$; see Section 3.1.3. Finally, $\mathbf{v} = \mathbf{Ay}$ is reduced mod q in line 4 of Algorithm 3.4 and, hence, $\mathbf{v}$ is in the centered range $-\lfloor q/2 \rfloor < \mathbf{Ay} \leq \lfloor q/2 \rfloor$. In conclusion, we get the desired condition vectors, $[\mathbf{Ay}]_L = \mathbf{Ec} + [\mathbf{Ay} - \mathbf{Ec}]_L$, which in turn means $[\mathbf{Az} - \mathbf{Tc}]_M = [\mathbf{Ay}]_M$ as argued above.

Moving further, we describe the differences between TESLA and the Bai-Galbraith signature schemes [24, 69] next.

Deterministic Signatures. We emphasize that signing is deterministic for each message m since the randomness is determined by the vector $\mathbf{y}$ which is deterministically computed by the secret key and the message to-be-signed. In the original scheme by Bai and Galbraith [24] the vector $\mathbf{y}$ was sampled uniformly random in $[-B, B]^n$. As long as we assume that the *PRF advantage* of $\mathsf{PRF}_2 \circ \mathsf{PRF}_1$ is somewhat "small", the security of TESLA still holds. We define the PRF advantage and PRF security in Definition 3.15 and formalize the security of TESLA with regard to the PRF security in Section 3.2.4. The idea to use a PRF to generate signatures deterministically has been deployed several times before [41, 135, 162]. The advantage of this approach is that a different randomness is used for different messages with very high probability. Hence, attacks that exploit a fixed randomness, such as in case of Sony's playstation 3 [56], are prevented. Another advantage is that there is no need to obtain a good pool of entropy at signing time which is shown to be difficult in practice [120].

Subroutine checkS During Key Generation. In contrast to earlier proposals [24, 69], we add an additional check in line 2, Algorithm 3.3. It ensures that no coefficient of the matrix $\mathbf{S}$ is too large, which allows for more concrete bounds during the security reduction. This is also done for qTESLA as explained in Section 3.1.2.

Additional Correctness Requirement. Moreover, in comparison to [24, 69], we add another condition in line 9, Algorithm 3.4 to ensure correctness of the scheme. The additional requirement is that the absolute value of each coordinate of $\mathbf{Ay} - \mathbf{Ec}$ is less than $\lfloor q/2 \rfloor - L_E$. We give an example why this property is necessary next. In particular, we construct an example such that all conditions in line 9, Algorithm 3.4 are fulfilled except for $\|\mathbf{Ay} - \mathbf{Ec}\|_\infty < \lfloor q/2 \rfloor - L_E$ and show that our example would lead to an invalid signature. In the following example, we use the parameters of TESLA-p-I (see Table 3.2), namely $q = 2^{31} - 19$, $d = 26$, and $L_E = 6703$. It is important to note that values mod q and mod 2^d are represented in $(-q/2, q/2]$ and $(-2^{d-1}, 2^{d-1}]$, respectively. Let

$$\alpha = (\mathbf{Ay})_i = -1\ 073\ 741\ 810 \in (-q/2, q/2] = [-1\ 073\ 741\ 814, 1\ 073\ 741\ 815]$$

for some $i \in \{0, ..., n-1\}$ and hence,

$$[\alpha]_L = \alpha \mod 2^d = -33\ 554\ 418.$$

Moreover, let

$$\varepsilon = (\mathbf{Ec})_i = 6\ 700 \in [-L_E, L_E] = [-6\ 703, 6\ 703].$$

Furthermore, we define

$$\nu = \alpha - \varepsilon \mod q.$$

Given the above values it follows that

$$\nu = -1\ 073\ 741\ 810 - 6\ 700 = -1\ 073\ 748\ 510 = 1\ 073\ 735\ 119 \mod q.$$

We emphasize that α is a value close to $-q/2$, while ν is close to $q/2$. Furthermore, we compute

$$[\nu]_L = \nu \mod 2^d = 33\ 547\ 727.$$

Hence, the property $|[\nu]_L| = |[\alpha - \varepsilon]_L| =< 2^{d-1} - L_E = 33\ 547\ 729$ holds. Nevertheless, $[\nu]_L = 33\ 547\ 727 \neq -33\ 554\ 418 = [\alpha]_L$ and hence, a signature with the i-th coefficient as above would have been rejected. This additional correctness requirement is also present in qTESLA which we describe next.

3.1.2 The Signature Scheme qTESLA

After describing the signature scheme TESLA, we now turn to its more efficient variant qTESLA which is constructed over ideal lattices. We first describe the signature scheme qTESLA in Section 3.1.2.1. Moreover, we explain the differences between TESLA and qTESLA in Section 3.1.2.2. Lastly, we describe a variant of qTESLA called ring-TESLA in Section 3.1.2.3.

3.1.2.1 Description of qTESLA

Analogously to TESLA, the description of qTESLA depends on the following system parameters: κ, n, k, q, σ, L_E, L_S, B, d, and h. We describe the parameters and their relations to each other in Section 3.1.3. Moreover, we define the following functions used in the description of qTESLA:

- The PRF $\mathsf{PRF}_1 : \{0,1\}^\kappa \to \{0,1\}^{\kappa,k+3}$. This function takes as input a seed $\mathsf{pre\text{-}seed}$ that is κ bits long and maps it to $(k+3)$ seeds of κ bits each. We instantiate the PRF with SHAKE [87].
- The PRF $\mathsf{PRF}_2 : \{0,1\}^\kappa \times \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^\kappa$. This function takes as inputs the seed seed_y and the random value r, each κ bits long, and a message m and maps them to the seed rand of κ bits. This PRF is also instantiated using SHAKE.
- The generation function of the public polynomials $a_1, ..., a_k$ $\mathsf{GenA} : \{0,1\}^\kappa \to \mathcal{R}_q^k$. This function takes as input the seed seed_a that is κ bits long and maps it to k polynomials $a_i \in \mathcal{R}_q^\times$. The function GenA is realized as an eXtendable Output Function (XOF) that is instantiated with cSHAKE [137].
- The hash function $\mathsf{H} : \mathcal{R}_q^k \times \{0,1\}^* \to \{0,1\}^\kappa$. This function takes as inputs k polynomials $v_1, ..., v_k \in \mathcal{R}_q$ and computes $[v_1]_M, ..., [v_k]_M$. The result is then hashed together with a message m to a κ-bit string. We adopt cSHAKE for function H.

- The encoding function $\mathsf{Enc} : \{0,1\}^{\kappa} \to \{-1,0,1\}^{h,2}$. This function encodes a κ-bit hash value c' as a polynomial $c \in \mathbb{H}$, where $\mathbb{H}$ is the set of polynomials with n coefficients in $\{-1,0,1\}$ and exactly h non-zero coefficients. We realize Enc following an algorithm originally proposed in [81, Section 4.4].
- The sampling function of the polynomial y, $\mathsf{ySampler} : \{0,1\}^{\kappa} \times \mathbb{Z} \to \mathcal{R}_{q,[B]}$. This function samples a polynomial $y \in \mathcal{R}_{q,[B]}$ taking as inputs a κ-bit seed rand and a nonce $S \in \mathbb{Z}_{>0}$. Its realization is based on a XOF that is instantiated with cSHAKE.

Next we describe the key generation, signature generation, and verification of qTESLA. The pseudo-code of the respective algorithms is given in Algorithm 3.6, 3.9, and 3.10.

Key Generation. First, a pre-seed is sampled to generate $\mathsf{seed}_s, \mathsf{seed}_{e_1}, \ldots, \mathsf{seed}_{e_k}$, seed_a, $\mathsf{seed}_y \leftarrow \mathsf{PRF}_1(\mathsf{pre\text{-}seed})$. Afterwards, seed_a is expanded to generate the polynomials $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$ which are in $\mathcal{R}_q^{\times}$. Next the secret polynomials $s, e_1, ..., e_k$ are sampled with Gaussian distribution (using seed_s, $\mathsf{seed}_{e_1}, \ldots, \mathsf{seed}_{e_k}$) until all polynomials fulfill the requirements checked through checkS and checkE which are depicted in Algorithm 3.7 and 3.8, respectively. Finally, the secret signing key $\mathsf{sk} \leftarrow (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$ and the public verification key $\mathsf{pk} \leftarrow (\mathsf{seed}_a, b_1, ..., b_k)$ are returned.

Algorithm 3.6 Key generation of qTESLA

Require: -
Ensure: Secret key $\mathsf{sk} = (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$ and public key $\mathsf{pk} = (\mathsf{seed}_a, b_1, ..., b_k)$

1: $\mathsf{pre\text{-}seed} \leftarrow_{\$} \{0,1\}^{\kappa}$
2: $\mathsf{seed}_s, \mathsf{seed}_{e_1}, \ldots, \mathsf{seed}_{e_k}, \mathsf{seed}_a, \mathsf{seed}_y \leftarrow \mathsf{PRF}_1(\mathsf{pre\text{-}seed})$
3: $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
4: **do**
5: $\quad s \leftarrow_{\sigma} \mathcal{R}$
6: **while** $\mathsf{checkS}(s) \neq 0$
7: **for** $i = 1, ..., k$ **do**
8: $\quad$ **do**
9: $\quad\quad e_i \leftarrow_{\sigma} \mathcal{R}$
10: $\quad$ **while** $\mathsf{checkE}(e_i) \neq 0$
11: $\quad b_i \leftarrow a_i s + e_i \mod q$
12: $\mathsf{sk} \leftarrow (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$
13: $\mathsf{pk} \leftarrow (\mathsf{seed}_a, b_1, ..., b_k)$
14: **return** sk, pk

Signature Generation. To sign a message m, first a counter is initialized with zero and a random seed $r \leftarrow_{\$} \{0,1\}^{\kappa}$ is sampled. Afterwards, $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, r, \mathsf{m})$ is computed and a pseudo-random polynomial y is obtained

Algorithm 3.7 checkE in qTESLA

Require: $e \in \mathcal{R}$
Ensure: $\{0,1\}$ ▷ true, false

1: **if** $\sum_{i=1}^{h} \max_i(e) > L_E$ **then**
2: **return** 1
3: **return** 0

Algorithm 3.8 checkS in qTESLA

Require: $s \in \mathcal{R}$
Ensure: $\{0,1\}$ ▷ true, false

1: **if** $\sum_{i=1}^{h} \max_i(s) > L_S$ **then**
2: **return** 1
3: **return** 0

by computing $y \leftarrow \mathsf{ySampler}(\mathsf{rand}, \mathsf{counter}) \in \mathcal{R}_{q,[B]}$. Then the values $v_i \leftarrow a_i y \mod q$ are computed for $i = 1, ..., k$. The higher order bits of the v_i's and the message m are hashed, yielding the hash value c'. We obtain the sparse polynomial $c = \mathsf{Enc}(c')$. Next, $z \leftarrow sc + y$ is computed. Afterwards, rejection sampling is applied to make sure that the signature does not leak any information about the secret s, i.e., if $z \notin \mathcal{R}_{q,[B-L_S]}$ then the signing algorithm discards (c', z) and repeats all steps. Similarly, if $\|[w_i]_L\|_\infty \geq 2^{d-1} - L_E$ or $\|w_i\|_\infty \geq \lfloor q/2 \rfloor - L_E$ then (c', z) is discarded and all steps are repeated to ensure correctness. Finally, it returns the signature (c', z) on m.

Algorithm 3.9 Signature generation of qTESLA

Require: Message m and secret key $\mathsf{sk} = (s, e_1, ..., e_k, \mathsf{seed}_a, \mathsf{seed}_y)$
Ensure: Signature $\mathsf{s} = (c', z)$

1: $\mathsf{counter} \leftarrow 0$
2: $r \leftarrow_\$ \{0,1\}^\kappa$
3: $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, r, \mathsf{m})$
4: $y \leftarrow \mathsf{ySampler}(\mathsf{rand}, \mathsf{counter})$
5: $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
6: **for** $i = 1, ..., k$ **do**
7: $v_i = a_i y \mod q$
8: $c' \leftarrow \mathsf{H}([v_1]_M, ..., [v_k]_M, \mathsf{m})$
9: $c \leftarrow \mathsf{Enc}(c')$
10: $z \leftarrow y + sc$
11: **if** $z \notin \mathcal{R}_{q,[B-L_S]}$ **then**
12: $\mathsf{counter}{+}{+}$
13: restart at step 4
14: **for** $i = 1, ..., k$ **do**
15: $w_i \leftarrow v_i - e_i c \mod q$
16: **if** $\|[w_i]_L\|_\infty \geq 2^{d-1} - L_E \vee \|w_i\|_\infty \geq \lfloor q/2 \rfloor - L_E$ **then**
17: $\mathsf{counter}{+}{+}$
18: restart at step 4
19: **return** (c', z)

Verification. The algorithm, upon input of a message m and a signature (c', z), first computes $c \leftarrow \mathsf{Enc}(c')$. Afterwards, it expands seed_a to generate $a_1, ..., a_k \in \mathcal{R}_q^\times$ and computes $w_i' = a_i z - b_i c \mod q$ for $i = 1, ..., k$. It returns 0 if $c = H([w_1]_M, ..., [w_k]_M, \mathsf{m})$ and $z \in \mathcal{R}_{q,[B-L_S]}$; otherwise, it returns -1.

Algorithm 3.10 Verification of qTESLA

Require: Message m, signature $\mathsf{s} = (c', z)$, and public key $\mathsf{pk} = (\mathsf{seed}_a, b_1, ..., b_k)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $c \leftarrow \mathsf{Enc}(c')$
2: $a_1, ..., a_k \leftarrow \mathsf{GenA}(\mathsf{seed}_a)$
3: **for** $i = 1, ..., k$ **do**
4: $\quad w_i \leftarrow a_i z - b_i c \mod q$
5: **if** $z \notin \mathcal{R}_{q,[B-L_S]} \vee c \neq H([w_1]_M, ..., [w_k]_M, \mathsf{m})$ **then**
6: $\quad$ **return** -1
7: **return** 0

Due to the similarities of TESLA and qTESLA, the correctness of qTESLA follows easily from the correctness of TESLA described above. Hence, we do not prove the correctness of qTESLA separately, but move on to elaborate on the differences of TESLA and qTESLA.

3.1.2.2 Differences Between TESLA and qTESLA

The signature schemes TESLA and qTESLA follow the same construction principle and can be seen as variants of each other. However, the designs differ in some points. We discuss the advantages and disadvantages of the respective design choices in the following paragraphs.

Standard vs. Ideal Lattices. The scheme TESLA is constructed over standard lattices and its security relies on the hardness of the M-LWE problem, see Definition 2.4 and Theorem 3.1. In contrast, qTESLA is constructed over ideal lattices and its security relies on the hardness of the R-LWE problem, see Definition 2.6 and Theorem 3.16. R-LWE was introduced by Lyubashevsky, Peikert, and Regev in 2012 [155] to enable the construction of more efficient lattice-based schemes that come with good security guarantees. This can also be seen in our construction as we explain next. The secret key of TESLA consists of two matrices with small entries and of dimension $n \times n'$ and $m \times n$, and a seed of size κ; the secret key of qTESLA consists of two seeds of size κ and $k + 1$ polynomials with small entries and of degree $n - 1$. Since k is much smaller than n' and m ($k \leq 5$ and $n', m \geq 390$ in all our proposed parameter sets), the secret key of qTESLA is much smaller than

the one of TESLA. We refer to Section 3.1.3 and 3.3.3 for a detailed description and concrete instantiations of the parameters, respectively. Similarly, the public key of qTESLA is smaller than the public key of TESLA. Namely, the public key of qTESLA consists of k polynomials in $\mathcal{R}_q$ and one seed of size κ, while the public key of TESLA is an $(m \times n)$-dimensional matrix with entries in $\mathbb{Z}_q$. Moreover, the run-time of qTESLA is generally faster than the run-time of TESLA because fast arithmetic operations such as fast polynomial multiplication can be implemented instead of matrix-vector multiplication (we refer to Section 3.4 for benchmarks of the implementation of TESLA and qTESLA). However, fast polynomial multiplications such as the NTT also come with a disadvantage, e.g., parameter choices are less flexible. For example, to enable the NTT it must hold that $q \mod 2n = 1$.

Another possible disadvantage of ideal-lattice-based schemes such as qTESLA is the additional algebraic structure. Several attacks such as [57, 67, 90, 102] have exploited the algebraic structure of some ideal lattices and constructions based on them. However, so far these attacks do not seem to be applicable to instantiations used in qTESLA.

Deterministic vs. Probabilistic Signature Generation. TESLA is a deterministic signature scheme, i.e., for the same message always the same signature is returned. In contrast, qTESLA signatures are probabilistically generated. One advantage of deterministic signature schemes is that in many cases attacks that use fixed randomness, such as the attack against Sony's playstation 3 [56], are prevented. This is because the needed randomness is freshly generated by construction, e.g., from the message to-be-signed and a fixed secret seed. For example, see line 2 of TESLA's signature generation in Algorithm 3.4. Also, no access to a source of high-quality randomness is needed for deterministic signature schemes at signing time. However, some deterministic signatures such as ECDSA [183], EdDSA [41], or Dilithium [82] but also an early deterministic variant of qTESLA, are vulnerable to an easy-to-implement fault attack as described in [111, 179] and Section 3.4.1. Our approach used in qTESLA prevents this fault attack as well as fixed-randomness attacks as explained next. To generate the randomness y, first a fresh randomness r is sampled (see line 2 in Algorithm 3.9). Afterwards, a PRF is applied to r, a value seed_y that is part of the secret key, and the message m (see line 3 in Algorithm 3.9). By using seed_y and m as part of the input to the PRF, it is ensured that a different randomness is used when different messages are signed. Hence, fixed-randomness attacks are prevented. Additionally, the random value r guarantees that at each signing operation a different y is used (with high probability). Thus, the above-mentioned fault attack against deterministic signature schemes is prevented. Interestingly, the purpose of the value r is solely to ensure the use of different randomness. The quality of the randomness, however, is

guaranteed by the PRF. Hence, qTESLA does not rely on high-quality randomness but only on low-quality randomness in line 2, Algorithm 3.9.

Storage of the Public Key. The signature scheme TESLA is parametrized by a matrix $\mathbf{A}$ that is assumed to be publicly available to all parties. Hence, the matrix or some representation of it does not need to be included in the secret or public key, yielding smaller key sizes. In contrast, the key generation, signature generation, and verification algorithm of qTESLA expand a seed seed_a that is given in the secret and public key to generate the polynomials $a_1, ..., a_k$. The use of fresh $a_1, ..., a_k$ per key pair makes the introduction of backdoors more difficult and reduces drastically the scope of all-for-the-price-of-one attacks [15, 30]. Moreover, storing only a seed instead of the full polynomials permits to save bandwidth since we only need κ bits to store seed_a instead of the $kn\lceil \log_2(q) \rceil$ bits that are required to represent the full polynomials.

3.1.2.3 Description of ring-TESLA

We now move on to describe the signature scheme ring-TESLA. The signature scheme qTESLA can be seen as a generalization and optimization of the earlier scheme ring-TESLA. Hence, we only describe ring-TESLA briefly for later reference in Chapter 4. We depict the key generation, signing, and verfication algorithm without further explanation in Algorithm 3.11, 3.12, and 3.13, respectively, and explain the most important improvements of qTESLA compared to its predecessor ring-TESLA next.

Correctness. The additional requirement $\|w_i\|_\infty < \lfloor q/2 \rfloor - L_E$ to ensure correctness of the signature scheme is missing in line 9 of Algorithm 3.12.

Security reduction. As it is shown in Section 3.2, qTESLA's security reduction has been proven in the QROM while ring-TESLA's reduction has been proven in the ROM.

Security estimations. The security estimations of qTESLA's instantiations, given in Section 3.3.3, are with respect to state-of-the-art quantum attacks while classical estimations are used for ring-TESLA's instantiations.

LWE samples. The number of R-LWE samples is more flexible in qTESLA while it is fixed to two samples (a_1, a_2, b_1, b_2) in ring-TESLA.

Implementation security. The implementation of qTESLA is secured against timing- and cache-side-channel attacks as explained in Section 3.4.1.

Algorithm 3.11 Key generation of ring-TESLA

Require: -
Ensure: Secret key $\mathsf{sk} = (s, e_1, e_2, a_1, a_2)$ and public key $\mathsf{pk} = (a_1, a_2, b_1, b_2)$

1: $a_1, a_2 \leftarrow_{\$} \mathcal{R}_q$
2: $s, e_1, e_2 \leftarrow_{\sigma} \mathbb{Z}^n$
3: **if** $\mathsf{checkE}(e_1) = 0 \vee \mathsf{checkE}(e_2) = 0$ **then**
4: restart in line 2
5: $b_1 \leftarrow a_1 s + e_1 \mod q$
6: $b_2 \leftarrow a_2 s + e_2 \mod q$
7: $\mathsf{sk} \leftarrow (s, e_1, e_2, a_1, a_2)$, $\mathsf{pk} \leftarrow (a_1, a_2, b_1, b_2)$
8: **return** $(\mathsf{sk}, \mathsf{pk})$

Algorithm 3.12 Signature generation of ring-TESLA

Require: Message m, secret key $\mathsf{sk} = (s, e_1, e_2, a_1, a_2)$
Ensure: Signature $\mathsf{s} = (c', z)$

1: $y \leftarrow_{\$} \mathcal{R}_{q,[B]}$
2: $v_1 \leftarrow a_1 y \mod q$
3: $v_2 \leftarrow a_2 y \mod q$
4: $c' \leftarrow H\left(\left[v_1\right]_M, \left[v_2\right]_M, \mathsf{m}\right)$
5: $c \leftarrow \mathsf{Enc}(c')$
6: $z \leftarrow y + sc$
7: $w_1 \leftarrow v_1 - e_1 c \mod q$
8: $w_2 \leftarrow v_2 - e_2 c \mod q$
9: **if** $\left[w_1\right]_L, \left[w_2\right]_L \notin \mathcal{R}_{q,[2^{d-1}-L_E]} \vee z \notin \mathcal{R}_{q,[B-U]}$ **then**
10: restart in line 1
11: **return** (z, c')

3.1.3 System Parameters

Having described the design of TESLA and qTESLA, we now turn to explicate the system parameters and their respective requirements to achieve a correct and secure signature scheme. As indicated by the descriptions above, most of the parameters used for TESLA and qTESLA are similar. Therefore, if appropriate, we describe the respective parameters together and explain the differences, if any, in the relevant passages. In general, the parameters for qTESLA are optimized for efficiency while the parameters for TESLA are chosen to be more conservative from an attackers point of view.

Algorithm 3.13 Verification of ring-TESLA

Require: Message m, signature $\mathsf{s} = (c', z)$, public key $\mathsf{pk} = (a_1, a_2, b_1, b_2)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $c \leftarrow \mathsf{Enc}(c')$
2: $w_1' \leftarrow a_1 z - b_1 c \mod q$
3: $w_2' \leftarrow a_2 z - b_2 c \mod q$
4: $c'' \leftarrow H\left(\left[w_1'\right]_M, \left[w_2'\right]_M, \mathsf{m}\right)$
5: **if** $c' = c'' \wedge \ z \in \mathcal{R}_{q,[B-U]}$ **then**
6: **return** 0
7: **else**
8: **return** -1

Table 3.1 summarizes all system parameters, including the relevant bounds for TESLA and qTESLA. If a parameter is not used in one of the signature schemes we write "-"; if no requirement is needed we leave the table entry empty. Concrete parameter values for each of the proposed parameter sets are compiled in Table 3.2, Section 3.3.3.

We start with the security parameter and the output length of the hash function. The security parameter λ is chosen to be the (targeted) bit security of a given instantiation. The parameter κ defines the output length of the hash function, determines the inputs and outputs of the used PRFs and the encoding function Enc. It has to hold that $\kappa \geq \lambda$. This is consistent with the use of the hash in a Fiat-Shamir style signature scheme such as TESLA or qTESLA. In the Fiat-Shamir paradigm for signatures [94], preimage resistance is relevant while collision resistance is much less, given that we take the hash size to be enough to resist preimage attacks. In a scenario that excludes Grover's quantum algorithm [112], a hash function with an output length of λ is expected to have preimage resistance of 2^λ. For example, this scenario is considered in NIST's security category with the highest security requirements (category 5, see [165, Section 4.A.4]). In [165] Grover's algorithm is ruled out because "Grover's algorithm requires a long-running serial computation, which is difficult to implement in practice". Hence, it is presumed that the quantum speed-up by Grover's algorithm in a practical attack is less than it is theoretically estimated. When considering the theoretical quadratic acceleration of Grover's algorithm, the preimage resistance is only $\approx 2^{\lambda/2}$. In such a case, the hash output length should be 2λ for an aspired security level of λ.

One of the most important parameters is $n \in \mathbb{Z}_{>0}$. For TESLA this is the number of columns of the matrix $\mathbf{A}$ and for qTESLA it is the number of polynomial coefficients, i.e., the polynomial degree is $n-1$. To be able to use the efficient NTT for polynomial multiplication in the ring $\mathcal{R}_q$ for qTESLA, we restrict ourselves to

Table 3.1: Description and bounds of TESLA's and qTESLA's system parameters

Param.	Description	Requirement for TESLA	Requirement for qTESLA
λ	security parameter		
q_h, q_s	number of hash and sign queries	2^{λ}, $2^{\lambda/2}$	2^{128}, 2^{64}
n	dimension		power-of-two
n'	dimension		-
σ, ξ	standard deviation	$\sigma > 2\sqrt{n}$	$\sigma = \frac{\xi}{\sqrt{2\ln 2}}$
k	#R-LWE samples		-
m	#LWE samples	-	
q	modulus	$q > 4B$ $q^m \geq \lvert\Delta\mathbb{S}\rvert \cdot \lvert\Delta\mathbb{L}\rvert \cdot \lvert\Delta\mathbb{H}\rvert$, $q^m \geq 2^{md+4\lambda+1}(q_h+q_s)^2 q_s^3$	$q = 1 \mod 2n$, $q > 4B$ $q^{nk} \geq \lvert\Delta\mathbb{S}\rvert \cdot \lvert\Delta\mathbb{L}\rvert \cdot \lvert\Delta\mathbb{H}\rvert$, $q^{nk} \geq 2^{nkd+4\lambda+1}(q_s+q_h)^2 q_s^3$
h	#non-zero entries in output of Enc	$2^h\binom{n'}{h} \geq 2^{3\lambda}$ (classically) $2^h\binom{n'}{h} \geq 2^{5\lambda}$ (quantumly)	$2^h \cdot \binom{n}{h} \geq 2^{2\lambda}$
κ	output length hash function H	$\kappa \geq 2\lambda$	$\kappa \geq \lambda$
L_E, η_E	bound in checkE	$\eta_E \cdot h \cdot \sigma$	
L_S, η_S	bound in checkS	$\eta_S \cdot h \cdot \sigma$	
U	influences rej. prob. in sign	$\lceil 14\sqrt{h}\sigma \rceil$	L_S
B	determines randomness in sign	$\geq 14n\sqrt{h}\sigma$	$B \geq \frac{k\cdot\sqrt[n]{M}+2L_S-1}{2(1-\sqrt[k\cdot n]{M})}$, near to power-of-two
d	number of rounded bits	$\left(1-\frac{2\cdot L_E+1}{2^d+1}\right)^m \geq 0.3$	$\left(1-\frac{2\cdot L_E+1}{2^d+1}\right)^{k\cdot n} \geq 0.3$, $d > \log_2(B)$
$\lvert\Delta\mathbb{H}\rvert$	see definition below in the text	$\sum_{j=0}^{h}\sum_{i=0}^{h-j}\binom{m}{2i}2^{2i}\binom{m-2i}{j}2^j$	$\sum_{j=0}^{h}\sum_{i=0}^{h-j}\binom{kn}{2i}2^{2i}\binom{kn-2i}{j}2^j$
$\lvert\Delta\mathbb{S}\rvert$		$(4(B-U)+1)^n$	$(4(B-L_S)+1)^n$
$\lvert\Delta\mathbb{L}\rvert$		$(2^d+1)^m$	$(2^d+1)^{nk}$
$\lvert\mathsf{sig}\rvert$	size signature [bit]	$n\lceil\log_2(2(B-U))\rceil + \kappa$	$n(\lceil\log_2(B-L_S)\rceil) + \kappa$
$\lvert\mathsf{pk}\rvert$	size public key [bit]	$mn'\lceil\log_2(q)\rceil$	$kn(\lceil\log_2(q)\rceil) + \kappa$
$\lvert\mathsf{sk}\rvert$	size secret key [bit]	$(nn'+mn')\lceil\log_2(t\sigma)\rceil + \kappa$	$n(k+1)(\lceil\log_2(t\cdot\sigma)\rceil) + 2\kappa$

a polynomial degree of a power-of-two, i.e., $n = 2^l$ for $l \in \mathbb{N}$.

Compared to [24, 69], we introduce the parameter n' as the column dimension of the secret matrices $\mathbf{S}$ and $\mathbf{E}$ for TESLA to get more flexibility in the choice of parameters. The value of n' influences the parameters h and the secret key size.

Additionally, the number of R-/M-LWE samples is denoted by k or m. In particular, $k \in \mathbb{Z}_{>0}$ is the number of R-LWE samples for qTESLA and m is the row dimension of the matrix $\mathbf{A}$. A larger k or m allows to reduce the size of the modulus q.

It is important to note that no requirement is stated for n, m, or k. However, implicitly it is required that bit hardness of the corresponding R-/M-LWE instance (defined through n, m or k, σ, and q) is at least λ bits. We describe the relation

between bit hardness of R-/M-LWE and bit security of the signature schemes in Section 3.3.2.

Furthermore, the standard deviation of the centered discrete Gaussian distribution that is used to sample the secret and error in the LWE problem, i.e., the entries of the matrices $\mathbf{S}$ and $\mathbf{E}$ for TESLA and the coefficients of the polynomials $s, e_1, ..., e_k$ for qTESLA, is denoted by σ. We choose our parameters for TESLA such that the quantum reduction from the worst-case gapSVP to LWE holds (see [191, Lemma 4.2] or [24, Theorem 1]), i.e., we choose $\sigma > 2\sqrt{n}$. For efficiency reasons, we do not instantiate qTESLA in the same way but rely on the average hardness of R-LWE as usually done for ideal-lattice-based schemes such as [13, 27, 81, 82, 222]. In the implementation of qTESLA, the fast Gaussian sampler introduced in [81] is used. For this reason, it is necessary to choose $\sigma = \frac{\xi}{\sqrt{2 \ln 2}}$ for some $\xi \in \mathbb{Z}_{>0}$.

The parameter h defines the number of non-zero elements in the output of the encoding function Enc. To ensure security of the encoding function we require $2^h \cdot \binom{n'}{h} \geq 2^\lambda$ for TESLA (resp., $2^h \cdot \binom{n}{h} \geq 2^\lambda$ for qTESLA). In order to reach small additional terms in Equations (3.41) (quantum) and (3.42) (classical) of our security reduction we require an even larger value for h. Namely, we choose $2^h \cdot \binom{n}{h} \geq 2^{2\lambda}$. Choosing h according to this bounds implies to be left with the summand $q_h/2^\lambda$ in Equation (3.41) (resp., $q_h/2^{2\lambda}$ in Equation (3.42)). The ratio $q_h/2^\lambda$ also corresponds to an an adversary's ability to break the preimage resistance of a hash function and thus, we find it an acceptable bound for an efficient scheme like qTESLA. For TESLA, however, we go one step further and aim at even smaller additionally terms in the corresponding equations, namely we choose h such that the additional terms are less or equal $2^{-\lambda}$.

Moreover, the values L_E and L_S denote the bounds used in the evaluation functions checkE and checkS, respectively. Bounding the size of the secret and error, restricts the size of the key space. To compensate the resulting potential security loss, we choose a larger bit hardness as explained in Section 3.3.2. Additionally, L_E impacts the rejection probability during the signature generation as follows. If one increases the value of L_E, the acceptance probability during key generation increases while the acceptance probability during signature generation decreases. The same holds true for TESLA with the parameter U which occurs in line 9 of Algorithm 3.4. For qTESLA, however, we simplified the parameters and choose $U = L_S$ at the expense of loosing more security bits because of the increased rejection probability in the key generation. We determine the best trade-off (in terms of run-time) between the acceptance probability in the key generation and the signature generation experimentally.

Besides L_E, L_S (in qTESLA), and U (in TESLA), also the parameter B, d, and M determine the rejection probability during the signature generation (see line 9 in

Algorithm 3.4 or line 11 of Algorithm 3.9). Let M be a value of our choosing. The parameter B defines the interval of the randomness during signature generation. The probability that $\mathbf{z} \in [-B+U, B-U]^n$ (resp., $z \in \mathcal{R}_{q,[B-L_S]}$) is given by

$$\left(\frac{2B-2U+1}{2B+1}\right)^m \geq M \text{ (resp., } \left(\frac{2B-2L_S+1}{2B+1}\right)^{k\cdot n} \geq M\text{)}.$$

Hence, we determine B for TESLA by

$$B \geq \frac{\sqrt[m]{M}+2L_S-1}{2(1-\sqrt[m]{M})} \text{ (resp., } B \geq \frac{\sqrt[k\cdot n]{M}+2L_S-1}{2(1-\sqrt[k\cdot n]{M})} \text{ for qTESLA)},$$

depending on the value of M. We select the rounding value d to be larger than $\log_2(B)$ and such that the acceptance probability of the check $\|[w]_L\|_\infty \geq 2^{d-1}-L_E$ is upper bounded by 0.7.

In order the give a concrete instantiation, the number of hash and sign queries q_h and q_s have to be chosen as well. For the conservative parameters of TESLA we choose $q_h \leq 2^\lambda$ and $q_s \leq 2^{\lambda/2}$, since a hash query is merely the evaluation of a publicly available function and hence the adversary can use all its computational power to pose hash queries. The number of sign queries is somewhat limited since it involves more complicated operations. We refer to [141, Section 7] for a discussion. For qTESLA we use another approach and follow NIST's proposals [165, Section 4.A.4], i.e., we choose the number of classical queries to the sign oracle to be $q_s = 2^{64}$ for all our parameter sets. Moreover, we choose the number of queries of a hash function to be $q_h = 2^{128}$.

Finally, the modulus of the LWE problem q, which depends on all other parameters, can be determined. The value of q is chosen to fulfill several bounds and assumptions that are required by the security reduction or to implement the schemes more efficiently. To achieve a sensible description of the signature schemes q is chosen larger than $4B$. Furthermore, to be able to use fast polynomial multiplication for qTESLA we choose q to be a prime integer such that $q \mod 2n = 1$. Moreover, to choose parameter sets according to the security reduction, it is convenient to enable a simplification of our security statement given in Section 3.2. To do so, we ensure that

$$q^m \geq |\Delta\mathbb{S}| \cdot |\Delta\mathbb{L}| \cdot |\Delta\mathbb{H}| \text{ and } q^{nk} \geq |\Delta\mathbb{S}| \cdot |\Delta\mathbb{L}| \cdot |\Delta\mathbb{H}|$$

for TESLA and qTESLA, respectively, with the following definition of sets:

- $\mathbb{S}$ is the set of vectors $\mathbf{z} \in \mathbb{Z}_q^n$ such that $\mathbf{z} \in [-B+U, B-U]^n$ (resp., the set of polynomials $z \in \mathcal{R}_{q,[B-L_S]}$),
- $\Delta\mathbb{S}$ is the set $\{\mathbf{z}-\mathbf{z}' : \mathbf{z}, \mathbf{z}' \in \mathbb{S}\}$ (resp., $\{z-z' \ : \ z, z' \in \mathbb{S}\}$),
- $\mathbb{H}$ is the set of vectors $c \in \{-1, 0, 1\}^{n'}$ with exactly h non-zero entries (resp., the set of polynomials $c \in \mathcal{R}_{q,[1]}$ with exactly h non-zero coefficients),

- $\Delta\mathbb{H}$ is the set $\{\mathbf{c}-\mathbf{c}' : \mathbf{c},\mathbf{c}' \in \mathbb{H}\}$ (resp., $\Delta\mathbb{H} = \{c - c' \ : \ c, c' \in \mathbb{H}\}$),
- $\Delta\mathbb{L}$ is the set $\left\{\mathbf{x}-\mathbf{x}' : \mathbf{x},\mathbf{x}' \in \mathbb{Z}_q^m \text{ and } [\mathbf{x}]_M = [\mathbf{x}']_M\right\}$ (resp., $\{x - x' : x, x' \in \mathcal{R} \text{ and } [x]_M = [x']_M\}$).

The sizes of the respective sets are summarized in Table 3.1 and derived in Section 3.2. According to Theorem 3.16, Section 3.2, the following equation has to hold for qTESLA:

$$\frac{2^{3\lambda+nkd+1}q_s^3(q_s+q_h)^2}{q^{nk}} \leq 2^{-\lambda} \Leftrightarrow q \geq \left(2^{4\lambda+nkd+1}q_s^3(q_s+q_h)^2\right)^{1/nk}.$$

For TESLA it has to hold correspondingly $q \geq \left(2^{4\lambda+md+1}q_s^3(q_h+q_s)^2\right)^{1/m}$.

We are now in a position to determine the key and signature sizes. The theoretical sizes of the signatures and public keys are given in Table 3.1. In the following we explain the secret key size. To determine the size of the secret key we note that for $t > 0$ it holds that [154]

$$Pr_{x\leftarrow_\sigma\mathbb{Z}}\left[|x| > t\sigma\right] \leq 2e^{-t^2/2}.$$

We choose t such that this probability is less or equal $2^{-\lambda}$. For example, if $t = 13.4$ then the probability $Pr_{x\leftarrow_\sigma\mathbb{Z}}\left[|x| > t\sigma\right]$ is less or equal to 2^{-128}. Therefore, the theoretical size of the secret key for TESLA is given by

$$(nn' + mn')\lceil\log_2(t\sigma)\rceil + \kappa \text{ bits}$$

and the size of qTESLA's secret key is

$$n \cdot (\lceil\log_2(t\sigma)\rceil) + k \cdot n \cdot (\lceil\log_2(t \cdot \sigma)\rceil) + 2\kappa \text{ bits},$$

with t chosen appropriately for the respective value of λ.

We summarize the dependencies of the TESLA and qTESLA parameters in Figure 3.2 and Figure 3.3, respectively, where parameters are sorted by the respective number of parameters they depend on. For example, the parameter λ does not depend on other parameters and hence, λ is a *1st level* parameter. In contrast, the parameter U depends on 1st and 2nd level parameters and hence, it is a 3rd level parameter. Interestingly, the secret key is a 3rd level parameter, whereas the public key is at the 8th level, which is the highest level there is.

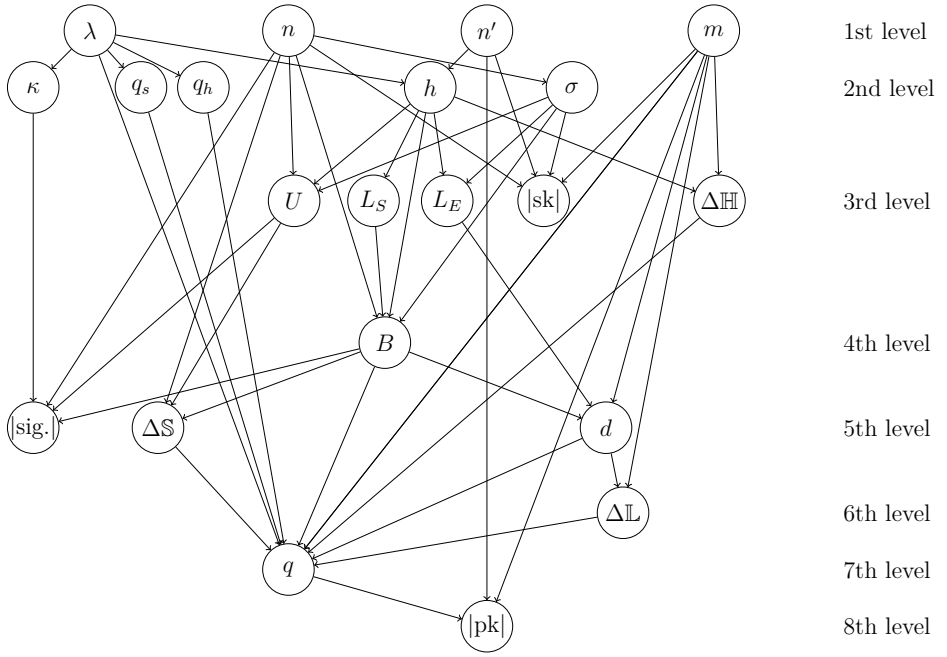


Figure 3.2: Dependencies of the TESLA parameters, inspired by [108, Figure 7.2]

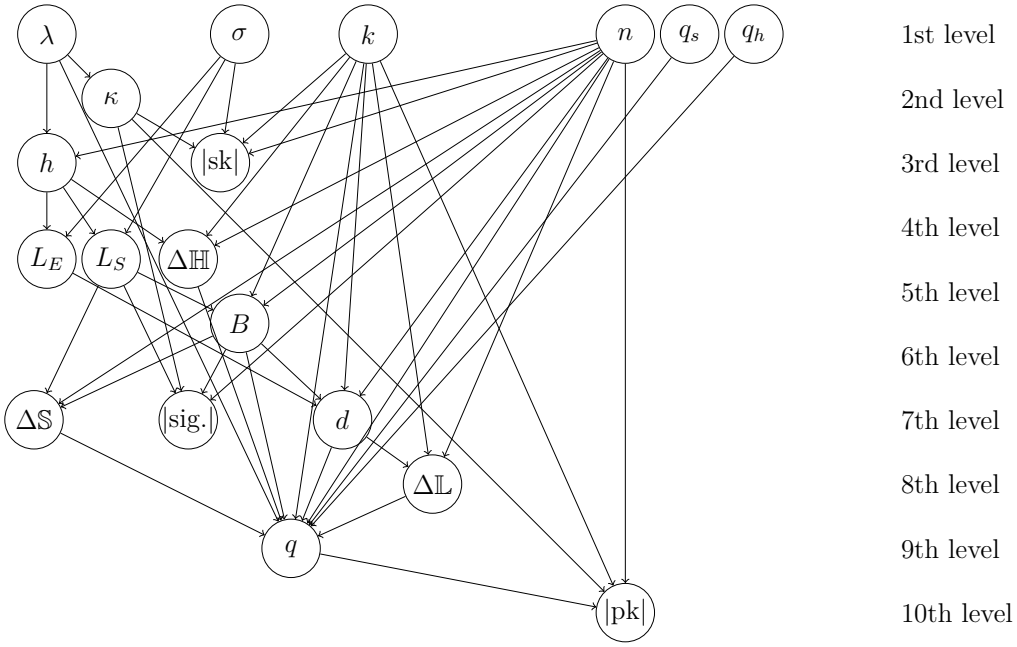


Figure 3.3: Dependencies of the qTESLA parameters, inspired by [108, Figure 7.2]

3.2 Security Reductions

In this section, we provide the security reductions from M-LWE or R-LWE to TESLA or qTESLA, respectively. We start with explaining the idea of the security reduction of TESLA. Afterwards, we present the technical details of the reduction in Section 3.2.1, 3.2.2, 3.2.3, and 3.2.4. Lastly, we give the security reduction of qTESLA which follows the reduction of TESLA closely except that we need to introduce one conjecture.

Our main theorem on the security of TESLA informally states that as long as M-LWE can not be solved in time t and with success probability ε then no adversary $\mathcal{A}$ exists that can forge signatures of TESLA in time t' and with success probability ε', if $\mathcal{A}$ is allowed to make at most q_h hash and q_s sign queries. The security reduction is proven in the QROM. The main theorem is as follows.

Theorem 3.1 (Security of TESLA)**.** *Let q, m, n, n', h, d, B, L_E, L_S, U, σ, λ, and κ be the parameters of TESLA that are convenient[2] (according to Definition 3.14 in Section 3.2.4) and that satisfy the bounds in Table 3.1. If M-LWE is (t, ε)-hard then TESLA is $(t', q_h, q_s, \varepsilon')$-EUF-CMA with $t' \approx t$ in*
(i) the QROM with

$$\varepsilon' < \varepsilon + \frac{3}{2^\lambda} + \frac{2^{md+3\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 + 2(q_h + 1)\sqrt{\frac{1}{2^h\binom{n'}{h}}}, \tag{3.8}$$

and in (ii) the ROM with

$$\varepsilon' < \varepsilon + \frac{3}{2^\lambda} + \frac{2^{md+3\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 + q_h\frac{1}{2^h\binom{n'}{h}}. \tag{3.9}$$

We proof this theorem in the following subsections but explain the overall idea first. The security reduction presented by Bai and Galbraith for their signature scheme employs the Forking Lemma [180]. As such, it is non-tight and it involves re-programming, so it holds in the ROM but is not known to hold in the QROM. In order to avoid the non-tightness inherent in the use of the Forking Lemma, we take an approach that was introduced by Katz and Wang to obtain tightly-secure signatures from the decisional Diffie-Hellman problem [135].

The idea is to use the underlying hardness assumption to show that "real", properly-formed public keys for the signature scheme are indistinguishable from "lossy", malformed public keys. The task of forging a signature for a lossy key

[2]It is not necessary that the parameters of TESLA are convenient in order to derive negligibly small upper bounds on ε'; the definition of convenience merely facilitates a simplified statement of those bounds.

is then somehow proven to be intractable. Any attacker must therefore fail to forge when given a lossy public key. Thus, any attacker who succeeds in forging a signature when given a real public key can be used to distinguish real keys from lossy keys, contradicting the underlying hardness assumption.

A similar approach has been proposed by Abdalla, Fouque, Lyubashevsky, and Tibouchi [1]. However, security reductions obtained by applying Abdalla et al.'s framework are guaranteed to hold only in the ROM. In order to fully recover our security reduction from this framework, one must first re-prove Abdalla et al.'s framework in the QROM. Recently, this has been done by Kiltz, Lyubashevsky, and Schaffner [139]. We take a different approach and give a security reduction that is tailored to TESLA and hence, yields tighter bounds.

3.2.1 Overview of the Security Reduction for TESLA

We start by explaining the structure of our proof. For simplicity, we consider a simplified scheme of TESLA given in Algorithm 3.14, 3.15, and 3.16. In particular, we assume that the hash function H has range $\mathbb{H}$, i.e., we ignore the encoding function Enc and we assume that the randomness $\mathbf{y} \leftarrow_{\$} [-B, B]^n$ instead of using PRF_1 and PRF_2 to compute $\mathbf{y} \in [-B, B]^n$. As long as we assume that PRF_1 and PRF_2 are secure PRFs and that Enc is instantiated such that it preserves a bit security of λ, the security reduction of the simplified TESLA transfers to TESLA as defined in Section 3.1.1. We formalize this in Section 3.2.4.

In the remainder of the proof we say that an integer vector $\mathbf{y}$ is B-short if each entry is at most B in absolute value. We call an integer vector $\mathbf{w}$ well-rounded if $\mathbf{w}$ is $(\lfloor q/2 \rfloor - L_E)$-short and $[\mathbf{w}]_M$ is $(2^{d-1} - L_E)$-short.

Algorithm 3.14 KeyGen–Simplified key generation of TESLA

Require: -
Ensure: Public key $(\mathbf{A}, \mathbf{B})$ and secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$

1: $\mathbf{A} \leftarrow_{\$} \mathbb{Z}_q^{m \times n}$
2: $\mathbf{S} \leftarrow_{\sigma} \mathbb{Z}_q^{n \times n'}$ and $\mathbf{E} \leftarrow_{\sigma} \mathbb{Z}_q^{m \times n'}$
3: **if** $\mathbf{E}$ has a row whose h largest entries sum to L_E or more **then**
4: restart in line 2
5: **if** $\mathbf{S}$ has a row whose h largest entries sum to L_S or more **then**
6: restart in line 2
7: $\mathbf{B} \leftarrow \mathbf{AS} + \mathbf{E} \mod q$
8: **return** public key $(\mathbf{A}, \mathbf{B})$ and secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$

The proof is by a simulation argument, which we briefly sketch next. Let $\mathcal{F}$ be a forger that forges signatures of the TESLA scheme with probability $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$,

Algorithm 3.15 Sign–Simplified signature generation of TESLA

Require: Message m and secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$
Ensure: Signature $(\mathbf{z}, \mathbf{c})$

1: $\mathbf{y} \leftarrow_{\$} [-B, B]^n$
2: $\mathbf{c} \leftarrow \mathsf{H}([\mathbf{Ay}]_M, \mathsf{m})$
3: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$
4: **if** $\mathbf{z}$ is not $(B-U)$-short **then**
5: restart in line 1
6: **if** $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded **then**
7: restart in line 1
8: **return** signature $(\mathbf{z}, \mathbf{c})$

Algorithm 3.16 Verify–Simplified verification of TESLA

Require: Message m, public key $(\mathbf{A}, \mathbf{B})$, and signature $(\mathbf{z}, \mathbf{c})$
Ensure: "Accept" or "reject"

1: **if** $\mathbf{z}$ is not $(B-U)$-short **then**
2: **return** "reject"
3: **if** $\mathsf{H}([\mathbf{Az} - \mathbf{Bc}]_M, \mathsf{m}) \neq \mathbf{c}$ **then**
4: **return** "reject"
5: **return** "accept"

where forge$(\mathbf{A}, \mathbf{B})$ denotes the event that $\mathcal{F}$ forges a signature on input $(\mathbf{A}, \mathbf{B})$, which is a yes- or a no-instance of M-LWE. As we describe in Section 2.2.2, we call $(\mathbf{A}, \mathbf{B}) \in \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n'}$ a yes-instance if there exists an $\mathbf{S} = (\mathbf{s_1}, ..., \mathbf{s_{n'}})$ with $\mathbf{s_1}, ..., \mathbf{s_{n'}} \in \mathbb{Z}_q^n$ and $(\mathbf{A}, \mathbf{B})$ are m M-LWE samples from the distribution $\mathcal{D}_{\mathbf{S},\chi}$, see Definition 2.4. Otherwise, i.e., when $(\mathbf{A}, \mathbf{B}) \leftarrow_{\$} \mathbb{Z}_q^{m \times n} \times \mathbb{Z}_q^{m \times n'}$, we call $(\mathbf{A}, \mathbf{B})$ a no-instance.

We build an M-LWE solver $\mathcal{S}$ whose run-time is close to that of $\mathcal{F}$ and who solves M-LWE with success bias close to $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$. It then follows from the presumed hardness of M-LWE that $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$ must be small.

The M-LWE solver $\mathcal{S}$ acts as follows. Given an M-LWE input $(\mathbf{A}, \mathbf{B})$, the M-LWE solver $\mathcal{S}$ treats $(\mathbf{A}, \mathbf{B})$ as a TESLA public key; $\mathcal{S}$ runs $\mathcal{F}$ on input $(\mathbf{A}, \mathbf{B})$ and outputs "yes" if and only if $\mathcal{F}$ succeeds in forging a TESLA signature. In order to run $\mathcal{F}$, $\mathcal{S}$ must respond to $\mathcal{F}$'s queries to the hash and sign oracles by simulating these oracles without knowledge of a secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ corresponding to $(\mathbf{A}, \mathbf{B})$. (Indeed, a secret key might not even exist if $(\mathbf{A}, \mathbf{B})$ is a no-instance of M-LWE.) In order to run $\mathcal{F}$, the M-LWE solver $\mathcal{S}$ must respond in some way to $\mathcal{F}$'s quantum queries to the hash oracle and to $\mathcal{F}$'s classical queries to the sign

oracle. Our description of $\mathcal{S}$ includes a procedure for responding to these queries as described in Algorithm 3.17.

Algorithm 3.17 M-LWE solver $\mathcal{S}$ using a TESLA forger $\mathcal{F}$

Require: An M-LWE instance $(\mathbf{A}, \mathbf{B})$
Ensure: "Yes" or "no"

1: Invoke the forger $\mathcal{F}$ with public key $(\mathbf{A}, \mathbf{B})$.
Whenever $\mathcal{F}$ makes a hash or sign query, simulate that query as follows:
Classical sign queries. Execute Simulated-sign (Algorithm 3.18).
Quantum hash queries. Apply a quantum circuit that implements a random but fixed $2q_h$-wise independent function, except on inputs that have been re-programmed by Simulated-sign.
2: Eventually, $\mathcal{F}$ produces a purported forgery.
If that forgery is legit then output "yes", otherwise output "no".

Classical queries made by $\mathcal{F}$ to the signing oracle are simulated by $\mathcal{S}$ as specified in "Simulated-sign" depicted in Algorithm 3.18. Quantum queries made by $\mathcal{F}$ to the hash oracle are simulated by $\mathcal{S}$ according to the construction of Zhandry based on $2q_h$-wise independent functions [221]. (Alternately, if the performance of the simulator is a concern then one could instead use a quantum-resistant PRF [45].)

Algorithm 3.18 Simulated-sign

Require: Message m and public key $(\mathbf{A}, \mathbf{B})$
Ensure: Signature $\mathsf{s} = (\mathbf{z}, \mathbf{c})$

1: $\mathbf{z} \leftarrow_{\$} [-B + U, B - U]^n$
2: $\mathbf{c} \leftarrow_{\$} \mathbb{H}$
3: **if** $\mathbf{Az} - \mathbf{Bc}$ is not well-rounded **then**
4: restart in line 1
5: re-program the hash oracle H so that $\mathsf{H}([\mathbf{Az} - \mathbf{Bc}]_M, \mathsf{m}) = \mathbf{c}$
6: **return** $(\mathbf{z}, \mathbf{c})$

Figure 2.2 in Section 2.4 depicts the definition of the quantum random oracle as a unitary channel that applies the linear map $|x, t, z\rangle \mapsto |x, t \oplus \mathsf{H}(x), z\rangle$ on standard basis states. In the following proof it is, however, sufficient to assume that for a hash function $\mathsf{H} : X \to Y$ the user has access to a unitary channel that applies the linear map $|x, y\rangle \mapsto |x, y + \mathsf{H}(x)\rangle$ on standard basis states, where $x \in X$ and $y \in Y$. Hence, we omit the workspace register.

$\mathcal{S}$ solves M-LWE with a success bias close to $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$ as a consequence of the following facts, which are proven in subsequent sections and brought together in Section 3.2.4:

Section 3.2.2: For yes-instances of M-LWE, the probability with which $\mathcal{S}$ outputs "yes" is close to $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$.

Section 3.2.3: For no-instances of M-LWE, $\mathcal{F}$ successfully forges (and hence, $\mathcal{S}$ outputs "yes") with only negligible probability.

3.2.1.1 Notation and Sizes for Various Sets of Vectors

We now elaborate on notations and sets that are used throughout the security reduction. We define/recall the following sets of integer vectors:

$\mathbb{Y}$: The set of vectors $\mathbf{y} \in \mathbb{Z}_q^n$ such that $\mathbf{y}$ is B-short.
$\Delta\mathbb{Y}$: $\{\mathbf{y} - \mathbf{y}' : \mathbf{y}, \mathbf{y}' \in \mathbb{Y}\}$.
$\mathbb{S}$: The set of vectors $\mathbf{z} \in \mathbb{Z}_q^n$ such that $\mathbf{z}$ is $(B-U)$-short.
$\Delta\mathbb{S}$: $\{\mathbf{z} - \mathbf{z}' : \mathbf{z}, \mathbf{z}' \in \mathbb{S}\}$.
$\mathbb{H}$: The set of vectors $\mathbf{c} \in \{-1, 0, 1\}^{n'}$ with exactly h non-zero entries.
$\Delta\mathbb{H}$: $\{\mathbf{c} - \mathbf{c}' : \mathbf{c}, \mathbf{c}' \in \mathbb{H}\}$.
$\mathbb{W}$: The set $\{[\mathbf{w}]_M : \mathbf{w} \in \mathbb{Z}_q^m\}$ obtained from the most significant bits of the vector coefficients.
$\Delta\mathbb{L}$: $\left\{\mathbf{x} - \mathbf{x}' : \mathbf{x}, \mathbf{x}' \in \mathbb{Z}_q^m \text{ and } [\mathbf{x}]_M = [\mathbf{x}']_M\right\}$.

The sizes of some of these sets are listed below:

$$\#\mathbb{Y} = (2B+1)^n, \qquad \#\Delta\mathbb{Y} = (4B+1)^n, \tag{3.10}$$

$$\#\mathbb{S} = (2(B-U)+1)^n, \qquad \#\Delta\mathbb{S} = (4(B-U)+1)^n, \tag{3.11}$$

$$\#\mathbb{H} = \binom{n'}{h} 2^h, \text{ and} \tag{3.12}$$

$$\#\Delta\mathbb{L} = (2^d+1)^m. \tag{3.13}$$

The size of $\Delta\mathbb{H}$ is computed as follows.

Lemma 3.2 (Size of $\Delta\mathbb{H}$)**.**

$$\#\Delta\mathbb{H} = \sum_{k=0}^{h} \sum_{i=0}^{h-k} \binom{n'}{2i} 2^{2i} \binom{n'-2i}{k} 2^k.$$

Proof. For each $k = 0, \ldots, h$ let $\Delta\mathbb{H}_k \subset \Delta\mathbb{H}$ denote the set of vectors in $\Delta\mathbb{H}$ with exactly k entries in $\{-2, 2\}$ and exactly $n'-k$ entries in $\{-1, 0, 1\}$. We can observe that $\#\Delta\mathbb{H} = \sum_{k=0}^{h} \#\Delta\mathbb{H}_k$.

We fix k. Then for each $i = 0, \ldots, h-k$ let $\Delta\mathbb{H}_{k,i} \subset \Delta\mathbb{H}_k$ we denote the set of vectors in $\Delta\mathbb{H}_k$ with exactly $2i$ entries in $\{-1, 1\}$. Observe that $\#\Delta\mathbb{H}_k = \sum_{i=0}^{h-k} \#\Delta\mathbb{H}_{k,i}$. Finally, one can count the number of elements in each $\Delta\mathbb{H}_{k,i}$ as

$$\#\Delta\mathbb{H}_{k,i} = \binom{n'}{2i} 2^{2i} \binom{n'-2i}{k} 2^k,$$

from which the lemma follows. □

3.2.2 Yes-Instances of M-LWE

In this section we establish a lower bound on the probability

$$\Pr\left[\mathcal{S} \text{ outputs "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ yes-instance of M-LWE}\right] \tag{3.14}$$

in terms of $\Pr\left[\text{forge}(\mathbf{A}, \mathbf{B})\right]$. This is accomplished by proving that the simulated oracles are indistinguishable from the real oracles.

To prove the indistinguishability, we consider an arbitrary distinguisher $\mathcal{D}$ who, like the forger $\mathcal{F}$, makes at most q_h quantum queries to the hash oracle and at most q_s classical queries to the signing oracle. Unlike $\mathcal{F}$, however, $\mathcal{D}$'s goal is merely to distinguish the real oracles from the simulated oracles.

3.2.2.1 Adaptively Chosen Queries

An arbitrary distinguisher could adaptively and arbitrarily interleave its hash and sign queries. To facilitate our analysis we wish to model the distinguisher in such a way that sign queries occur at fixed, predictable points throughout the protocol. This goal is accomplished by a continuous accounting method for hash queries that we describe as follows.

Standard formalism specifies that a quantum oracle for $\mathsf{H} : X \to Y$ is implemented by a unitary channel $|x, y\rangle \mapsto |x, y + \mathsf{H}(x)\rangle$. We modify this formalism so that the unitary channel for H is a controlled unitary channel. Specifically, the channel acts on an additional qubit which dictates whether the unitary channel is applied or not:

$$\begin{aligned} |\text{off}, x, y\rangle &\mapsto |\text{off}, x, y\rangle\,, \\ |\text{on}, x, y\rangle &\mapsto |\text{on}, x, y + \mathsf{H}(x)\rangle\,. \end{aligned}$$

Consider a new type of distinguisher with the following properties:

1. The distinguisher makes $q_h q_s$ hash queries instead of q_h hash queries.
2. Exactly one sign query occurs after every q_h hash queries.
3. For each choice of hash oracle H, the distinguisher's total *query magnitude*, as defined in Section 3.2.2.6, on query states with the control qubit set to $|\text{on}\rangle$ over all $q_h q_s$ hash queries does not exceed q_h.

A distinguisher of this form is called a *live-switch distinguisher*. For later convenience, we refer to the query magnitude on states with the control qubit set to $|\text{on}\rangle$ as the *query magnitude on the live-switch*.

Intuitively this corresponds to a live-switch distinguisher that can make *partial* queries to the hash oracle. If it's query state has only α amplitude on the live-switch then the distinguisher is *charged* for only an $|\alpha|^2$-fraction of a query.

It is clear that any ordinary distinguisher who makes q_h hash queries and q_s sign queries, adaptively chosen and interleaved, could be simulated by a live-switch distinguisher. Thus, live-switch distinguishers are at least as powerful as ordinary distinguishers, and possibly more powerful. We will prove indistinguishability against a live-switch distinguisher, which proves a stronger statement than strictly necessary.

The benefit of the live-switch distinguisher is that sign queries occur at fixed points throughout the protocol, namely one sign query after every q_h hash queries. This property allows us to partition the interaction into q_s *blocks*. Each block consists of q_h quantum queries to the hash oracle, followed by a single classical query to the sign oracle. We prove security of each block and then claim security of the entire interaction inductively.

3.2.2.2 The Distinguisher's State–a First Look

To begin, we consider the state of $\mathcal{D}$'s system immediately prior to the signing oracle query in the first block. At this point in the interaction the real and simulated oracles are perfectly identical–both respond to the first q_h hash queries in accordance with some fixed choice of hash oracle H. Let ρ_{H} denote the state of $\mathcal{D}$'s system at this point in the interaction, conditioned on H. The signing oracle (both real and simulated) acts as follows on $\mathcal{D}$'s system:

1. Measure the message register, resulting in outcome m.
2. Select a signature $(\mathbf{z}, \mathbf{c})$ for message m.
3. Prepare an output register in the classical basis state $|\mathsf{m}, (\mathbf{z}, \mathbf{c})\rangle$.

These actions can be viewed as a quantum channel. If the signing oracle is the original (or *real*) signing oracle then the signature $(\mathbf{z}, \mathbf{c})$ is a function of private randomness and the hash oracle H. In this case, the channel is denoted $\Psi_{\text{real},\mathsf{H}}$. If the signing oracle is a simulated signing oracle then the signature $(\mathbf{z}, \mathbf{c})$ is a function only of private randomness. In this case, the channel is denoted Ψ_{sim}.

Thus, the state of $\mathcal{D}$'s system at the end of the first block, conditioned on the choice of H, is either $\Psi_{\text{sim}}(\rho_{\mathsf{H}})$ or $\Psi_{\text{real},\mathsf{H}}(\rho_{\mathsf{H}})$. We will argue that the state $\Psi_{\text{sim}}(\rho_{\mathsf{H}})$ is δ-close to a probabilistic mixture over re-programmed hash oracles H' of states of the form $\Psi_{\text{real},\mathsf{H}'}(\rho_{\mathsf{H}'})$.

This δ-closeness is preserved by the hash queries in the second block of the interaction, since both the real and simulated hash oracles remain consistent with H' in this block. Let $\rho_{2,\mathsf{H}'}$ denote the state of $\mathcal{D}$'s system immediately prior to the signing oracle query in the second block. As above, we have that $\Psi_{\text{sim}}(\rho_{2,\mathsf{H}'})$ is δ-close to a mixture of states of the form $\Psi_{\text{real},\mathsf{H}''}(\rho_{2,\mathsf{H}''})$.

Continuing inductively, we see that the state of $\mathcal{D}$'s system at the end of an interaction with simulated oracles is $q_s\delta$-close to a probabilistic mixture over hash oracles of states of $\mathcal{D}$'s system at the end of an interaction with real oracles.

Averaging over the choice of initial hash oracle H, we then see that the simulated oracles are indistinguishable from the real oracles. We formalize these arguments in subsequent sections.

3.2.2.3 Mid-Sign

Consider the signing oracle Mid-sign of Algorithm 3.19. Mid-sign should be viewed as a hybrid of Simulated-sign (Algorithm 3.18) and the real signing oracle depicted in Algorithm 3.15.

Algorithm 3.19 Mid-sign

Require: Message m, public key $(\mathbf{A}, \mathbf{B})$, and secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$
Ensure: Signature $(\mathbf{z}, \mathbf{c})$

1: $(\mathbf{y}, \mathbf{c}) \leftarrow_{\$} \mathbb{Y} \times \mathbb{H}$ uniformly at random
2: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$
3: **if** $\mathbf{z} \notin \mathbb{S}$ **then**
4: restart in line 1
5: **if** $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded **then**
6: restart in line 1
7: re-program the hash oracle H so that $\mathsf{H}([\mathbf{Ay}]_M, \mathsf{m}) = \mathbf{c}$
8: **return** $(\mathbf{z}, \mathbf{c})$

In this section, we prove that Mid-sign (Algorithm 3.19) and Simulated-sign (Algorithm 3.18) are identical. This fact can be stated in terms of quantum channels as follows. Let Ψ_{mid} denote the channel described by Algorithm 3.19. The claim of this section is that $\Psi_{\text{mid}} = \Psi_{\text{sim}}$. To prove this statement, we first define the following sets for each choice of $\mathbf{c} \in \mathbb{H}$:

$$\text{good}_{\text{sim}}(\mathbf{c}) = \{\mathbf{z} \in \mathbb{S} : \mathbf{Az} - \mathbf{Bc} \text{ is well-rounded}\},$$
$$\text{good}_{\text{mid}}(\mathbf{c}) = \{\mathbf{y} \in \mathbb{Y} : \mathbf{y} + \mathbf{Sc} \in \mathbb{S} \text{ and } \mathbf{A}(\mathbf{y} + \mathbf{Sc}) - \mathbf{Bc} \text{ is well-rounded}\}.$$

We begin with a simple observation on these sets.

Lemma 3.3. *The mapping* $f : \mathbf{y} \mapsto \mathbf{y} + \mathbf{Sc}$ *is a bijection from* $\text{good}_{\text{mid}}(\mathbf{c})$ *to* $\text{good}_{\text{sim}}(\mathbf{c})$ *with inverse* $f^{-1} : \mathbf{z} \mapsto \mathbf{z} - \mathbf{Sc}$.

Proof. It is clear that $f^{-1}f$ is the identity function on $\text{good}_{\text{mid}}(\mathbf{c})$. It remains to prove the following:

1. For each $\mathbf{y} \in \text{good}_{\text{mid}}(\mathbf{c})$ it holds that $f(\mathbf{y}) \in \text{good}_{\text{sim}}(\mathbf{c})$.
2. For each $\mathbf{z} \in \text{good}_{\text{sim}}(\mathbf{c})$ it holds that $f^{-1}(\mathbf{z}) \in \text{good}_{\text{mid}}(\mathbf{c})$.

To prove item 1 we show (i) $f(\mathbf{y}) \in \mathbb{S}$, and (ii) $\mathbf{A}f(\mathbf{y}) - \mathbf{Bc}$ is well-rounded. Both items are immediate from the definitions of $\text{good}_{\text{mid}}(\mathbf{c})$ and f.

To prove item 2 we show (i) $f^{-1}(\mathbf{z}) \in \mathbb{Y}$, and (ii) $\mathbf{A}f^{-1}(\mathbf{z}) - \mathbf{Ec}$ is well-rounded. Item (i) follows from the fact that $\mathbf{z}$ is $(B-U)$-short and $\mathbf{Sc}$ is U-short. Item (ii) is immediate from the definitions of $\text{good}_{\text{sim}}(\mathbf{c})$ and f^{-1}. □

We now prove this section's claim.

Proposition 3.4 (Equivalence of Mid-sign and Simulated-sign)**.** *The observable behavior of Mid-sign (Algorithm 3.19) is statistically identical to that of Simulated-sign (Algorithm 3.18). In terms of quantum channels, we have* $\Psi_{\text{mid}} = \Psi_{\text{sim}}$.

Proof. The observable effects of both the Simulated-sign and Mid-sign algorithms can be summarized as follows. Given a message m as input, the algorithm selects (i) a signature $(\mathbf{z}_{\mathsf{m}}, \mathbf{c}_{\mathsf{m}})$ as output, and (ii) a vector $\mathbf{w}_{\mathsf{m}}$ inducing a hash input $(\mathbf{w}_{\mathsf{m}}, \mathsf{m})$ upon which the hash oracle is re-programmed. Thus, to establish statistical equivalence between Simulated-sign and Mid-sign it suffices to prove that, for each choice of message m, the joint distribution over $(\mathbf{z}_{\mathsf{m}}, \mathbf{c}_{\mathsf{m}}, \mathbf{w}_{\mathsf{m}})$ induced by the two algorithms Mid-sign and Simulated-sign is identical.

Fix an arbitrary message m and let $(Z_{\text{sim}}, C_{\text{sim}}, W_{\text{sim}})$, $(Z_{\text{mid}}, C_{\text{mid}}, W_{\text{mid}})$ denote the joint random variables representing the observable behavior of Simulated-sign and Mid-sign, respectively, on input message m. We argue that the joint random variables $(Z_{\text{sim}}, C_{\text{sim}})$, $(Z_{\text{mid}}, C_{\text{mid}})$ are identical. The proposition will then follow from the observation that the hash input $\mathbf{w}$ to be re-programmed is specified in both algorithms by the same deterministic function of $(\mathbf{z}, \mathbf{c})$. Specifically, in Simulated-sign we have $\mathbf{w} = [\mathbf{Az} - \mathbf{Bc}]_M$, whereas in Mid-sign we have $\mathbf{w} = [\mathbf{Ay}]_M$. Since $\mathbf{Ay} - \mathbf{Ec}$ is well-rounded, we have

$$\mathbf{w} = [\mathbf{Ay}]_M = [\mathbf{Ay} - \mathbf{Ec}]_M = [\mathbf{A}(\mathbf{y} + \mathbf{Sc}) - \mathbf{Bc}]_M = [\mathbf{Az} - \mathbf{Bc}]_M$$

as desired. Moreover, we argue that

$$\Pr[Z_{\text{mid}} = \mathbf{z} \mid C_{\text{mid}} = \mathbf{c}] = \Pr[Z_{\text{sim}} = \mathbf{z} \mid C_{\text{sim}} = \mathbf{c}]$$

for each choice of $\mathbf{c} \in \mathbb{H}$. In Simulated-sign, conditioned on a choice of $\mathbf{c}$, the vector $\mathbf{z}$ is chosen uniformly among those $\mathbf{z} \in \text{good}_{\text{sim}}(\mathbf{c})$. In Mid-sign, conditioned on a choice of $\mathbf{c}$, the vector $\mathbf{y}$ is chosen uniformly among those $\mathbf{y} \in \text{good}_{\text{mid}}(\mathbf{c})$ and the vector $\mathbf{z}$ is computed as $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$. It follows from Lemma 3.3 that $\mathbf{z}$ is uniform on $\text{good}_{\text{sim}}(\mathbf{c})$, as desired.

Next, we argue that

$$\Pr[C_{\text{mid}} = \mathbf{c}] = \Pr[C_{\text{sim}} = \mathbf{c}]$$

for each choice of $\mathbf{c} \in \mathbb{H}$, from which it follows that the joint random variables $(Z_{\text{sim}}, C_{\text{sim}})$, $(Z_{\text{mid}}, C_{\text{mid}})$ are identical. It follows from Lemma 3.3 that $\#\,\text{good}_{\text{mid}}(\mathbf{c}) = \#\,\text{good}_{\text{sim}}(\mathbf{c})$ for each $c \in \mathbb{H}$. Thus,

$$\Pr[C_{\text{mid}} = \mathbf{c}] = \frac{\#\,\text{good}_{\text{mid}}(\mathbf{c})}{\sum_{\mathbf{c}'} \#\,\text{good}_{\text{mid}}(\mathbf{c}')} = \frac{\#\,\text{good}_{\text{sim}}(\mathbf{c})}{\sum_{\mathbf{c}'} \#\,\text{good}_{\text{sim}}(\mathbf{c}')} = \Pr[C_{\text{sim}} = \mathbf{c}]$$

as desired. □

3.2.2.4 Consistent-Mid-Sign

Broadly speaking, Mid-sign (Algorithm 3.19) behaves like the real signature generation (Algorithm 3.15) except that $\mathbf{c}$ is selected freshly at random instead of according to some hash oracle H. It is tempting to claim that the only difference between Mid-sign and Sign is that repeated invocations of Sign always use the same hash oracle H, whereas each invocation of Mid-sign switches to another hash oracle H' that differs from H on a small number of randomly selected inputs.

However, there is a small probability that the random choices in a given execution of Mid-sign are not consistent with *any* hash oracle. To understand how such an inconsistency can occur, observe that each candidate $(\mathbf{y}, \mathbf{c})$ selected by Mid-sign induces an associated claim about the underlying hash oracle, namely that $\mathsf{H}([\mathbf{Ay}]_M, \mathsf{m}) = \mathbf{c}$. Suppose Mid-sign rejects one candidate pair $(\mathbf{y}, \mathbf{c})$ because $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded before finally accepting another candidate pair $(\mathbf{y}', \mathbf{c}')$. If $[\mathbf{Ay}]_M = [\mathbf{Ay}']_M$ but $\mathbf{c} \neq \mathbf{c}'$ then these two candidates represent conflicting claims about the underlying hash oracle.

To address this problem we present a new signing oracle Consistent-mid-sign in Algorithm 3.20 and argue that its observable behavior is close to that of Mid-sign (Algorithm 3.19). This fact can be stated in terms of quantum channels as follows. Let $\Psi_{\text{c-mid}}$ denote the channel described by Algorithm 3.20. The claim of this section is that $\Psi_{\text{c-mid}} \approx \Psi_{\text{mid}}$, meaning that $\Psi_{\text{c-mid}}(\rho) \approx \Psi_{\text{mid}}(\rho)$ for input states ρ.

The only difference between Consistent-mid-sign and Mid-sign is that each invocation of Consistent-mid-sign remembers the random candidate pairs it selected throughout the invocation and alters them as needed so as to maintain consistency with a hash oracle. Thus, in order to prove $\Psi_{\text{c-mid}} \approx \Psi_{\text{mid}}$ it suffices to prove that only a negligibly small fraction of the random choices made by Mid-sign leads to an inconsistency that is corrected in Consistent-mid-sign.

A sequence $r = \{(\mathbf{y}_i, \mathbf{c}_i)\}_{i=1}^{\infty}$ of random choices made by Mid-sign leads to an inconsistently derived signature only if there exists $k \geq 2$ such that the following conditions hold:

1. $\mathbf{Ay}_1 - \mathbf{Ec}_1, \ldots, \mathbf{Ay}_{k-1} - \mathbf{Ec}_{k-1}$ are not well-rounded,
2. $\mathbf{Ay}_k - \mathbf{Ec}_k$ is well-rounded, and
3. $[\mathbf{Ay}_k]_M \in \{[\mathbf{Ay}_1]_M, \ldots, [\mathbf{Ay}_{k-1}]_M\}$.

Algorithm 3.20 Consistent-mid-sign

Require: Message m, public key $(\mathbf{A}, \mathbf{B})$, and secret key $(\mathbf{S}, \mathbf{E}, \mathbf{A})$
Ensure: Signature $(\mathbf{z}, \mathbf{c})$

1: initialize the dictionary $\mathcal{A} \subset (\mathbb{W} \mapsto \mathbb{H})$ to the empty dictionary $\mathcal{A} = \emptyset$
2: $\mathbf{y} \leftarrow_{\$} \mathbb{Y}$
3: **if** $[\mathbf{Ay}]_M \in \mathcal{A}$ **then**
4: $\quad \mathbf{c} \leftarrow \mathcal{A}[[\mathbf{Ay}]_M]$
5: **else**
6: $\quad \mathbf{c} \leftarrow_{\$} \mathbb{H}$
7: $\quad$ add $\mathcal{A}[[\mathbf{Ay}]_M] \leftarrow \mathbf{c}$ to the dictionary $\mathcal{A}$
8: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{Sc}$
9: **if** $\mathbf{z} \notin \mathbb{S}$ **then**
10: $\quad$ restart in line 2
11: **if** $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded **then**
12: $\quad$ restart in line 2
13: re-program the hash oracle H so that $\mathsf{H}([\mathbf{Ay}]_M, \mathsf{m}) = \mathbf{c}$
14: **return** $(\mathbf{z}, \mathbf{c})$

Consider the event that a random sequence r meets conditions 1–3 for some choice of $k \geq 2$, and let inconsistent(r) denote the infinite disjunction of these events over all $k \geq 2$.[3] Then it holds that

$$\frac{1}{2} \left\| \Psi_{\text{c-mid}}(\rho) - \Psi_{\text{mid}}(\rho) \right\|_{\text{tr}} < \Pr_r \left[\text{inconsistent}(r) \right]. \tag{3.15}$$

Hence, we seek an upper bound on the probability of event inconsistent(r) over the choice of r. To determine the upper bound, we define the following quantities for each choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$:

nwr$(\mathbf{A}, \mathbf{E})$:	The probability over $(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}$ that $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded,
coll$(\mathbf{A}, \mathbf{E})$:	The maximum over all $\mathbf{w} \in \mathbb{W}$ of the probability over $(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}$ that $[\mathbf{Ay} - \mathbf{Ec}]_M = \mathbf{w}$.

[3] In item 3 it suffices to look for a collision only between $[\mathbf{Ay}_k]_M$ and any previous $[\mathbf{Ay}_i]_M$; we do not need to look for a collision among arbitrary $[\mathbf{Ay}_i]_M = [\mathbf{Ay}_j]_M$. This is because such a collision among the bad entries is statistically identical to as if $\mathbf{c}_i = \mathbf{c}_j$. Namely, $[\mathbf{Ay}_i]_M$ is rejected regardless of whether $\mathbf{c}_i = \mathbf{c}_j$. If however, $\mathbf{c}_j$ changes $[\mathbf{Ay}_i]_M$ from bad to good then that difference will be detected at $k = j$. Thus, there's no need to check for this when $k > j$.

In symbols, these quantities are written as

$$\mathrm{nwr}(\mathbf{A},\mathbf{E}) = \Pr_{(\mathbf{y},\mathbf{c})\in\mathbb{Y}\times\mathbb{H}}\left[\mathbf{Ay}-\mathbf{Ec} \text{ not well-rounded}\right],$$
$$\mathrm{coll}(\mathbf{A},\mathbf{E}) = \max_{\mathbf{w}\in\mathbb{W}}\left\{\Pr_{(\mathbf{y},\mathbf{c})\in\mathbb{Y}\times\mathbb{H}}\left[\left[\mathbf{Ay}-\mathbf{Ec}\right]_M=\mathbf{w}\right]\right\}.$$

In Section 3.2.2.8 and 3.2.2.9, we prove bounds on these quantities that hold with high probability over the choice of TESLA keys $(\mathbf{A},\mathbf{B})$, $(\mathbf{S},\mathbf{E},\mathbf{A})$.

Broadly speaking, $\mathrm{nwr}(\mathbf{A},\mathbf{E})$ should be viewed as a constant that is noticeably smaller than 1. For example, $\mathrm{nwr}(\mathbf{A},\mathbf{E}) = 1/2$. By contrast $\mathrm{coll}(\mathbf{A},\mathbf{E})$ is negligibly small. We prove the following.

Proposition 3.5 (Probability of Inconsistency)**.** *For each choice of TESLA keys* $(\mathbf{A},\mathbf{B})$*,* $(\mathbf{S},\mathbf{E},\mathbf{A})$ *it holds that*

$$\Pr_r\left[\text{inconsistent}(r)\right] \le \mathrm{coll}(\mathbf{A},\mathbf{E})\frac{\mathrm{nwr}(\mathbf{A},\mathbf{E})}{\left(1-\mathrm{nwr}(\mathbf{A},\mathbf{E})\right)^2}$$

over sequences $r = \{(\mathbf{y}_i,\mathbf{c}_i)\}_{i=1}^{\infty}$ *of random choices made by Mid-sign.*

Proof. For each $k \ge 2$ the probability with which events 1 and 2 hold is

$$\mathrm{nwr}(\mathbf{A},\mathbf{E})^{k-1}(1-\mathrm{nwr}(\mathbf{A},\mathbf{E})).$$

In the remainder of this proof we use the notation $(\mathbf{y},\mathbf{c}) \in \mathrm{WR}(\mathbf{A},\mathbf{E})$ to mean that $\mathbf{Ay}-\mathbf{Ec}$ is well-rounded. Then, conditioned on events 1 and 2, the probability of event 3 is

$$\Pr_{\substack{(\mathbf{y}_1,\mathbf{c}_1),\ldots,(\mathbf{y}_k,\mathbf{c}_k)\notin\mathrm{WR}(\mathbf{A},\mathbf{E})\\(\mathbf{y}_k,\mathbf{c}_k)\in\mathrm{WR}(\mathbf{A},\mathbf{E})}}\left[\bigvee_{i=1}^{k-1}\left[\mathbf{Ay}_k\right]_M=\left[\mathbf{Ay}_i\right]_M\right] \tag{3.16}$$
$$\le (k-1)\max_{\mathbf{w}\in\mathbb{W}}\left\{\Pr_{(\mathbf{y},\mathbf{c})\in\mathrm{WR}(\mathbf{A},\mathbf{E})}\left[\left[\mathbf{Ay}\right]_M=\mathbf{w}\right]\right\} \tag{3.17}$$
$$\le (k-1)\frac{\max_{\mathbf{w}\in\mathbb{W}}\left\{\Pr_{(\mathbf{y},\mathbf{c})\in\mathbb{Y}\times\mathbb{H}}\left[\left[\mathbf{Ay}-\mathbf{Ec}\right]_M=\mathbf{w}\right]\right\}}{\Pr_{(\mathbf{y},\mathbf{c})\in\mathbb{Y}\times\mathbb{H}}\left[\mathbf{Ay}-\mathbf{Ec} \text{ is well-rounded}\right]} \tag{3.18}$$
$$= (k-1)\frac{\mathrm{coll}(\mathbf{A},\mathbf{E})}{1-\mathrm{nwr}(\mathbf{A},\mathbf{E})}. \tag{3.19}$$

Thus,

$$\begin{aligned}\Pr_r\left[\text{inconsistent}(r)\right] &\le \sum_{k=2}^{\infty}\mathrm{nwr}(\mathbf{A},\mathbf{E})^{k-1}(1-\mathrm{nwr}(\mathbf{A},\mathbf{E}))(k-1)\frac{\mathrm{coll}(\mathbf{A},\mathbf{E})}{1-\mathrm{nwr}(\mathbf{A},\mathbf{E})}\\ &= \mathrm{coll}(\mathbf{A},\mathbf{E})\sum_{k=2}^{\infty}(k-1)\,\mathrm{nwr}(\mathbf{A},\mathbf{E})^{k-1}.\end{aligned}$$

The proposition then follows from the formula for the derivative of a geometric progression. □

An immediate corollary of Proposition 3.5 and Equation (3.15) is that for all states ρ

$$\|\Psi_{\text{c-mid}}(\rho) - \Psi_{\text{mid}}(\rho)\|_{\text{tr}} < 2 \Pr_r [\text{inconsistent}(r)] \leq 2\,\text{coll}(\mathbf{A}, \mathbf{E}) \frac{\text{nwr}(\mathbf{A}, \mathbf{E})}{(1 - \text{nwr}(\mathbf{A}, \mathbf{E}))^2}.$$

3.2.2.5 Consistent-Mid-Sign is a Mixture of Real Sign Oracles

In the previous section we introduced the signing oracle Consistent-mid-sign (Algorithm 3.20) and claimed that it behaves exactly like Sign (Algorithm 3.15) with the following exception: Repeated invocations of Sign always use the same hash oracle H, whereas each invocation of Consistent-mid-sign switches to another hash oracle H' that differs from H on a small fraction of randomly selected inputs.

We formalize this claim in the following paragraph. Therefore, we must introduce some notation. For each message m define the symbols

$\mathbf{y}_{\mathsf{m}}$:	A sequence $\{\mathbf{y}_{\mathsf{m},i}\}_{i=1}^{\infty}$ of elements drawn randomly from $\mathbb{Y}$.
$\mathbf{c}_{\mathsf{m}}(\mathbf{y}_{\mathsf{m}})$:	A sequence $\{\mathbf{c}_{\mathsf{m},i}\}_{i=1}^{\infty}$ of elements drawn randomly from $\mathbb{H}$ subject to the constraint that if $[\mathbf{A}\mathbf{y}_{\mathsf{m},i}]_M = [\mathbf{A}\mathbf{y}_{\mathsf{m},j}]_M$ then $\mathbf{c}_{\mathsf{m},i} = \mathbf{c}_{\mathsf{m},j}$.

The output of Consistent-mid-sign on input m is a deterministic function of the random data $\mathbf{y}_{\mathsf{m}}, \mathbf{c}_{\mathsf{m}}(\mathbf{y}_{\mathsf{m}})$. Specifically, let $k(\mathsf{m})$ denote the minimum index for which $\mathbf{A}\mathbf{y}_{\mathsf{m},k(\mathsf{m})} - \mathbf{E}\mathbf{c}_{\mathsf{m},k(\mathsf{m})}$ is well-rounded. Then Consistent-mid-sign outputs the signature $(\mathbf{y}_{\mathsf{m},k(\mathsf{m})} + \mathbf{S}\mathbf{c}_{\mathsf{m},k(\mathsf{m})}, \mathbf{c}_{\mathsf{m},k(\mathsf{m})})$. For shorthand, write $\tau_{\mathsf{m}} = (\mathbf{y}_{\mathsf{m}}, \mathbf{c}_{\mathsf{m}}(\mathbf{y}_{\mathsf{m}}))$. Moreover, let $\mathbf{y} = \{\mathbf{y}_{\mathsf{m}}\}_{\mathsf{m}}$ and $\mathbf{c}(\mathbf{y}) = \{\mathbf{c}_{\mathsf{m}}(\mathbf{y}_{\mathsf{m}})\}_{\mathsf{m}}$ denote selections of random data for each possible message m. For shorthand, write $\tau = (\mathbf{y}, \mathbf{c}(\mathbf{y}))$ so that the behavior of Consistent-mid-sign on all inputs is completely specified by τ.

For each choice of hash oracle H and random data τ, we consider the hash oracle H_τ that agrees with H everywhere except that $\mathsf{H}_\tau([\mathbf{A}\mathbf{y}_{\mathsf{m},i}]_M, \mathsf{m}) = \mathbf{c}_{\mathsf{m},i}$ for each message m and each $i = 1, \ldots, k(\mathsf{m})$. In other words,

$$\mathsf{H}_\tau(\mathbf{w}, \mathsf{m}) = \begin{cases} \mathbf{c}_{\mathsf{m},i} & \text{if } \mathbf{w} = [\mathbf{A}\mathbf{y}_{\mathsf{m},i}]_M \text{ for some } i \in \{1, \ldots, k(\mathsf{m})\} \\ \mathsf{H}(\mathbf{w}, \mathsf{m}) & \text{otherwise.} \end{cases},$$

The behavior of Sign (Algorithm 3.15) with hash oracle H_τ on all inputs is completely specified by $\mathbf{y}$ and H_τ. Moreover, the behavior of Sign with hash oracle H_τ and random data $\mathbf{y}$ is *identical* to the behavior of Consistent-mid-sign with random data τ. Furthermore, for each choice of random data τ we define the following quantum channels:

$\Psi_{\text{c-mid},\tau}$: The quantum channel representing the actions of Consistent-mid-sign (Algorithm 3.20) with randomness τ.

$\Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}$: The quantum channel representing the actions of Sign (Algorithm 3.15) with hash oracle H_τ and randomness $\mathbf{y}$.

The previous observations establish

$$\Psi_{\text{c-mid},\tau} = \Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}$$

for each choice of τ. Because $\Psi_{\text{c-mid}}$ is simply a uniform mixture of channels $\Psi_{\text{c-mid},\tau}$, it follows that[4]

$$\Psi_{\text{c-mid}} = \sum_\tau \Pr[\tau]\, \Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}.$$

3.2.2.6 Re-Programming of Hash Oracles is Hard to Detect

Thus far we have proved that Consistent-mid-sign behaves like a mixture of real sign oracles *when viewed in isolation*. That is, for all states ρ we have

$$\Psi_{\text{c-mid}}(\rho) = \sum_\tau \Pr[\tau]\, \Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}(\rho).$$

But we must extend this proof so that it holds even in the presence of independent information on the underlying hash oracle. In particular, for any hash oracle H and any state ρ_{H} prepared using only a tractable number of queries to H, we must show that

$$\Psi_{\text{c-mid}}(\rho_{\mathsf{H}}) \approx \sum_\tau \Pr[\tau]\, \Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}(\rho_{\mathsf{H}_\tau}).$$

To establish this claim it suffices to show that $\rho_{\mathsf{H}} \approx \rho_{\mathsf{H}_\tau}$ with high probability over the choice of τ.

This claim is proven by an application of [36, Theorem 3.3], called the BBBV Theorem. To do so, we first introduce the formalism necessary to state this theorem. Suppose ρ_{H} was prepared by some party $\mathcal{R}$ using t queries to some hash oracle $\mathsf{H} : X \to Y$. For each $i = 1, \ldots, t$ let ρ_i denote the state of $\mathcal{R}$'s system immediately prior to the ith query to H. For each hash input $x \in X$ let

$$\mathcal{Q}_{\mathcal{R}(\mathsf{H})}(x) = \sum_{i=1}^{t} \operatorname{tr}\left(|x\rangle\langle x|\, \rho_i\right)$$

denote the *query magnitude* on input x for $\mathcal{R}$'s interaction with hash oracle H, where tr is the trace of the matrices $|x\rangle\langle x|$ and ρ_i. For a definition of the trace, we refer to Section 2.3. The BBBV Theorem (or rather, a consequence of it) is as follows.

[4] Strictly speaking, $\Pr[\tau]$ is zero because it represents the uniform distribution over a countably infinite set. To be correct, we should switch to a probability measure on τ and use an integral instead of a summation over τ.

Theorem 3.6 ([36, Theorem 3.3])**.** *The following holds for each $\varepsilon > 0$. Suppose ρ_{H} was prepared by some party $\mathcal{R}$ using t queries to some hash oracle $\mathsf{H} : X \to Y$. Let H' be a hash oracle that agrees with H except on a subset $X' \subset X$ of inputs with the property that*

$$\sum_{x \in X'} \mathcal{Q}_{\mathcal{R}(\mathsf{H})}(x) \leq \frac{\varepsilon^2}{t}.$$

Furthermore, let $\rho_{\mathsf{H}'}$ be the state prepared when $\mathcal{R}$ uses hash oracle H' instead of H. It holds that $\|\rho_{\mathsf{H}'} - \rho_{\mathsf{H}}\|_{\mathrm{tr}} \leq \varepsilon$.

We are now ready to prove the claim of this section.

Proposition 3.7 (Re-Programming in TESLA)**.** *The following holds for each choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ and each $\delta > 0$. Suppose ρ_{H} was prepared by some party $\mathcal{D}$ using t queries to the hash oracle H. Let τ be random data and let H_τ be a hash oracle derived from H and τ as described in Section 3.2.2.5. Let $\rho_{\mathsf{H}'}$ be the state prepared when $\mathcal{D}$ uses the hash oracle H' instead of H. Then $\|\rho_{\mathsf{H}_\tau} - \rho_{\mathsf{H}'}\|_{\mathrm{tr}} < \delta$ except with probability at most*

$$\frac{t^2}{\delta^2} \frac{\mathrm{coll}(\mathbf{A}, \mathbf{E})}{1 - \mathrm{nwr}(\mathbf{A}, \mathbf{E})}$$

over the choice of τ.

Proof. By Theorem 3.6 it suffices to prove that the quantity

$$\sum_{\mathsf{m}} \sum_{i=1}^{k(\mathsf{m})} \mathcal{Q}_{\mathcal{D}(\mathsf{H})} \left([\mathbf{A}\mathbf{y}_{\mathsf{m},i}]_M , \mathsf{m} \right) \tag{3.20}$$

is at most δ^2/t with high probability over the choice of τ. To prove this statement, for each message m let

$$X_{\mathsf{m}} = \{([\mathbf{A}\mathbf{y}]_M , \mathsf{m}) : \mathbf{y} \in \mathbb{Y}\}$$

denote the set of hash inputs for message m that are candidates for re-programming. Furthermore, let

$$t_{\mathsf{m}} = \sum_{x \in X_{\mathsf{m}}} \mathcal{Q}_{\mathcal{D}(\mathsf{H})}(x)$$

denote the total query magnitude for message m. Observe that $t = \sum_{\mathsf{m}} t_{\mathsf{m}}$. The quantity (3.20) is maximized if for each message m all the query magnitude t_{m} alloted to message m is placed on the element $\mathbf{w} \in \mathbb{W}$ most likely to collide with $[\mathbf{A}\mathbf{y}]_M$ when $\mathbf{y} \in \mathbb{Y}$ is chosen uniformly at random. In this case, the quantity (3.20) is at most

$$\sum_{\mathsf{m}} k(\mathsf{m}) t_{\mathsf{m}} \, \mathrm{coll}(\mathbf{A}, \mathbf{E}).$$

By Markov's inequality we have

$$\Pr_{\tau}\left[\sum_{\mathsf{m}} k(\mathsf{m}) t_{\mathsf{m}} \operatorname{coll}(\mathbf{A},\mathbf{E}) \geq \frac{\delta^2}{t}\right] \leq \frac{t}{\delta^2} \operatorname*{Ex}_{\tau}\left[\sum_{\mathsf{m}} k(\mathsf{m}) t_{\mathsf{m}} \operatorname{coll}(\mathbf{A},\mathbf{E})\right] \tag{3.21}$$

$$= \frac{t}{\delta^2} \operatorname{coll}(\mathbf{A},\mathbf{E}) \sum_{\mathsf{m}} t_{\mathsf{m}} \operatorname*{Ex}_{\tau_{\mathsf{m}}}\left[k(\mathsf{m})\right]. \tag{3.22}$$

Thus, it suffices to bound the expected number $k(\mathsf{m})$ of entries one must view from a given list τ_{m} before encountering an entry $(\mathbf{y},\mathbf{c})$ for which $\mathbf{Ay}-\mathbf{Ec}$ is well-rounded. We have

$$\operatorname*{Ex}_{\tau_{\mathsf{m}}}\left[k(\mathsf{m})\right] = \sum_{k=1}^{\infty} k \operatorname{nwr}(\mathbf{A},\mathbf{E})^{k-1}(1-\operatorname{nwr}(\mathbf{A},\mathbf{E})) = \frac{1}{1-\operatorname{nwr}(\mathbf{A},\mathbf{E})}$$

where the final equality follows from the formula for the derivative of a geometric progression. The proposition follows from $\sum_{\mathsf{m}} t_{\mathsf{m}} = t$. □

3.2.2.7 The Distinguisher's State–Revisited

Recall from Section 3.2.2.2 the state ρ_{H}, which is the state of $\mathcal{D}$'s system immediately prior to the sign query in the first block. Let $\kappa_1 \leq q_h$ denote the query magnitude on the live-switch for the hash oracle in the first block. We proved the following statements in the previous sections:

$$\begin{aligned}
\Psi_{\text{sim}}(\rho_{\mathsf{H}}) &= \Psi_{\text{mid}}(\rho_{\mathsf{H}}),\\
\|\Psi_{\text{mid}}(\rho_{\mathsf{H}}) - \Psi_{\text{c-mid}}(\rho_{\mathsf{H}})\|_{\text{tr}} &< 2\operatorname{coll}(\mathbf{A},\mathbf{E})\frac{\operatorname{nwr}(\mathbf{A},\mathbf{E})}{(1-\operatorname{nwr}(\mathbf{A},\mathbf{E}))^2},\\
\Psi_{\text{c-mid}}(\rho_{\mathsf{H}}) &= \sum_{\tau} \Pr[\tau]\, \Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}(\rho_{\mathsf{H}}),\\
\Pr_{\tau}\left[\|\rho_{\mathsf{H}_\tau} - \rho_{\mathsf{H}}\|_{\text{tr}} > \varepsilon\right] &< \frac{\kappa_1^2}{\varepsilon^2}\frac{\operatorname{coll}(\mathbf{A},\mathbf{E})}{1-\operatorname{nwr}(\mathbf{A},\mathbf{E})}.
\end{aligned}$$

We conclude that

$$\left\|\Psi_{\text{sim}}(\rho_{\mathsf{H}}) - \sum_{\tau}\Pr[\tau]\,\Psi_{\text{real},\mathsf{H}_\tau,\mathbf{y}}(\rho_{\mathsf{H}_\tau})\right\|_{\text{tr}} < \delta(\kappa_1)$$

where

$$\delta(\kappa_1) = 2\operatorname{coll}(\mathbf{A},\mathbf{E})\frac{\operatorname{nwr}(\mathbf{A},\mathbf{E})}{(1-\operatorname{nwr}(\mathbf{A},\mathbf{E}))^2} + \varepsilon + \frac{\kappa_1^2}{\varepsilon^2}\frac{\operatorname{coll}(\mathbf{A},\mathbf{E})}{1-\operatorname{nwr}(\mathbf{A},\mathbf{E})}. \tag{3.23}$$

The final two terms of (3.23) are due to the fact that a $\frac{\kappa^2}{\varepsilon^2}\frac{\operatorname{nwr}(\mathbf{A},\mathbf{E})}{(1-\operatorname{nwr}(\mathbf{A},\mathbf{E}))^2}$-fraction of τ lead to a hash oracle H_τ for which $\|\rho_{\mathsf{H}} - \rho_{\mathsf{H}_\tau}\|_{\text{tr}} > \varepsilon$, in which case we assume

that $\rho_{\mathsf{H}}, \rho_{\mathsf{H}_\tau}$ are perfectly distinguishable. For all other τ it holds that ρ_{H} and ρ_{H_τ} are ε-close.

Equation (3.23) means that the state of $\mathcal{D}$'s system at the end of the first block of an interaction with simulated oracles is $\delta(\kappa_1)$-close to a probabilistic mixture over states of $\mathcal{D}$'s system, each of which could have been obtained from an interaction with real hash oracles.

As suggested in Section 3.2.2.2, we continue inductively throughout the q_s blocks. As with κ_1, let $\kappa_2, \ldots, \kappa_{q_s}$ denote the query magnitude on the live-switch for blocks two through q_s. Define

$$\delta_{\text{yes}} = \sum_{i=1}^{q_s} \delta(\kappa_i)$$

and observe that the state of $\mathcal{D}$'s system at the end of an interaction with simulated oracles is δ_{yes}-close to a probabilistic mixture over states obtained from an interaction with real hash oracles.

We compute an upper bound on δ_{yes} next. Moreover, because each block includes information from hash queries in previous blocks plus one additional hash query learned from the signing oracle, we have

$$\kappa_{i+1} \geq \kappa_i + 1.$$

Because $\mathcal{D}$ is permitted at most q_h total query magnitude on the live-switch for its hash queries, we have

$$\kappa_i \leq q_h + i - 1.$$

It is clear that δ_{yes} is maximized when $\kappa_1 = q_h$, which corresponds to a distinguisher who makes all q_h hash queries before making any of the q_s sign queries. Taking a loose bound $q_h + q_s$ for each κ_i, we obtain

$$\delta_{\text{yes}} < q_s \left(2\,\text{coll}(\mathbf{A}, \mathbf{E}) \frac{\text{nwr}(\mathbf{A}, \mathbf{E})}{(1 - \text{nwr}(\mathbf{A}, \mathbf{E}))^2} + \varepsilon + \frac{(q_h + q_s)^2}{\varepsilon^2} \frac{\text{coll}(\mathbf{A}, \mathbf{E})}{1 - \text{nwr}(\mathbf{A}, \mathbf{E})} \right). \quad (3.24)$$

Finally, because the real and simulated oracles are δ_{yes}-close, as desired, it follows that

$$\Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ yes-instance of M-LWE}\right] > \Pr\left[\text{forge}(\mathbf{A}, \mathbf{B})\right] - \delta_{\text{yes}}.$$

3.2.2.8 Probability of Well-Roundedness

We now turn to prove the intermediate result about well-roundedness used in Proposition 3.5. Let ϕ denote the probability that a random vector in $\mathbb{Z}_q^m$ is not well-rounded:

$$\phi = \Pr_{x \in \mathbb{Z}_q^m}\left[x \text{ not well-rounded}\right] \leq m \left(\frac{2L_E}{2^d} + \frac{2L_E}{q} \right). \quad (3.25)$$

The quantity ϕ is a function of the TESLA parameters q, m, d, L_E. It is a constant that is noticeably smaller than 1.

Recall the definition of $\mathrm{nwr}(\mathbf{A}, \mathbf{E})$. For TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ define $\mathrm{nwr}(\mathbf{A}, \mathbf{E})$ as the probability over $(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}$ that $\mathbf{Ay} - \mathbf{Ec}$ is not well-rounded:

$$\mathrm{nwr}(\mathbf{A}, \mathbf{E}) = \Pr_{(\mathbf{y},\mathbf{c}) \in \mathbb{Y} \times \mathbb{H}} [\mathbf{Ay} - \mathbf{Ec} \text{ not well-rounded}] .$$

We prove the following.

Lemma 3.8 (Probability of Well-Roundedness)**.** *The following holds for all $K > 0$. With probability $1 - 1/K$ over the choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ it holds that*

$$\mathrm{nwr}(\mathbf{A}, \mathbf{E}) \leq \phi + \sqrt{\frac{K(q+1)}{\#\mathbb{Y}}}.$$

Proof. Our strategy is to bound the variance of $\mathrm{nwr}(\mathbf{A}, \mathbf{E})$ over the choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ and use Chebyshev's inequality. By definition it holds that

$$\operatorname*{Var}_{(\mathbf{A},\mathbf{E})} [\mathrm{nwr}(\mathbf{A}, \mathbf{E})] = \operatorname*{Ex}_{(\mathbf{A},\mathbf{E})} \left[\mathrm{nwr}(\mathbf{A}, \mathbf{E})^2\right] - \operatorname*{Ex}_{(\mathbf{A},\mathbf{E})} [\mathrm{nwr}(\mathbf{A}, \mathbf{E})]^2 .$$

So it suffices to compute the expectation of $\mathrm{nwr}(\mathbf{A}, \mathbf{E})$ and an upper bound on the expectation of $\mathrm{nwr}(\mathbf{A}, \mathbf{E})^2$. We begin by computing the expectation of $\mathrm{nwr}(\mathbf{A}, \mathbf{E})$. We have

$$\begin{aligned}
&\operatorname*{Ex}_{(\mathbf{A},\mathbf{E})} [\mathrm{nwr}(\mathbf{A}, \mathbf{E})] \\
&= \sum_{(\mathbf{A},\mathbf{E})} \Pr[(\mathbf{A}, \mathbf{E})] \frac{1}{\#\mathbb{Y}\#\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \mathrm{bool}\,[\mathbf{Ay} - \mathbf{Ec} \text{ not well-rounded}] \\
&= \frac{1}{\#\mathbb{Y}\#\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \sum_{(\mathbf{A},\mathbf{E})} \Pr[(\mathbf{A}, \mathbf{E})]\, \mathrm{bool}\,[\mathbf{Ay} - \mathbf{Ec} \text{ not well-rounded}] \\
&= \frac{1}{\#\mathbb{Y}\#\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \Pr_{(\mathbf{A},\mathbf{E})} [\mathbf{Ay} - \mathbf{Ec} \text{ not well-rounded}] .
\end{aligned}$$

(Here we have used the notation $\mathrm{bool}\,[s]$ for any statement s that can be either true or false to mean that $\mathrm{bool}\,[s] = 1$ if the statement is true and $\mathrm{bool}\,[s] = 0$ otherwise.) We aim to bound the probability

$$\Pr_{(\mathbf{A},\mathbf{E})} [\mathbf{Ay} - \mathbf{Ec} \text{ not well-rounded}] \tag{3.26}$$

for each fixed choice of $(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}$. There are two cases:

1. If $\mathbf{y} \neq 0$ then $\mathbf{Ay}$ is a uniformly random vector in $\mathbb{Z}_q^m$. So too is $\mathbf{Ay} - \mathbf{Ec}$, since $\mathbf{c}$ is fixed and $\mathbf{E}$ is independent of $\mathbf{A}$. In this case, the probability (3.26) equals ϕ.
2. If $\mathbf{y} = 0$ then the probability (3.26) equals 0, since $-\mathbf{Ec}$ is well-rounded for all $\mathbf{E}, \mathbf{c}$.

Case 2 occurs with probability $1/\#\mathbb{Y}$, from which it follows that

$$\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{nwr}(\mathbf{A},\mathbf{E})\right] = \left(1 - \frac{1}{\#\mathbb{Y}}\right)\phi. \tag{3.27}$$

Next, we compute an upper bound on the expectation of $\mathrm{nwr}(\mathbf{A},\mathbf{E})^2$. Similar to the above, we have

$$\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{nwr}(\mathbf{A},\mathbf{E})^2\right] = \frac{1}{(\#\mathbb{Y}\#\mathbb{H})^2}\sum_{(\mathbf{y},\mathbf{c}),(\mathbf{y}',\mathbf{c}')}\underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[\mathbf{Ay} - \mathbf{Ec}, \mathbf{Ay}' - \mathbf{Ec}' \text{ not well-rounded}\right]$$

and so we aim at bounding the probability

$$\underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[\mathbf{Ay} - \mathbf{Ec}, \mathbf{Ay}' - \mathbf{Ec}' \text{ not well-rounded}\right] \tag{3.28}$$

for each fixed choice of $(\mathbf{y},\mathbf{c}), (\mathbf{y}',\mathbf{c}') \in \mathbb{Y} \times \mathbb{H}$. There are again two cases:

1. If $\mathbf{y}, \mathbf{y}'$ are non-zero and linearly independent then $\mathbf{Ay}, \mathbf{Ay}'$ are uniformly random vectors in $\mathbb{Z}_q^m$; so too are $\mathbf{Ay} - \mathbf{Ec}, \mathbf{Ay}' - \mathbf{Ec}'$. In this case, the probability (3.28) equals ϕ^2.
2. If $\mathbf{y}, \mathbf{y}'$ are linearly dependent then the probability (3.28) is at most 1.

Case 2 occurs with probability at most $(q+1)/\#\mathbb{Y}$, from which it follows that

$$\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{nwr}(\mathbf{A},\mathbf{E})^2\right] \leq \left(1 - \frac{q+1}{\#\mathbb{Y}}\right)\phi^2 + \frac{q+1}{\#\mathbb{Y}}.$$

Combining these bounds on the expectation of $\mathrm{nwr}(\mathbf{A},\mathbf{E})$ and $\mathrm{nwr}(\mathbf{A},\mathbf{E})^2$ (and employing the inequality $1 - (q+1)/\#\mathbb{Y} < (1 - 1/\#\mathbb{Y})^2$), we obtain the inequality

$$\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Var}}\left[\mathrm{nwr}(\mathbf{A},\mathbf{E})\right] \leq \frac{q+1}{\#\mathbb{Y}}.$$

By Chebyshev's inequality it then holds that

$$\underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[\left|\mathrm{nwr}(\mathbf{A},\mathbf{E}) - \underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{nwr}(\mathbf{A},\mathbf{E})\right]\right| \geq \sqrt{\frac{K(q+1)}{\#\mathbb{Y}}}\right] \leq \frac{1}{K}.$$

The lemma follows from the expression (3.27) for the expectation of $\mathrm{nwr}(\mathbf{A},\mathbf{E})$. □

3.2.2.9 Probability of Repetition

Now we move on to prove another intermediate result, namely about the bound of $\mathrm{coll}(\mathbf{A}, \mathbf{E})$. Let ψ denote the probability that a random vector $\mathbf{x} \in \mathbb{Z}_q^m$ is in $\Delta\mathbb{L}$:

$$\psi = \Pr_{\mathbf{x} \in \mathbb{Z}_q^m} [\mathbf{x} \in \Delta\mathbb{L}] \leq \left(\frac{2^d}{q}\right)^m . \tag{3.29}$$

The quantity ψ is a function of the TESLA parameters q, m, d. It is negligibly small for our concrete choices of parameters. Moving further, we recall the definition of $\mathrm{coll}(\mathbf{A}, \mathbf{E})$. For TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ define $\mathrm{coll}(\mathbf{A}, \mathbf{E})$ as the maximum over all $\mathbf{w} \in \mathbb{W}$ of the probability over $(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}$ that $[\mathbf{A}\mathbf{y} - \mathbf{E}\mathbf{c}]_M = \mathbf{w}$:

$$\mathrm{coll}(\mathbf{A}, \mathbf{E}) = \max_{\mathbf{w} \in \mathbb{W}} \left\{ \Pr_{(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H}} [[\mathbf{A}\mathbf{y} - \mathbf{E}\mathbf{c}]_M = \mathbf{w}] \right\} .$$

Next we prove the following lemma.

Lemma 3.9 (Probability of Repetition)**.** *The following holds for all $K > 0$. With probability $1 - 1/K$ over the choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ it holds that*

$$\mathrm{coll}(\mathbf{A}, \mathbf{E}) \leq K\psi.$$

Before proving Lemma 3.9, we define the set

$$\mathbb{G}(\mathbf{A}, \mathbf{E}) = \{(\mathbf{y}, \mathbf{c}) \in \Delta\mathbb{Y} \times \Delta\mathbb{H} : \mathbf{A}\mathbf{y} - \mathbf{E}\mathbf{c} \in \Delta\mathbb{L}\} .$$

Some basic facts about the set $\mathbb{G}(\mathbf{A}, \mathbf{E})$ are listed below in Lemma 3.10.

Proof of Lemma 3.9. Let $\mathrm{coll}'(\mathbf{A}, \mathbf{E})$ denote the probability over $(\mathbf{y}, \mathbf{c}) \in \Delta\mathbb{Y} \times \Delta\mathbb{H}$ that $(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})$:

$$\mathrm{coll}'(\mathbf{A}, \mathbf{E}) = \Pr_{(\mathbf{y}, \mathbf{c}) \in \Delta\mathbb{Y} \times \Delta\mathbb{H}} [(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})] .$$

It follows from Lemma 3.10 that $\mathrm{coll}'(\mathbf{A}, \mathbf{E}) \geq \mathrm{coll}(\mathbf{A}, \mathbf{E})$. Thus, it suffices to prove the lemma with $\mathrm{coll}'(\mathbf{A}, \mathbf{E})$ in place of $\mathrm{coll}(\mathbf{A}, \mathbf{E})$. Moreover, it follows from Lemma 3.10 that $\#\mathbb{G}(\mathbf{A}, \mathbf{E}) \geq \#\Delta\mathbb{H}$. Hence, it holds that $\mathrm{coll}'(\mathbf{A}, \mathbf{E}) \geq 1/\#\Delta\mathbb{Y}$. Our strategy is to bound the expectation of the positive random variable $\mathrm{coll}'(\mathbf{A}, \mathbf{E}) - 1/\#\Delta\mathbb{Y}$ over the choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ and use Markov's inequality. In order to do so, we first compute the expectation of

$\mathrm{coll}'(\mathbf{A}, \mathbf{E})$:

$$\begin{aligned}\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{coll}'(\mathbf{A}, \mathbf{E})\right] &= \sum_{(\mathbf{A},\mathbf{E})} \Pr\left[(\mathbf{A}, \mathbf{E})\right] \frac{1}{\#\Delta\mathbb{S}\#\Delta\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \mathrm{bool}\left[(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})\right] \\ &= \frac{1}{\#\Delta\mathbb{S}\#\Delta\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \sum_{(\mathbf{A},\mathbf{E})} \Pr\left[(\mathbf{A}, \mathbf{E})\right] \mathrm{bool}\left[(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})\right] \\ &= \frac{1}{\#\Delta\mathbb{S}\#\Delta\mathbb{H}} \sum_{(\mathbf{y},\mathbf{c})} \underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})\right].\end{aligned}$$

So we aim at bounding the probability

$$\underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[(\mathbf{y}, \mathbf{c}) \in \mathbb{G}(\mathbf{A}, \mathbf{E})\right] \tag{3.30}$$

for each fixed choice of $(\mathbf{y}, \mathbf{c}) \in \Delta\mathbb{Y} \times \Delta\mathbb{H}$. There are again two cases:

1. If $\mathbf{y} \neq 0$ then $\mathbf{Ay}$ is a uniformly random vector in $\mathbb{Z}_q^m$. So too is $\mathbf{Ay} - \mathbf{Ec}$, since $\mathbf{c}$ is fixed and $\mathbf{E}$ is independent of $\mathbf{A}$. In this case, the probability (3.30) is exactly ψ.
2. If $\mathbf{y} = 0$ then the probability (3.30) is exactly 1.

Case 2 occurs with probability exactly $1/\#\Delta\mathbb{Y}$ over the choice of $(\mathbf{y}, \mathbf{c})$. It follows that

$$\underset{(\mathbf{A},\mathbf{E})}{\mathrm{Ex}}\left[\mathrm{coll}'(\mathbf{A}, \mathbf{E})\right] = \left(1 - \frac{1}{\#\Delta\mathbb{Y}}\right)\psi + \frac{1}{\#\Delta\mathbb{Y}}.$$

Then by Markov's inequality we have

$$\underset{(\mathbf{A},\mathbf{E})}{\Pr}\left[\mathrm{coll}'(\mathbf{A}, \mathbf{E}) \geq K\left(1 - \frac{1}{\#\Delta\mathbb{Y}}\right)\psi\right] \leq \frac{1}{K}.$$

That is, with probability at least $1 - 1/K$ over the choice of TESLA keys $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ it holds that

$$\mathrm{coll}'(\mathbf{A}, \mathbf{E}) \leq K\left(1 - \frac{1}{\#\Delta\mathbb{Y}}\right)\psi$$

from which the lemma follows. □

Lemma 3.10. *For all TESLA keys* $(\mathbf{A}, \mathbf{B})$, $(\mathbf{S}, \mathbf{E}, \mathbf{A})$ *and all* $\mathbf{w} \in \mathbb{W}$ *it holds that*

$$\#\mathbb{G}(\mathbf{A}, \mathbf{E}) \geq \#\Delta\mathbb{H}, \text{ and} \tag{3.31}$$
$$\#\mathbb{G}(\mathbf{A}, \mathbf{E}) \geq \#\left\{(\mathbf{y}, \mathbf{c}) \in \mathbb{Y} \times \mathbb{H} : [\mathbf{Ay} - \mathbf{Ec}]_M = \mathbf{w}\right\}. \tag{3.32}$$

Proof. The Inequality (3.31) is straightforward: For each $\mathbf{c}, \mathbf{c}' \in \mathbb{H}$ we have $[\mathbf{Ec}]_M = [\mathbf{Ec}']_M = [0]_M$, from which it follows that $(0, \mathbf{c} - \mathbf{c}') \in \mathbb{G}(\mathbf{A}, \mathbf{E})$.

It remains to prove the Inequality (3.32). Let $(\mathbf{y}, \mathbf{c}), (\mathbf{y}', \mathbf{c}')$ be elements of $\mathbb{Y} \times \mathbb{H}$ with $[\mathbf{Ay} - \mathbf{Ec}]_M = [\mathbf{Ay}' - \mathbf{Ec}']_M = \mathbf{w}$. We claim that $(\mathbf{y} - \mathbf{y}', \mathbf{c} - \mathbf{c}')$ is in $\mathbb{G}(\mathbf{A}, \mathbf{E})$. It is clear that $\mathbf{y} - \mathbf{y}' \in \Delta\mathbb{Y}$ and $\mathbf{c} - \mathbf{c}' \in \Delta\mathbb{H}$. It remains to verify $\mathbf{A}(\mathbf{y} - \mathbf{y}') - \mathbf{E}(\mathbf{c} - \mathbf{c}') \in \Delta\mathbb{L}$. We have

$$\mathbf{A}(\mathbf{y} - \mathbf{y}') - \mathbf{E}(\mathbf{c} - \mathbf{c}') = \mathbf{Ay} - \mathbf{Ec} - (\mathbf{Ay}' - \mathbf{Ec}').$$

Since $\mathbf{Ay} - \mathbf{Ec}$ and $\mathbf{Ay}' - \mathbf{Ec}'$ have the same high bits, it must be that $\mathbf{A}(\mathbf{y} - \mathbf{y}') - \mathbf{E}(\mathbf{c} - \mathbf{c}')$ is the difference of two vectors from $[-(2^{d-1} - 1), 2^{d-1}]^m$, from which it follows that $\mathbf{A}(\mathbf{y} - \mathbf{y}') - \mathbf{E}(\mathbf{c} - \mathbf{c}') \in \Delta\mathbb{L}$.

If $(\mathbf{y}_1, \mathbf{c}_1), \ldots, (\mathbf{y}_k, \mathbf{c}_k)$ are distinct elements of $\mathbb{Y} \times \mathbb{H}$ with $[\mathbf{Ay}_i - \mathbf{Ec}_i]_M = \mathbf{w}$ for each $i = 1, \ldots, k$ then $(0, 0), (\mathbf{y}_1 - \mathbf{y}_2, \mathbf{c}_1 - \mathbf{c}_2), \ldots, (\mathbf{y}_1 - \mathbf{y}_k, \mathbf{c}_1 - \mathbf{c}_k)$ must be distinct elements of $\mathbb{G}(\mathbf{A}, \mathbf{E})$[5]. We have thus listed k distinct elements of $\mathbb{G}(\mathbf{A}, \mathbf{E})$, from which the lemma follows. □

3.2.3 No-Instances of M-LWE

In this section, we prove that the probability

$$\Pr[\mathcal{S} \text{ output “yes”} \mid (\mathbf{A}, \mathbf{B}) \text{ no-instance of M-LWE}] \tag{3.33}$$

is small. Our strategy is to identify a correspondence between valid message-signature pairs and “good” inputs to the hash oracle. We then argue that, with high probability over the choice of M-LWE no-instances $(\mathbf{A}, \mathbf{B})$ and hash oracle H, the number of good inputs is a very small fraction of the total number of inputs.

Moreover, whether a given input is good is determined solely by its corresponding output from the hash oracle, implying that the only way to discover good inputs is to perform a search through an unstructured space.

Thus, a computationally bounded forger cannot expect to find a good input (and hence, a valid forgery), even with quantum access to the random oracle. This argument establishes the claim that the M-LWE solver $\mathcal{S}$ outputs “yes” only with small probability, as desired.

3.2.3.1 Correspondence Between Valid Signatures and Good Hash Inputs

Let $\mathbf{w} \in \mathbb{W}$, and let m be an arbitrary message. For any fixed choice of random oracle H and M-LWE no-instance $(\mathbf{A}, \mathbf{B})$ the hash input $(\mathbf{w}, \mathsf{m})$ is called *good for* $\mathsf{H}, \mathbf{A}, \mathbf{B}$ if there exists $\mathbf{z} \in \mathbb{S}$ with

$$[\mathbf{Az} - \mathbf{B}\,\mathsf{H}(\mathbf{w}, \mathsf{m})]_M = \mathbf{w}.$$

[5] By contrast with no-instances, we cannot guarantee that the negations are distinct.

Proposition 3.11 (Correspondence Between Valid Signatures and Good Hash Inputs)**.** *If* $(\mathsf{m}, (\mathbf{z}, \mathbf{c}))$ *is a valid message-signature pair for public key* $(\mathbf{A}, \mathbf{B})$ *and hash oracle* H *then* $([\mathbf{Az} - \mathbf{Bc}]_M, \mathsf{m})$ *is good for* $\mathsf{H}, \mathbf{A}, \mathbf{B}$*.*

Proof. We write $\mathbf{w} = [\mathbf{Az} - \mathbf{Bc}]_M$. Because $(\mathbf{z}, \mathbf{c})$ is a valid signature for m we have

$$\mathsf{H}(\mathbf{w}, \mathsf{m}) = \mathsf{H}([\mathbf{Az} - \mathbf{Bc}]_M, \mathsf{m}) = \mathbf{c}.$$

Then it holds that

$$[\mathbf{Az} - \mathbf{B}\,\mathsf{H}(\mathbf{w}, \mathsf{m})]_M = [\mathbf{Az} - \mathbf{Bc}]_M = \mathbf{w}$$

as desired. □

A corollary of Proposition 3.11 is that the ability to find a message-signature pair $(\mathsf{m}, (\mathbf{z}, \mathbf{c}))$ that is valid for public key $(\mathbf{A}, \mathbf{B})$ using q_h classical or quantum queries to H implies the ability to find a hash input $(\mathbf{w}, \mathsf{m})$ that is good for $\mathsf{H}, \mathbf{A}, \mathbf{B}$ using the same number of classical or quantum queries to H.

3.2.3.2 The Fraction of Good Hash Inputs

We wish to bound the probability over hash oracles H and M-LWE no-instances $(\mathbf{A}, \mathbf{B})$ that a non-negligible fraction of hash inputs $(\mathbf{w}, \mathsf{m})$ are good. In order to do this, define the sets

$$\mathbb{M} = \{(\mathbf{w}, \mathsf{m}) : \mathbf{w} \in \mathbb{W}, \mathsf{m} \text{ is a message}\}, \text{ and}$$
$$\mathbb{M}(\mathsf{H}, \mathbf{A}, \mathbf{B}) = \{(\mathbf{w}, \mathsf{m}) \in \mathbb{M} : (\mathbf{w}, \mathsf{m}) \text{ is good for } \mathsf{H}, \mathbf{A}, \mathbf{B}\}.$$

For ease of exposition we presume a fixed, large upper bound such as 2^{2^λ} on the size of $\mathbb{M}$ although strictly speaking $\mathbb{M}$ and $\mathbb{M}(\mathsf{H}, \mathbf{A}, \mathbf{B})$ are infinite sets. After all, no computationally bounded forger could possibly query the hash oracle on inputs whose bit length exceeds 2^λ. Under this presumption, $\mathbb{M}$ is a finite set and so $\#\mathbb{M}$ is a positive integer. The ratio

$$\frac{\#\mathbb{M}(\mathsf{H}, \mathbf{A}, \mathbf{B})}{\#\mathbb{M}} \tag{3.34}$$

is the fraction of inputs that are good.

Our goal is to show that the ratio given in Equation (3.34) is negligibly small with high probability over the choice of $\mathsf{H}, \mathbf{A}, \mathbf{B}$. In order to prove this, we define for each message $(\mathbf{w}, \mathsf{m}) \in \mathbb{M}$ the boolean random variable

$$X_{(\mathbf{w},\mathsf{m})} = \begin{cases} 1 & \text{if } (\mathbf{w}, \mathsf{m}) \text{ is good for } \mathsf{H}, \mathbf{A}, \mathbf{B}, \\ 0 & \text{otherwise,} \end{cases}$$

and observe

$$\frac{\#\mathbb{M}(\mathsf{H},\mathbf{A},\mathbf{B})}{\#\mathbb{M}} = \frac{1}{\#\mathbb{M}} \sum_{(\mathbf{w},\mathsf{m})\in\mathbb{M}} X_{(\mathbf{w},\mathsf{m})},$$

which is an average over boolean random variables. Moreover, the random variables $X_{(\mathbf{w},\mathsf{m})}$ are independent and so we may apply Hoeffding's bounds to obtain

$$\Pr_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[\frac{\#\mathbb{M}(\mathsf{H},\mathbf{A},\mathbf{B})}{\#\mathbb{M}} - \operatorname*{Ex}_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[\frac{\#\mathbb{M}(\mathsf{H},\mathbf{A},\mathbf{B})}{\#\mathbb{M}}\right] \geq \delta\right] \leq e^{-2\#\mathbb{M}\delta^2}. \tag{3.35}$$

Because $\#\mathbb{M}$ is very large relative to other TESLA parameters, we may choose δ so small that it can safely be assumed to equal zero. For example, if $\#\mathbb{M} = 2^{2^\lambda}$ then the probability (3.35) is negligibly small even if δ is as small as $2^{-2^{\lambda-2}}$. Thus, the ratio (3.34) is almost certain to be very close to its expectation

$$\operatorname*{Ex}_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[\frac{\#\mathbb{M}(\mathsf{H},\mathbf{A},\mathbf{B})}{\#\mathbb{M}}\right]. \tag{3.36}$$

This expectation equals

$$\frac{1}{\#\mathbb{M}} \sum_{(\mathbf{w},\mathsf{m})\in\mathbb{M}} \operatorname*{Ex}_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[X_{(\mathbf{w},\mathsf{m})}\right],$$

and by definition it holds that

$$\operatorname*{Ex}_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[X_{(\mathbf{w},\mathsf{m})}\right] = \Pr_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[(\mathbf{w},\mathsf{m}) \text{ is good for } \mathsf{H},\mathbf{A},\mathbf{T}\right].$$

It remains to bound this probability for each hash input $(\mathbf{w},\mathsf{m})$.

3.2.3.3 Good Hash Inputs are Rare

For each choice of $\mathbf{w} \in \mathbb{W}$ and M-LWE no-instance $(\mathbf{A},\mathbf{B})$, we define the set $\mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B}) \subset \mathbb{H}$ as

$$\mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B}) = \{c \in \mathbb{H} \mid \exists z \in \mathbb{S} : [\mathbf{A}\mathbf{z} - \mathbf{B}\mathbf{c}]_M = \mathbf{w}\}.$$

Observe that a hash input $(\mathbf{w},\mathsf{m})$ is good for $\mathsf{H},\mathbf{A},\mathbf{B}$ if and only if $\mathsf{H}(\mathbf{w},\mathsf{m}) \in \mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B})$. Thus,

$$\Pr_{\mathsf{H},(\mathbf{A},\mathbf{B})}\left[(\mathbf{w},\mathsf{m}) \text{ is good for } \mathsf{H},\mathbf{A},\mathbf{B}\right] = \operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\frac{\#\mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B})}{\#\mathbb{H}}\right].$$

We prove the following.

Proposition 3.12 (Good Hash Inputs are Rare)**.** *For all* $\mathbf{w} \in \mathbb{W}$ *it holds that*

$$\operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\max_{\mathbf{w}\in\mathbb{W}}\left\{\frac{\#\mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B})}{\#\mathbb{H}}\right\}\right] \leq \frac{1}{2\#\mathbb{H}}\left(1+\frac{\#\Delta\mathbb{H}\#\Delta\mathbb{S}\#\Delta\mathbb{L}}{q^m}\right).$$

Proof. Define the set

$$\mathbb{D}(\mathbf{A},\mathbf{B}) = \{\mathbf{b}\in\Delta\mathbb{H} : \exists\mathbf{y}\in\Delta\mathbb{S} \text{ with } \mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\}. \tag{3.37}$$

In Lemma 3.13 below we prove

$$\#\mathbb{H}(\mathbf{w},\mathbf{A},\mathbf{B}) \leq \frac{\#\mathbb{D}(\mathbf{A},\mathbf{B})+1}{2}$$

for all $\mathbf{w}\in\mathbb{W}$. Thus, it suffices to bound the expectation

$$\operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\frac{\#\mathbb{D}(\mathbf{A},\mathbf{B})+1}{2\#\mathbb{H}}\right] = \frac{1}{2\#\mathbb{H}}\left(1+\operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\#\mathbb{D}(\mathbf{A},\mathbf{B})\right]\right).$$

It holds that

$$\begin{aligned}
\operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\#\mathbb{D}(\mathbf{A},\mathbf{B})\right] &= \frac{1}{\#(\mathbf{A},\mathbf{B})}\sum_{(\mathbf{A},\mathbf{B})}\#\left\{\mathbf{b}\in\Delta\mathbb{H} : \exists\mathbf{y}\in\Delta\mathbb{S} \text{ with } \mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right\}\\
&= \frac{1}{\#(\mathbf{A},\mathbf{B})}\sum_{(\mathbf{A},\mathbf{B})}\sum_{\mathbf{b}\in\Delta\mathbb{H}}\operatorname{bool}\left[\exists\mathbf{y}\in\Delta\mathbb{S} \text{ with } \mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right]\\
&\leq \frac{1}{\#(\mathbf{A},\mathbf{B})}\sum_{(\mathbf{A},\mathbf{B})}\sum_{\mathbf{b}\in\Delta\mathbb{H}}\sum_{\mathbf{y}\in\Delta\mathbb{S}}\operatorname{bool}\left[\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right]\\
&= \sum_{\mathbf{b}\in\Delta\mathbb{H}}\sum_{\mathbf{y}\in\Delta\mathbb{S}}\frac{1}{\#(\mathbf{A},\mathbf{B})}\sum_{(\mathbf{A},\mathbf{B})}\operatorname{bool}\left[\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right]\\
&= \sum_{\mathbf{b}\in\Delta\mathbb{H}}\sum_{\mathbf{y}\in\Delta\mathbb{S}}\Pr_{(\mathbf{A},\mathbf{B})}\left[\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right].
\end{aligned}$$

For each fixed choice of $\mathbf{y}\in\Delta\mathbb{S}$, $\mathbf{b}\in\Delta\mathbb{H}$[6], if $\mathbf{A},\mathbf{B}$ are uniformly random matrices then $\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}$ is a uniformly random vector from $\mathbb{Z}_q^m$. Thus, the probability $\Pr_{(\mathbf{A},\mathbf{B})}\left[\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right]$ is simply the probability that a random vector lands in $\Delta\mathbb{L}$. That is,

$$\Pr_{(\mathbf{A},\mathbf{B})}\left[\mathbf{A}\mathbf{y}-\mathbf{B}\mathbf{b}\in\Delta\mathbb{L}\right] = \frac{\#\Delta\mathbb{L}}{q^m}.$$

Thus, the expectation becomes

$$\operatorname*{Ex}_{(\mathbf{A},\mathbf{B})}\left[\#\mathbb{D}(\mathbf{A},\mathbf{B})\right] \leq \sum_{\mathbf{b}\in\Delta\mathbb{H}}\sum_{\mathbf{y}\in\Delta\mathbb{S}}\frac{\#\Delta\mathbb{L}}{q^m} = \frac{\#\Delta\mathbb{H}\#\Delta\mathbb{S}\#\Delta\mathbb{L}}{q^m} \tag{3.38}$$

as desired. □

[6]Except for the choice that $\mathbf{y}=0$ and $\mathbf{b}=0$. However, the probability that $\mathbf{b}=0$ is negligible.

Lemma 3.13. *Let* $\mathbb{D}(\mathbf{A}, \mathbf{B})$ *be as defined in* (3.37)*. For all M-LWE no-instances* $(\mathbf{A}, \mathbf{B})$ *and all* $\mathbf{w} \in \mathbb{W}$ *it holds that*

$$\#\mathbb{H}(\mathbf{w}, \mathbf{A}, \mathbf{B}) \leq \frac{\#\mathbb{D}(\mathbf{A}, \mathbf{B}) + 1}{2}.$$

Proof. Let $\mathbf{c}, \mathbf{c}' \in \mathbb{H}(\mathbf{w}, \mathbf{A}, \mathbf{B})$ as witnessed by $\mathbf{z}, \mathbf{z}' \in \mathbb{S}$, respectively. We claim that $\mathbf{c} - \mathbf{c}'$ is in $\mathbb{D}(\mathbf{A}, \mathbf{B})$. It is clear that $\mathbf{c} - \mathbf{c}' \in \Delta\mathbb{H}$ and $\mathbf{z} - \mathbf{z}' \in \Delta\mathbb{S}$. It remains to verify $\mathbf{A}(\mathbf{z} - \mathbf{z}') - \mathbf{B}(\mathbf{c} - \mathbf{c}') \in \Delta\mathbb{L}$. We have

$$\mathbf{A}(\mathbf{z} - \mathbf{z}') - \mathbf{B}(\mathbf{c} - \mathbf{c}') = \mathbf{A}\mathbf{z} - \mathbf{B}\mathbf{c} - (\mathbf{A}\mathbf{z}' - \mathbf{B}\mathbf{c}').$$

Since $\mathbf{A}\mathbf{z} - \mathbf{B}\mathbf{c}$ and $\mathbf{A}\mathbf{z}' - \mathbf{B}\mathbf{c}'$ have the same high bits, it must be that $\mathbf{A}(\mathbf{z} - \mathbf{z}') - \mathbf{B}(\mathbf{c} - \mathbf{c}')$ is the difference of two vectors from $[-(2^{d-1} - 1), 2^{d-1}]^m$, from which it follows that $\mathbf{A}(\mathbf{z} - \mathbf{z}') - \mathbf{B}(\mathbf{c} - \mathbf{c}') \in \Delta\mathbb{L}$. A similar argument proves that the negation $\mathbf{c}' - \mathbf{c} \in \mathbb{D}(\mathbf{A}, \mathbf{B})$.

If $\mathbf{c}_1, \ldots, \mathbf{c}_k$ are distinct elements of $\mathbb{H}(\mathbf{w}, \mathbf{A}, \mathbf{B})$ then $0, \mathbf{c}_1 - \mathbf{c}_2, \ldots, \mathbf{c}_1 - \mathbf{c}_k$ must be distinct elements of $\mathbb{D}(\mathbf{A}, \mathbf{B})$. Similarly, the negations $\mathbf{c}_2 - \mathbf{c}_1, \ldots, \mathbf{c}_k - \mathbf{c}_1$ are also distinct elements of $\mathbb{D}(\mathbf{A}, \mathbf{B})$. To see that $\mathbf{c}_1 - \mathbf{c}_2, \ldots, \mathbf{c}_1 - \mathbf{c}_k$ are all distinct from their negations, observe that

$$\mathbf{c}_1 - \mathbf{c}_i = -(\mathbf{c}_1 - \mathbf{c}_j) \implies 2\mathbf{c}_1 = \mathbf{c}_i + \mathbf{c}_j \implies \mathbf{c}_i = \mathbf{c}_j$$

where the final implication follows from the fact that the entries of $\mathbf{c}_1, \mathbf{c}_i, \mathbf{c}_j$ are all in $\{-1, 0, 1\}$. We have thus listed $2k - 1$ distinct elements of $\mathbb{D}(\mathbf{A}, \mathbf{B})$, from which the lemma follows. □

3.2.3.4 Forgers Cannot Forge on M-LWE No-Instances

Proposition 3.12 provides a bound on the fraction δ_{no} of hash inputs that are good. Moreover, since the goodness of a hash input $(\mathbf{w}, \mathsf{m})$ depends solely on whether $\mathsf{H}(\mathbf{w}, \mathsf{m})$ is in $\mathbb{H}(\mathbf{w}, \mathbf{A}, \mathbf{B})$, the set of all good hash inputs is a randomly selected set. Thus, the only way to find a good hash input is via search through an unstructured space.

It then follows from lower bounds for quantum search [54] that any algorithm making no more than q_h quantum queries to H finds a good hash input—and thus a valid TESLA forgery—with probability no larger than

$$2(q_h + 1)\sqrt{\delta_{\text{no}}}.$$

We therefore obtain

$$\Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ no-instance of M-LWE}\right] \leq 2(q_h + 1)\sqrt{\delta_{\text{no}}}.$$

3.2.4 Conclusion of the Security Reduction

Assuming that no algorithm with run-time comparable to that of $\mathcal{S}$ can solve M-LWE with success bias exceeding ε, we have:

$$\begin{aligned}\varepsilon \geq \Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ yes-instance of M-LWE}\right] \\ - \Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ no-instance of M-LWE}\right].\end{aligned}$$

We know that

$$\Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ yes-instance of M-LWE}\right] \geq \Pr\left[\text{forge}\,(\mathbf{A}, \mathbf{B})\right] - \delta_{\text{yes}}.$$

Moreover, against a quantum forger, we showed that

$$\Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ no-instance of M-LWE}\right] \leq 2(q_h + 1)\sqrt{\delta_{\text{no}}},$$

implying that

$$\begin{aligned}\Pr\left[\text{forge}(\mathbf{A}, \mathbf{B})\right] &\leq \Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ yes-instance of M-LWE}\right] + \delta_{\text{yes}} \\ &\leq \varepsilon + \Pr\left[\mathcal{S} \text{ output "yes"} \mid (\mathbf{A}, \mathbf{B}) \text{ no-instance of M-LWE}\right] + \delta_{\text{yes}} \\ &\leq \varepsilon + 2(q_h + 1)\sqrt{\delta_{\text{no}}} + \delta_{\text{yes}}.\end{aligned}$$

Against a *classical* forger, we can remove the quadratic speedup on the lower bound query complexity, and the probability becomes

$$\Pr\left[\text{forge}(\mathbf{A}, \mathbf{B})\right] \leq \epsilon + q_h \cdot \delta_{\text{no}} + \delta_{\text{yes}}.$$

We make some simplifying assumptions on the choice of TESLA parameters. These assumptions are not necessary in order to derive a negligibly small upper bound on the forger's success probability; they merely facilitate a simplified statement of the upper bound.

Definition 3.14 (Convenient TESLA Parameters)**.** *TESLA parameters are convenient if the following bounds hold:*

$$\phi + \sqrt{\frac{2^{\lambda}(q+1)}{\#\mathbb{Y}}} \leq 1/2 \tag{3.39}$$

$$\#\Delta\mathbb{H}\#\Delta\mathbb{S}\#\Delta\mathbb{L} \leq q^m, \tag{3.40}$$

with ϕ denoting the probability that a random vector in $\mathbb{Z}_q^m$ is not well-rounded (see Section 3.2.2.8).

We now incorporate our bounds on δ_{yes} and δ_{no} in order to derive an explicit upper bound on the forger's success probability.

By applying Lemma 3.8 with $K = 2^\lambda$, as well as Equation (3.39) in Definition 3.14, we observe that with probability $1-2^{-\lambda}$ over the choice of $(\mathbf{A}, \mathbf{E})$, $\text{nwr}(\mathbf{A}, \mathbf{E}) \leq 1/2$. Incorporating this into Equation (3.24), we see that

$$\delta_{\text{yes}} \leq q_s\gamma + 4q_s \,\text{coll}(\mathbf{A}, \mathbf{E}) \left(1 + \frac{(q_h + q_s)^2}{2\gamma^2}\right)$$

for some probability γ. From Lemma 3.9, we have a bound on $\text{coll}(\mathbf{A}, \mathbf{E})$ that holds with probability $1 - 1/K_{\text{coll}}$, implying

$$\delta_{\text{yes}} \leq q_s\gamma + 4q_s \left(\frac{2^d}{q}\right)^m K_{\text{coll}} \left(1 + \frac{(q_h + q_s)^2}{2\gamma^2}\right).$$

At this point, we note that our result on this bound holds for whatever γ we may choose. As we want the first term to be exponentially small, we will select $\gamma = \frac{1}{2^\lambda q_s}$. Then, using the simplification that $1 + \frac{(q_h+q_s)^2}{2\gamma^2} \approx \frac{(q_h+q_s)^2}{2\gamma^2}$ we get

$$\delta_{\text{yes}} \leq \frac{1}{2^\lambda} + \frac{2^{md+2\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 K_{\text{coll}}.$$

From Proposition 3.12 we have

$$\delta_{\text{no}} \leq \frac{1}{2\#\mathbb{H}} \left(1 + \frac{\#\Delta\mathbb{H}\#\Delta\mathbb{S}\#\Delta\mathbb{L}}{q^m}\right).$$

Using Equation (3.40) of Definition 3.14 we can simplify this bound on δ_{no} to

$$\delta_{\text{no}} \leq \frac{1}{\#\mathbb{H}}.$$

Finally, we substitute this and our bound for δ_{yes} into Equation (3.39). We also note that $\#\mathbb{H} = 2^h\binom{n'}{h}$. Additionally, we also must consider the probability with which our bounds do not hold. Doing this, we get that $\Pr[\text{forge}(\mathbf{A}, \mathbf{B})]$ is at most

$$\varepsilon + \frac{1}{2^\lambda} + \frac{2^{md+2\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 K_{\text{coll}} + 2(q_h + 1)\sqrt{\frac{1}{2^h\binom{n'}{h}}} + \frac{1}{K_{\text{nwr}}} + \frac{1}{K_{\text{coll}}}.$$

Then by choosing each K-value to be 2^λ, we get that this is equal to

$$\varepsilon + \frac{3}{2^\lambda} + \frac{2^{md+3\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 + 2(q_h + 1)\sqrt{\frac{1}{2^h\binom{n'}{h}}}. \tag{3.41}$$

Classically, we can similarly derive that the adversary's success is bounded by

$$\varepsilon + \frac{3}{2^\lambda} + \frac{2^{md+3\lambda+1}}{q^m}(q_h + q_s)^2 q_s^3 + q_h\frac{1}{2^h\binom{n'}{h}}. \tag{3.42}$$

From Simplified TESLA to TESLA. We now move on to explicate the difference between the simplified (Algorithm 3.15) and the original signature generation in TESLA (Algorithm 3.4). To do so, we first define the *PRF security* and *PRF advantage*.

Definition 3.15 (PRF Security)**.** *Let* $F : \mathsf{Keys} \times \mathsf{In} \to \mathsf{Out}$ *be a PRF. Define* $\mathsf{Func}[\mathsf{In}, \mathsf{Out}]$ *to be the set of all functions* $f : \mathsf{In} \to \mathsf{Out}$*. Furthermore, let* $\mathcal{A}$ *be a quantum adversary interacting with* F *in the Game* $\mathrm{Expt}_F^{\mathsf{PRF\text{-}SEC}}$ *given in Figure 3.4.*

We say that F *is a* $(t_{\mathsf{PRF}}, \varepsilon_{\mathsf{PRF}})$*-secure PRF (*PRF-SEC*) if for all quantum adversaries* $\mathcal{A}$ *running in time* t_{PRF}*, the advantage*

$$\mathsf{Adv}_F^{\mathsf{PRF\text{-}SEC}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}_F^{\mathsf{PRF\text{-}SEC}}(\mathcal{A}) = 1\right] \leq \varepsilon_{\mathsf{PRF}}.$$

$\underline{\mathrm{Expt}_F^{\mathsf{PRF\text{-}SEC}}(\mathcal{A})\text{:}}$
1: $k \leftarrow_{\$} \mathsf{Keys}_F$
2: $b \leftarrow_{\$} \{0,1\}$
3: $f \leftarrow_{\$} \mathsf{Func}[\mathsf{In}, \mathsf{Out}]$
4: $b' \leftarrow \mathcal{A}^{\mathcal{O}_F}()$
5: return $[b = b']$

$\underline{\text{Classical } \mathcal{O}_F(x)\text{:}}$
1: if $b = 1$:
2: return $f(x)$
3: else
4: return $F(k, x)$

$\underline{\text{Quantum } \mathcal{O}_F(\sum_{x,t,z} \psi_{x,t,z} |x,t,z\rangle)\text{:}}$
1: return state $\sum_{x,t,z} \psi_{x,t,z} |x, t \oplus \mathcal{O}_F(x), z\rangle$

Figure 3.4: PRF security experiment against an adversary $\mathcal{A}$

Define the signature scheme $\mathcal{S} = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$. KeyGen' and Verify' are given in Algorithm 3.14, and 3.16. They are the same as the original TESLA algorithms given in Algorithm 3.3 and 3.5, respectively. Sign' is given in Algorithm 3.15 and differs from the original TESLA signature generation (Algorithm 3.4) as it chooses $\mathbf{y} \leftarrow_{\$} [-B, B]^n$ instead of computing it by PRF_1 and PRF_2. Let $\varepsilon_{\mathsf{EUF\text{-}CMA}}$ be the success probability to break the EUF-CMA security of $\mathcal{S}$.

The success probability ε'' against TESLA is upper bounded by the sum of the success probability where the adversary distinguishes $\mathsf{PRF}_2 \circ \mathsf{PRF}_1$ from a truly random function and the success probability where the adversary does not distinguish the two functions but breaks the unforgeability of $\mathcal{S}$. In addition, the success probability against TESLA is depending on the acceptance probability of checkE and checkS, since these functions reduce the key space. Hence it finally holds that

$$\varepsilon'' \leq (\varepsilon_{\mathsf{PRF}} + \varepsilon_{\mathsf{EUF\text{-}CMA}}(1 - \varepsilon_{\mathsf{PRF}}))\, \delta_{\mathsf{KeyGen}},$$

where δ_{KeyGen} is the probability that checkE and checkS accept a secret key $(\mathbf{E}, \mathbf{S})$.

Moving on from the security reduction of TESLA, we turn to the security reduction of qTESLA next.

3.2.5 Security Reduction for qTESLA

In the following paragraphs we prove the security of qTESLA. Similar to the case of TESLA, our reduction gives a tight and explicit reduction in the QROM from the hardness R-LWE to the EUF-CMA security of qTESLA. In contrast to TESLA, however, Theorem 3.16 holds assuming a *conjecture*, as explained below.

Theorem 3.16 (Security of qTESLA)**.** *Let the parameters be as in Table 3.1. Furthermore, assume that Conjecture 3.17 holds. If there exists an adversary $\mathcal{A}$ that forges a signature of the signature scheme qTESLA described in Section 3.1.2 in time t_Σ and with success probability ϵ_Σ, then there exists a reduction R that solves the R-LWE$_{n,k,q,\chi}$ problem in time $t_{LWE} \approx t_\Sigma$ with*

$$\epsilon_\Sigma \leq \frac{2^{3\lambda+nkd+1}\ q_s^3(q_s+q_h)^2}{q^{nk}} + \frac{2q_h+5}{2^\lambda} + \epsilon_{LWE}$$

with parameters as in Table 3.1.

Except for a few differences, the proof of Theorem 3.16 follows the idea of TESLA's security reduction. Hence, we do not repeat the entire proof here but only explain the differences, namely the computation of the two probabilities $\mathrm{coll}(a,e)$ and $\mathrm{nwr}(a,e)$.

For simplicity we assume that the randomness is sampled uniformly random in $\mathcal{R}_{q,[B]}$ as before in case of TESLA. For our discussion we define/recall the following sets:

$\mathbb{Y}$:	The set of polynomials $y \in \mathcal{R}_{q,[B]}$.
$\mathbb{S}$:	The set of polynomials $z \in \mathcal{R}_{q,[B-L_S]}$.
$\mathbb{H}$:	The set of polynomials $c \in \mathcal{R}_{q,[1]}$ with exactly h non-zero coefficients.
$\mathbb{W}$:	The set $\{[w]_M : w \in \mathcal{R}_q\}$.
$\Delta\mathbb{L}$:	The set $\{x - x' : x, x' \in \mathcal{R}$ and $[x]_M = [x']_M\}$.

Furthermore, we call a polynomial w well-rounded if w is in $\mathcal{R}_{q,[\lfloor q/2 \rfloor - L_E]}$ and $[w]_M \in \mathcal{R}_{q,[(2^{d-1}-L_E)]}$. We define the following quantities for keys $(a_1, ..., a_k, t_1, ..., t_k)$, $(s, e_1, ..., e_k)$, where we denote $\overrightarrow{a} = (a_1, ..., a_k)$ and $\overrightarrow{e} = (e_1, ..., e_k)$:

$$\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e}) = \Pr_{(y,c)\in\mathbb{Y}\times\mathbb{H}}[a_i y - e_i c \text{ not well-rounded for at least one } i \in \{1, ..., k\}]$$

$$\mathrm{coll}(\overrightarrow{a}, \overrightarrow{e}) = \max_{(w_1,...,w_k)\in\mathbb{W}^k} \left\{ \Pr_{(y,c)\in\mathbb{Y}\times\mathbb{H}}[[a_1 y - e_1 c]_M = w_1, ..., [a_k y - e_k c]_M = w_k] \right\}.$$

Informally speaking $\mathrm{nwr}(\overrightarrow{a}, \overrightarrow{e})$ refers to the probability over random (y, c) that $a_i y - e_i c$ is not well-rounded for some i. This quantity varies as a function of

$a_1, ..., a_k, e_1, ..., e_k$. In contrast to the case of TESLA, we cannot upper bound this in general in the ring setting. Hence, we first assume that $\text{nwr}(\overrightarrow{a}, \overrightarrow{e}) < 3/4$ and afterwards check experimentally that this holds true. Indeed, our acceptance probability δ_{Sign} is at least 1/4 for our parameter sets qTESLA-p-I and qTESLA-p-III as we determine experimentally (see Table 3.2 for the concrete probabilities). Hence, the bound $\text{nwr}(\overrightarrow{a}, \overrightarrow{e}) < 3/4$ holds.

Secondly, we need to bound the probability $\text{coll}(\overrightarrow{a}, \overrightarrow{e})$. In Lemma 3.8 of Section 3.2.2.8, the corresponding probability $\text{coll}(\mathbf{A}, \mathbf{E})$ for standard lattices is upper bounded. Unfortunately, we were not able to transfer the proof to the ring setting for the following reason. In the proof of Lemma 3.8, it is used that if the randomness $\mathbf{y}$ is not equal to $\mathbf{0}$, the vector $\mathbf{Ay}$ is uniformly random distributed over $\mathbb{Z}_q$ and hence, also $\mathbf{Ay} - \mathbf{Ec}$ is uniformly random distributed over $\mathbb{Z}_q^m$. This does not necessarily holds if the *polynomial* y is chosen uniformly in $\mathcal{R}_{q,[B]}$. Moreover, in Equation (3.29), Section 3.2.2.9, ψ denotes the probability that a random vector $\mathbf{x} \in \mathbb{Z}_q^m$ is in $\Delta\mathbb{L}$:

$$\psi = \Pr_{\mathbf{x} \in \mathbb{Z}_q^m} [\mathbf{x} \in \Delta\mathbb{L}] \leq \left(\frac{2^d}{q}\right)^m . \tag{3.43}$$

The quantity ψ is a function of the TESLA parameters q, m, d, and it is negligibly small for the chosen parameters. However, we cannot prove a similar statement for the signature scheme qTESLA over ideals. Instead, we need to *conjecture* the following.

Conjecture 3.17. *Let I be a non-zero ideal in $\mathcal{R}_q$ and let $r \in \mathcal{R}_q$ be a fixed choice of ring elements. Then, it holds that the probability that $x + r \in \Delta\mathbb{L}$ for a uniformly distributed element $x \leftarrow_{\$} I$ is negligibly small.*

The intuition behind our conjecture is as follows. Let ψ_I denote the probability that a random element from the ideal I lands in $\Delta\mathbb{L}$. We know that ψ_I is small when the ideal $I = \mathcal{R}_q$, i.e., a negligibly small fraction of elements from $\mathcal{R}_q$ are in $\Delta\mathbb{L}$. Furthermore, the set $\Delta\mathbb{L}$ appears to have no relationship with the ideal structure of the ring, so it seems reasonable to view each ideal as a *random* subset of $\mathcal{R}_q$ in the following sense: No larger or smaller portion of elements in the ideal I is in $\Delta\mathbb{L}$ than that portion of elements of $\mathcal{R}_q$ that is in $\Delta\mathbb{L}$.

Hence, the corresponding statement described above and needed in Lemma 3.8 translates for qTESLA to the following. If $y \neq 0$ then $a_i y$ is a uniformly random element of some non-zero ideal I for all i. The polynomial c is fixed and the polynomials $e_1, ..., e_k$ are independent of the polynomials $a_1, ..., a_k$, and y. Hence, by our conjecture (with $x = a_i y$ and $r = e_i c$) it holds that the probability of Equation (3.30) in Section 3.2.2.9 is negligibly small. Thus, assuming that our conjecture holds true, Lemma 3.8 and hence, the security reduction for TESLA holds for qTESLA as well.

3.3 Bit Security and Parameter Selection

In this section, we determine concrete instantiations of TESLA and qTESLA. As shown in Section 3.2, the security of our signature schemes relies on the hardness of the M-/R-LWE problem. Therefore, we start this section by explaining how to estimate the bit hardness of LWE instances. Afterwards, we move on to describe how to derive secure instantiations of the signature schemes, when given concrete LWE instances of a certain hardness. Finally, we present instances of TESLA and qTESLA.

3.3.1 Hardness Estimation of LWE

We start by explaining why we estimate R-LWE using the same attacks as for LWE. Afterwards, we explain current state-of-the-art classical and quantum attacks against LWE and finally, we describe the software tool *LWE-Estimator* [11] which we use for our estimations.

Estimation of the R-LWE Hardness. Recent results exploit the algebraic structure of some ideal lattices [42, 57, 67, 68, 88, 90, 102]. However, so far they do not seem to weaken the hardness of R-LWE instances that are used for qTESLA. As a consequence, we estimate the hardness of R-LWE using state-of-the-art LWE solvers described next.

Classical Algorithms. Currently, the following basic attacks on (decisional) LWE are known: the embedding approach, the decoding attack, the distinguishing attack, the algorithm by Blum, Kalai, and Wassermann (BKW) [44], and the Arora-Ge-Algorithm [21] described briefly in the following paragraphs. We refer to [8, 11] for a deeper contemplation of the state-of-the-art hardness estimations.

The *embedding approach* is to solve an LWE instance by reducing it to an instance of uSVP. In particular, during the attack a lattice is defined such that the error term of an LWE instance is embedded in that lattice (see for example [9, 24, 39] for a description of such lattices). In the end, the short error term is found as a short vector of the constructed lattice via basis reduction such as BKZ [20, 64, 100]. During the BKZ algorithm, an SVP-oracle is queried that returns solutions of SVP instances of dimension β. The dimension β is also called the block size. To answer the queries to the SVP-oracle, algorithms such sieving [147] or enumeration [64] are used.

During the *decoding attack*, an LWE instance $(\mathbf{A}, \mathbf{As} + \mathbf{e})$ is seen as an instance of the Bounded Distance Decoding (BDD) problem [151]. The idea of the attack is to first use lattice reduction algorithms such as BKZ, and to find the closest

lattice vector to a target vector via the nearest plane algorithm by Babai [23] (or improved variants such as [151, 152]) afterwards.

The *distinguishing attack*'s goal is to solve an instance $(\mathbf{A}, \mathbf{b})$ of decisional LWE. In order to do, basis reduction such as BKZ is applied to find a short vector $\mathbf{v}$ in $\Lambda_q^{\perp}(\mathbf{A})$. Depending on whether $\langle \mathbf{b}, \mathbf{v} \rangle$ is small or large, the decisional LWE problem can be solved.

The *BKW algorithm* and the *algorithm by Arora and Ge* require a large number of LWE samples to be applied efficiently. Although the number of required samples was crucially reduced, for both BKW [79, 116, 140] and the Arora-Ge algorithm [6], our proposed instances give far less LWE samples than required for the two attacks. Hence, we do not consider them further.

After describing the classical algorithms to solve LWE, we explain recent quantum LWE solvers next.

Quantum Algorithms. State-of-the-art quantum attacks on LWE make (black-box) use of Grover's quantum search algorithm [112] to speed up classical LWE solvers. Laarhoven et al. [148] investigate and compare the impact of Grover's quantum search algorithm on different SVP solvers and propose a new quantum SVP solver with a run-time (in the dimension β) of $2^{0.268\beta+o(1)}$. Furthermore, Aono and Nguyen [19] recently published a quantum enumeration algorithm to be used as an SVP solver.

Another recent quantum attack, called quantum hybrid attack, was presented by Göpfert, van Vredendaal, and Wunderer [109] in 2017. This hybrid attack is most efficient on LWE with very small secret and error, e.g., binary or ternary coefficients. Since the coefficients of the secret and error of TESLA or qTESLA are chosen Gaussian distributed, the attack is not efficient for our instances.

The LWE-Estimator. Albrecht, Player, and Scott presented the LWE-Estimator which is a software to estimate the hardness of LWE given the matrix dimension n, the modulus q, the relative error rate $\alpha = \frac{\sqrt{2\pi}\sigma}{q}$, and the number m of LWE samples. The LWE-Estimator then estimates the bit hardness regarding the fastest LWE solvers currently known, i.e., it outputs a lower bound on the number of operations an attack needs to break a given LWE instance. In particular, the following attacks are considered in the LWE-Estimator: the meet-in-the-middle exhaustive search, the BKW algorithm [7, 44, 116, 140], the dual lattice attacks recently published in [5], the enumeration approach by Linder and Peikert [151], the distinguishing attack described in [9, 25], and the Arora-Ge algorithm [21] using Gröbner bases [6]. Furthermore, it uses the latest analysis to compute the block sizes used in the lattice basis reduction BKZ [10]. Moreover, the aforementioned quantum speed-ups for the sieving algorithm used in BKZ [147] are considered as

well. In particular, they estimate the run-time as $2^{0.268\beta+16.40}$.

The LWE-Estimator is the result of many different contributions and contributors. It is open source and hence easily checked and maintained by the community. Hence, we find the LWE-Estimator to be a suitable tool to estimate the hardness of our chosen LWE instances. For our estimations, we use the LWE-Estimator with commit-id `9302d42`. It allows to choose different cost models to estimate the run-time of the BKZ algorithm. Since for our TESLA and qTESLA instances, the above-mentioned embedding approach is the most efficient attack and BKZ is used as a subroutine in this attack, the choice of the cost model impacts the overall run-time of the attack greatly. In fact, Albrecht et al. show that depending on the BKZ cost model, the bit hardness of the same problem instance can differ by several hundred bits [8]. Essentially, the BKZ cost model is defined by the run-time of one call to the SVP oracle and the number of so-called BKZ rounds. For our instances, we choose a conservative approach that is supported by practical experiments. In particular, we assume that the number of needed BKZ rounds is $8d$ with lattice dimension d since experimentally lattice bases are sufficiently reduced after $8d$ BKZ rounds [11]. Furthermore, we choose the BKZ run-time estimation of [147] since the resulting run-time is the smallest on our instances when taking quantum algorithms into account. In particular, let n, m (resp., $n \cdot k$ for qTESLA), α, and q be given. Then we estimate the hardness using the command `estimate_lwe(n,alpha,q,m,reduction_cost_model=BKZ.qsieve)`.

Next we explain how to determine the bit security of our signature schemes given the bit hardness of R-/M-LWE instances.

3.3.2 Correspondence Between Security and Hardness

After we described how to estimate the bit hardness of a given LWE instance, we now go on describing the relation between hardness and security in case of TESLA and qTESLA.

The reductionist approach to prove security of a cryptographic scheme essentially consists in building an efficient reduction that turns any successful adversary against the scheme into one that solves some computationally hard problem. The hardness of breaking the scheme and of solving the underlying problem are often expressed asymptotically. When a scheme is to be deployed in the real world, however, for a security analysis to be realistic it is essential that run-times and success probabilities are estimated in a more explicit way. Given an explicit security reduction, the scheme can be instantiated according to the security reduction. This implies the following: By virtue of our security reduction, these parameters strictly correspond to an instance of the problem. That is, the reduction provably guarantees that our scheme has the selected security level as long as the corresponding problem instance provides a certain hardness level. In other words, hardness statements for

instances of the LWE problem have a provable consequence for the security levels of TESLA and qTESLA.

The security reduction given in Section 3.2 provides a reduction from the hardness of M-LWE (resp., R-LWE) and bounds explicitly the forging probability with the success probability of the reduction. More formally, let $\varepsilon_{\mathcal{F}}$ and $t_{\mathcal{F}}$ denote the success probability and the run-time of a forger $\mathcal{F}$ against our signature scheme and let $\varepsilon_{\mathcal{S}}$ and $t_{\mathcal{S}}$ denote analogous quantities for the reduction $\mathcal{S}$ presented in Section 3.2. We say that LWE is η-bit hard if $t_{\mathcal{S}}/\varepsilon_{\mathcal{S}} \geq 2^{\eta}$; and we say that the signature scheme is λ-bit secure if $t_{\mathcal{F}}/\varepsilon_{\mathcal{F}} \geq 2^{\lambda}$.

We aim at choosing parameters such that $\varepsilon_{\mathcal{S}} \approx \varepsilon_{\mathcal{F}}$ and $t_{\mathcal{S}} \approx t_{\mathcal{F}}$, i.e., we choose parameters according to our tight security reduction. Hence, the bit hardness of the (R-)LWE instance is about the same as the bit security of our signature scheme.

For TESLA, however, we lose $\lceil \log_2(n') \rceil$ bits of security since TESLA is actually based on M-LWE instead of LWE and the reduction from M-LWE to LWE looses this number of bits as explained in Section 2.2.

Additionally, depending on the instantiation the size of the key space is decreased by $\lceil |\log_2(\delta_{\mathsf{KeyGen}})| \rceil$ bit. For example, if the acceptance probability of the key generation is 0.2, i.e., the key space is decreased by 80%, then we lose at most 3 bits, since $2^{-3} \leq 0.2 \leq 2^{-2}$.

In summary, for TESLA we choose an LWE instances of $\lambda + \lceil \log_2(n') \rceil + \lceil |\log_2(\delta_{\mathsf{KeyGen}})| \rceil$ bit hardness to achieve concrete instances of λ bit security. For qTESLA, the corresponding (R-)LWE instances give at least $\lambda + \lceil |\log_2(\delta_{\mathsf{KeyGen}})| \rceil$ bit of hardness against currently known attacks.

Instantiations According to the Reductions from SVP to LWE. As observed by Chatterjee et al. [59], existing worst-case to average-case reductions from SIVP or GapSVP to LWE are highly non-tight. We are not aware of a proposal for a concrete instantiation of a cryptosystem based on LWE with the property that the proposed parameters were selected according to such a reduction. Instead, it is common to instantiate schemes based on the best known algorithms for solving LWE as described above.

For TESLA and qTESLA, we take care to instantiate the scheme according to their security reductions from R-/M-LWE. However, we do not instantiate them according to reductions from underlying lattice problems, since due to the non-tightness of these reductions, corresponding parameters would yield much less efficient schemes. To instantiate a scheme according to the given worst-to-average-case reductions is an interesting research direction.

Table 3.2: Parameters of TESLA and qTESLA; key and signature sizes are reported in bytes (B); $\kappa = 256$ for all parameter sets

Param.	TESLA			qTESLA	
	TESLA-p-I	TESLA-p-II	TESLA-p-III	qTESLA-p-I	qTESLA-p-III
λ	96	96	128	95	160
q_h, q_s	2^{96}, 2^{48}	2^{96}, 2^{48}	2^{128}, 2^{64}	2^{128}, 2^{64}	
n	804	800	1 300	1 024	2 048
n'	600	700	1036	-	-
σ	57	60	73	8.5	8.5
k	-	-	-	4	5
m	4972	4000	4788	-	-
q	$2^{31} - 19$	9 608 718 773 $\approx 2^{33.16}$	40 582 171 961 $\approx 2^{35.24}$	485 978 113 $\approx 2^{29}$	1 129 725 953 $\approx 2^{30}$
h	42	69	88	25	40
L_E	6 703	11 592	17 987	554	901
L_S	33 516	57 960	89 936	554	901
U	5 172	6 978	9 588	-	-
B	$2^{22} - 1$	$2^{23} - 1$	$2^{24} - 1$	$2^{21} - 1$	$2^{23} - 1$
d	26	27	27	22	24
δ_{KeyGen}	1	-	-	0.59	0.44
δ_{Sign}	0.154	-	-	0.26	0.28
sig. size [B]	2 343	2 432	4 095	2 848	6 176
pk size [B]	11 559 900	11 900 000	22 321 656	14 880	39 712
sk size [B]	4 332 000	4 200 000	7 883 920	4 544	10 816
with t	11.6	11.6	13.4	11.6	15
quant. LWE hardness [bit]	-	109	154	123	247
class. LWE hardness [bit]	116	117	167	132	270

3.3.3 Instantiations of TESLA and qTESLA

In this section, we present different instantiations of TESLA and qTESLA. Overall, we propose three parameter sets for TESLA and five sets for qTESLA. In particular, we present the set TESLA-p-I that is chosen to give a *classical* bit security of 96. This set is merely chosen to enable a comparison with the GPV signature scheme [104] in Section 3.5. The other two sets TESLA-p-II and TESLA-p-III are chosen with a targeted *quantum* security level of 96 and 128 bit, respectively. All three TESLA parameter sets are chosen according to the respective security reductions and as explained in Section 3.1.3 and summarized in Table 3.2. To be able to compare not only the key and signature size but also the run-time of TESLA with, e.g., the GPV signature scheme, we report the cycle counts of Alkim's implementation of TESLA-p-I in Section 3.4.2. However, no implementation exists for TESLA-p-II and TESLA-p-III. This also implies that we cannot state

the acceptance probabilities δ_{KeyGen} and δ_{Sign} of the key generation and signature generation algorithms of TESLA-p-II and TESLA-p-III, since these probabilities would have to be determined experimentally. As mentioned before, the parameters of TESLA are chosen to be conservative (from an attacker's point of view).

In contrast, qTESLA's instantiations have been optimized for efficiency. We propose the qTESLA parameter set qTESLA-p-I and qTESLA-p-III[7] of *quantum* security levels 96 and 160. Both sets are chosen according to the respective security reductions and as explained in Section 3.1.3. The qTESLA set qTESLA-h-I, qTESLA-h-III-speed, and qTESLA-h-III-size depicted in Table 3.3, however, are chosen heuristically as explained next.

Table 3.3: Heuristic parameters of qTESLA; key and signature sizes are reported in B; $q_h = 2^{128}$, $q_s = 2^{64}$, $\kappa = 256$ for all parameter sets

Param.	qTESLA-h-I	qTESLA-h-III-speed	qTESLA-h-III-size
λ	95	160	160
n	512	1 024	1 024
k	1	1	1
σ	23.78	10.2	8.49
q	$4\,205\,569 \approx 2^{22}$	$8\,404\,993 \approx 2^{23}$	$4\,206\,593 \approx 2^{22}$
h	30	48	48
L_E	1 586	1 147	910
L_S	1 586	1 233	910
B	$2^{20} - 1$	$2^{21} - 1$	$2^{20} - 1$
d	21	22	21
δ_{Sign}	0.14	0.21	0.09
δ_{KeyGen}	0.45	0.60	0.39
sig size[B]	1 376	2 848	2 720
pk size [B]	1 504	3 104	2 976
sk size (with t) [B]	1 216 ($t = 11.6$)	2 112 ($t = 15$)	1 856 ($t = 15$)
quant. LWE hardness [bit]	97	164	169
class. LWE hardness [bit]	104	178	188

Heuristic Parameters. Choosing parameters according to the security statements, as described above, implies to follow specific security requirements and to take a reduction loss into account. This affects the performance and sizes of the scheme. As a consequence, instantiations from the literature are sometimes chosen under the assumption that the bit security of the scheme is the same as the bit hardness of the underlying computational problem, e.g., [24, 69, 81, 82].

[7]The names of qTESLA's parameter sets indicate the targeted security categories of NIST [164].

In order to indicate how the performance of qTESLA performs to other efficient ideal-lattice-based schemes that do not take existing security reductions into account, we propose three additional parameter sets, namely qTESLA-h-I, qTESLA-h-III-speed, and qTESLA-h-III-size, which are chosen heuristically. In this case, we assume that the security level of an instantiation of the scheme by a certain parameter set directly corresponds to the hardness level of the instance of the underlying lattice problem that corresponds to these parameters, without taking into account the security reduction. In this case we *assume* that Theorem 3.2.5 still holds for these concrete parameter sets. We present our heuristic parameter sets for qTESLA in Table 3.3. The parameter sets qTESLA-h-III-speed and qTESLA-h-III-size target the same security level but they are optimized for different purposes: qTESLA-h-III-speed is optimized to give very fast run-times whereas qTESLA-h-III-size is optimized for small key and signature sizes. We exemplify how to optimize parameters for the two different purposes next. The parameters L_S and L_E are the most important parameters to tweak the size of the signature and keys, or the run-time of the algorithms: The larger L_S and L_E are, the larger the acceptance probability during key and signature generation and hence, the faster the signing of a message will be. However, it also holds that the larger the parameter L_S is, the larger the signature size will be. This can also be seen in the parameter sets qTESLA-h-III-speed and qTESLA-h-III-size. Namely, L_S of qTESLA-h-III-size is smaller than L_S of qTESLA-h-III-speed. Hence, the acceptance probability δ_{Sign} and the signature size of qTESLA-h-III-size are smaller than those of qTESLA-h-III-speed. Moreover, a smaller L_S yields a smaller B. Similarly, a smaller L_E gives a smaller value for d. Both, B and d impact the size of q and hence, the size of the public key. Again this can be seen in the case of qTESLA-h-III-speed and qTESLA-h-III-size: As mentioned above, L_S and L_E of qTESLA-h-III-size are smaller than those of qTESLA-h-III-speed. Consequently, B, d, and the public key of qTESLA-h-III-size are smaller than those of qTESLA-h-III-speed. Lastly, for the same values of σ and n, a smaller value for q gives a larger bit hardness of the corresponding LWE instance in general. This means, that for the same value of n and λ, and the smaller q, the value of σ for qTESLA-h-III-size can be chosen smaller than for qTESLA-h-III-speed to give the same level of bit hardness. This in turn, decreases the size of the secret key as can be seen in Table 3.3.

3.4 Implementation and Performance

After describing the schemes and presenting parameter sets, we turn to the implementation security of TESLA and qTESLA and report on different existing implementations.

3.4.1 Implementation Security

Besides the theoretical security against computational attacks, such as lattice reduction, it is important for a cryptographic scheme to be secure against implementation attacks. These attacks come in two flavors: side-channel and fault analysis attacks.

In this section we discuss the resistance of TESLA and qTESLA against certain implementation attacks. Due to its fast run-times and smaller key sizes, qTESLA is much more suitable for a variety of applications. As a consequence, qTESLA might be exposed to different physical attacks. Hence, the implementation of qTESLA is carefully protected against prominent implementation attacks as explained in the following paragraphs.

In both, TESLA and qTESLA, the Gaussian sampler, arguably the most complex part of the scheme, is restricted to key generation. This reduces drastically the attack surface to carry out recent timing, cache, and power attacks, such as [91, 110], against our implementations of TESLA and qTESLA. Additionally, qTESLA's Gaussian sampler is relatively simple and it is implemented in a *constant-time* manner. In practice, this means that the implementation avoids the use of secret address accesses and conditional branches based on secret information. Other functions of qTESLA, such as polynomial arithmetic operations, are also implemented in constant-time. Additionally, the qTESLA implementation of the rejection sampling in the signature generation (line 11 of Algorithm 3.9) protects the sign of each coefficient of z by doing a masked conversion to obtain its absolute value before checking against a unique positive bound.

It is interesting to note that qTESLA has implicitly an additional line of defense, thanks to its non-deterministic nature. To generate the polynomial y, a PRF is applied to fresh randomness r, a value seed_y, and the message m. The use of seed_y makes qTESLA resilient to a catastrophic failure of the Random Number Generator (RNG) during generation of the fresh randomness (e.g., a similar failure of ECDSA signatures was demonstrated in [56]). The randomness guarantees the use of a fresh y at each signing operation, which makes a timing attack (or even more generally, any side-channel attack) more difficult to carry out. More importantly, this implicitly protects against some powerful and easy-to-carry out fault attacks, as explained next.

Recently, some studies have exposed the vulnerability of lattice-based schemes to fault attacks. We describe a simple yet powerful attack that falls in this category of attacks [111]. Assume that line 3 of Algorithm 3.9 is computed without the random value r, i.e., as $\mathsf{rand} \leftarrow \mathsf{PRF}_2(\mathsf{seed}_y, \mathsf{m})$. Assume that a signature (z, c) is generated for a given message m. Afterwards, a signature is requested again for the same message m, but this time, a fault is injected on the computation of the hash value c yielding the value c_{faulted}. The corresponding faulted signature is then $(z_{\text{faulted}}, c_{\text{faulted}})$. Computing $z - z_{\text{faulted}} = sc - sc_{\text{faulted}} = s(c - c_{\text{faulted}})$, reveals the

secret s since $c - c_{\text{faulted}}$ is known to the attacker. As stated in [179], this attack has broad implications since it is generically applicable to deterministic *Schnorr-like* signatures [203].

It is easy to see that, to prevent this (and other similar) fault attacks, every signing operation should use fresh randomness, as precisely specified in line 3 of Algorithm 3.9. This makes qTESLA implicitly resilient to this line of attacks.

In an earlier description of qTESLA, the scheme was specified as a deterministic signature scheme and, hence, was susceptible to the fault attacks described in [111]. Also the scheme TESLA might be vulnerable to this attack since it is deterministic.

In summary, qTESLA is protected against most commonly exploited side-channel and easy-to-carry-out fault attacks. As a future research direction, the countermeasures against more advanced cache-, power-, and electromagnetic-side-channel and fault attacks, described in Chapter 4 should be implemented.

3.4.2 Experimental Results

To evaluate the performance of our schemes and the respective proposed parameter sets, we present the experimental results of existing implementations of TESLA and qTESLA.

3.4.2.1 Experimental Results of TESLA

We first report on the experimental results of the TESLA software implementation by Alkim that targets the Intel Haswell microarchitecture. The software makes use of fast AVX2 instructions on vectors of 4 double-precision floating-point numbers.

Table 3.4 reports on the benchmarking results for TESLA-p-I. The benchmarks are obtained on an 2.40 GHz Intel Core i5-6300U (Skylake) processor while disabling Intel Turbo Boost and hyper-threading. Benchmarks of TESLA for signing are computed over 100 000 signatures; benchmarks of TESLA for verification are computed over 100 verifications, and the key generation reports the median/average of two runs.

We give a comparison of TESLA with other standard-lattice-based signature schemes in Section 3.5. Moreover, we compare the obtained benchmarks of TESLA with those of qTESLA next.

3.4.2.2 Experimental Results of qTESLA

We now move on to report on the experimental results of the qTESLA implementation by Akleylek, Alkim, Barreto, Longa, Ricardini, and Zanon that is a simple yet efficient reference implementation written exclusively in C. It is protected against several implementation attacks such as timing and cache attacks as

Instantiation	Key Generation	Sign	Verify
TESLA-p-I	165 168 022	79 486	7 308
	(165 168 022)	(87 970)	(7 553)

Table 3.4: Performance (in thousands of cycles) of the software implementation of TESLA on a 2.40 GHz Intel Core i5-6300U (Skylake) processor; results are stated as the median and average (in parenthesis)

previously described in Section 3.4.1. This implementation is publicly available on `https://github.com/qtesla/qTesla`. We present benchmarks that have been obtained on different platforms or/and from different libraries in the following paragraphs. Namely, we first state cycle counts that are obtained from the original qTESLA reference implementation on two different CPUs. Afterwards, we report on the benchmarks obtained from the qTESLA implementation integrated in the library *liboqs* on an Intel Core i7-4500U (Haswell) and from the integration in the library *pqm4* on a microcontroller. Lastly, we present benchmarks obtained from qTESLA on the *VexRiscV* processor on an FPGA.

Performance of the Original Reference Implementation. The following benchmarks are obtained on two different desktop computers: a 3.40 GHz Intel Core i7-6700 (Skylake) processor and a 3.40 GHz Intel Core i7-4770 (Haswell) processor. See Table 3.5 and 3.6 for the results of the software implementation. As usual Intel Turbo Boost and hyper-threading were disabled. The results in the two tables correspond to a relatively simple implementation of qTESLA. Nevertheless, they demonstrate that the scheme is practical for most applications. In particular, the signature generation of the reference implementation of qTESLA-p-I is about 70 times faster than TESLA-p-I which makes use of the fast AVX instructions. Verification of qTESLA-p-I is about 13 times faster than the corresponding TESLA-p-I algorithm. From the reported results it can be seen that the heuristic parameter sets of qTESLA are even more efficient: Signing with qTESLA-h-I is about 2.7 times faster than signing with qTESLA-p-I. Likewise, signature generation with qTESLA-p-III is about 7.5 times slower than qTESLA-h-III-speed (and about 4 times slower than qTESLA-h-III-size). We give a comparison of qTESLA with other ideal-lattice-based signature schemes in Section 3.5.

Performance of the qTESLA Implementation in liboqs. Furthermore, qTESLA has been integrated in the library liboqs [16] of the *Open Quantum Safe* (OQS) project [161]. The open source library liboqs provides a testing and benchmarking framework for C implementations of post-quantum secure KEMs and signature

Instantiation	Security [bit]	Key Generation	Sign	Verify
qTESLA-p-I	95	6 678 (6 880)	1 259 (1 590)	505 (505)
qTESLA-p-III	160	30 597 (32 573)	5 057 (6 242)	2 556 (2 556)
qTESLA-h-I	95	1,583 (1,727)	467 (626)	99 (99)
qTESLA-h-III-speed	160	3 576 (3 873)	663 (866)	202 (202)
qTESLA-h-III-size	160	6 057 (6 284)	1 236 (1 714)	204 (205)

Table 3.5: Performance (in thousands of cycles) of the reference implementation of qTESLA on a 3.40 GHz Intel Core i7-6700 (Skylake) processor; results are stated as the median and average (in parenthesis)

Instantiation	Security [bit]	Key Generation	Sign	Verify
qTESLA-p-I	95	6 857 (7 194)	1 348 (1 705)	544 (545)
qTESLA-p-III	160	31 904 (34 488)	5 488 (7 127)	2 715 (2 784)
qTESLA-h-I	95	1,657 (1,851)	513 (714)	106 (107)
qTESLA-h-III-speed	160	3 707 (4 120)	804 (1 110)	238 (241)
qTESLA-h-III-size	160	6 162 (6 554)	1 402 (1 993)	221 (227)

Table 3.6: Performance (in thousands of cycles) of the reference implementation of qTESLA on a 3.40 GHz Intel Core i7-4770 (Haswell) processor; results are stated as the median and average (in parenthesis)

schemes. In Table 3.7 we state the benchmarks that are obtained from the qTESLA liboqs implementation on an Intel Core i7-4500U (Haswell) at 1.8GHz processor while disabling Intel Turbo Boost and hyper-threading. Benchmarks of liboqs and hence, the cycle counts in Table 3.7, are derived as follows. First as many iterations of one algorithm as possible are computed for three seconds, e.g., the key

generation of qTESLA-p-III is run 157 times, the signature generation is run 607 times, and the verification is run 1616 times. Then the mean of the measured cycle counts is returned (along with other information that we do not report here). A comparison of Table 3.7 and Table 3.5 reveals that the key generation of the liboqs implementation of qTESLA is 1.29 to 1.33 times slower, the signing algorithm is 1.30 to 1.66 times slower, and verification is 1.31 to 1.60 times slower that the original qTESLA implementation.

Instantiation	Security [bit]	Key Generation	Sign	Verify
qTESLA-p-I	95	9 318	2 728	848
qTESLA-p-III	160	45 886	11 834	4 445
qTESLA-h-I	95	2 435	1 056	157
qTESLA-h-III-speed	160	5 322	1 442	316
qTESLA-h-III-size	160	8 700	3 039	317

Table 3.7: Performance (in thousands of cycles) of the liboqs implementations of qTESLA on a Intel Core i7-4500U (Haswell) at 1.8GHz processor; results are stated as the median

Performance of the qTESLA Implementation in pqm4. Additionally, Alkim ported the reference implementation of qTESLA-h-I, qTESLA-h-III-speed, and qTESLA-h-III-size to the post-quantum cryptographic library targeting the ARM Cortex-M4 family of microcontrollers, called pqm4 [132]. The benchmarks in Table 3.8 are generated with the benchmarking and testing framework provided by pqm4. The benchmarks are obtained from 100 executions. The results show that qTESLA running on a microcontroller is about 12 to 16 times slower than running on an Intel processor. Nevertheless, these results show that qTESLA is suitable for running on microcontrollers.

Performance of the qTESLA Implementation on FPGA-Based VexRiscV. To further show the practicability of qTESLA, we report on the cycle counts of a qTESLA implementation running on the CPU VexRiscV [50]. VexRiscV is an open source FPGA implementation of the RISC-V CPU architecture. This architecture was designed to use the Reduced Instruction Set Computing (RISC) principles. To run qTESLA on the VexRiscV platform, Deng, Szefer, Tian, and Wang compiled the reference implementation of the signature generation and verification for qTESLA-h-I using the VexRiscV compiler to receive a list of instructions that is executable on the VexRiscV CPU. The execution of qTESLA on an FPGA-based CPU is one of the first steps towards a dedicated qTESLA module in modern CPUs (or

Instantiation	Key Generation		Sign		Verify	
qTESLA-h-I	AVG:	16 894	AVG:	8 233	AVG:	1 281
(of bit security 95)	MIN:	8 109	MIN:	1 720	MIN:	1 277
	MAX:	48 319	MAX:	49 637	MAX:	1 306
qTESLA-h-III-size	AVG:	56 075	AVG:	24 229	AVG:	2 531
(of bit security 160)	MIN:	22 907	MIN:	3 709	MIN:	2 515
	MAX:	153 716	MAX:	122 874	MAX:	2 566
qTESLA-h-III-speed	AVG:	36 689	AVG:	13 203	AVG:	2 582
(of bit security 160)	MIN:	20 936	MIN:	3 682	MIN:	2 576
	MAX:	109 099	MAX:	43 998	MAX:	2 607

Table 3.8: Performance (in thousands of cycles) of the reference implementation of qTESLA compiled and run on an ARM Cortex-M4; results for the minimum, maximum, and average are stated

smaller but more frequently used modules such as a module for NTT computations). Table 3.9 reports the benchmarking results for qTESLA-h-I. Running qTESLA-h-I on VexRiscV used about 46% of the available memory, while the stack size was set to 28 KB and the total memory region size is set as 320KB. This corresponds to the size of the available on-chip Random-Access Memory (RAM) on a Cyclone V DE1-SoC FPGA that was used for the experiments. Comparing the results depicted in Table 3.9 and 3.5 shows that the implementation of qTESLA-h-I is 32 times slower than the corresponding implementation benchmarked on an Intel Core i7-6700 processor. Verification is about 106 times slower than on the Intel processor. The implementation of qTESLA-h-I is the only qTESLA parameter set that has been executed on VexRiscV so far. The reason why the other sets have not yet been compiled for VexRiscV is that the available on-chip RAM size is too small to be able to fit the qTESLA design with larger parameters. Hence, some variables would be overwritten during the execution, disrupting the results. Possible approaches to solve this issue in the future could be to switch to an FPGA with larger on-chip RAM size or to optimize the reference implementation of qTESLA with respect to memory usage to make the implementation of qTESLA more friendly to embedded system use cases. Currently, the C reference implementation optimized for speed.

3.5 Comparison with Other Signature Schemes

In this section, we compare TESLA and qTESLA with other signature schemes from the literature. Table 3.10, hence, provides an overview of selected lattice-based and

Instantiation	Sign	Verify
qTESLA-h-I	15 343	10 522

Table 3.9: Performance of the reference implementation on VexRiscV; cycle counts are rounded to the nearest 10^3 cycles

other post-quantum signature schemes, along with their properties. Additionally, a similar comparison of TESLA, qTESLA, and the classical signature schemes RSA and ECDSA is presented in Table 3.11. In both the tables, the following properties of signature schemes have been considered: the underlying hardness assumption, the existence of a security reduction in the ROM or QROM, the existence of a *tight* reduction in the (Q)ROM, the estimated bit security (regarding classical or quantum attacks), the key and signature size in bytes, and the cycle counts (in thousands of cycles) of signature generation and verification.

We state in the table the theoretical key and signature size independently of the compression used by the corresponding software. Within our table, we report the cycle counts from the software analysis obtained from the project *SafeCrypto* [200] whenever possible to get a fair and reasonable comparison. In particular, the cycle counts of pqNTRUSign, FALCON, Dilithium, SPHINCS$^+$, and MQDSS are taken from [200]. We report the average for signing and the median for verify. The reason for not reporting the median for the signing performance is that often the median is overly optimistic for signing. This holds in particular for lattice-based signatures that use rejection sampling. However, the cycle counts of GPV, GPV-poly, GLP, and BLISS are stated as in the respective literature since they do not appear in [200]. The benchmarks for RSA and ECDSA are obtained from OpenSSL 1.1.0h on a desktop computer (Intel Core i7-4500U, Haswell at 1.8GHz) while disabling Intel Turbo Boost and hyper-threading. In both the tables, we denote if parameters have been chosen according to the security reduction, such as for TESLA and qTESLA, and if the bit security was estimated against classical *and* quantum attacks. In the following paragraphs we present a comparison of qTESLA and TESLA with the schemes listed in Tables 3.10 and 3.11.

Comparison with Unstructured-Lattice-Based Schemes. In Table 3.10, we list two signature schemes constructed over unstructured lattices: TESLA and the GPV scheme. These two schemes are the only unstructured-lattice-based schemes that haven been proven secure in the QROM and are instantiated according to their security reductions. As can be seen in the table, TESLA outperforms GPV in run-time and size, as exemplified in the table through the signature size of GPV, that is about 12 times larger. While TESLA's key sizes are still several magnitudes larger than the corresponding key sizes of ideal-lattice-based schemes,

the signatures are of about the same size. In particular, the signature sizes are even smaller than qTESLA's signatures. One reason for this is the more flexible, and in this case smaller, choice of n. We now discuss the comparison of qTESLA to other (ideal-)lattice-based signatures next.

Comparison with Ideal-Lattice-Based Schemes. GPV-poly is a variant of GPV over ideal lattices. Similarly to the case of TESLA explained above, qTESLA is much more efficient in run-time and size than GPV-poly.

The signature schemes BLISS and GLP are two of the most efficient schemes in Table 3.10. However, the two schemes are not instantiated against state-of-the-art attacks, e.g., they do not consider recent quantum speed-ups such as [148], and they are more vulnerable to implementation attacks such as [91, 110, 177]. This raises an interesting question on how would updated and protected implementations compare with recent schemes such as FALCON, Dilithium, or pqNTRUSign. The three latter schemes are arguably the most interesting schemes for a comparison with qTESLA, since they are, along with qTESLA, submitted to the post-quantum cryptography standardization process initiated by NIST [164][8]. To enable a fair comparison of parameter sets with similar bit securities, we do not report the bit securities stated in the literature. Instead we report the estimations presented in [8] under the same cost model as used for the security estimations of TESLA and qTESLA. In particular, we state the bit security of TESLA, qTESLA, Dilithium, FALCON, and pqNTRUprime assuming a BKZ cost model of $0.265\beta + 16.4 + \log_2(8d)$ with β being the BKZ blocksize and d being the lattice dimension.

Comparing qTESLA-h-III-size or qTESLA-h-III-speed with pqNTRUprime, shows that qTESLA is faster, while public key and signature sizes of pqNTRUPrime are slightly smaller.

In comparison with FALCON, qTESLA-p-III's keys and signature are between 3 to 33 times larger. Moreover, qTESLA's verification is about 4 times slower than FALCON's verify. However, qTESLA's signature generation is about 1.3 times faster than FALCON's algorithms. The most important advantage of qTESLA compared to FALCON, however, is qTESLA's very simple design. Along side with this, qTESLA is more easily protected against timing and cache side channels. Conversely, an analysis of FALCON's vulnerabilities to implementation attacks has yet to be conducted. In particular, FALCON makes heavily use of Gaussian sampling–a building block that has been targeted by power and cache-side-channel attacks before [110, 177].

A comparison with Dilithium shows that, in regards to its design, Dilithium

[8]The lattice-based signature scheme DRS [178] was also submitted to NIST. However, Ducas and Yu already presented a statistical attack against DRS [219]. To our knowledge, no update has been given so far.

is closest to qTESLA since it is also a Fiat-Shamir signature scheme based on a variant of LWE. According to Table 3.10, qTESLA-h-III-speed outperforms Dilithium-recommended. In particular, qTESLA's signature generation is about three times faster while verification is about twice as fast as Dilithium's algorithms. Moreover, the size of qTESLA's secret key is only 2/3 of Dilithium's secret. Lastly, although a simple yet powerful attack can be used to recover Dilithum's secret key [111], in case of qTESLA this is not possible since qTESLA is it is not vunerable to this attack. However, qTESLA's keys are about twice as large as Dilithium's keys.

Comparison with Selected Other Post-Quantum Schemes. For our comparison with other post-quantum signature schemes we picked two well-known schemes which were also submitted to NIST in 2017, namely SPHINCS$^+$–a stateless hash-bashed scheme–and MQDSS–a multivariate scheme. Comparing qTESLA with SPHINCS$^+$ and MQDSS, draws a well-known picture for post-quantum cryptography: While the key sizes of SPHINCS$^+$ and MQDSS are much smaller than those of qTESLA (they are about the same size as ECDSA keys), the run-time to generate qTESLA-p-III's signatures is much smaller (about 140 times faster compared to SPHINCS$^+$). Furthermore, SPHINCS$^+$ and MQDSS signatures are larger than the signatures of respective instantiations of qTESLA.

Hence, to sum up, in the realm of post-quantum signatures, qTESLA seems to be a fine candidate that offers a good trade-off between run-time, signature size, and key sizes. However, the choice of a concrete scheme depends very much on the application scenario.

Comparison with Selected Classical Schemes. Moving on to Table 3.11, we summarize the benchmarks and sizes for different RSA and ECDSA instantiations in comparison to qTESLA and TESLA. Comparing qTESLA-p-III and RSA-3072 shows that the run-times of qTESLA's signing are faster than the run-times of RSA. This becomes clearer when comparing the heuristic parameter set qTESLA-h-III-speed with the run-times of RSA-3072. In this case, the verification algorithm is also as fast as the RSA verification. Nevertheless, qTESLA has larger signature and key sizes, e.g., RSA-2048's signatures are about 5 times smaller than qTESLA-h-I's signatures.

However, overall the larger key and signature sizes of qTESLA when compared to RSA do not seem to lead to irresolvable problems when substituting RSA in the current PKI. To exemplify, Kampanakis et al. [131] investigated the viability of post-quantum signatures submitted to NIST [164] in X.509 certificates and its applications such as the TLS protocol or Internet Key Exchange (IKEv2) protocols. Based on their experimental findings they concluded, that the protocol

transmission overhead introduced by the certificate size of post-quantum signatures were acceptable. In addition, Herath and Stebila [B10] tested the compatibility of different browsers and software libraries with hybrid signatures as we desxribe in Chapter 5. They found that most TLS connections and S/MIME applications that use hybrid certificates accept extensions/attributes of the size of qTESLA's keys. Nevertheless, software and libraries would likely have to be adapted to the larger key sizes as the Gnu Privacy Guard (GPG) software and the library libgcrypt only support keys up to a key size of four to 15 kilo bits by default. Hence, the implementations in GPG and libgcrypt have to be changed accordingly to support larger key sizes as used in qTESLA-p-I and qTESLA-p-III, e.g., a larger secure memory has to be used in GPG and libgcrypt.

However, some applications require key and signature sizes that are even smaller than RSA sizes and hence, qTESLA sizes. Some examples are signatures with advanced functionality that are implemented on low-end embedded devices. Ambrosin et al [17], for example, constructed an aggregate network attestation protocol. To be able to implement such an attestation protocol for tiny sensor devices, their construction is based on pairings, leading to overall communication cost of 100-200 B. This is well below the qTESLA sizes.

In this chapter, we explained our signature schemes TESLA and qTESLA and discussed their advantages and limitations. In the next chapter, we analyze the vulnerabilities of lattice-based signature schemes regarding implementation attacks.

Table 3.10: Overview of state-of-the-art post-quantum signature schemes

	Scheme/ Software	Comp. Assum.	ROM? Tight?	QROM? Tight?	Security [bit]	Key Size [B]		Sig. Size [B]	Cycle counts [k-cycles]	
Selected lattice-based signatures	GPV[a] [27, 69, 104]	SIS	✓ ✓	✓ ✓	96	pk: 28 508 160 sk: 12 353 536		30 106	sign: verify:	287 500 48 300
	TESLA-p-I [a] (this thesis)	LWE	✓ ✓	✓ ✓	108	pk: 11 559 900 sk: 4 332 000		2 343	sign: verify:	79 486 7 553
	GPV-poly[a] [27, 69, 104]	R-SIS	✓ ✓	✓ ✓	96	pk: 48 435 sk: 24 474		30 822	sign: verify:	71 300 9 200
	GLP [69, 114]	DCK	✓ ✗	✗ ✗	75–80	vk: 1 475 sk: 203		1 119	sign: verify:	452 34
	Bliss-bI [80, 81]	R-SIS, NTRU	✓ ✗	✗ ✗	128	pk: 896 sk: 256		717	sign: verify:	≈358 102
	pqNTRUSign (Uniform) [61, 200]	NTRU LWT/LWR[c]	✓ ✓	✗ ✗	183[b]	pk: 2 048 sk: 2 604		2 048	sign: verify:	202 185 2 533
	FALCON-512[a] [96, 200]	NTRU	✓ ✓	✓ ✓	158[b]	pk: 897 sk: 4 097		617	sign: verify:	8 360 640
	Dilithium-medium [82, 200]	module SIS, module LWE	✓ ✓	✓ ✓	122[b]	pk: 1 184 sk: 2 800		2 044	sign: verify:	1 185 291
	Dilithium-recommended [82, 200]	module SIS, module LWE	✓ ✓	✓ ✓	160[b]	pk: 1 472 sk: 3 504		2 701	sign: verify:	2 754 466
	qTESLA-p-I [a] (this thesis)	R-LWE	✓ ✓	✓ ✓	95[b]	pk: 14 880 sk: 4 544		2 848	sign: verify:	1 590 505
	qTESLA-p-III [a] (this thesis)	R-LWE	✓ ✓	✓ ✓	160[b]	pk: 39 712 sk: 10 816		6 176	sign: verify:	6 242 2 556
	qTESLA-h-I (this thesis)	R-LWE	✓ ✓	✓ ✓	95[b]	pk: 1 504 sk: 1 216		1 376	sign: verify:	626 99
	qTESLA-h-III-size (this thesis)	R-LWE	✓ ✓	✓ ✓	160[b]	pk: 2 976 sk: 1 856		2 720	sign: verify:	1 714 204
	qTESLA-h-III-speed (this thesis)	R-LWE	✓ ✓	✓ ✓	160[b]	pk: 3 104 sk: 2 112		2 848	sign: verify:	866 202
Selected PQ signatures	SPHINCS$^+$-128f[a] (Haraka) [40, 200]	Hash collisions 2nd preimage	✓ ✓	✓ ✓	128[d]	pk: 32 sk: 64		8 080	sign: verify:	872 790 32 632
	MQDSS-31-64 [62, 200]	Multivariate Quadratic system	✓ ✗	✗ ✗	128[d]	pk: 88 sk: 48		67 800	sign: verify:	266 840 554 688

[a] Parameters are chosen according to given security reduction in the ROM/QROM.

[b] Bit security analyzed against classical and quantum adversaries with BKZ cost model $0.265\beta + 16.4 + \log_2(8d)$ [8].

[d] Learning With Truncation (LWT) and Learning with Rounding (LWR) problem [61]

[d] Bit security analyzed against classical and quantum adversaries.

Table 3.11: Comparison of TESLA and qTESLA with RSA and ECDSA

Scheme/ Software	Comp. Assum.	Security [bit]	Key Size [B]		Sig. Size [B]	Cycle counts [k-cycles]	
TESLA-p-I (this thesis)	LWE	108	pk:	11 559 900	2 343	sign:	79 486
			sk:	4 332 000		verify:	7 553
qTESLA-p-I (this thesis)	R-LWE	95[a]	pk:	14 880	2 848	sign:	1 590
			sk:	4 544		verify:	505
qTESLA-p-III (this thesis)	R-LWE	160[a]	pk:	39 712	6 176	sign:	6 242
			sk:	10 816		verify:	2 556
qTESLA-h-I (this thesis)	R-LWE	95[a]	pk:	1 504	1 376	sign:	626
			sk:	1 216		verify:	99
qTESLA-h-III-size (this thesis)	R-LWE	160[a]	pk:	2 976	2 720	sign:	1 714
			sk:	1 856		verify:	204
qTESLA-h-III-speed (this thesis)	R-LWE	160[a]	pk:	3 104	2 848	sign:	866
			sk:	2 112		verify:	202
RSA-2048	Integer Factorization	112[b]	pk:	256	256	sign:	2 095
			sk:	256		verify:	92
RSA-3072	Integer Factorization	128[b]	pk:	384	384	sign:	9 895
			sk:	384		verify:	194
RSA-7680	Integer Factorization	192[b]	pk:	960	960	sign:	185 938
			sk:	960		verify:	1 161
ECDSA (P-192)	Elliptic Curve DL	96[b]	pk:	48	48	sign:	180
			sk:	48		verify:	720
ECDSA (P-256)	Elliptic Curve DL	128[b]	pk:	64	64	sign:	180
			sk:	64		verify:	360
ECDSA (P-384)	Elliptic Curve DL	192[b]	pk:	96	96	sign:	540
			sk:	96		verify:	2 520

[a] Bit security analyzed against classical and quantum adversaries.

[b] Broken against quantum computers (bit security analyzed against classical adversaries).

4 Implementation Security of Lattice-Based Signature Schemes

When introducing new cryptographic schemes in practice, care must be taken to ensure that they are secure against mathematical cryptanalysis. Cryptographic *implementations* must additionally be protected against implementation attacks, considering that the secret key is stored on physical devices [172]. They can be categorized into side-channel or fault attacks [156] and are further exemplified through power, time, electromagnetic, and cache side channels [144] or zeroing, randomizing, and skipping fault attacks [190]. Moreover, lattice-based signature schemes appear to be particularly vulnerable to cache-side-channel and fault attacks. Recent investigations [110, 176, 177, 199] highlight the difficulty in implementing lattice-based signatures without cache side channels. In addition, the most efficient lattice-based signature schemes are based on Fiat-Shamir identification [94] which were targeted in one of the first fault attacks presented by Boneh et al. [47].

In this chapter we develop a cache-side-channel resistant implementation of ring-TESLA. Furthermore, we analyze the signature schemes GLP [114], BLISS [81], and ring-TESLA with respect to fault attacks.

We start this chapter with Section 4.1 on cache side channels. First, we explain the attacker models that we consider for our analysis. We then identify potential vulnerabilities through manual code inspection in Section 4.1.2. Lastly, we implement effective countermeasures in Section 4.1.3. Moving forward, Section 4.2 on fault attacks starts with a description of the analyzed signature schemes and an approach to reduce the number of faults. Subsequently, we analyze BLISS, ring-TESLA, and GLP with respect to zeroing, randomizing, and skipping faults in Section 4.2.3, 4.2.4, and 4.2.5, respectively. In the final part of this section, we present countermeasures for each of the found attacks and explain their realization using ring-TESLA as an example. Finally, Section 4.3 briefly discusses the vulnerability of lattice-based signatures against other side channels.

This chapter is based on the publications [B7] (FDTS 2016) and [B8] (FPS 2017), and [B11] (CODES+ISSS 2017). Section 4.3 is original in this thesis.

4.1 Vulnerability Against Cache-Side-Channel Attacks

In this section, we analyze potential cache side channels of lattice-based implementations at the example of the signature generation in ring-TESLA for the four attacker models that are described in Section 4.1.1.

Mantel, Schickel, and Weber used an extension of the software tool CacheAudit [78] to compute upper leakage bounds of 2.6 bit to 51.6 bit depending on the attacker model. Informally, bit leakage quantifies how much the uncertainty of an attacker is removed when observing the execution of algorithms (with respect to certain attacker models). For a formal definition of bit leakage we refer to [78]. The computed leakage bounds are sound, i.e., conservative with respect to the attacker models, but they might not be tight. Hence, we first inspect the C implementation of ring-TESLA manually to investigate if the determined bit leakage corresponds to an actual concern and, if so, to detect possible cache side channels. Afterwards, we augment the ring-TESLA implementation by countermeasures and argue for the effectiveness of the countermeasures. Experimental results by Mantel, Schickel, and Weber using CacheAudit show indeed a decreased number of bits leaked.

The detection and mitigation of cache side channels not only hardens the ring-TESLA implementation. Other lattice-based constructions, use techniques that are similar to the ones that have been analyzed in ring-TESLA.

4.1.1 Attacker Models

In the following paragraphs, we introduce the four attack models that we consider in our analysis.

As explained in Section 2.5, a cache-side-channel vulnerability might occur if the behavior of the cache depends on secret data. Attacks on cryptographic implementations have exploited secret-dependence in the trace of cache hits and misses [3,103], the time taken for cache hits and misses [37], and the final cache state of an execution [169]. According to these different attacks, cache side channels are categorized as time-driven, trace-driven, or access-driven in the literature [3, 103]. We consider the following four attacker models, following Doychev et al. [78] who distinguish between two different access-driven attacker models:

accd This attacker model captures attackers who can observe the *number of memory blocks* in each cache set in the final cache state after a program execution. It is a generalization of techniques like Evict+Time and Prime+Probe [169].

acc This attacker model captures attackers who can observe the *position of each memory* block in the final cache state. It captures techniques like Flush+Reload [218].

trace This attacker model captures trace-based attackers who can observe the *trace of cache hits and misses* that occur during one program execution. It corresponds to the trace-based attacks in [3, 103] where such traces of hits and misses have been exploited.

time This attacker model captures time-based attackers who can observe the *run-time of one program execution*. For instance, the attack described in [37] models the amount of cache hits and cache misses that occur based on the run-time.

Possible sources for cache side channels are, for example, branchings that depend on secret values or early termination of `while`- or `for`-loops.

4.1.2 Manual Analysis of the Implementation

We now move on to analyze the C implementation of the ring-TESLA signature generation to check if the potential leakage detected by Mantel, Schickel, and Weber corresponds to an actual concern.

The signature scheme ring-TESLA is described in Section 3.1.2.3; Listing 4.1 shows the parts of the signature generation function `crypto_sign` that are most important for our analysis, leaving out variable declarations. To detect potential cache side channels, we analyze all subroutines in `crypto_sign` that get secret values as input or return secret values as their output, e.g., the subroutine `random_y` does not depend on secret information but nevertheless the output has to be kept secret. The following variables have to be kept secret during the execution of `crypto_sign`:

- the secret key `sk`: the secret key consists of three secret polynomials s, e_1, e_2, which are sometimes extracted from `sk`, e.g., Listing 4.12, line 3.
- the randomness `vec_y`: learning information about the randomness might give information about secret data, e.g., in Listing 4.1, line 22 the input `vec_y` of `poly_add(vec_y, vec_y, Sc)` corresponds to the randomness y used during signature generation (see Algorithm 3.12), `Sc` corresponds to sc, and the output value `vec_y` corresponds to z. Hence, information about y might reveal information about sc since $z = y + sc$ is returned as part of the signature.
- the polynomials `vec_v1` and `vec_v2` to compute the hash value: `vec_v1` corresponds to the product of `vec_y` and `poly_a1` (similarly for `vec_v2`). Hence, learning information about `vec_v1` or `vec_v2` might reveal information about `vec_y`, which is also secret as explained above. The polynomials `poly_a1` and `poly_a2`, however, are publicly known.
- the hash value `c` and the coefficient vector of the corresponding encoded polynomial `pos_list`: the two values have to be kept secret until line 24 in Listing 4.1. In line 24 it is decided whether the potential signature (computed

in line 22) is returned together with the value `c` (and hence `c` and `pos_list` become public information) or if all computed values are discarded. An attacker should not learn the values of the discarded polynomials, e.g., `c` or `pos_list`. If the attacker learns values of many discarded `pos_list` or `c` there exists a potential attack as described in the analysis of the subroutines `test_rejection` and `test_w`.

- the polynomials `E1c`, `E2c`, and `Sc`: these values correspond to the polynomials e_1c, e_2c, and sc, respectively, in pseudo-code notation (see Algorithm 3.12). Hence, information about these values might yield information about the secret polynomials e_1, e_2, or s.

The parameters `PARAM_N`, `PARAM_SIGMA`, `PARAM_Q`, `PARAM_B`, `PARAM_W`, `PARAM_D`, and `PARAM_U` are constants and publicly known.

```
1  [...]
2  while(1){
3   sample_y(vec_y);
4   poly_mul_fixed(vec_v1, vec_y, poly_a1);
5   poly_mul_fixed(vec_v2, vec_y, poly_a2);
6   random_oracle(c, vec_v1, vec_v2, m, mlen);
7   generate_c(pos_list, c);
8
9   computeEc(E1c, sk+sizeof(int)*PARAM_N, pos_list);
10  poly_sub(vec_v1,vec_v1, E1c);
11  if (test_w(vec_v1) != 0){
12   continue;
13  }
14
15  computeEc(E2c, sk+sizeof(int)*PARAM_N*2, pos_list);
16  poly_sub(vec_v2,vec_v2, E2c);
17  if (test_w(vec_v2) != 0){
18   continue;
19  }
20
21  computeEc(Sc, sk, pos_list);
22  poly_add(vec_y, vec_y, Sc);
23  if (test_rejection(vec_y) != 0){
24   continue;
25  }
26
27  for(i=0; i<mlen; i++){
28   sm[i]=m[i];
29  }
30  *smlen = CRYPTO_BYTES + mlen;
31  compress_sig(sm+mlen, c, vec_y);
32  return 0;
33 }
```

Listing 4.1: Code of signature generation of ring-TESLA in `crypto_sign`

In what follows, we first present the subroutines that are not vulnerable to cache-side-channel attacks according to our manual analysis and give our reasoning. Afterwards, we describe potential cache side channels in the subroutines

generate_c, test_w, test_rejection, and compute_Ec. Based on the analysis of the subroutines we discuss the vulnerability of the entire routine crypto_sign.

4.1.2.1 Analysis of the Subroutine sample_y

The implementation of the subroutine sample_y is given in Listing 4.2. The function is called once in the sign algorithm in line 3 in Algorithm 4.1 via sample_y(vec_y). The result poly mat_y in Listing 4.2 of the subroutine has to be kept secret.

The cache behavior during the execution of this subroutine does not depend on the values of mat_y for the following reason. The only values loaded in the cache are pos, buf, val, and potentially the elements of mat_y that are changed during the routine. However, each of these values is loaded into the cache only once during the run of the subroutine and is not overwritten by any other value since these elements are small enough to fit in the cache at the same time. Hence it stays in the cache during the execution of the subroutine. The number of accesses to the individual values is not constant due to the branchings in line 8 and line 14. However, it depends only on the number of tries it takes to obtain valid values for pos and val, and the number of tries is not secret.

```
1  void sample_y(poly mat_y){
2   int32_t val;
3   unsigned char buf[3*PARAM_N+68];
4   int pos=0, i=0;
5
6   fastrandombytes(buf,3*PARAM_N+68);
7   do{
8    if(pos == 3*PARAM_N+66){
9     fastrandombytes(buf,3*PARAM_N+68);
10    pos = 0;
11   }
12   val  = (*(int32_t *)(buf+pos)) & 0x7fffff;
13
14   if(val < 0x7fffff){
15    mat_y[i++] = val-PARAM_B;
16   }
17   pos+=3;
18  }
19  while(i< PARAM_N);
20 }
```

Listing 4.2: Code of the subroutine sample_y

Two more subroutines are called during sample_y, namely fastrandombytes and randombytes. Both do not reveal information about mat_y as explained next. The subroutine fastrandombytes is given in Listing 4.3. There are no branchings, loops, or memory accesses that depend on secret values and hence, all cache hits and misses are independent of the computed/sampled values. Furthermore, the call to crypto_stream does not pass any secret information to crypto_stream since

`rlen` is a public parameter and `key` is random. Hence, there should be no cache side channels in the subroutine `fastrandombytes`.

```
1  void fastrandombytes(unsigned char *r, unsigned long long rlen){
2   unsigned long long n=0;
3   int i;
4   if(!init){
5    randombytes(key, crypto_stream_KEYBYTES);
6    init = 1;
7   }
8   crypto_stream(r,rlen,nonce,key);
9
10  for(i=0;i<8;i++){
11   n ^= ((unsigned long long)nonce[i]) << 8*i;
12  }
13  n++;
14  for(i=0;i<8;i++){
15   nonce[i] = (n >> 8*i) & 0xff;
16  }
17 }
```

Listing 4.3: Code of the subroutine `fastrandombytes`

The subroutine `randombytes` is given in Listing 4.4. There are also no cache side channels in this subroutines since the only branchings in line 2 until line 6 do not depend on secret (but on random) information.

```
1  void randombytes(unsigned char *x,unsigned long long xlen){
2   while (xlen > 0){
3    if (!outleft){
4     if (!++in[0]){
5      if (!++in[1]){
6       if (!++in[2]){
7        ++in[3];
8       }
9      }
10    }
11    surf();
12    outleft = 8;
13   }
14   *x = out[--outleft];
15   ++x;
16   --xlen;
17  }
18 }
```

Listing 4.4: Code of the subroutine `randombytes`

4.1.2.2 Analysis of the Subroutine `poly_mul_fixed`

The implementation of the subroutine `poly_mul_fixed` is given in Listing 4.5. The function is called twice during signing in Listing 4.1, namely in line 4 and 5 via

`poly_mul_fixed(vec_v1,vec_y, poly_a1)` and `poly_mul_fixed(vec_v2,vec_y, poly_a2)`, respectively. The input value that has to be kept secret is `x` (corresponding to `vec_y` in both calls) in Listing 4.5. Later on in the subroutine also the value `result` has to be kept secret.

The only conditional branching depending on `x` is in line 4, Listing 4.5. However, this does not lead to a cache side channel since the used values in the `then`-branch only depend on `result` and `x`–the two values the `if`-condition depends on. Hence, by a cache hit the attacker just learns that `results != x` which does not give any additional information about the secret values.

```
1  void poly_mul_fixed(poly  result, const poly  x, const poly  a){
2   unsigned int i;
3
4   if (result != x){
5    for(i = 0; i < PARAM_N; i++){
6     result[i] = x[i];
7    }
8   }
9   mul_coefficients(result, psis);
10  bitrev_vector(result);
11  ntt(result, omegas);
12
13  for(i=0; i<PARAM_N; i++){
14   result[i] = fmodq(result[i] * a[i]);
15  }
16
17  bitrev_vector(result);
18  ntt(result, omegas_inv);
19  mul_coefficients(result, psis_inv);
20 }
```

Listing 4.5: Code of the subroutine `poly_mul_fixed`

There are no further loops, conditions, or access indices depending on `x` or `result`. They only depend on publicly known constants. However, there are four more calls of subroutines with (secret) input value `result`, namely `ntt`, `bitrev_vector`, `mul_coefficients`, and `fmodq`. The three subroutines `ntt`, `bitrev_vector`, and `mul_coefficients` are implementations of (pre-)computations during the NTT which we do not display here. The subroutine of `fmodq` is given in Listing 4.6. All four subroutines have control flow and memory accesses that are independent of secret values. The computed operations do not depend on the values of `result` but only on constants or public parameters such as `PARAM_N` or `PARAM_Q`. Hence, the subroutine `poly_mul_fixed` is not vulnerable to cache-side-channel attacks.

4.1.2.3 Analysis of the Subroutine `random_oracle`

The implementation of the subroutine `random_oracle` is given in Listing 4.7. It is called once in `crypto_sign` in line 6 via `random_oracle(c,vec_v1,vec_v2,m,mlen)`.

```
1  static int32_t fmodq(int64_t x){
2   int64_t u = (x * 206175203327);
3   u &= (((int64_t)1<<39)-1);
4   u *= PARAM_Q;
5   u += x;
6   return u>>39;
7  }
```

Listing 4.6: Code of the subroutine `fmodq`

The input values that have to be kept secret are `v1` and `v2` in Listing 4.7.

Due to the following reasons no information about the two values is obtained by a cache side channel. The control flow and memory accesses are independent of the value of the secrets. Furthermore, the subroutine `compress_v` has control flow and memory accesses independent of secret information as can be seen in Listing 4.8. For example, every coefficient of `v1` is computed modulo `PARAM_Q` (corresponding to the modulus q in pseudo-code notation) independently of whether the value is already in the interval $(-q/2, q/2]$ or not. Also, the subroutine `crypto_hash_sha512` does not contain secret-dependent branches, loops, or access indices. We do not display the (very long) source code here. Overall, there should be no cache side channels present in the subroutine `random_oracle`.

```
1  void random_oracle(unsigned char *c_bin, poly v1, poly v2, const unsigned char *m,
       unsigned long long mlen){
2   int32_t t1[PARAM_N],t2[PARAM_N];
3   unsigned long long i;
4   unsigned char buf[2*PARAM_N+mlen];
5   compress_v(t1, v1);
6   compress_v(t2, v2);
7   for(i=0; i<PARAM_N; i++){
8    buf[i] = t1[i];
9   }
10  for(i=0; i<PARAM_N; i++){
11   buf[i+PARAM_N] = t2[i];
12  }
13  for(i=0; i<mlen; i++){
14   buf[i+2*PARAM_N] = m[i];
15  }
16  crypto_hash_sha512(c_bin, buf, mlen+2*PARAM_N);
17 }
```

Listing 4.7: Code of the subroutine `random_oracle`

4.1.2.4 Analysis of the Subroutines `poly_sub` and `poly_add`

The implementations of the subroutines `poly_sub` and `poly_add` are given in Listing 4.9 and Listing 4.10, respectively. They are called three times during signing: `poly_sub(vec_v1,vec_v1, E1c)` in line 10, `poly_sub(vec_v2,vec_v2,E2c)` in

```
1 static void compress_v(int32_t t[PARAM_N], poly v){
2  int i;
3  for(i=0;i<PARAM_N;i++){
4   int32_t c = v[i] % PARAM_Q;
5   t[i] = ((int64_t)(v[i]-c))>>PARAM_D;
6  }
7 }
```

Listing 4.8: Code of the subroutine `compress_v`

line 16, and `poly_add(vec_y,vec_y,Sc)` in line 22 in Algorithm 4.1. The two subroutines are exactly the same except for one summand in line 4 in both of the listings. Hence, we analyze them together next. The values `result` and `y` in Listing 4.9 and 4.10 have to be kept secret.

The two subroutines consist of a `for`-loop depending on the publicly known value `PARAM_N` and calls to the subroutine `fmodq`. No branchings, loops, or access indices depend on secret values (in the routines `poly_sub` and `poly_add`, and the subroutine `fmodq`, given in Listing 4.6). Hence, there are no cache side channels in these two subroutines.

```
1 void poly_sub(poly  result, const poly  x, const poly  y){
2  unsigned int i;
3  for(i = 0; i < PARAM_N; i++){
4   result[i] = fmodq((int64_t)(x[i] + (PARAM_Q-y[i]))<<39);}
5 }
```

Listing 4.9: Code of the subroutine `poly_sub`

```
1 void poly_add(poly  result, const poly  x, const poly  y){
2  unsigned int i;
3  for(i = 0; i < PARAM_N; i++){
4   result[i] = fmodq((int64_t)(x[i] + y[i])<<39);}
5 }
```

Listing 4.10: Code of the subroutine `poly_add`

4.1.2.5 Analysis of the Subroutine `compress_sig`

The implementation of the routine `compress_sig` is given in Listing 4.11. It is called once in `crypto_sign`, namely in line 31 via `compress_sig(sm+mlen,c,vec_y)`. In this subroutine the values `c` and `vec_y` have to be kept secret.

There are no branchings, loops, or access indices depending on these secret values since all operations are computed exactly the same way for different secret values. Hence, there are no cache side channels in `compress_sig`.

```
1  static void compress_sig(unsigned char *sm, unsigned char *c, poly vec_z){
2   int i,k;
3   int ptr =0;
4   int32_t t=0;
5
6   for (i=0; i<32; i++){
7    sm[ptr++] =c[i];
8   }
9   for(i=0; i<PARAM_N; i++){
10   t = (int32_t)vec_z[i];
11   for(k=0;k<3;k++){
12    sm[ptr++] = ((t>>(8*(k))) & 0xff);
13   }
14 }
```

Listing 4.11: Code of the subroutine `compress_sig`

4.1.2.6 Analysis of the Subroutine `compute_Ec`

The implementation of the subroutine `compute_Ec` is given in Listing 4.12. It is called three times during sign, namely `computeEc(E1c, sk+sizeof(int)*PARAM_N, pos_list)` in line 9, `computeEc(E2c, sk+sizeof(int)*PARAM_N*2, pos_list)` in line 15, and `computeEc(Sc, sk, pos_list)` in line 21. In Listing 4.12, the following values have to be kept secret: `Ec`, `sk`, `e`, `pos_list`, and `pos`.

Most loops and branchings do not depend on any of the secret values. However, there might be a possible side channels that reveals information of `pos` (and hence, of the values in `pos_list`) because of the cache hits/misses depending on `e`. In both loop bodies values are read from `e` (namely, either `e[j+PARAM_N-pos]` or `e[j-pos]`) such that in both loops together all entries of `e` are read. However, information might be gathered from the chronological order of cache hits and misses. We illustrate this using an example: The array `e` consists of `PARAM_N` many entries of type `int`, i.e., each entry of `e` is represented in 32 bit. We assume that one cache line is 64 byte as it is used in the first level cache of the Intel Skylake architecture [126] that is also used to benchmark qTESLA in Section 3.4.2. Then about 16 entries (depending on the alignment in the memory) of `e` fit into one cache line. Assume furthermore, `pos`=14. Then, under the trace-driven attacker model *trace*, an attacker sees one cache miss (on element `e[PARAM_N-14]`) and 13 cache hits (on elements `e[PARAM_N-13]`,..., `e[PARAM_N-1]`) during the loop in line 10.[9] However, two more entries of `e` are also already loaded in the cache, namely `e[PARAM_N-16]` and `e[PARAM_N-15]`. Thus, in the second loop in line 13, the attacker sees cache hits on these two elements. He might, hence, be able to determine the value of `pos` from the distribution of cache hits for the considered cache line over the loops.

[9] To simplify our explanation we assume that the corresponding cache line starts with `e[PARAM_N-16]` and ends with `e[PARAM_N-1]`.

```
1  static void computeEc(poly Ec, const unsigned char *sk, const uint32_t pos_list[
       PARAM_W]){
2   int i,j, pos, * e;
3   e = (int*)sk;
4   for(i=0;i<PARAM_N;i++){
5    Ec[i] = 0;
6   }
7
8   for(i=0;i<PARAM_W;i++){
9    pos = pos_list[i];
10   for(j=0;j<pos;j++){
11    Ec[j] += e[j+PARAM_N - pos];
12   }
13   for(j=pos;j<PARAM_N;j++){
14    Ec[j] -= e[j-pos];
15   }
16  }
17 }
```

Listing 4.12: Code of the subroutine `computeEc`

4.1.2.7 Analysis of the Subroutine `test_rejection`

The implementation of the subroutine `test_rejection` is given in Listing 4.13. It is called once in the sign algorithm in line 23 via `if (test_rejection(vec_y) != 0)`. The variable `poly_z` in Listing 4.13 has to be kept secret.

The subroutine `test_rejection` consists of a `for`-loop that loops independently of the secret over `i=0,...,PARAM_N`. Within the `for`-loop, there is a secret-dependent `if`-condition which leads to a potential cache-side-channel vulnerability as explained next.

We assume a strong trace-driven attacker model, i.e., the attacker has a sequence of occurred cache hits and misses. Furthermore, we assume that `poly_z` is already loaded in the cache before the `if`-condition is evaluated.[10] We first consider the case if the `if`-condition in line 4, Listing 4.13, holds never true. Then the value `0` is returned and the attacker gets a sequence of `PARAM_N` (or 2·`PARAM_N`–depending on the compilation) hits. This essentially means that all coefficients of `poly_z` are in the interval $[-B+U, B-U]$ and, hence, the corresponding signature is compressed and returned (see Listing 4.1). Next, we consider the other case, i.e., the absolute value of at least one of the coefficients of `poly_z` is larger than $B-U$. That means that the `if`-condition in Listing 4.13 holds true for some `i`$\in \{$`0`, ...,`PARAM_N`$\}$. Hence, `1` is returned in the `i`-th iteration and the attacker gets a sequence of only `PARAM_N`$-$`i` hits. Consequently, the attacker knows the exact index of the coefficient that violated the `if`-condition. Now we assume that the attacker also knows the values in the array `pos_list` (which corresponds to the polynomial

[10] Our arguments hold also true if we assume that `poly_z` is not loaded in the cache. In the ring-TESLA implementation, however, it is loaded in the cache, see line 22 in Listing 4.1.

$c = \mathsf{Enc}(\mathsf{H}([v_1]_M, [v_2]_M, \mu))$, see Figure 3.12) from another cache side channel. Then it is possible that the attacker finds out which coefficients of the secret s contributed to the `i`-th, large coefficient of `poly_z`. If an attacker learns the exact position `i` and the corresponding `pos_list` for many different values to the same secret key s then the attacker might receive enough information about the size of the entries in s to successfully break the scheme via a learning-the-parallelepiped-attack [84, 166]. At least, this information might decrease the secret key space considerably.

```
1 static int test_rejection(poly poly_z){
2  int i;
3  for(i=0; i<PARAM_N; i++){
4   if(poly_z[i]<-(PARAM_B-PARAM_U)||poly_z[i]>(PARAM_B-PARAM_U)){
5    return 1;
6   }
7  }
8  return 0;
9 }
```

Listing 4.13: Code of the subroutine `test_rejection`

4.1.2.8 Analysis of the Subroutine `test_w`

The implementation of the routine `test_w` is given in Listing 4.14. It is called twice in the sign algorithm: in line 11 via `if (test_w(vec_v1) != 0)` and in line 17 via `if (test_w(vec_v2) != 0)`. The values `poly_w`, `val`, and `left` in Listing 4.14 have to be kept secret.

There exists a potential vulnerability in the subroutine `test_w` that is similar to the cache side channel described for `test_rejection` in Section 4.1.2.7. In the subroutine `test_w`, the cache-side-channel vulnerability comes from the early abortion depending on `left` in line 15 of Listing 4.14. Namely, when the `if`-condition in line 15 holds for some `i` and the corresponding `abs(left)`, `-1` is immediately returned, a trace-driven attacker might learn the exact index `i`.

4.1.2.9 Analysis of the Subroutine `generate_c`

The implementation of the subroutine `generate_c` is given in Listing 4.15. It is called once in the signature generation in line 7 via `generate_c(pos_list, c)`. The values `pos_list`, `c`, and `pos` in Listing 4.15 have to be kept secret.

There are no branchings or loops depending on secret values, except for one `if`-condition on `c[pos]` in line 15 of Listing 4.15 which leads to a possible side channel. To explain the side channel, we first explain the cache policy *no-write-allocate* first. In case of using no-write-allocate policy, data is only loaded in the cache if the data is read. In contrast, if data should only be written–without reading it

```
1  static int test_w(poly poly_w){
2   int i;
3   int64_t left, right, val;
4   for(i=0; i<PARAM_N; i++){
5    val = (int64_t) poly_w[i];
6    val = val % PARAM_Q;
7    if (val < 0){
8     val = val + PARAM_Q;
9    }
10   left = val;
11   left = left % (1<<(PARAM_D));
12   left -= (1<<PARAM_D)/2;
13   left++;
14   right = (1<<(PARAM_D-1))-PARAM_REJECTION;
15   if (abs(left) > right){
16    return -1;
17   }
18  }
19  return 0;
20 }
```

Listing 4.14: Code of the subroutine `test_w`

first–the respective data is not loaded in the cache. Now in case of ring-TESLA, if a cache with no-write-allocate policy is used, the values `c[i]` are not cached in line 7. Consequently, an attacker might be able to find out which elements `c[i]` are cached in line 15 and hence, to learn information about the values of `pos`. Together with the (potential) vulnerability in `test_rejection`, an attacker might be able to successfully break the scheme via a learning-the-parallelepiped-attack [84, 166].

4.1.2.10 Combined Analysis of the Overall Signature Generation

The most important parts of the implementation of `crypto_sign` are depicted in Listing 4.1. In the signature generation, most operations, branchings, or loops are independent of secret values. Exceptions are the branchings in line 11, 17, and 23 in Listing 4.1: They depend on secret values and hence, the length of the observed trace of cache hits and misses depends on the branches that were taken.

What does this mean from a cryptographic viewpoint? If we assume that the subroutine `test_w` does not have cache side channels then the attacker does not learn more information about the secret if he knows whether or not the condition in line 11 holds. The attacker would only learn that `vec_v1` does not fulfill the conditions needed for a valid signature. However, the attacker does not learn why exactly the condition was not fulfilled, i.e., the attacker does not learn the corresponding index on which the `if`-condition failed. Furthermore, since the value `vec_v1` depends on `vec_y` and the value `vec_y` is discarded if the `if`-condition in line 11 does not hold, the attacker does not gain any additional information about the secret. The same explanation also holds for the branchings in line 17 and line

```
1  void generate_c(uint32_t *pos_list, unsigned char *c_bin){
2   int32_t c[PARAM_N]; int cnt =0; int pos;
3   [...]
4   crypto_stream(r, R_LENGTH, nonce, c_bin);
5
6   for(i=0; i<PARAM_N; i++){
7    c[i] = 0;
8   }
9   i=0;
10  while(i<PARAM_W){
11   pos = 0;
12   pos = (r[cnt]<<8) | (r[cnt+1]);
13   pos &= PARAM_N-1;
14   cnt += 2;
15   if (c[pos] == 0){
16    pos_list[i] = pos;
17    c[pos]=1;
18    i++;
19    cnt++;
20   }
21  }
22 }
```

Listing 4.15: Code of the subroutine `generate_c`

23.

In summary, this means that there exists a potential cache side channel that we probably cannot mitigate, but it does not affect the security of the signature scheme as long as the subroutines `test_w` and `test_rejection` or `generate_c` do not reveal secret information.

4.1.3 Mitigation of the Vulnerabilities

In this section we propose and implement countermeasures to mitigate the side channels identified in Section 4.1.2.

4.1.3.1 Adaptation of Vulnerable Routines

We identified possible cache side channels in the routines `test_w`, `test_rejection`, `computeEc`, and `generate_c`. As explained in Section 4.1.2.10, revealing information in the routine `generate_c` is only a concern if corresponding information is gathered in `test_w` or `test_rejection`. Hence, it is sufficient to mitigate potential cache side channels in `test_w`, `test_rejection`, and `computeEc`.

Our proposed adaption of the subroutine `test_rejection` is given in Listing 4.16 and described as follows. The routine `test_rejection` returns 0 or 1 depending on whether all coefficients of the potential signature are of the correct size. In our adapted code, we introduce an auxiliary variable `res` and collect the result, i.e., 0 or 1, in `res`. Then, instead of returning early in case of a failed test as in

the original implementation, we return `res` after `PARAM_N` iterations. Using this adaption, the cache behavior does not depend on the index of the coefficient that revokes the rejection of `poly_z` anymore.

```
1 int test_rejection(poly poly_z){
2  int i; int res;
3  res = 0;
4  for(i=0; i<PARAM_N; i++){
5    res |= (poly_z[i] < -(PARAM_B-PARAM_U));
6    res |= (poly_z[i] > (PARAM_B-PARAM_U));
7   }
8   return res;
9  }
```

Listing 4.16: Adaption of the subroutine `test_rejection`

Next we turn to the adaption of the subroutine `test_w` that is shown in Listing 4.17. Our adaption follows the same idea as used in Listing 4.16. In particular, we also save the result, i.e., whether 0 or -1 is returned, in the auxiliary variable `res`, instead of returning -1 as soon as a coefficient causes rejection.

```
1  int test_w(poly poly_w){
2   [...]
3   for(i=0; i<PARAM_N; i++) {
4     val = poly_w[i]; val = val % PARAM_Q;
5     val += (((unsigned int)val & 0x80000000) >> 31)*PARAM_Q;
6     left = val;
7     left = left % (1<<(PARAM_D));
8     left -= (1<<PARAM_D)/2; left++;
9     right = (1<<(PARAM_D-1))-PARAM_REJECTION;
10    res |= (abs(left) - right > 0);
11  }
12  return -res;
13 }
```

Listing 4.17: Adaption of the subroutine `test_w`

Finally, we propose to adapt the subroutine `computeEc` as depicted in Listing 4.18 and described as follows. In the subroutine `computeEc`, we add preloading of the variable `e`. This ensures that the sequence of cache hits and misses does not depend on the secret-dependent order of accesses, since all coefficients are loaded in the cache (under the assumption that no process interferes with the cache during the ring-TESLA execution). This implies for instance that a trace-driven attacker can observe `PARAM_W` many cache hits independent of the actual coefficient index.

By code inspection, our proposed modifications should remove the cache side channels in the three subroutines.

```
1  void computeEc([...]){
2   [...]
3   for(i=0;i<PARAM_N;i++){
4    Ec[i] = 0;
5   }
6   for(i=0;i<PARAM_N;i++){
7    tmp = e[i];
8   }
9   for(i=0;i<PARAM_W;i++){
10   pos = pos_list[i];
11   for(j=0;j<pos;j++){
12    Ec[j] += e[j+PARAM_N - pos];
13   }
14   for(j=pos;j<PARAM_N;j++){
15    Ec[j] -= e[j-pos];
16   }
17  }
18 }
```

Listing 4.18: Adaption of the subroutine `computeEc`

4.1.3.2 Analysis of the Effectiveness of the Mitigations

The effectiveness of most of our countermeasures is also supported by another round of program analysis with CacheAudit by Mantel, Schickel, and Weber. The individual leakage bounds computed with CacheAudit on the unmitigated and the adapted implementations of `test_w`, `test_rejection`, `computeEc`, and `crypto_sign` with respect to the four considered attacker models are listed in Table 4.1.

Table 4.1: Leakage bounds [bit]

	Unmitigated routines				Mitigated routines			
	acc	*accd*	*trace*	*time*	*acc*	*accd*	*trace*	*time*
`test_w`	31	31	**49152**	19.3	0	0	0	0
`test_rejection`	**31**	**31**	10.1	10.1	0	0	0	0
`computeEc`	0	0	**20**	5.9	0	0	**19**	4.4
`crypto_sign`	12.9	2.6	**51.6**	9.5	8.1	1.6	**48.6**	9.0

For `test_w` and `test_rejection`, CacheAudit returned an upper bound on the bit leakage of 0 bit for all four attacker models. Hence, the potential cache side channels are effectively removed. For `computeEc` the picutre is different: Regarding the attacker models *acc* and *accd*, the potential vulnerability is removed as well, since 0 bit of leakage are obtained. However, upper bounds of 19 bit and 4.4 bit leakage for the attacker models *trace* and *time*, respectively, are reported. The bounds determined by Mantel, Schickel, and Weber are upper bounds, but they are

not necessarily tight. According to our manual analysis, the preloading of `e` should mitigate the cache side channel in `computeEc`, because it makes the caching of `e` independent of secrets. Hence, it is possible that a refinement of CacheAudit is needed to prove that no cache side channels occurs in `computeEc`. This is, however, out of the scope of this thesis.

Based on our manual inspection of the individual subroutines, there might be two possible sources for the remaining *potential* leakage reported by CacheAudit. One of the sources is the routine `generate_c`, where, as discussed before, the remaining leakage is harmless because we mitigated the cache side channels in `test_w` and `test_rejection`. The second source is the rejection sampling in `crypto_sign` which should not pose a vulnerability since only the number of sign iterations might be revealed but no information about the secret values. Currently, no approach to avoid rejection sampling for signature schemes that use the Fiat-Shamir approach, such as ring-TESLA, qTESLA, TESLA, and others such as [24, 30, 81, 82]. Moreover, the results of `computeEc` are propagated further through the implementation because of the rejection sampling.

Finally, it is important to note that the leakage bounds refer to the decrease of the uncertainty about the secret key, i.e., about the three polynomials that all together are saved in more than one kilo-byte (KB). By construction of the R-LWE problem, the potential leakage of at most 49 bit of the secret key, reported by Mantel, Schickel, and Weber, does not immediately translate to the bit hardness of LWE (resp., the bit security of ring-TESLA).

This section focused on analyzing and mitigating cache side channels. Moving on to the next section, we turn to another implementation attack, namely fault attacks against lattice-based signature schemes.

4.2 Susceptibility to Fault Attacks

This section provides a thorough analysis of the lattice-based signature schemes GLP by Güneysu et al. [114], BLISS by Ducas et al. [81], and ring-TESLA described in Section 3.1.2.3 with respect to fault attacks. We consider first-order fault attacks regarding randomizing[11], skipping, and zeroing faults. We explore the reasons for the vulnerability and resistance of the key generation, signature generation, and verification algorithms. Furthermore, we propose countermeasures for each of the developed attacks and realize the countermeasures for ring-TESLA as an example.

For our analysis, we use the pseudo-code descriptions and the publicly available software of the three signature schemes, i.e., the implementation of BLISS [81] in C++, the implementation of the GLP scheme [115] in C, and the implementation of ring-TESLA in C. Since the implementation of ring-TESLA and the GLP scheme use the benefits of the AVX instructions, the X86 disassembled code of the respective code lines is also considered occasionally.

A summary of the analyzed attacks and the respective vulnerabilities of the three signature schemes is depicted in Table 4.2. In the table, we enumerate the number of needed faults if the scheme is vulnerable and we write "(❍)" if it is vulnerable but only with a huge number of needed faults. Furthermore, we write "❍" if a scheme is not vulnerable to the respective attack and "-" if the attack is not applicable to the respective scheme. The table shows the algorithms targeted by the fault attacks, i.e., key generation (KG), signature generation (S), or verification (V). Moreover, the column "Referred to as" summarizes the abbreviation used in later sections. It must be noted that certain effects can be achieved with different kinds of fault attacks, e.g., a value of a variable can be set to zero during the execution with a zeroing fault or a skipping fault. While such fault attacks are only listed once in Table 4.2, they are mentioned and explained in all relevant paragraphs throughout this section. Moving further, we present the analyzed schemes BLISS and GLP next.

4.2.1 Description of the Analyzed Signature Schemes

In this subsection, we describe the signature schemes, namely the GLP scheme and BLISS, which are then analyzed further within this section. The scheme ring-TESLA is stated in Algorithm 3.11, 3.12, and 3.13 in Section 3.1.2.3. For later reference, we use the instantiation ring-TESLA-I [B1], i.e., $n = 512$, $\sigma = 30$, $q = 8399873$, $B = 2^{21} - 1$, $h = 11$, $d = 21$, and $U = 993$ and ring-TESLA-II [B1], i.e., $n = 512$, $\sigma = 48$, $q = 33550337$, $B = 2^{22} - 1$, $h = 19$, $d = 23$, and $U = 2848$.

[11] We do not analyze bit flips separately since they are either covered by randomizing faults or considered unrealistic [106] in practice.

Table 4.2: Comparison of the GLP scheme, BLISS, and ring-TESLA with respect to their vulnerability to the attacks described in this section

Fault Attack	Algorithm	Referred to as	GLP	BLISS	ring-TESLA	Section
Rand. of secret[a]	S	R-S-Sec	24	354	❍	4.2.4.1
Rand. of error	S	R-S-Err	❍	❍	❍	4.2.4.2
Rand. of modulus	S	R-S-Mod	❍	❍	❍	4.2.4.3
Rand. of randomness	S	R-S-Rand	❍	❍	❍	
Skip of mod-reduction	KG	S-KG-Mod	❍	-	❍	4.2.5.1
Skip of addition	KG	S-KG-Add	1	1	1	
Skip of rejection	S	S-S-Rej	(❍)	(❍)	(❍)	4.2.5.2
Skip of addition	S	S-S-Add	1	❍	❍	
Skip of mod-reduction	S	S-S-Mod	❍	-	❍	
Skip of correctness check	V	S-V-Cor	1	1	1	4.2.5.3
Skip of size check	V	S-V-Size	1	1	❍	
Zero. of secret	KG	Z-KG-Sec	1	-	1	4.2.3.1
Zero. of randomness[b]	S	Z-KG-Ran	1	2	1	4.2.3.2
Zero. of hash value	S	Z-S-HVal	❍	❍	❍	4.2.3.3
Zero. of hash polynomial	V	Z-S-HPoly	1	1	1	4.2.3.4

[a] Number of needed faults computed with $r = 4$.
[b] Number of needed faults computed with $r = 1$.

4.2.1.1 GLP

The GLP scheme is depicted in Algorithm 4.1, 4.2, and 4.3. The secret key consists of two polynomials $s, e \leftarrow_{\$} \mathcal{R}_{q,[1]}$ with ternary coefficients, i.e., with coefficients in $\{-1, 0, 1\}$; the public key is a tuple of $a \leftarrow_{\$} \mathcal{R}_q$ and $b = as + e \mod q$. On input message m, the sign algorithm first samples $y_1, y_2 \leftarrow_{\$} \mathcal{R}_{q,[k]}$. Afterwards, the most significant bits of $ay_1 + y_2$ and m are hashed with the hash function H. Then the signature polynomials z_1 and z_2 are computed. To hide the secret, rejection sampling is applied, i.e., z_2 is compressed to $z_2^\star$ and the signature is returned only with some probability. The verification algorithm checks the size of z_1 and $z_2^\star$, and the equality of c and $\mathsf{H}([az_1 + z_2^\star - bc]_M, \mathsf{m})$. The compression algorithm $\mathsf{compress}$ is constructed with the following property [114, Lemma 3.1]. For any q, n, k with $\frac{2nk}{q} > 1$, $z \in \mathcal{R}_{q,[k]}$, and $y \leftarrow_{\$} \mathcal{R}_q$, the algorithm $\mathsf{compress}(y, z, q, k)$ outputs $z' \in \mathcal{R}_{q,[k]}$ such that $[y + z]_M = [y + z']_M$. For detailed information about the parameters and $\mathsf{compress}$, we refer to the original work.

The security of GLP is based on the DCK problem that is defined in Section 2.2. In the remainder of the section, we use the instantiation GLP-Set-I with $n = 512$ and $q = 8383489$, which gives a bit security of at least 71 bit [217].

Algorithm 4.1 Key generation of the GLP scheme

Require: -
Ensure: Secret key $\mathsf{sk} = (s, e)$ and public key $\mathsf{pk} = (a, b)$

1: $s, e \leftarrow_{\$} \mathcal{R}_{q,[1]}$
2: $a \leftarrow_{\$} \mathcal{R}_q$
3: $b \leftarrow as + e \mod q$
4: $\mathsf{sk} \leftarrow (s, e)$, $\mathsf{pk} \leftarrow (a, b)$
5: **return** $(\mathsf{sk}, \mathsf{pk})$

Algorithm 4.2 Signature generation of the GLP scheme

Require: Message m, secret key $\mathsf{sk} = (s, e)$, and polynomial a
Ensure: Signature $(z_1, z_2^\star, c)$

1: $y_1, y_2 \leftarrow_{\$} \mathcal{R}_{q,[k]}$
2: $c \leftarrow \mathsf{H}\left(\left[ay_1 + y_2\right]_M, \mathsf{m}\right)$
3: $z_1 \leftarrow y_1 + sc$
4: $z_2 \leftarrow y_2 + ec$
5: **if** $z_1, z_2 \notin \mathcal{R}_{k-32}$ **then**
6: restart in line 1
7: **else**
8: $z_2^\star \leftarrow \mathsf{compress}(az_1 - bc, z_2, p, k - 32)$
9: **return** $(z_1, z_2^\star, c)$

Algorithm 4.3 Verification of the GLP scheme

Require: Message m, public key $\mathsf{pk} = (a, b)$, and signature $(z_1, z_2^\star, c)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $c' \leftarrow \mathsf{H}\left(\left[az_1 + z_2^\star - bc\right]_M, \mathsf{m}\right)$
2: **if** $c = c' \wedge\ z_1, z_2^\star \in \mathcal{R}_{k-32}$ **then**
3: **return** 0
4: **else**
5: **return** -1

4.2.1.2 BLISS

Moving further, we depict the scheme BLISS in Algorithm 4.4, 4.5, and 4.6. The key pair is chosen NTRU-like, i.e., the public key is $\mathsf{pk} = (a_1, a_2) = \left(2\frac{2g+1}{f} \mod q, q - 2\right)$, where $g \leftarrow_{\$} F_{d_1,d_2} = \{\sum_{i=0}^{n-1} h_i x^i | h_i \in \{-2, -1, 0, 1, 2\},\ |\{h_i = \pm 1\}| = d_1,\ |\{h_i = \pm 2\}| = d_2\}$ and $f \leftarrow_{\$} F_{d_1,d_2}^{\times}$. The secret key sk consists of $\mathsf{sk} = (s_1, s_2)^T = (f, 2g + 1)^T$. Furthermore, the vectors (a_1, a_2), $(s_1, s_2)^T$, and $\xi \in \mathbb{Z}$ are chosen

such that $(a_1, a_2)(s_1, s_2)^T = q = -q \bmod 2q$, $\xi(q-2) = 1 \bmod 2q$, and hence, $\xi(a_1, a_2) = (\xi a_1, 1) \bmod 2q$. To sign a message m, random polynomials y_1 and y_2 are sampled with Gaussian distribution. Then, a hash value c is computed from the randomness, the public key, ξ, and the message m with the hash function H. The rounding function $\lfloor \cdot \rceil_M$ rounds elements mod $2q$ instead of mod q as for GLP and ring-TESLA. After computing the hash, the value $b \leftarrow_\$ \{0,1\}$ is chosen, the polynomials $z_1 = y_1 + (-1)^b s_1 c$ and $z_2 = y_2 + (-1)^b s_2 c$ are computed, rejection sampling is applied, and z_2 is compressed to $z_2^\star$. During verification of the signature $(z_1, z_2^\star, c)$, the sizes of z_1 and $z_2^\star$, and the equality of c and $\mathsf{H}\left(\lfloor \xi a_1 z_1 + \xi q c \bmod 2q \rceil_M + z_2^\star \bmod q, \mathsf{m}\right)$ are checked.

The security of BLISS is based on (NTRU-like instantiations of) the R-SIS problem, defined in Section 2.2. Ducas et al. [81] provide two parameter sets that are estimated to give 124 bit of security, namely BLISS-I with $n = 512$, $\sigma = 215$, $q = 12289$ and BLISS-II with $n = 512$, $\sigma = 107$, and $q = 12289$. Furthermore, we emphasize that in the instantiations BLISS-I and BLISS-II, $d_2 = 0$. Hence, it holds for the secret polynomials $s_1 = \sum_{i=0}^{n} s_{1,i} x^i$ and $s_2 = \sum_{i=0}^{n} s_{2,i} x^i$ that

$$s_{j,i} \in \begin{cases} \{-1, 0, 1\} & \text{if } j = 1, \\ \{-1, 1, 3\} & \text{if } j = 2, i = 0, \\ \{-2, 0, 2\} & \text{if } j = 2, i \in \{1, ..., n-1\}. \end{cases}$$

Algorithm 4.4 Key generation of BLISS

Require: -
Ensure: Secret key $\mathsf{sk} = (s_1, s_2)^T$ and public key $\mathsf{pk} = (a_1, a_2)$

1: $f, g \leftarrow_\$ F_{d_1,d_2}$
2: **if** $N_\lambda(\mathbf{S}) \geq 5C^2(\lceil \delta_1 n \rceil + 4\lceil 4\delta_2 n \rceil)\kappa$ **then**
3: restart in line 1
4: $a_q = (2g+1)/f \mod q$
5: **if** f not invertible **then**
6: restart in line 1
7: $(s_1, s_2)^T \leftarrow (f, 2g+1)^T$
8: $(a_1, a_2) = (2a_q, q-2) \bmod 2q$
9: $\mathsf{sk} \leftarrow (s_1, s_2)^T$, $\mathsf{pk} \leftarrow (a_1, a_2)$
10: **return** $(\mathsf{sk}, \mathsf{pk})$

4.2.2 Reducing the Number of Necessary Faults

After the description of the signature schemes in the previous section, we now elaborate on how to combine fault attacks and algorithms that solve lattice problems

Algorithm 4.5 Signature generation of BLISS; define the value $\nu = (M \exp(-\|Sc\|_2^2/(2\sigma^2)) cosh(\langle z, Sc\rangle/\sigma^2)))$

Require: Message m, secret key $S = (s_1, s_2)^T$, and $A = (a_1, q-2)$
Ensure: Signature $(z_1, z_2^\star, c)$

1: $y_1, y_2 \leftarrow_\sigma \mathcal{R}$
2: $u = \xi a_1 y_1 + y_2 \bmod 2q$
3: $c \leftarrow \mathsf{H}(\lfloor u \rceil_M, \mathsf{m})$
4: $b \leftarrow_\$ \{0,1\}$
5: $z_1 \leftarrow y_1 + (-1)^b s_1 c$
6: $z_2 \leftarrow y_2 + (-1)^b s_2 c$
7: Continue with probability $1/\nu$
8: $z_2^\star \leftarrow (\lfloor u \rceil_M - \lfloor u - z_2 \rceil_M \mod p)$
9: **return** $(z_1, z_2^\star, c)$

Algorithm 4.6 Verification of BLISS

Require: Message m, public key $A = (a_1, q-2)$, and signature $(z_1, z_2^\star, c)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $c' \leftarrow \mathsf{H}(\lfloor \xi a_1 z_1 + \xi q c \bmod 2q \rceil_M + z_2^\star \bmod p, \mathsf{m})$
2: **if** $c = c' \wedge \ \|(z_1|2^d z_2^\star)\|_2 \leq B_2 \wedge \|(z_1|2^d z_2^\star)\|_\infty \leq B_\infty$ **then**
3: **return** 0
4: **else**
5: **return** -1

such as LWE or SIS to reduce the number of faults that are necessary to recover the secret. Revealing all coefficients of the secret of a lattice problem with high dimension sometimes requires, depending on the fault attack, a very large amount of injected faults. Instead, it is sufficient to reveal just enough coefficients using fault attacks such that LWE and SIS solvers can be applied to recover the rest of the secret coefficients. We move on to describe our hybrid approach for the LWE problem next.

Let $\mathbf{As} + \mathbf{e} = \mathbf{b} \mod q$ be an LWE instance, with $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, $\mathbf{s} \in \mathbb{Z}_q^n$, and $\mathbf{e} \in \mathbb{Z}_q^m$. Assume that k coefficients of the secret $\mathbf{s}$ are known. Without losing generality, we can assume that the first k coefficients of $\mathbf{s}$ are known, since the samples of an LWE instance can be reordered. Rewriting yields

$$(\mathbf{A_1}|\mathbf{A_2})\left(\mathbf{s_1}|\mathbf{s_2}\right)^T + \mathbf{e} = \mathbf{A_1 s_1} + \mathbf{A_2 s_2} + \mathbf{e} = \mathbf{b},$$

with $\mathbf{A_1} \in \mathbb{Z}_q^{m \times k}$, $\mathbf{A_2} \in \mathbb{Z}_q^{m \times (n-k)}$, and $\mathbf{s_1} \in \mathbb{Z}_q^k, \mathbf{s_2} \in \mathbb{Z}_q^{n-k}$ where $\mathbf{s_1}$ is known. Let $\mathbf{b'} = \mathbf{b} - \mathbf{A_1 s_1}$. Thus, $\mathbf{A_2 s_2} + \mathbf{e} = \mathbf{b'} \mod q$ defines an LWE instance where the dimension of the secret vector is $n - k$.

The following paragraphs now focus on explaining how to find a lower bound on the value of k. First, we choose the number of operations T the LWE solver should compute. For example, we assume that the LWE solver computes $T = 2^{50}$ operations to solve the remaining LWE instance.

To estimate the hardness of LWE, we first compute the BKZ blocksize β such that the run-time of BKZ $T_{BKZ}(\beta)$ is equal to T as follows. As in Section 3.3.1, we estimate $T_{BKZ}(\beta) = 2^{0.27\beta+16.40}$ which corresponds to the quantum-sieving algorithm by Laarhoven [148]. Hence, β can be computed as

$$\beta = \left\lfloor \frac{\log_2(T) - 16.40}{0.27} \right\rfloor = \left\lfloor \frac{50 - 16.40}{0.27} \right\rfloor = \lfloor 124.4 \rfloor = 124.$$

Under the Gaussian heuristic [64] and the geometric series assumption [202], the following correspondence between the block size β and the Hermite-delta δ_0 can be given [63]:

$$\delta_0 \approx \left(\frac{\beta}{2\pi e} (\pi\beta)^{\frac{1}{\beta}} \right)^{\frac{1}{2(\beta-1)}}.$$

Given $\beta = 124$, $\delta_0 = 1.0083$ (rounded to the fourth decimal place).

In the embedding approach, the LWE instance is reduced to an instance of the uSVP [9, 24]. To do so, an embedding lattice Λ of dimension d is defined so that the error vector $\mathbf{e}$ is embedded in Λ. A short vector can be found if [10, 15]

$$\sqrt{\frac{\beta}{d}} \| \mathbf{e} \mid 1 \|_2 \leq \delta_0^{2\beta - d} \cdot \det(\Lambda)^{\frac{1}{d}}. \tag{4.1}$$

Two different ways to define Λ were proposed, namely the standard and dual embedding approach that are described as follows.

In the standard embedding approach, the uSVP solver is applied on the lattice

$$\Lambda = \Lambda_q(\mathbf{A_{st}}) \text{ with } \mathbf{A_{st}} = \begin{pmatrix} \mathbf{A_2} & \mathbf{b'} \\ 0 & 1 \end{pmatrix} \in \mathbb{Z}_q^{(m+1)\times(n-k+1)}.$$

The definition of the q-ary lattice $\Lambda_q(\mathbf{A_{st}})$ is given in Section 2.1. Hence, $\dim(\Lambda_q(\mathbf{A_{st}})) = d = m + 1$ and $\det(\Lambda_q(\mathbf{A_{st}})) = q^{m-n+k}$. Thus, Equation (4.1) gives the following inequality

$$\sqrt{\frac{\beta}{m+1}} \| \mathbf{e} \mid 1 \|_2 \leq \delta_0^{2\beta - m - 1} \cdot q^{\frac{m-n+k}{m+1}}. \tag{4.2}$$

During the dual embedding approach, we apply a uSVP solver on the lattice

$$\Lambda = \Lambda_q^{\perp}(\mathbf{A_D}) \text{ with } \mathbf{A_D} = \left(\mathbf{A_2} | \mathbf{I_m} | \mathbf{b'} \right) \in \mathbb{Z}_q^{m\times(n-k+m+1)}.$$

Hence, $\dim(\Lambda_q^{\perp}(\mathbf{A_D})) = d = n - k + m + 1$ and $\det\Big(\Lambda_q^{\perp}(\mathbf{A_D})\Big) = q^m$. Thus,

$$\sqrt{\frac{\beta}{n-k+m+1}}\|\mathbf{e} \mid 1\|_2 \leq \delta_0^{2\beta-n+k-m-1} \cdot q^{\frac{m}{n-k+m+1}}. \tag{4.3}$$

Finally, given n, m, $\|e\|_2$, $\beta = 124$, and $\delta_0 = 1.0083$, we can compute small values for k such that Equation (4.2) or Equation (4.3) is fulfilled. We emphasize that we are aware that the embedding approach is not always the fastest attack to solve LWE. Nevertheless, it yields an upper bound on the number of fault attacks needed.

Therefore, applying the hybrid approach to BLISS, ring-TESLA, and the GLP scheme shows that it is sufficient to reveal $k = 184$, $k = 217$, and $k = 21$, respectively. We explain the derivation of the values for k for each of the three schemes in the following subsections.

Hybrid Approach Applied to the GLP Scheme. As described in Section 4.2.1.1, the error used in the GLP scheme is ternary. Hence, the expected norm of the error is $\|e\|_2 = \sqrt{2/3 \cdot n}$. Furthermore, only a single LWE tuple is given, i.e., $m = n$. Substituting this in Equation (4.2) for the standard embedding approach, we get the inequality

$$\sqrt{\frac{\beta}{n+1}}\sqrt{2/3 \cdot n + 1} \leq \delta_0^{2\beta-n-1} \cdot q^{\frac{k}{n+1}}. \tag{4.4}$$

Moreover, Equation (4.3) for the dual embedding approach corresponds to

$$\sqrt{\frac{\beta}{2n-k+1}}\sqrt{2/3 \cdot n + 1} \leq \delta_0^{2\beta-2n+k-1} \cdot q^{\frac{n}{2n-k+1}}. \tag{4.5}$$

Finally, the minimal value for k to ensure that the inequality for the standard embedding approach above is fulfilled, is 142. Additionally, the minimal value to ensure that the equation holds for the dual embedding, is 21.

Hybrid Approach Applied to the Signature Scheme ring-TESLA. The hybrid-approach applied to the scheme ring-TESLA is similar to the case of GLP. The only difference is the distribution of the error and the number of given LWE samples. The coefficients of the polynomial e are sampled with Gaussian distribution with standard deviation σ. Hence, the expected length of the corresponding coefficient vector $\mathbf{e}$ of dimension n is given by $\|\mathbf{e}\|_2 = \frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})}\sqrt{2}\sigma$ [58]. If z is a positive integer, the Gamma function is defined as $\Gamma(z) = (n-1)!$; if z is a real number, the Gamma

functions is defined as $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}dt$. The number of given LWE samples is two, i.e., $m = 2n$. Thus, Equation (4.2) for the standard embedding approach gives

$$\sqrt{\frac{\beta}{2n+1}}\frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})}\sqrt{2}\sigma \leq \delta_0^{2\beta-2n-1} \cdot q^{\frac{n+k}{2n+1}}. \tag{4.6}$$

Similarly, Equation (4.3) for the dual embedding approach corresponds to

$$\sqrt{\frac{\beta}{3n-k+1}}\frac{\Gamma(\frac{n+1}{2})}{\Gamma(\frac{n}{2})}\sqrt{2}\sigma \leq \delta_0^{2\beta-3n+k-1} \cdot q^{\frac{2n}{3n-k+1}}. \tag{4.7}$$

Hence, $k = 217$ and $k = 280$ for the standard and the dual embedding approach, respectively.

Hybrid Approach Applied to the Signature Scheme BLISS. The signature scheme BLISS is based on the hardness of SIS. Hence, the hybrid approach applied to BLISS is slightly different than to GLP and ring-TESLA as explained next. First we define the rotation matrix of a vector $\mathbf{a} = (\mathbf{a}_0, ..., \mathbf{a}_{n-1})$: $\mathsf{rot}(\mathbf{a}) = (-\mathbf{a}_{n-1}, \mathbf{a}_0, \ldots, \mathbf{a}_{n-2})$ and $\mathsf{Rot}(\mathbf{a}) = (\mathbf{a}, \mathsf{rot}(\mathbf{a}), \mathsf{rot}^2(\mathbf{a}), \ldots, \mathsf{rot}^{n-1}(\mathbf{a})) \in \mathbb{Z}_q^{n\times n}$. Since we identify a polynomial a with its coefficient vector $\mathbf{a}$, we define $\mathsf{rot}(a) = \mathsf{rot}(\mathbf{a})$. Following the notation in Section 4.2.1.2, let $\mathbf{A} \in \mathbb{Z}_q^{n\times 2n}$ be the matrix defined by the rotation matrices of $2a_q$ and $a_2 = q - 2$, i.e., $\mathbf{A} = (\mathsf{Rot}(2a_q), \mathsf{Rot}(a_2))$. Furthermore, let $\mathbf{s} \in \mathbb{Z}_q^{2n}$ be the coefficient vector of the secret polynomials f and $2g+1$, i.e., $s_0 = f_0, ..., s_{n-1} = f_{n-1}, s_n = 2g_0+1, s_{n+1} = 2g_1, ..., s_{2n-1} = 2g_{n-1}$. Due to the sparse structure of the secret, the expected norm of $\mathbf{s}$ can be approximated by $\|\mathbf{s}\|_2 = \sqrt{5d_1}$.

In the following paragraph, we assume that we revealed k coefficients of $\mathbf{s}$ by fault attacks. We write the equation $\mathbf{As} = 0 \mod q$ as $\mathbf{A_1 s_1} + \mathbf{A_2 s_2} = 0 \mod q$ with $\mathbf{A_1} \in \mathbb{Z}_q^{n\times k}$, $\mathbf{A_2} \in \mathbb{Z}_q^{n\times 2n-k}$, $\mathbf{s_1} \in \mathbb{Z}_q^k$, and $\mathbf{s_2} \in \mathbb{Z}_q^{2n-k}$. This can be written as $\mathbf{A_2 s_2} = -\mathbf{b}$, where $-\mathbf{b} = -\mathbf{A_1 s_1}$. Define the set $L = \{\mathbf{w} \in \mathbb{Z}^{2n-k} \mid \mathbf{A_2 w} = -\mathbf{b} \mod q\} = \Lambda_q^\perp(\mathbf{A_2}) + \mathbf{u}$, where $\mathbf{u} \in \mathbb{Z}^{2n-k}$ such that $\mathbf{A_2 u} = -\mathbf{b}$. Finding $\mathbf{s_2} \in L$ is equivalent to solving CVP: $\mathsf{CVP}(\Lambda_q^\perp(\mathbf{A_2}), \mathbf{u})) = \mathbf{v} \in \Lambda_q^\perp(\mathbf{A_2})$, since the vector $\mathbf{u} - \mathbf{v}$ corresponds to $\mathbf{s_2}$. Instead of solving CVP in the kernel lattice $\Lambda_q^\perp(\mathbf{A_2})$, we can equivalently solve CVP in the image lattice $\Lambda_q\left(\overline{\mathbf{A_2}}\right)$, with $\overline{\mathbf{A_2}} \in \mathbb{Z}_q^{2n-k\times n-k}$ such that $\langle Ker(\mathbf{A_2})\rangle = \langle Im(\overline{\mathbf{A_2}})\rangle$. Hence, $\Lambda_q^\perp(\mathbf{A_2}) = \Lambda_q\left(\overline{\mathbf{A_2}}\right)$. We solve the CVP by embedding the vector $\mathbf{u}$ in the standard embedding lattice. As before, we define

$$\Lambda = \Lambda_q(\mathbf{A_{st}}) = \{\mathbf{w} \in \mathbb{Z}_q^{2n-k+1} \mid \mathbf{A_{st}x} = \mathbf{w} \mod q \text{ for some } \mathbf{x} \in \mathbb{Z}_q^{n-k+1}\},$$

with

$$\mathbf{A_{st}} = \begin{pmatrix} \overline{\mathbf{A_2}} & \mathbf{u} \\ 0 & 1 \end{pmatrix} \in \mathbb{Z}_q^{2n-k+1\times n-k+1}.$$

Hence, $\dim(\Lambda_q(\mathbf{A_{st}})) = d = 2n - k + 1$, and $\det(\Lambda_q(\mathbf{A_{st}})) = q^n$. The secret vector $\mathbf{s_2}$ is a short vector in this lattice. We can compute the norm as $\|\mathbf{s_2}\|_2 \leq \sqrt{5d_1 - 4k}$ and $d_1 = 154$. Substituting these values in Equation (4.1), yields

$$\sqrt{\frac{\beta}{2n-k+1}}\sqrt{770-4k} \leq \delta_0^{2\beta-2n+k-1} \cdot q^{\frac{n}{2n-k+1}}. \tag{4.8}$$

The minimal value for k such that the inequality holds true is $k = 184$. The dual embedding approach, however, is not efficiently applicable to BLISS.

4.2.3 Zeroing Faults

This section elaborates on zeroing fault attacks where it is assumed that the attacker can set the entire variable or a part thereof to zero. Although it has often been questioned if this is a realistic attack scenario, zeroing faults have been realized in practice [163]. In certain cases, zeroing attacks can be realized with skipping attacks as explained in Section 4.2.5.

4.2.3.1 Zeroing the Secret or Error Polynomial During Key Generation

In the following description, we assume that during the key generation of GLP the secret s is set to zero by a zeroing fault. Hence, the value $b = e \mod q$ (instead of $b = as + e \mod q$) is returned and the attacker knows the error polynomial e. In case of the GLP scheme, knowledge of e is enough to forge signatures, which is described as follows.

First, an attacker chooses $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$ and a message m. Afterwards, the attacker computes $c \leftarrow \mathsf{H}([ay_1 + y_2]_M, \mathsf{m})$, $z_2 = y_2 + ec$, and $z_2^\star$ as usual. Next, the attacker defines $z_1 = y_1$ (instead of $z_1 = y_1 + sc$) and returns the signature $\mathsf{s} = (z_1, z_2^\star, c)$. Easy computation shows that the signature would be accepted by the verify algorithm. Similarly, ring-TESLA signatures can also be forged.

The key generation and the sign algorithms in the publicly available software implementation of GLP and ring-TESLA do not test whether the keys are of the correct form, thereby leaving the attacks undetected. The described attack is, however, not applicable to BLISS since the public value $a_q = 0$ if one of the secret polynomials f, g is set to zero. Hence, the attacker gains no additional information.

Zeroing the error polynomial can be implemented as a skipping fault during key generation as explained in Section 4.2.5.1.

4.2.3.2 Zeroing the Randomness During Signature Generation

We describe a zeroing attack against ring-TESLA next. Moving further, we then describe similar attacks on BLISS and the GLP scheme. Since ideal-lattice-based

schemes have been observed to be particularly vulnerable to this attack, we also discuss its applicability to schemes over standard lattices.

Description of the Attack Against ring-TESLA. Let the secret polynomial be $s = \sum_{j=0}^{n-1} s_j x^j$. Moreover, let $(z_1, c_1), ..., (z_m, c_m)$ be m faulty signatures for m messages $\mathsf{m}_1, ..., \mathsf{m}_m$ with $z_i = \sum_{j=0}^{n-1} z_{i,j} x^j$ and $c_i = \sum_{j=0}^{n-1} c_{i,j} x^j$ where $c_{i,j} \in \{-1, 0, 1\}$ and $\|c_i\|_2^2 = h$ for $i = 1, ..., m$. Furthermore, let $y_1, ..., y_m$ be the faulty randomnesses with $y_i = \sum_{j=0}^{n-1} y_{i,j} x^j$. Lastly, let $r \in \{1, ..., n\}$ be the number of coefficients of y_i that are changed to zero via a zeroing fault. We assume that the coefficients are changed block wise, i.e., for some randomness y_i its coefficients $y_{i,j}, ..., y_{i,j+r}$ are changed for some $j \in \{0, ..., n-r-1\}$. However, the attacker cannot control which block of r coefficients is set to zero.

The idea of the attack is as follows: First, the attacker induces a zeroing fault on the randomness and checks which of the coefficients have been changed to zero (we explain later in this section how this is done). The attacker collects equations with $y_{i,j} = 0$, i.e., $(s_j, ..., s_0, -s_{n-1}, ..., -s_{j+1})(c_{i,0}, ..., c_{i,n-1})^T = z_{i,j}$. The attacker repeats these steps until every coefficient of $s_0, ..., s_{n-1}$ is at least once multiplied with a non-zero c_{i_k,j_k} in one of the collected equations. Let $\eta \geq n$ be the number of collected equations. Hence, the attacker receives the following system of equations, which can be solved uniquely:

$$\mathbf{C} \cdot (s_0, ..., s_{n-1})^T = (z_{i_1,j_1}, ..., z_{i_\eta,j_\eta})^T, \tag{4.9}$$

$$\text{with } \mathbf{C} = \begin{pmatrix} c_{i_1,j_1} & ... & c_{i_1,0} & -c_{i_1,n-1} & ... & -c_{i_1,j_1+1} \\ & ... & & & ... & \\ c_{i_\eta,j_\eta} & ... & c_{i_\eta,0} & -c_{i_\eta,n-1} & ... & -c_{i_\eta,j_\eta+1} \end{pmatrix}.$$

Next, we describe how an attacker can find out which coefficients of the randomness were changed to zero during the i-th fault attack. To simplify the explanation, we assume without loss of generality that $c_{i,0} = ... = c_{i,h-1} = 1$ and $c_{i,h} = ... = c_{i,n-1} = 0$. Each coefficient of $z_i = sc_i + y_i$ can be written as

$$\begin{aligned} z_{i,0} &= s_0 - s_{n-1} + ... - s_{n-h} + y_{i,0} \\ z_{i,1} &= s_1 + s_0 - s_{n-2} + ... - s_{n-h+1} + y_{i,1} \\ &\vdots \\ z_{i,n-1} &= s_{n-1} + s_{n-2} + ... + s_{n-h-1} + y_{i,n-1}. \end{aligned} \tag{4.10}$$

We define $z_{i,j} = \varsigma_{i,j} + y_{i,j}$ for $j = 0, ..., n-1$. Since $s_j \leftarrow_\sigma \mathbb{Z}$, the expectation value of $|s_j|$ is given by

$$\mathrm{Ex}[|s_j|] = \sigma\sqrt{\frac{2}{\pi}} \text{ for } j = 0, ..., n-1.$$

Furthermore, since $\|c\|_2^2 = h$, $\varsigma_{i,j}$ is Gaussian distributed with standard deviation $\sqrt{h}\sigma$ and

$$\mathrm{Ex}[|\varsigma_{i,j}|] = \sqrt{\frac{2h}{\pi}}\sigma.$$

Since the coefficients of $y_1, ..., y_m$ are chosen uniformly random in $[-B, B]$, via the triangle inequality it holds that

$$B - \sqrt{\frac{2h}{\pi}}\sigma \leq \mathrm{Ex}[|\varsigma_{i,j} + y_{i,j}|] \leq \sqrt{\frac{2h}{\pi}}\sigma + B.$$

For the parameter set ring-TESLA-II, i.e., with $B = 2^{22} - 1$, $h = 19$, and $\sigma = 52$, $\mathrm{E}[|z_{i,j}|]$ is given by

$$\mathrm{Ex}[|z_{i,j}|] \approx \begin{cases} 102, & \text{if } y_{i,j} = 0, \\ 2^{22}, & \text{if } y_{i,j} \neq 0. \end{cases}$$

Since the difference between the expectation values is large, it is reasonable to assume that the attacker can determine whether $y_{i,j}$ has been changed to zero.

Due to the rejection conditions in line 9 in Algorithm 3.12, the efficiency of the attack depends on when the fault was inserted. If y has been set to zero before $v_1 = a_1 y$ and $v_2 = a_2 y$ (cf. line 2 and 3, Algorithm 3.12) have been computed then the rejection sampling is disabled since the coefficients z, w_1, w_2 are then very small and will pass the `if`-condition in line 9 in Algorithm 3.12. Now we assume that the zeroing attack is induced after the computation of v_1 and v_2 in line 9 but before the computation of z in line 6. The rejection condition on z is thus, disabled but the condition on w_1 and w_2 is the same as in the case without zeroing. For the instantiation ring-TESLA-II with $n = 512$, $d = 23$, and $L_E = 2766$, the fault has to be repeated (expectedly) with a probability of about $(1 - 2L_E/2^d)^{2n} = \frac{1}{2}$.

The number of needed zeroing faults strongly depends on the number of zeroed coefficients r as explained next. Let f be the number of necessary successful fault inductions to reveal the secret and let S be the set of equations which are part of Equation (4.9). We assume that every successful fault induction adds r new equations to the set S, since the hash value c changes for every sign query. Hence, solving the following equation for f gives the number of necessary faults:

$$\frac{1}{3} \cdot \sum_{k=1}^{f} r - \frac{1}{2^n}(k-1) - \frac{1}{2^r} \geq n, \tag{4.11}$$

where the factor 3 corresponds to the rejection probability of 0.34 for the instantiation ring-TESLA-II. Now we assume that the attacker can set 12 bytes to zero. This corresponds to $r = 4$ since each coefficient of y can be saved in three bytes. Then $f = 384$. If $r = 1$ then $f = 1536$. If $r = n$, i.e., $y = 0$, only a single successful fault is necessary, since then the linear system of equations in Equation (4.10) can be solved uniquely with high probability.

Application to BLISS and GLP. The attack described above can also be applied to BLISS and the GLP scheme. We assume that the zeroing fault was induced on y_1 of the GLP scheme. As explained above, we can recover s. Afterwards, we can compute e by $e = b - as$. Because of the compression function the attack would not be effective if y_2 is faulty.

Similarly, we can assume that the zeroing fault was induced on y_1 during the sign algorithm of BLISS. As explained above, we can then recover s_1 and s_2 by since $s_2 = a_1 s_1$. For similar reasons as in case of GLP, the attack does not work effectively if y_2 instead of y_1 is faulty.

Application to Signature Schemes over Standard Lattices. The attack, when applied to schemes defined over standard lattices instead of ideal lattices, is far less efficient and only applicable for $r = n$. Let $r = n$, i.e., the randomness is equal to the zero vector. Moving further, let the notation and assumptions be as described above. The following system of equations then gives $n \cdot m$ equations (with $m > n$) and n^2 unknowns, and can be solved uniquely:

$$(z_1, ..., z_m)^T = \mathbf{S}\,(c_1, ..., c_m)^T\,,$$

where $\mathbf{S}$ is the secret matrix. Still it is less efficient than in the ring setting, since the attacker needs to induce at least $3n$ zeroing faults (again the factor 3 comes from the rejection probability). In the ring setting with the same assumption $r = n$, the attacker would have to induce (on average) three zeroing faults successfully.

In case $r < n$, the attack becomes in general not applicable to signature schemes over standard lattices, since the multiplication of matrices is not commutative–a condition that is needed in Equation (4.9).

We now move on to describe yet another zeroing attack that targets the signature generation.

4.2.3.3 Zeroing the Hash Value During the Signature Generation

Zeroing the hash value c during the sign algorithm does not lead to more information about the secret key since only the product sc occurs in the final signature. Hence, the attacker only gets access to the randomness, which changes for every run of the sign algorithm in BLISS, ring-TESLA, and the GLP scheme.

However, TESLA (as previously described in Section 3), is defined to be deterministic. Zeroing the hash value reveals secret information as described for TESLA in the following. Let $\mathsf{s_m} = (\mathbf{z_m} = \mathbf{S}\mathbf{c_m} + \mathbf{y_m}, \mathbf{c_m})$ be the TESLA signature of m. Querying a signature for m again while inducing a zeroing fault on $\mathbf{c_m}$ yields the faulty signature $\mathsf{s}_0 = (\mathbf{z_0} = \mathbf{y_m}, \mathbf{0})$. Thus, the attacker can compute $\mathbf{z_m} - \mathbf{z_0} = \mathbf{S}\mathbf{c_m}$. To determine $\mathbf{S}$ uniquely, n zeroing faults of this kind have to be

performed. The attack would work in a similar way on a deterministic variant of qTESLA or ring-TESLA using only one faulty and one correct signature. A similar but slightly more general attack was proposed in [179] against ECDSA [183] and EdDSA [41] signatures. It was applied to the signature schemes Dilithium [82] and a deterministic variant of qTESLA by Groot Bruinderink and Pessl [111] who further provide experimental verification of their attack on an ARM Cortex-M4 microcontroller. We discuss the attack on the deterministic variant of qTESLA and our implemented countermeasures in Section 3.4.1.

4.2.3.4 Zeroing Fault During the Verification Algorithm

After describing zeroing attacks targeting the signature generation, we now turn to zeroing attacks on the polynomial computed from the hash value c during verification. We use the pseudo-code description of ring-TESLA as an example. This attack works similarly on GLP and BLISS since it is the same mechanism as used in ring-TESLA, although this is not made explicit in their pseudo-code descriptions.

The goal of the attacker in our attack scenario is to force the verify algorithm to accept a (invalid) signature for a message m. To achieve this, the attacker chooses $z \leftarrow_{\$} \mathcal{R}_{q,[B-U]}$, computes $c' \leftarrow \mathsf{H}([a_1 z]_M, [a_2 z]_M, \mathsf{m})$, and returns (c', z) as the signature of m. During the verify algorithm, first the value $c \leftarrow F(c')$ is computed (see Algorithm 3.13). Assume c was set to zero via a fault attack during the verification. Hence, $w_1 = a_1 z$ and $w_2 = a_2 z$ (instead of $w_i = a_i z - b_i c$), and $c'' \leftarrow \mathsf{H}([a_1 z]_M, [a_2 z]_M, \mathsf{m})$. Thus, $c' = c''$ and the signature is accepted.

4.2.4 Randomizing Faults

In this section we elaborate on randomizing faults. A randomizing fault randomly changes the value of a variable that is processed in the attacked algorithm. This means that the attacker does not know the value of the variable after the attack but might benefit from knowing that it has been changed within a certain range. Depending on the attacker's abilities, the fault targets the entire variable or some parts of it [212]. We now analyze the effects of a randomizing fault targeting the secret polynomial, the error polynomial, the modulus, and the randomness during the signature generation.

4.2.4.1 Randomizing the Secret Polynomial

In 1996, Bao et al. [28] introduced a method to attack signature schemes with binary secret keys. In particular, they have shown how to attack RSA [196], the ElGamal scheme [89], and Schnorr signature schemes [203]. In this section, we describe how

to adjust the attack by Bao et al. to lattice-based Schnorr-like signature schemes. First, we explain our adapted attack against LWE-based schemes instantiated with binary secret over standard lattices. In addition, we also describe more evolved attacks on GLP and BLISS.

Let $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \mod q$ with $\mathbf{s} \in \{0,1\}^n$, $\mathbf{e} \leftarrow \chi$ with χ being some distribution over $\mathbb{Z}^m$, and $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$. The public key is $(\mathbf{A}, \mathbf{b})$ and the secret key consists of $(\mathbf{s}, \mathbf{e})$. The signature of a message m is computed as follows. Choose a random vector $\mathbf{y}$, compute the hash value $\mathbf{c} = \mathsf{H}([\mathbf{A}\mathbf{y}]_M, \mathsf{m})$, compute $\mathbf{z} = \mathbf{s}\mathbf{c} + \mathbf{y}$, and return the signature $\mathsf{s} = (\mathbf{z}, \mathbf{c})$.

Assume one coefficient of $\mathbf{s}$ is changed via a fault attack, i.e., the secret $\mathbf{s}' = (s_0, ..., s_{i-1}, s_i', s_{i+1}, ..., s_{n-1})^T$ is used to generate the faulty signature $\mathsf{s}' = (\mathbf{z}', \mathbf{c}) = (\mathbf{s}'\mathbf{c} + \mathbf{y}, \mathbf{c})$ of m. Now, the attacker tries different values $\alpha \in \{-1, 0, 1\}$ until $\mathsf{H}\left([\mathbf{A}\mathbf{z}' - \mathbf{b}\mathbf{c} - \mathbf{A}\mathbf{v}_{\mathbf{i},\alpha}\mathbf{c}]_M, \mathsf{m}\right) = \mathbf{c}$ holds, where $\mathbf{v}_{\mathbf{i},\alpha}$ is the zero vector except that the i-th entry is equal to α. Depending on the value of α and the index i, the attacker can determine the value of s_i:

$$\text{If } \alpha = \begin{cases} 1 & \text{then } s_i = 1, \\ -1 & \text{then } s_i = 0, \\ 0 & \text{then } s_i = s_i' \text{ and the attack needs to be repeated.} \end{cases}$$

Hence, in case of a successful fault attack, i.e., $s_i \neq s_i'$, an attacker finds out one coefficient of the secret for each injected fault. To our knowledge, there is no lattice-based signature scheme instantiated over binary-LWE. However, there exists a lattice-based encryption scheme with binary secret [55]. Therefore, through the description stated above we find ourselves standing in agreement with those being cautious about instantiations of schemes with binary-LWE.

Applying the Attack to the GLP Scheme. After describing the attack idea using a simplified signature scheme, we now describe a generalization of the attack by Bao et al. [28] to ternary secret keys, i.e., to secret keys with coefficients in $\{-1, 0, 1\}$. We explain our attack using GLP as an example since its secret key is ternary. Furthermore, we assume that the attack changes up to r consecutive coefficients instead of only a single coefficient of the secret. This is generally considered to be a more realistic scenario [106]. Let the faulty secret s' be written as $s' = s + \varepsilon$ with $\varepsilon = \sum_{i=j}^{j+r} \varepsilon_i x^i$, $0 \leq j < n$ where all $\varepsilon_i, s_i' \in \{-1, 0, 1\}$. The attack consists of three steps, namely inducing a randomizing fault, querying a signature on some message, i.e., $\mathsf{s}' = (z_1', z_2^\star, c) = (s'c + y, z_2^\star, c)$ with s' being the faulty secret, and analyzing the output by running a software implementation of the algorithm GeneralBao that is depicted in Figure 4.7. The attacker repeats these three steps until sufficiently many secret coefficients are determined, so that the hybrid approach described in Section 4.2.2 can be applied.

Next we describe the algorithm GeneralBao. It gets as input the public key pk, a signature s of a message m, the list LSecret where the determined coefficients of the secret are saved, and the list LDetermined where the information whether or not a coefficient is already determined is saved. The algorithm GeneralBao then returns updated lists LSecret and LDetermined.

Let α be the difference between the secret s and the faulty secret s', i.e., $\alpha = \sum_{i=0}^{n-1} \alpha_i x^i$ is a polynomial with $\alpha_i \in \{-2, -1, 0, 1, 2\}$. The attacker checks whether $\mathsf{H}([az_1 + z_2^\star - bc - a\alpha c]_M, \mathsf{m}) = c$ with $\alpha_i, ..., \alpha_{i+r} \in \{-2, -1, 0, 1, 2\}$ for $i \in \{0, ..., n-1-r\}$. Thereby, the attacker gains information about the value and index of s_i. The possible values for s_i, s_i', and α_i are shown in Table 4.3.

Table 4.3: Possible combinations for the coefficients of s, s', and α

s_i'	0	0	0	1	1	1	-1	-1	-1
s_i	0	1	-1	0	1	-1	0	1	-1
$\alpha_i = s_i' - s_i$	0	-1	1	1	0	2	-1	-2	0

As indicated by Figure 4.7, the procedure GeneralBao distinguishes between five different cases for each coefficient of α once the correct values of $\alpha_0, ..., \alpha_{n-1}$ are found, namely

$$\text{if } \alpha_i = \begin{cases} 2 & \text{then } s_i = -1, \\ -2 & \text{then } s_i = 1, \\ 1 & \text{then } s_i = 1 \text{ or } s_i = 0, \\ -1 & \text{then } s_i = -1 \text{ or } s_i = 0, \\ 0 & \text{then } s_i = s_i'. \end{cases}$$

Put differently, if $\alpha = \pm 1$ or $\alpha = 0$, the attacker cannot determine s_i uniquely. Let s_j be a coefficient which was changed during a fault attack such that $\alpha_{j,1} = \alpha_j = \pm 1$. We assume that s_j is changed again by another fault attack with difference $\alpha_{j,2}$. Then the attacker can determine s_j uniquely if $\alpha_{j,1} \neq \alpha_{j,2}$ and $\alpha_{j,2} \neq 0$. The list LDetermined is used for exactly this purpose, namely to remember which coefficients were changed but could not be determined uniquely.

As elaborated earlier in Section 4.2.2, at most $k = 21$ coefficients of s have to be determined using fault attacks to recover the entire secret polynomial via the hybrid approach. We analyze the expected number of faults that are necessary to determine $k = 21$ coefficients of s next. We assume that the index of the first of the r changed coefficients is chosen uniformly random in $\{0, ..., n-1\}$. Since r is much smaller than n, we assume furthermore that the changed (and hence the determined) coefficients are uniformly distributed over all coefficients $s_0, ..., s_{n-1}$. We assume that the j-th fault attack is induced after i_j coefficients have already

been determined uniquely. Following Table 4.3, the probability that a coefficient is changed such that it can be determined uniquely is 2/9. Then the number of newly determined coefficients after the j-th fault attack is given by $\frac{2}{9}r\frac{n-i_{j-1}}{n}$ where n is the number of secret coefficients. Moreover, we assume that the fault attack targets one byte. Since each coefficient can be saved in two bits, changing one byte corresponds to changing four coefficients, i.e., $r = 4$. Let f be the number of (expected) needed faults to determine $k = 21$ coefficients of the secret s. Solving the following equation for f gives $f = 24$ (expected) needed faults:

$$\sum_{j=1}^{f} \frac{2r}{9} \cdot \frac{512 - i_{j-1}}{512} \geq k,$$

with $i_0 = 0$ and $n = 512$. If $r = 1$ then $f = 96$ faults are necessary to recover the secret. A larger value for r is often more realistic and results in a smaller number of necessary faults. On the other hand the time to find the correct polynomial α takes much longer, since the hash function has to be queried $512 \cdot 5^r$ times for every fault.

Application to BLISS. As described earlier in Section 4.2.1.2, the secret keys of the instantiations BLISS-I and BLISS-II consist of two polynomials s_1, s_2 with sparse instantiation. Therefore, the attack on these instantiations of BLISS can be applied similarly to the attack on GLP. For the higher-security instantiations BLISS-III and BLISS-IV, $d_2 \in \{0.03, 0.06\}$, i.e., some coefficients are not in the set $\{-1, 0, 1\}$. Hence, extending our attack to these instantiations is possible but increases the run-time of the attack.

As in the description above, we assume r coefficients are changed during a successful fault. We also assume that the coefficients of the faulty secret polynomial(s) are in the set $\{-3, ..., 3\}$, since it can be assumed that each coefficient is saved in three bits. Let $\alpha = \sum_{i=0}^{n-1} \alpha_i x^i$. Given a faulty signature $\mathsf{s}' = (z_1, z_2^\star, c)$ of the message m, the attacker runs an algorithm similar to GeneralBao. The only difference is that the values of α lie in different intervals. If the fault has been induced on s_1 then the attacker checks $\mathsf{H}(\lfloor \xi a_1 z_1 + \xi q c - \xi \alpha c \rceil_M + z_2^\star, \mathsf{m}) = c$ for $\alpha_{i_1}, ..., \alpha_{i_r} \in \{-4, ..., 4\}$ for $i_1, ..., i_r \in \{0, ..., n-1\}$. If the fault has been induced on s_2 then the attacker checks $\mathsf{H}(\lfloor \xi a_1 z_1 + \xi q c - \alpha c \rceil_M + z_2^\star, \mathsf{m}) = c$ for $\alpha_{i_1}, ..., \alpha_{i_r} \in \{-5, ..., 5\}$ for $i_1, ..., i_r \in \{0, ..., n-1\}$[12]. As in the GLP attack described above, in some cases the secret coefficients can be determined uniquely. The probability that a coefficient of s_1 (resp., s_2) is determined uniquely is 2/21 (resp., 4/21).

[12]To be exact, if the fault was induced on s_2 then the attacker checks the hash values for $\alpha_0 \in \{-4, ..., 6\}$ and $\alpha_1, ..., \alpha_{n-1} \in \{-5, ..., 5\}$.

Algorithm 4.7 Algorithm GeneralBao to compute coefficients of the secret polynomial given a GLP signature computed with a faulty secret where at most r coefficients are changed by a randomizing fault

Require: $\mathsf{s} = (z_1, z_2^\star, c), \mathsf{m}, \mathsf{pk} = (a, b)$, list LDetermined, list LSecret; signature s is computed with a faulty secret

Ensure: LDetermined, LSecret

```
1: for i ∈ {0, ..., n − 1} do
2:     for α_i ∈ {0, −2, −1, 1, 2} do
3:         for α_{i+1} ∈ {0, −2, −1, 1, 2} do
4:             ...
5:             for For α_{i+r} ∈ {0, −2, −1, 1, 2} do
6:                 α_{i+r+1}, ..., α_{n−1} = 0
7:                 α = Σ_{i=0}^{n−1} α_i x^i
8:                 if (H([az_1 + z_2^⋆ − bc − aαc]_M, m) = c) then
9:                     for j ∈ {i, ..., i + r} do
10:                        if (α_j = 2) then
11:                            LSecret[j] = −1, LDetermined[j] = 2
12:                        if (α_j = −2) then
13:                            LSecret[j] = 1, LDetermined[j] = 2
14:                        if (α_j = −1) then
15:                            if (LDetermined[j] = 1) then
16:                                LSecret[j] = 0, LDetermined[j] = 2
17:                            else
18:                                LDetermined[j] = −1
19:                        if (α_j = 1) then
20:                            if (LDetermined[j] = −1) then
21:                                LSecret[j] = 0, LDetermined[j] = 2
22:                            else
23:                                LDetermined[j] = 1
24: return LDetermined, LSecret
```

To compute the number of successful faults needed, we make the same assumptions as before, i.e., we assume that the first index i_1 of the changed coefficients is distributed uniformly random over $\{0, ..., n-1\}$. Hence, we approximate the expected number of needed faults f by

$$\sum_{j=1}^{f} \frac{r}{7} \cdot \frac{n - i_{j-1}}{n} \geq k,$$

with $i_0 = 0$ and $n = 1024$. Hence, for $r = 1$ the expected number of faults to determine $k = 184$ coefficients of the secret (and hence, the entire secret using the hybrid approach described in Section 4.2.2) is $f = 1419$. For $r = 4$, $f = 354$ faults are necessary. We thus conclude, that the attack is less efficient on BLISS than on the GLP scheme, but it is still applicable.

Application to ring-TESLA. The coefficients of the secret $s = \sum_{i=0}^{n-1} s_i x^i$ of ring-TESLA are chosen Gaussian distributed with standard deviation σ (with $\sigma = 52$ for instantiation ring-TESLA-I). Hence, the possible values of α are in a large range. Even if we assume that $|s_i| \leq \sigma$ with high probability, the number of successful faults needed would be very large. Hence, this attack does not seem to be a threat for ring-TESLA in particular and for instantiations with Gaussian distribution in general.

4.2.4.2 Randomizing the Error Polynomial

An attack similar to the one described in Section 4.2.4.1, can be used to compute the (secret) error polynomial e. If e is recovered and the public key $(a, b = as + e \mod q)$ is known, s can be recovered as well. However, GLP is not vulnerable to this variant because of its compression and rounding functions. If the error is randomized, the equation $\mathsf{H}([az_1 + z_2^\star - bc - a\alpha x^i c]_M, \mathsf{m}) = c$ holds for several values of α. Hence, α cannot be determined uniquely. Additionally, the BLISS scheme is also not vulnerable to this attack since it is not based on LWE and hence, no error polynomial occurs.

Nevertheless, we would like to mention this attack to raise awareness during the construction and instantiation of schemes. To exemplify, instantiating ring-TESLA (which does not come with such a compression function) over ternary-LWE should be implemented very carefully.

4.2.4.3 Randomizing the Modulus

The randomizing the modulus does not seem to reveal any information that would lead to a forgery because the value of the faulty modulus would remain unknown. Furthermore, key and signature generation of lattice-based signature schemes are randomized at several points by construction. Hence, tools like the Chinese remainder theorem do not give access to the secret.

4.2.4.4 Randomizing the Randomness

As expected, also randomizing random values, e.g., the product ay or the hash output c, does not reveal information that lead to signature forgeries since such

values look (or are) random by default. An exception are randomizing faults on the hash output c for deterministic signatures which are described in Section 4.2.3.3.

4.2.5 Skipping Faults

In this section, we turn to skipping faults that aim at skipping selected lines of the program code. This can for example, be achieved via CPU clock glitching [43]. Next, we analyze different ways to exploit skipping faults during the key generation, the signature generation, and the verification algorithm.

4.2.5.1 Skipping During the Key Generation

In the following paragraphs, we elaborate on two skipping attacks targeting the key generation.

Skipping the Modulus Reduction. Let $\mathbf{A} \leftarrow_{\$} \mathbb{Z}_q^{n \times m}$, and let $\mathbf{s}, \mathbf{e}$ be chosen with some distribution over $\mathbb{Z}^n$ and $\mathbb{Z}^m$, respectively. Afterwards compute $\mathbf{b} = \mathbf{As} + \mathbf{e}$ without reducing modulo q. Solving SVP or CVP in the lattice $\Lambda = \{\mathbf{v} \in \mathbb{Z}^n \mid \mathbf{A^T w} = \mathbf{v} \text{ for some } \mathbf{w} \in \mathbb{Z}^m\}$ is often computationally easier than solving the same instance in $\Lambda_q(\mathbf{A}) = \{\mathbf{v} \in \mathbb{Z}^n \mid \mathbf{A^T w} = \mathbf{v} \mod q \text{ for some } \mathbf{w} \in \mathbb{Z}^m\}$, since $\det(\Lambda) \geq \det(\Lambda_q(\mathbf{A}))$. Hence, skipping the modulo operation during the key generation could potentially introduce a vulnerability. However, this fault attack is already prevented in the three considered signature schemes. We explain how the implementations are protected, using GLP as an example, next.

As indicated by Listing 4.19, the value `t` (corresponding to b in our notation) is computed without the reduction step. The modulo operation is performed in the subroutine `poly_pack`. Skipping line 11 of Listing 4.19 thwarts the modulo reduction. Afterwards, only the least significant 32 bits of (the faulty) `t` are saved in `r`. Hence, skipping line 11 leads to a randomizing fault on b which does not reveal secret information.

Skipping the Addition. In this attack we skip additions to compute a malformed public key. We first explain the attack using examples from the C implementation of the GLP scheme. However, the attack can also be successfully applied to ring-TESLA and BLISS as explained afterwards.

In the implementation of GLP, the public key is computed as follows (see Listing 4.19): First a and `s1` (corresponding to s in our notation) are multiplied and saved in the value `t` (corresponding to b in our notation). Afterwards, the error `s2` (corresponding to e in our notation) is added to `t`. Hence, skipping the second operation yields $b = as$ and an attacker can easily recover s by Gaussian reduction. It is important to note here that skipping line 1 in Listing 4.19 results in

```
1  poly_mul_a(t, s1);
2  poly_add_nored(t, t, s2);
3  poly_pack(pk, t);
4  [...]
5
6  void poly_pack(unsigned char r[3*POLY_DEG], const poly f){
7   int i;
8   signed long long t;
9   for(i=0;i<POLY_DEG;i++){
10   t = (unsigned long long)f[i];
11   t = ((t % PARAM_P) + PARAM_P) % PARAM_P;
12   r[3*i+0] = t       & 0xff;
13   r[3*i+1] = (t >>  8) & 0xff;
14   r[3*i+2] = (t >> 16) & 0xff;
15  }
16 }
```

Listing 4.19: Code of the GLP scheme for the computation of the public value $b = as + e \mod q$ in the subroutine `crypto_sign_keypair` and compression in the subroutine `poly_pack` in C; the value `t` corresponds to the value b, the value `s1` corresponds to s, and `s2` corresponds to e in our notation

an unallocated variable `t`, triggering a segmentation fault. Thus, no (predictable) information which could potentially be used by an attacker, is returned.

As indicated by the disassembled code of the addition in Listing 4.20, the command `poly_add_nored@PLT` is called in line 5. Hence, skipping this line results in $b = as$ as described above. This implies that although the attacker can compute s,

```
1                .loc 1 49 0
2  1254 4C89E2   movq   %r12, %rdx
3  1257 4C89F6   movq   %r14, %rsi
4  125a 4C89F7   movq   %r14, %rdi
5  25d E8000000  call   poly_add_noredPLT00
```

Listing 4.20: Assembly code corresponding to line 2 in Listing 4.19 of the GLP implementation

the error vector e remains unknown. However, in the signature generation of GLP, e is used to compute z_2. We illustrate now how an attacker can forge signatures anyway: The attacker chooses random polynomials y_1, y_2 and computes the hash value c for a message `m`, $z_1 = y_1 + sc$, and $z_2 = y_2$ (instead of $z_2 = y_2 + ec$). The attacker applies rejection sampling, compresses z_2 to $z_2^\star$, and returns the signature $\mathsf{s} = (z_1, z_2^\star, c)$. The verify algorithm accepts this signature `s` as explained next. Due to the rejection sampling, $z_1, z_2^\star \in \mathcal{R}_{q,[k-32]}$. Furthermore, it holds that $[az_1 + z_2^\star - bc]_M = [az_1 + z_2 - bc]_M = [asc + ay_1 + y_2 - asc]_M = [ay_1 + y_2]_M$. Thus, by skipping a single line an attacker can reveal s and also forge a signature

for any message m.

Listing 4.21 shows the implementation of ring-TESLA's public key computation b_1, b_2. Similar to the description of the GLP scheme, skipping line 2 or 4 yields $b_i = a_i s$. Hence, we can easily compute s. By definition of the signature generation of ring-TESLA, this is enough to forge signatures. The error polynomials e_1 and e_2 are not necessary during the sign algorithms since line 5 and 6 in Algorithm 3.12 can be substituted by testing whether the equation $[a_i z - b_i c]_M = [a_i y]_M$ holds for $i = 1, 2$.

```
1 poly_mul_fixed(poly_T1,poly_S,poly_a1);
2 poly_add(poly_T1,poly_T1,poly_E1);
3 poly_mul_fixed(poly_T2,poly_S,poly_a2);
4 poly_add(poly_T2,poly_T2,poly_E2);
```

Listing 4.21: Code of the signature scheme ring-TESLA for the computation of the public values b_1 and b_2 in C; the values `poly_T1` and `poly_T2` correspond to b_1 and b_2, respectively

The public key of BLISS consists of a polynomial $a_1 = 2a_q = 2(2g + 1)/f$, where f, g are polynomials with small coefficients. In the implementation of BLISS the public key is computed via the operations depicted in Listing 4.22. Skipping line 7 in Listing 4.22 yields $a_q = 1/f$. Since $a_1 = 2a_q$ is part of the public key, f can be recovered easily. Similarly to GLP, skipping an operation in the key generation does not reveal the polynomial g or $2g+1$. Given f, however, an attacker can forge signatures for any message m as explained next. A general secret-public-key pair of BLISS fulfills the equation $a_1 s_1 + (q-1)s_2 = q \mod 2q$ for $s_1 = f$ and $s_2 = 2g+1$. This can be written as $\frac{2s_2 s_1}{s_1} + (q-1)s_2 = q \mod 2q$. Hence, $s_2 = \frac{q}{q+1} \mod 2q$. As long as it is not checked whether s_2 is of the correct form during signing, signatures computed with the secret key $(f, q/(q-1) \mod 2q)$ will be verified with the public key $(2/f, q-2)$.

```
1 conv(pX, sk.s1);}        \\ pX=f
2 NTL::conv(aq, pX);}      \\ aq=f
3 NTL::inv(aq, aq);}       \\ aq=1/f
4 NTL::ZZ\_pE tmp;}
5 conv(pX, -(sk.s2));}     \\ pX=-(2g+1)
6 NTL::conv(tmp, pX);}     \\ tmp=-(2g+1)
7 NTL::mul(aq, aq, tmp);} \\ aq=-(2g+1)/f
```

Listing 4.22: Code of the signature scheme BLISS for the computation of the public value a_1 in C++

4.2.5.2 Skipping During the Signature Generation

The signature generation of BLISS, GLP, and ring-TESLA can be implemented as a short and simple sequence of (polynomial) additions and multiplications. Furthermore, signing is randomized in several different operations. Hence, the number of vulnerabilities introduced by skipping faults is low. We describe two skipping attacks during the sign algorithm in the following paragraphs.

Skipping the Rejection Condition. Lyubashevsky first applied rejection sampling (introduced by von Neumann [214]) to lattice-based signature schemes to assure the statistical independence of the secret key and the signatures generated with this secret. As a result, learning-the-parallelepiped-attacks introduced by Nguyen and Regev [166] and improved by Ducas and Nguyen [84] further, are prevented. Ducas and Nguyen have needed about 8000 signatures to reveal the secret in their attack on NTRUSign [121]. In case of BLISS, ring-TESLA, and the GLP scheme, the rejection sampling is implemented as an `if`-condition, which would have to be skipped in order to circumvent the rejection sampling. Skipping this condition for executions of the signature generation might introduce the same security flaw as used by the attacks described in [84, 166]. Since these attacks exploit the special structure of NTRU-lattices, BLISS might be especially vulnerable because its keys are chosen in an NTRU-like manner. However, to find out the exact number of needed faults, the mentioned attacks have to be adapted to BLISS, ring-TESLA, and the GLP scheme which we leave for future inquisition.

Skipping the Addition of the Randomness. In the implementation of the GLP scheme, the values z_1 and z_2 are computed in two steps (see Algorithm 4.2): First s (resp., e) and c are multiplied. Afterwards, sc and y_1 (resp., ec and y_2) are added, as can be seen in Listing 4.23. Hence, skipping line 5 (resp., line 9) in Listing 4.23 yields $z_1 = sc$ (resp., $z_2 = ec$). As previously stated, the assembly code

```
1  poly_setrandom_maxk(y1);
2  poly_setrandom_maxk(y2);
3  [...]
4  poly_mul(z1,c,s1);
5  poly_add_nored(z1,z1,y1);
6  poly_coeffreduce(z1);
7  [...]
8  poly_mul(z2,c,s2);
9  poly_add_nored(z2,z2,y2);
10 poly_coeffreduce(z2);
```

Listing 4.23: Code of the GLP implementation for the computation of the signature values z_1 and z_2 in C

of the respective code lines corresponds to jumping to another operation. Hence, this attack gives the same result as zeroing the entire randomness as described in Section 4.2.3.2. This means that an attacker can recover the secret key by skipping a single line.

A similar attack is not possible in case of the implementations of ring-TESLA and BLISS as we explain next. In the implementation of ring-TESLA the value sc is added to the value y as can be seen in Listing 4.24. Hence, skipping the line in Listing 4.24 yields $z = y$ (the value `vec_y` is the output value in the implementation). Since the randomness changes for every run of the sign algorithm, the attacker does not learn additional information about the secret.

```
1 poly_add(vec_y, vec_y, Sc);
```

Listing 4.24: Code of ring-TESLA for the addition of sc and y in C

Similarly the attack does not recover additional secret information when applied to BLISS. As Listing 4.25 indicates, the values of y_1 and y_2 are written in z_1 and z_2, respectively. Hence, skipping the addition in lines 11 until 19 of Listing 4.25 yields $z_1 = y_1$ and $z_2 = y_2$. Since the randomness y_1, y_2 changes for every run of the sign algorithm, this does not reveal additional secret information. Skipping lines 2 until 4 yields z_1, z_2 with unknown values to the attacker.

```
1  for (i=0; i<N; i++){
2   signOutput.z1[i] = sampler->SamplerGaussian();
3   ay[i] = signOutput.z1[i]*W[i];
4   signOutput.z2[i] = sampler->SamplerGaussian();
5  }
6  [...]
7  mult_by_c(sc1, sk.ls1, false, sk.offset, signOutput.indicesC);
8  mult_by_c(sc2, sk.ls2, true, sk.offset, signOutput.indicesC);
9  [...]
10 if (random->getRandomBit()){
11  for (i=0; i<N; i++){
12   signOutput.z1[i]-=sc1[i];
13   signOutput.z2[i]-=sc2[i];
14  }
15 } else{
16  for (i=0; i<N; i++){
17   signOutput.z1[i]+=sc1[i];
18   signOutput.z2[i]+=sc2[i];
19  }
20 }
```

Listing 4.25: Code of BLISS for the computation of the signature values z_1 and z_2 (without compression) in C++

Skipping the Modulus Reduction. Skipping the reduction modulo q during the signature generation does not reveal information about the secret key since during the computation of $z = y + sc$ (the only value that is returned and depending on the secret) no modulo reduction is computed in all of the signature schemes. Moreover, the modulo operation is computed very often during the sign algorithm. As a consequence, it is rather difficult to skip all the modulo operations during the computation using fault attacks.

4.2.5.3 Skipping During the Verification

To prevent the installation of malicious malware, cryptographic signatures of software updates are used. In addition, it is also necessary to ensure that the verification of these signatures is computed correctly [204]. As a result of this observation, we also analyze skipping attacks during the verification algorithm. In what follows, we identify two possibilities to force acceptance of invalid signatures for messages m via skipping attacks.

In the signature schemes BLISS, GLP, and ring-TESLA, the verify algorithm consists essentially of computing a hash value c', checking whether c' is the same as the input value c (called the correctness check), and checking whether z (resp., z_1 and $z_2^\star$) is small enough (called the size check). It is important to note here that we do not consider skipping the computation of the encoding function of the hash value c, since this would lead to an unallocated value. However, we already consider zeroing c in Section 4.2.3.4.

Skipping the Correctness Check. We first describe how to force acceptance of an invalid signature via skipping the correctness check. An adversary chooses c uniformly at random and chooses z (reps., z_1 and $z_2^\star$) small enough and of the expected form (e.g., correct number of zero-coefficients), such that the size check of the verification accepts. Afterwards, the attacker computes the hash value c'. Hence, skipping the correctness check yields an acceptance of the (invalid) signature of any message.

In the software of ring-TESLA, the correctness check is realized as a simple `if`-condition shown in Listing 4.26, where `c_sig` corresponds to c' in our notation and returning `-1` corresponds to the rejection of a signature.

```
1 if(memcmp(c,c_sig,32)) return -1;
```

Listing 4.26: Code of ring-TESLA of the correctness check during verify in C

In case of BLISS and the GLP scheme, the correctness checks are realized as `if`-conditions for each entry of c. We depict the respective lines of the GLP implementation in Listing 4.27 as an example. Therefore, the skipping fault could be realized as an early loop-abort during the `for`-loop after the first iteration.

Hence, the invalid signature is accepted as long as c, z_1, and $z_2^\star$ are chosen such that $c[0] = c'[0]$.

```
1 for(i=0;i<20;i++){
2  if(sm[i] != h[i]){
3   goto fail;
4  }
5 }
```

Listing 4.27: Code of the correctness check in the GLP scheme in C; `sm[0],...,sm[19]` corresponds to c and `h` corresponds to c' in our notation; `goto fail` corresponds to rejecting a signature

Skipping the Size Check. First, we explain the attack against GLP. Moving further, we then explain the attack against BLISS and discuss why it cannot be applied to ring-TESLA.

Against GLP, the attack works as follows. The attacker chooses $y_1, y_2 \leftarrow_\$ \mathcal{R}_{q,[k]}$ and computes the hash value c for a chosen message. Afterwards, the attacker computes $z_1 = a^{-1}(ay_1 + bc)$ (recall that the polynomial a is invertible) and $z_2 = y_2$. Easy computation shows that as long as the size check is skipped, the signature $\mathsf{s} = (z_1, z_2^\star, c)$ is accepted. In case of GLP, the size check is realized as a simple `if`-condition. This is also the case for BLISS, but by construction of the verify algorithm, two `if`-conditions have to be checked for z_1 as indicated by Algorithm 4.6.

The attack, however, does not work for ring-TESLA for the following reason: To accept the signature (z, c) the equation $c = \mathsf{H}\left([w_1']_M, [w_2']_M, \mathsf{m}\right)$ has to hold true. Therefore $[w_i']_M = [a_i z - b_i c]_M = [a_i y]_M$ has to hold for $i = 1, 2$, i.e., the signature value z has to fulfill two equations during verification. As described for GLP, the value z would be uniquely determined by either a_1 or a_2. Hence, the probability that z would fulfill both equations is very small.

After describing all our found attacks, we now turn to propose countermeasures.

4.2.6 Countermeasures

In what follows, we describe countermeasures to prevent fault attacks for each of the successful attacks described earlier in Section 4.2.3, 4.2.4, and 4.2.5 against BLISS, ring-TESLA, and the GLP scheme. We give an overview about the success of the analyzed attacks and their respective abbreviations in Table 4.2. It is crucial that a countermeasure cannot be easily circumvented by another fault attack. Hence, implementations of countermeasures should always consider preventions against

all three kinds of attacks. Towards the end of the section, we demonstrate how to implement countermeasures for ring-TESLA as an example.

4.2.6.1 Countermeasures Against Randomizing Faults

In the following paragraphs, we propose countermeasures against randomizing the secret polynomial during the signature generation, called R-S-Sec and described in Section 4.2.4.1.

A simple countermeasure is to check the correctness of the secret key during signing, e.g., by simple correctness checks or comparisons. However, depending on the fault attack and the implementation of the countermeasure this might not mitigate the fault attack. For example, if the randomizing fault is implemented as a skipping attack, a naive implementation of a comparison might be skipped as well.

Our approach is somewhat different: Let a^{-1} be the inverse polynomial of a in $\mathcal{R}_q$, s' be the faulty secret, s be the original secret, $b' = as' + e \mod q$, and $b = as + e \mod q$. In line 3 of Algorithm 4.2 of the GLP signature generation, $z_1 \leftarrow y_1 + s'c$ is computed if the randomizing attack was successful. We thus, propose to change line 3 to $z_1 = a^{-1}(b' - b)c + s'c + y_1$. Since it holds that

$$\begin{aligned} z_1 &= a^{-1}(b' - b)c + s'c + y_1 \\ &= a^{-1}(as' + e - as - e)c + s'c + y_1 \\ &= sc + y_1, \end{aligned}$$

we always return a signature generated with the correct secret key even if the randomizing attack R-S-Sec successfully changed the secret. As long as our adaption is implemented with respect to the guidelines mentioned below, the countermeasure should not induce vulnerabilities against other described skipping or zeroing attacks.

Conversely, a disadvantage of our countermeasure is that the public key b has to be given as input to the sign algorithm. Hence, the input sizes are increased. Furthermore, the inverse of a has to be computed during signing or has to be given as an additional input or constant.

Our analysis in Section 4.2.4.1 indicates that instantiations such as DCK or NTRU are more vulnerable to this randomizing attack. Hence, instantiating BLISS or the GLP scheme over R-LWE or R-SIS would strengthen the security of these schemes against the randomizing attacks as well. It is highly probable that this would greatly decrease the efficiency.

The error term can be protected against the corresponding randomizing attack R-S-Err, described in Section 4.2.4.2, using a similar approach, if necessary.

4.2.6.2 Countermeasures Against Skipping Faults

We now illustrate the countermeasures to prevent the skipping attacks presented in Section 4.2.5.

Skipping the Addition of Secret Polynomials During Key Generation. We start with a countermeasure to prevent skipping additions, called S-KG-Add, to compute malformed public keys.

One possible countermeasure is to define a new variable that is used to store the resulting sum. Listing 4.2.1a shows the original addition of the GLP key generation, our adaption to mitigate the attack is given in Listing 4.2.1b. Skipping line 2 in Listing 4.2.1b does not lead to a successful attack since the value `b2` would not be allocated, a segmentation fault would be triggered, and hence, no additional information about the secret is revealed in case of first-order attacks. If an even stronger attacker model is considered where the attacker can skip instructions and is able to read the memory after execution, the value `b1` should be discarded or overwritten as a precaution.

```
1 poly_mul_a(b,s);
2 poly_add_nored(b,b,e);
```

a: Original code

```
1 poly_mul_a(b1,s);
2 poly_add_nored(b2,b1,e);
```

b: Adapted code

Listing 4.2.1: Code of the addition during the key generation of GLP in C

Besides the countermeasure mentioned above, it should be ensured that only correctly formed or random-looking keys are returned. Moving further, we now describe a more general but slower method to prevent returning a malformed public key b such as $b = as \mod q$, $b = e \mod q$, or $b = 0$ for GLP as an example in Figure 4.8. The implementations of BLISS and ring-TESLA can be adapted similarly.

If b is not faulty then $\nu = 1$ and the correct elements s and e are returned. If b is faulty the secret and the public key do not correspond to each other. Hence, signed messages cannot be verified with the corresponding faulty b, but at least no signatures can be forged either.

Skipping the Addition of Randomness During Signature Generation. We now propose a countermeasure to prevent skipping additions during the signature generation to compute malformed signatures, called S-S-Add. One approach to prevent such skipping attacks is to add secret information to random information and not the other way around. Listing 4.2.2 shows the implementation of GLP as an example. Skipping line 4 in Listing 4.2.2a (original code) results in $z_1 = as$ and gives access to s. Skipping line 4 in Listing 4.2.2b (adapted code), however, results

Algorithm 4.8 Adapted key generation of the GLP scheme

Require: -
Ensure: Secret key $\mathsf{sk} = (s, e)$ and public key $\mathsf{pk} = (a, b)$

1: $s, e \leftarrow_{\$} \mathcal{R}_{q,[1]}$
2: $a \leftarrow_{\$} \mathcal{R}_q$
3: $b \leftarrow as + e \mod q$
4: $u \leftarrow_{\$} \mathbb{Z}_q$
5: $\nu \leftarrow \frac{\|b-as\|_2+u}{\|e\|_2+u}$
6: **if** $s = \nu s \wedge e = \nu e$ **then**
7: **return** $sk = (\nu s, \nu e)$
8: **else**
9: restart in line 1

in $z_1 = y_1$. Since y_1 changes for every signature generation, the attacker does not gain additional information about the secret. This is already realized in the implementations of BLISS and ring-TESLA. A similar countermeasure is proposed above against a skipping attack during key generation. As above, it is important to note that our proposed countermeasure should prevent first-order skipping attacks. If an even stronger attacker model is considered, however, the value `v1` should be discarded or overwritten as a precaution. Otherwise the attacker might be able to read the secret value `v1` from the memory after execution.

```
1 poly_setrandom_maxk(y1);
2 [...]
3 poly_mul(z1,c,s1);
4 poly_add_nored(z1,z1,y1);
5 poly_coeffreduce(z1);
```

a: Original code

```
1 poly_setrandom_maxk(z1);
2 [...]
3 poly_mul(v1,c,s1);
4 poly_add_nored(z1,z1,v1);
5 poly_coeffreduce(z1);
```

b: Adapted code

Listing 4.2.2: Code of the addition of randomness and secret during the signature generation of GLP in C

Skipping the Rejection Sampling During Signature Generation. Due to the large number of successful faults needed, skipping the rejection sampling for each signature generation does not seem to be a realistic attack scenario (see the attack description of S-S-Rej). Nevertheless, we now discuss a countermeasure that increases the number of successful faults needed further in case of BLISS.

The rejection sampling is realized as an `if`-condition in ring-TESLA and GLP. The signature is returned if the `if`-condition holds true. In the disassembled code this corresponds to returning the signature if the zero flag is equal to 1. This means

that when the `if`-condition is skipped by fault, a signature is returned if and only if the zero flag was set equal to 1 in an earlier computation. It is reasonable to assume that this happens with probability 0.5. Hence, a realization of the `if`-condition such as in ring-TESLA and the GLP scheme does not prevent the skipping attack in all cases, but it increases the (expected) number of necessary fault injections by the number of two. In case of the BLISS implementation, the rejection sampling is realized as `if`-condition(s) such that a signature is rejected if the `if`-condition holds true. Hence, skipping this `if`-condition means to skip the rejection sampling. Reformulating the `if`-condition as it is done for ring-TESLA and the GLP scheme makes this skipping attack more complicated in case of BLISS.

Skipping the Correctness or the Size Check During Verification. To prevent against attacks that skip the correctness or the size check during verification such as S-V-Cor or S-V-Size described in Section 4.2.5.3, we combine several countermeasures. The adapted pseudo-code of the verification is displayed for the GLP scheme in Figure 4.9 as an example. The implementations of BLISS and ring-TESLA can be adapted in a similar way.

The idea of the countermeasure is to return the result of the size and correctness check directly without depending on `if`-conditions. For this, we introduce the subroutine `CheckVerify` that returns 0 if the signature values have the correct size and the computed hash value is the same as the input value (see line 2 in Algorithm 4.3). Otherwise `CheckVerify` returns 1.

To hamper that our countermeasures are circumvented by using higher-order skipping faults, we shuffle the order of the coefficients of $z_1, z_2^\star, c$ using the routine shuffle().

Algorithm 4.9 Adapted verification of the GLP scheme

Require: Message m, public key (a, b), and signature $(z_1, z_2^\star, c)$
Ensure: $\{0, -1\}$ ▷ accept, reject signature

1: $c' \leftarrow H\left(\left[az_1 + z_2^\star - bc\right]_M, \mathsf{m}\right)$
2: $c'_{sh} \leftarrow \mathsf{shuffle}(c')$
3: $z'_{1,sh} \leftarrow \mathsf{shuffle}(z_1)$
4: $z'_{2,sh} \leftarrow \mathsf{shuffle}(z_2^\star)$
5: **return** $-\mathsf{CheckVerify}(z'_{1,sh}, z'_{2,sh}, c'_{sh})$

We also use randomization in the routine CheckVerify() displayed in Algorithm 4.10. Namely, we randomize the order in which the size and correctness are checked for each of the coefficients. Moreover, we introduce two lists result_z and result_c with n entries equal to 1. The lists are then used to save the respective results of the size and the correctness checks for each coefficient. For example,

Algorithm 4.10 Pseudo-code of CheckVerify()

Require: Polynomials z_1, z_2, and c
Ensure: $\{0, 1\}$

1: $r \leftarrow_{\$} \{0,1\}^n$
2: $\text{result}_z = [1]^n$
3: $\text{result}_c = [1]^n$
4: **for** $i = 0, ..., n-1$ **do**
5: **if** r=0 **then**
6: $\text{result}_z[i] = (\mathsf{CheckSize}(z_1[i]) \wedge \mathsf{CheckSize}(z_2[i]))$
7: $\text{result}_c[i] = \mathsf{CheckCorrectness}(c[i])$
8: **else**
9: $\text{result}_c[i] = \mathsf{CheckCorrectness}(c[i])$
10: $\text{result}_z[i] = (\mathsf{CheckSize}(z_1[i]) \wedge \mathsf{CheckSize}(z_2[i]))$
11: $\text{result} = \text{result}_c + \text{result}_z$
12: **return** $1 - [[0]^n = \text{result}]$

CheckSize() returns 0 if the absolute value of $z_j[i]$ is small enough. Otherwise it returns 1. CheckCorrectness() is defined similarly for the correctness check. The results are added in line 11 in Algorithm 4.10. If the signature is valid, the resulting list result equals the list $[0, ..., 0]$ and the value 0 is returned. If the signature is invalid, at least one entry of the resulting list result is non-zero and 1 is returned. Thus, due to the different lists corresponding to the check results and randomization, skipping attacks and even loop-aborts as described in [92] are prevented.

4.2.6.3 Countermeasures Against Zeroing Faults

Zeroing faults can often be categorized as randomizing or skipping faults. This means that often the countermeasures described in Section 4.2.6.1 and Section 4.2.6.2 are sufficient to mitigate the zeroing attacks. If a zeroing fault is not caused by higher-order skipping or randomizing faults, we can prevent it by simply checking whether the values of the secret or error polynomial (Section 4.2.3.1), the randomness during signing (Section 4.2.3.2), the hash value (Section 4.2.3.3), or the encoding polynomial (Section 4.2.3.4) are zero.

4.2.6.4 Implementation of Countermeasures for ring-TESLA

In the following paragraphs, we illustrate how to adapt the implementation of ring-TESLA to mitigate the attacks described in Section 4.2.3, 4.2.4, and 4.2.5. Moreover, we discuss the efficiency of our implemented countermeasures. The

original implementation of ring-TESLA is vulnerable to the following attacks: S-KG-Add, Z-KG-Sec, Z-KG-Ran, S-S-Rej, S-V-Cor, and Z-S-HPoly.

Mitigation Against S-KG-Add and Z-KG-Sec. We implement a countermeasure to prevent skipping the addition of the error during the key generation described earlier in Section 4.2.5.1. The countermeasure also mitigates the zeroing attack described in Section 4.2.3.1. We show the original, vulnerable key generation algorithm of ring-TESLA alongside of our adaption in Listing 4.2.3. If line 1 (resp, line 3) in Listing 4.2.3b is skipped, `poly_T1` (resp., `poly_T2`) is uninitialized and the private and public key do not correspond to each other. Hence, signatures might not be verified correctly, but at least the attacker would not learn additional information about the secret.

```
1 poly_mul_fixed(poly_T1,poly_S,poly_a1);
2 poly_add(poly_T1,poly_T1,poly_E1);
3 poly_mul_fixed(poly_T2,poly_S,poly_a2);
4 poly_add(poly_T2,poly_T2,poly_E2);
```

a: Original code

```
poly_mul_fixed(poly_A1S,poly_S,poly_a1);
poly_add(poly_T1,poly_A1S,poly_E1);
poly_mul_fixed(poly_A2S,poly_S,poly_a2);
poly_add(poly_T2,poly_A2S,poly_E2);
```

b: Adapted code

Listing 4.2.3: Original and adapted key generation of ring-TESLA

Mitigation Against Z-KG-Ran. To mitigate zeroing attacks against the randomness `poly vec_y` in the signature generation of ring-TESLA, we implement the countermeasure shown in Listing 4.2.4. The randomness is sampled by calling `sample_y(vec_y)`. To implement our countermeasure, we introduce the function `count_zeroes()`, shown in Listing 4.2.4b. In our adaption the randomness is resampled if too many coefficients are equal to zero. Hence, zeroing of many coefficients is prevented.

For the instantiation ring-TESLA-II, we check that no more than eight coefficients are set to zero for the following reason. The polynomial `vec_y` consists of 512 integer coefficients that are sampled uniformly random over $[-2097151, 2097151]$. Hence, the probability that more than eight coefficients of a sampled polynomial are equal to zero is less than 2^{-128}. Since the security parameter of ring-TESLA-II is $\lambda = 128$, we consider our change in the probability distribution to be negligible. For other parameter sets, the bound in line 3 in Listing 4.2.4a has to be adapted.

Mitigation Against S-S-Rej. We implement the following countermeasure against skipping the rejection sampling within the signature compression function. The function `compress_sig()` copies the hash value and the polynomial `vec_y`, coefficient by coefficient into an array of bytes. Thus, we propose adding the function

```
1  poly vec_y;
2  sample_y(vec_y);
3  if (count_zeroes(vec_y) > 8){
4    // restart in line 2
5   }
6  }
```

a: Adapted Code

```
1  int count_zeroes(poly p){
2    int zeroes = 0;
3    for (int i = 0; i < PARAM_N; i++){
4      if (p[i] == 0.0){
5       zeroes++;
6      }
7    }
8    return zeroes;
9  }
```

b: Implementation of `count_zeros()`

Listing 4.2.4: Adapted code of sampling the randomness during the signature generation of ring-TESLA and implementation of `count_zeros()`

`fmodb_u()` to the original code. The corresponding adaption of `compress_sig()` and our implementation of `fmodb_u()` are illustrated in Listing 4.2.5.

In case no fault has been introduced, `fmodb_u()` does not alter any coefficient because due to the original rejection sampling all coefficients (in absolute value) are smaller than $B - U$. Assuming that the rejection sampling has been skipped and some coefficients are (in absolute value) larger than $B - U$, the function `fmodb_u()` changes the coefficients that are too large and, hence, prevents revealing secret information. In particular, it computes each coefficient modulo $B - U$. Our countermeasure thus, results in an invalid signature.

To circumvent this countermeasure, the call to the function `fmodb_u()` could be skipped in every iteration of the loop, which is executed hundreds of times. (If the entire loop is skipped, the signature will be empty.) Hence, circumventing our countermeasure is very difficult.

```
1  static void compress_sig(unsigned char *sm,
       unsigned char *c,double vec_z[PARAM_N
     ]){
2   int i,k;
3   int ptr =0;
4   int32_t t=0;
5
6   //store the hash value
7   for (i=0; i<32; i++){
8    sm[ptr++] =c[i];
9   }
10  for(i=0; i < PARAM_N; i++){
11   t = (int32_t) fmodb_u(vec_z[i]);
12  }
13  for(k=0;k<4;k++){
14   sm[ptr++] = ((t>>(8*(k))) & 0xff);
15  }
16 }
```

a: Adapted `compress_sig()`

```
1  static double fmodb_u(double x){
2   int modulus=PARAM_B - PARAM_U;
3   if(x < -modulus){
4    return x + modulus;
5   } else{
6    if(x > modulus){
7     return x - modulus;
8    } else{
9     return x;
10   }
11 }
```

b: Implementation of `fmodb_u`

Listing 4.2.5: Adapted `compress_sig()` and implementation of `fmodb_u()`

Mitigation Against S-V-Cor**.** During the verification algorithm of ring-TESLA, the input value `c_sig` and the computed hash polynomial `c` are compared. If they are equal, the function `memcmp()`[13] returns 0 which results in accepting the signature (see Listing 4.26), otherwise it returns a positive integer which results in rejecting the signature. An attacker who wants his victim to accept an invalid signature must force the verification algorithm to return 0. This can be achieved by a skipping attack. Upon considering only first-order attacks, we can prevent the skipping attack by adapting the implementation as in Listing 4.2.6a. The function still returns 0 if the signature is valid and, by definition of `memcmp()`, a non-zero value otherwise (albeit not -1). The disassembled code in Listing 4.2.6b shows that failing to call `memcmp()` returns the content of `%rax`, which is modified in line 2 and hence, which is very unlikely to be 0.

```
1 [...]
2 *mlen = smlen-CRYPTO_BYTES;
3 memcpy(m, sm, *mlen);
4 return memcmp(c,c_sig,32);
5
```

a: Adapted code

```
1  [...]
2  leaq  -112(%rbp), %rcx
3  leaq  -48(%rbp), %rax
4  movl  $32, %edx
5  movq  %rcx, %rsi
6  movq  %rax, %rdi
7  call  _memcmp
8  testl %eax, %eax
9  je  L139
10 movl  $-1, %eax
11 jmp L140
12 L139:
13 [...]
14 movl  $0, %eax
15 L140:
16 [...]
17 ret
```

b: Assembly code

Listing 4.2.6: Adapted verification and corresponding assembly code

Efficiency of the Countermeasures. Moving along to investigate the efficiency of our proposed countermeasures we implemented them and compared the execution times with the original ring-TESLA implementation. The benchmarks of the original and the adapted ring-TESLA implementation were recorded on a 4.0 GHz Intel Core i7-6700k. To compile our code, we used GCC 7.1.0. The benchmarks are given in clock cycles and are averaged over 2,000 runs for the key generation and 20,000 runs for signature generation and verification. Since compiler optimizations might remove some of the countermeasures, we report two benchmarks in Table 4.4, namely with and without the optimization `-Ofast`.

[13] We refer to `http://www.cplusplus.com/reference/cstring/memcmp/` for a description of `memcmp()`.

As shown in Table 4.4, the difference in the execution times of the original and adapted implementation is small. The largest time overhead is introduced by the countermeasure optimization S-KG-Add during key generation where the adapted implementation is 1.05 slower than the original implementation.

Table 4.4: Performance (in cycles) of the original and adapted ring-TESLA code

	Algorithm	Key generation	Sign		Verify
	Mitigated attack	S-KG-Add, Z-KG-Sec	Z-KG-Ran	S-S-Rej	S-V-Cor
w/o compiler opt.	Original	55 308 534	1 735 398	1 735 398	467 321
	Adapted	58 345 806	1 768 674	1 743 670	467 762
	Factor*	1.0549	1.0192	1.0048	1.0009
w/ compiler opt.	Original	33 429 812	69 843	69 843	71 299
	Adapted	35 247 385	71 354	71 075	73 739
	Factor*	1.0544	1.0216	1.0176	1.0342

* Rounded to four decimal positions

In this section, we explain different fault attacks on lattice-based signature schemes, propose possible countermeasures, and provide an example implementation of these countermeasures. In the next section section, we elaborate on timing, power, and electromagnetic attacks.

4.3 Vulnerability Against Other Implementation Attacks

After analyzing the vulnerability of lattice-based signature schemes against cache-side-channel and fault attacks, we now discuss other important implementation attacks. In particular, we summarize the existing results from the literature about power, electromagnetic (EM), and timing attacks and discuss their applicability to our signature schemes. An introduction to these three kinds of attack is presented in Section 2.5. An overview of published implementation attacks against lattice-based signature schemes, PKE, and key exchange protocols is given in Table 4.5.

4.3.1 Timing Attacks

We start with a discussion on timing side channels. As indicted by Table 4.5, the only attacks that exploit timing side channels target the encryption scheme NTRUEncrypt. One of the reasons for this is that lattice-based schemes are often easily protected against timing side channels; their arithmetic operations are

Algorithm \ Attack	Power/EM	Timing	Cache	Fault
NTRUEnc./Sign [121]	[150, 215, 224]	[207, 213]		[12, 130]
NTT	[184]			
LWEEncrypt [155]	[171, 192, 193]			
Frodo [52], Newhope [15]	[22]			[92]
BLISS [81]	[91]		[110, 176, 177, 199]	[92], Sect. 4.2
GLP [114]				[92], Sect. 4.2
Dilithium [82]				[111]
TESLA/qTESLA/ ring-TESLA			Sec. 4.1	[92],Sect. 4.2 [111]

Table 4.5: State-of-the-art implementation attacks against lattice-based schemes

rather simple and branchings seldom depend on secret information as stated by Pöppelmann and Güneysu [181]:

> Our implementation of Ring-LWEEncrypt is fully pipelined and has no data-dependent operations. The processor core does not support any branches and Gaussian sampling based on the inverse transform operates in constant time. Summarizing, all cryptographic operations of our core are timing-invariant.

Moreover, there exist several implementations of the LWEEncrypt scheme [155] that claim to provide resistance against timing attacks, such as [181, 182], or that prevent most prominent points of attack such as secret-dependent branches, table lookups, polynomial multiplication, or modular reductions, such as [14, 51]. Another example is the cryptographic lattice library by Microsoft[14] that is "fully protected against timing and cache attacks" [66]. In [18] the lattice-based open-source cryptographic libraries FHEW [83] and HElib [117, 118] are analyzed regarding timing side channels. The authors concluded that they did not find significant differences in the execution time depending on the secret values.

Furthermore, several works propose constant-time implementations of two building blocks that are present in almost all lattice-based schemes, namely the NTT for polynomial multiplication [205] or discrete Gaussian sampling [133, 198].

Most of the results summarized above can be transfered to lattice-based signature schemes. For example, the execution time of all subroutines in the signature generation of Dilithium [82] and qTESLA are independent of the secret key. However,

[14]The library [66] currently consists of an implementation of a lattice-based key exchange [15].

a major difference between lattice-based signature schemes and, e.g., encryption schemes is the usage of rejection sampling during the signature generation in many efficient lattice-based signatures schemes such as [24, 30, 81, 82], TESLA, and qTESLA. For example, in line 11 of the signature generation of qTESLA in Algorithm 3.9, the potential signature $\mathsf{s} = (c', z)$ is rejected if $z \notin \mathcal{R}_{q,[B-L_S]}$. Since, the rejection probability depends on random values, the signature generation does not run in constant time. However, this branching should not lead to a timing attack since it does not depend on secret information as explained as follows. If the potential signature is rejected, all random values are discarded and no additional information about the final signature oder the secret is revealed; if the potential signature is returned (approximations of) the values w_i become public.

In summary, lattice-based signature schemes are easy to be implemented to be resistant against timing attacks. However, to guarantee timing-side-channel resistance, program analysis tools such as [74] (similarly to the tool CacheAudit [78] used in Section 4.1) should be considered.

4.3.2 Power and Electromagnetic Attacks

In the following paragraphs we elaborate on power and electromagnetic attacks. Power attacks, such as DPA, are very powerful since they can be implemented with rather low-end techniques [193]. As indicated by Table 4.5, analyzing and mitigating power attacks that target lattice-based encryption, signatures, and key exchange protocols is a very active field of research.

The sources for electromagnetic side channels often resemble those for power side channels, as electromagnetic fields are generated by current flowing through respective wires. However, the experimental setup and the results to be expected differ since "the design of special probes and the development of advanced measurement methods that focus very accurately selected points of the chip" enable to measure local electromagnetic emanation [101]. Gandolfi, Mourtel, and Olivier [101] show that electromagnetic attacks are practical in many cases even when power analysis is too inaccurate to obtain definite conclusions. This observation is also supported by [91, 184]: The theoretical descriptions of the attacks against lattice-based primitives use power (or sometimes electromagnetic) side channels. However, the practical attacks are carried out using electromagnetic side channels, e.g., [91, Section 3.3] and [184, Section 4.2].

In the following paragraphs, we discuss if and how power attacks and electromagnetic attacks presented for lattice-based encryption schemes might be adapted to signature schemes such as qTESLA.

Primas, Pessl, and Mangard [184] explain that masking countermeasures such as [192, 193] seem to protect lattice-based encryption schemes against differential analysis since en- and decryption consists mainly of linear operations. The same

holds for most lattice-based signature schemes and hence, similar masking methods should be implemented as a precaution against differential analysis. However, masking might not be a sufficient countermeasure to prevent simple analysis or correlation analysis as described in the following.

Primas et al. [184] presented an attack against naive implementations of the NTT that is used in the majority of ideal-lattice-based signature schemes. Their attack exploits a secret-dependent difference of the execution time during the modular reduction. Implementing the NTT such that it runs in constant time, such as proposed by Oder et al. [168] or realized for qTESLA, might hamper the attack. However, due to sophisticated versions of the attack it might still be possible and, hence, make the implementation of additional countermeasures such as blinding or shuffling necessary [184, Section 7.3]. To ensure resistance against such attacks, a careful analysis of the implementations and corresponding countermeasures is necessary.

Additionally, the attacks presented by Espiteau et al. [91] target the Gaussian sampling, the realization of the rejection sampling, and sparse multiplication, i.e., the multiplication of a polynomial in $\mathcal{R}_q$ and a sparse polynomial with only a small and fixed number of non-zero coefficients, during signature generation of BLISS [81]. In qTESLA as well as in [24, 69, 82, 114], the Gaussian sampling has been replaced by uniform sampling. This implies that the rejection sampling in these schemes can be implemented much easier than in BLISS. In particular, the first two attacks by Espiteau et al. are not applicable to our scheme. However, qTESLA and [69, 114] also use sparse polynomial multiplication which is targeted by Espiteau et al.'s third attack as explained next. Sparse multiplication is used, e.g., in line 10 and 16 in Algorithm 3.9 of qTESLA. The attack by Espiteau et al. has two steps: First the ordering of the coefficients of c is exploited using a power analysis of one `if`-condition or some additional computation; secondly, the actual multiplication is attacked. Since our implementation of sparse multiplication runs in constant time and without secret-dependent branchings, the ordering of the coefficients is not available to the attacker. Hence, this attack (at least not in a straightforward way) is not applicable to qTESLA.

Moreover, Aysu et al. [22] also attack schoolbook polynomial and matrix implementation used in the key exchange protocols Frodo [52] and Newhope [15]. During the attack the ephemeral secret key is recovered from the ephemeral public key computed using classical polynomial/matrix multiplication and sent to the other party. Since schoolbook multiplication is used, e.g., in the sparse multiplication during qTESLA's signature generation, a careful analysis to investigate qTESLA's vulnerability to this attack is necessary.

Furthermore, the attack by Park and Han [171] exploits naive modular reductions that only reduces a value if its absolute value is larger than the modulus. Since

our realizations of the modular reduction does not branch depending on the size of the value to be reduced, this attack should not be applicable to qTESLA.

We conclude this chapter after elaborating on the security of lattice-based signature schemes and their implementations. In the next chapter, we present hybrid combiners whose security is based on both classical and post-quantum schemes in the next chapter.

5 Hybrid Signatures and KEMs

At present, researchers, industry, and standardization bodies find themselves in a predicament: On the one hand, demand to protect today's communication from the quantum threat and the expected lengthy time frame to complete widespread deployment of new algorithms call for beginning the transition from classical to post-quantum cryptography as soon as possible [160]; on the other hand, there is a lack of confidence in the concrete security of efficient post-quantum schemes because the security has not been studied as long as RSA's security, for example. Due to their wide deployment, standards and protocols such as X.509 for certificates or the TLS protocol for secure channels, are of particular interest for a fast and secure transition to post-quantum cryptography.

One approach to solve the above-mentioned dilemma are so-called hybrid schemes. They combine two or more schemes of the same kind, e.g., signatures, such that the combined scheme is secure as long as one of the components is secure.

In this chapter we construct secure hybrid signature schemes and hybrid KEM since signature schemes and KEMs are extremely important building blocks in standards and protocols. We give security reductions for all constructions in a novel adversary model which is a unification and extension of current quantum security models.

We start this chapter by introducing our family of novel security notions for signatures and KEMs in Section 5.1. In the same section, we also show implications and separations to justify our new model. Moving further, we present our combiners to construct hybrid signatures and KEMs, and prove the security reductions in Section 5.2 and 5.3, respectively. In the respective sections, we also discuss the applicability of our combiners. Our constructions can be used in the following protocols: X.509, CMS as part of S/MIME for secure e-mail, and TLS.

This chapter is based on the publications [B10] (PQCrypto 2017) and [B5] (submitted to ASIACRYPT 2018). This chapter also contains examples to identify how hybrid signatures might be used in PKI standards by Herath and Stebila.

5.1 The Two-Stage Adversary Model

In this section, we introduce a novel security model that allows to distinguish between adversaries with evolving quantum capabilities over time. Our model captures the security models introduced in Section 2.3 and adds a security notion for the following scenario. One may believe that no adversary today has a sufficiently powerful quantum computer to break any cryptographic assumption, and that it may still be some decades before a full-fledged quantum computer is built. In that scenario, one would want to protect today's communications against attacks in which the (currently classical) attacker records encrypted communications today and, once a quantum computer is available, attempts to extract the secret keys from the corresponding collected data such as key exchange transcripts or certificates.

In the remainder of this chapter, we call oracles that compute their responses using public information *public oracles* and oracles that use secret information *secret oracles*. Public oracles are for example encapsulation or verification oracles; secret oracles are for example decryption, decapsulation, or sign oracles.

To model quantum adversaries for our different quantum security notions, we distinguish between classical and quantum power in four different ways:

(i) The adversary could locally be running a classical or quantum computer during the stage in which the adversary can interact with secret oracles.
(ii) The adversary's interaction with secret oracles could be classical or quantum.
(iii) The adversary could locally be running a classical or quantum computer after the interaction with secret oracles has finished.
(iv) The adversary's interaction with the random oracle (if any) could be classical or quantum.

Adversaries in the QROM can be modeled to have classical or quantum access to the random oracle [45]. While both options lead to valid definitions, giving the adversary quantum access to the random oracle is clearly the stronger option. Moreover, it seems sensible to allow the adversary quantum access to the random oracle since it is meant to capture idealized public hash functions that can be implemented by an adversary in practice. Depending on the adversary's power this implementation can be classical or quantum. Hence, we assume that the adversary has quantum random oracle access whenever it is quantum, eliminating the fourth option above. Likewise the responses of public oracles can always be computed locally. Hence, we assume that the adversary has quantum public oracle assess whenever it is quantum.

To model the three above-mentioned distinctions (i), (ii), and (iii), we consider a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, in which $\mathcal{A}_1$ runs having access to the secret oracle, then terminates and passes a state st to $\mathcal{A}_2$, which does not have access to the secret oracle. Both adversaries have access to public oracles at all times.

Let $\mathsf{X}, \mathsf{Z} \in \{\mathsf{C}, \mathsf{Q}\}$ and $\mathsf{y} \in \{\mathsf{c}, \mathsf{q}\}$. We will use the terminology "$\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary" to denote that $\mathcal{A}_1$ is either classical ($\mathsf{X} = \mathsf{C}$) or quantum ($\mathsf{X} = \mathsf{Q}$), that $\mathcal{A}_1$'s access to its secret oracles is either classical ($\mathsf{y} = \mathsf{c}$) or quantum ($\mathsf{y} = \mathsf{q}$), and that $\mathcal{A}_2$ is either classical ($\mathsf{Z} = \mathsf{C}$) or quantum ($\mathsf{Z} = \mathsf{Q}$). In the (Q)ROM, $\mathcal{A}_1$ and $\mathcal{A}_2$ can query the random oracle in superposition, if they are quantum; this is independent of y but depends on X and Z.

The following combinations of classical and quantum adversaries in the two-stage setting are meaningful in a real world context. The notions *fully classical*, *post-quantum*, and *fully quantum* cover the notions existing in the literature and are explained in Section 2.3 where we also give examples for constructions that are proven secure with respect to these notions. The notion *future quantum* is novel.

$\mathsf{C}^{\mathsf{c}}\mathsf{C}$ corresponds to the quantum scenario *fully classical*.

$\mathsf{C}^{\mathsf{c}}\mathsf{Q}$ is called *future quantum* since it models the scenario with a currently classical but potentially future quantum adversary. For example, it models the scenario of a system which is in use today where parties eventually stop signing new documents. However, the signed documents need to remain unforgeable for a long time, even after quantum computers become available. In this scenario, the adversary uses only a classical computer during the period of time it is interacting with the secret (and public) oracles. At a later point in time, the adversary may use a quantum computer, but no longer has access to its secret (but only to its public) oracles.

$\mathsf{Q}^{\mathsf{c}}\mathsf{Q}$ corresponds to the quantum scenario *post-quantum*.

$\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$ corresponds to the quantum scenario *fully quantum*.

Our family of notions $\mathsf{C}^{\mathsf{c}}\mathsf{C}, \mathsf{C}^{\mathsf{c}}\mathsf{Q}, \mathsf{Q}^{\mathsf{c}}\mathsf{Q}, \mathsf{Q}^{\mathsf{q}}\mathsf{Q}$ form a natural hierarchy as we show in Section 5.1.2. In the next section, we define the security notions of signature schemes and KEMs in our two-stage adversary model.

5.1.1 Security Definitions in the Two-Stage Model

We consider our two-stage adversary model in the context of security notions for signature schemes and KEMs since we prove the security of our hybrid signature schemes and KEM in the two-stage model. Moreover, we also adapt traditional security definitions that are needed to construct our combiners in Section 5.2 and 5.3 to our two-stage model. In particular, we adapt the unforgeability of Message Authentication Codes (MACs), the security of PRFs, and the security of dual PRFs.

5.1.1.1 Security of Signature Schemes Against Quantum Adversaries

Our definition of EUF-CMA follows the formulation of Boneh and Zhandry [49] (see Section 2.4 for a formal definition) but separates out the adversary to be a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$, where $\mathcal{A}_1$ (of type X) interacts with either a signing oracle (of

type y) and outputs an intermediate state st (of type X), which $\mathcal{A}_2$ (of type Z) then processes. The input to $\mathcal{A}_1$ and the output of $\mathcal{A}_2$ are always classical. As mentioned above, in the (Q)ROM, $\mathcal{A}_1$ and $\mathcal{A}_2$ can query the random oracle in superposition, if the respective adversaries are quantum. Figure 5.1 shows our unified definition for EUF-CMA parameterized for any of the four types of adversaries in the standard model, i.e., without access to a random (or hash) oracle, and in the (Q)ROM for a signature scheme $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$. For a definition of (quantum) random and signing oracles we refer to Section 2.4. We define the advantage as

$$\mathrm{Adv}_{\mathcal{S}}^{\mathsf{X^yZ\text{-}EUF\text{-}CMA}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}_{\mathcal{S}}^{\mathsf{X^yZ\text{-}EUF\text{-}CMA}}(\mathcal{A}) = 0\right].$$

Given the advantage, the security definition of $\mathsf{X^yZ\text{-}EUF\text{-}CMA}$ follows easily from the EUF-CMA definition in Section 2.4.1.

Additionally, the strongly unforgeable variant $\mathsf{X^yZ\text{-}sEUF\text{-}CMA}$ can be obtained by adapting line 5 of Figure 5.1 in an analogous way to the discussion in Section 2.4.

$\underline{\mathrm{Expt}_{\mathcal{S}}^{\mathsf{X^yZ\text{-}EUF\text{-}CMA}}(\mathcal{A})}$:

0:
1: $q_S \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $st \leftarrow \mathcal{A}_1^{\mathcal{O}_S^{\mathsf{y}}}(\mathsf{pk})$
4: $((\mathsf{m}_1^*, \mathsf{s}_1^*), \ldots, (\mathsf{m}_{q_S+1}^*, \mathsf{s}_{q_S+1}^*)) \leftarrow \mathcal{A}_2(st)$
5: if $\forall\, i, j \in [1, q_S + 1], i \neq j$:
$(\mathsf{Verify}(\mathsf{pk}, \mathsf{m}_i^*, \mathsf{s}_i^*) = 0\ ) \wedge (\mathsf{m}_i^* \neq \mathsf{m}_j^*)$:
6: return 1
7: else
8: return 0

$\underline{\mathrm{Expt}_{\mathcal{S}}^{\mathsf{X^yZ\text{-}EUF\text{-}CMA}}(\mathcal{A}_1, \mathcal{A}_2)}$:

0: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{S}}$
1: $q_H \leftarrow 0$, $q_S \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $st \leftarrow \mathcal{A}_1^{\mathcal{O}_S^{\mathsf{y}}, \mathcal{O}_H^{\mathsf{X}}}(\mathsf{pk})$
4: $((\mathsf{m}_1^*, \mathsf{s}_1^*), \ldots, (\mathsf{m}_{q_S+1}^*, \mathsf{s}_{q_S+1}^*)) \leftarrow \mathcal{A}_2^{\mathcal{O}_H^{\mathsf{Z}}}(st)$
5: if $\forall\, i, j \in [1, q_S + 1], i \neq j$:
$(\mathsf{Verify}(\mathsf{pk}, \mathsf{m}_i^*, \mathsf{s}_i^*) = 0\ ) \wedge (\mathsf{m}_i^* \neq \mathsf{m}_j^*)$:
6: return 1
7: else
8: return 0

Figure 5.1: Unified security experiment for $\mathsf{X^yZ\text{-}EUF\text{-}CMA}$ in the standard model (left) and (Q)ROM (right) against a two-stage $\mathsf{X^yZ}$ adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

5.1.1.2 Security of KEMs Against Quantum Adversaries

First, we adapt the traditional definitions of IND-CCA and IND-CPA security of KEMs against quantum adversaries. Subsequently, we also define OW-CCA and OW-CPA of KEMs against quantum adversaries.

IND-CCA Security Against Quantum Adversaries. Our definition of IND-CCA follows the traditional IND-CCA formulation given in Definition 2.15 but separates out the adversary to be a two-stage adversary $(\mathcal{A}_1, \mathcal{A}_2)$, in which $\mathcal{A}_1$ (of type X)

runs having access to the decapsulation oracle (of type y), then terminates and passes a state st (of type X) to $\mathcal{A}_2$ (of type Z), which does not have access to the decapsulation oracle. The input to $\mathcal{A}_1$ and the output of $\mathcal{A}_2$ are always classical. As before, in the (Q)ROM, $\mathcal{A}_1$ and $\mathcal{A}_2$ can query the random oracle in superposition, if the respective adversaries are quantum.

Figure 5.2 shows the security experiment for indistinguishability of keys of a KEM $\mathcal{K} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ under chosen-ciphertext attacks for a two-stage $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ in the standard model and in the (Q)ROM. For a definition of (quantum) random and decapsulation oracles we refer to Section 2.4. For every notion $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}$, we define the corresponding advantage

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}}(\mathcal{A}) = \left| \Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}}(\mathcal{A}) \Rightarrow 1\right] - \frac{1}{2} \right|.$$

Given the advantage, the security definition of $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}$ follows easily from the IND-CCA definition given in Definition 2.15.

$\mathrm{Expt}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}}(\mathcal{A})$:	$\mathrm{Expt}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}}(\mathcal{A})$:
0:	0: $H \leftarrow\!\!\$\ \mathcal{H}_{\mathcal{K}}$
1: $q_D \leftarrow 0$	1: $q_D \leftarrow 0,\ q_H \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$	2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$	3: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
4: $\kappa_1^* \leftarrow\!\!\$\ K$	4: $\kappa_1^* \leftarrow\!\!\$\ K$
5: $b \leftarrow\!\!\$\ \{0,1\}$	5: $b \leftarrow\!\!\$\ \{0,1\}$
6: $st \leftarrow \mathcal{A}_1^{\mathcal{O}_D^{\mathsf{y}}}(\mathsf{pk}, c^*, \kappa_b^*)$	6: $st \leftarrow \mathcal{A}_1^{\mathcal{O}_H^{\mathsf{X}}, \mathcal{O}_D^{\mathsf{y}}}(\mathsf{pk}, c^*, \kappa_b^*)$
7: $b' \leftarrow \mathcal{A}_2(st)$	7: $b' \leftarrow \mathcal{A}_2^{\mathcal{O}_H^{\mathsf{Z}}}(st)$
8: return $[b = b']$	8: return $[b = b']$

Figure 5.2: Unified security experiment for $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-IND-CCA}$ in the standard model (left) and in the (Q)ROM (right) against a two-stage adversary $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

IND-CPA Security Against Quantum Adversaries. In the standard model, the IND-CPA adversary $\mathcal{A}$ is simply treated as a quantum algorithm. For IND-CPA in the (Q)ROM, we assume that the adversary has quantum random oracle access whenever it is quantum, as before. In Figure 5.3, we give a unified definition of classical and quantum IND-CPA, denoted $\mathsf{Z}\text{-IND-CPA}$, where Z is either C (for classical) or Q (for quantum). We define the corresponding advantage

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{Z}\text{-IND-CPA}}(\mathcal{A}) = \left| \Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{Z}\text{-IND-CPA}}(\mathcal{A}) = 1\right] - \frac{1}{2} \right|.$$

Given the advantage, the security definition of $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}$ follows easily from the IND-CPA definition given in Definition 2.14.

For consistency with our IND-CCA notion, we occasionally also use the notation $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}$ instead of $\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}$ for IND-CPA security. In such cases we sometimes refer to both as $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{ATK}$ with $\mathsf{atk} \in \{\mathsf{cpa}, \mathsf{cca}\}$. We stress, however, that X and y are irrelevant in case of IND-CPA, and that they are there solely for notational uniformity.

$\underline{\mathrm{Expt}_{\mathcal{K}}^{\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}}(\mathcal{A})}$:	$\underline{\mathrm{Expt}_{\mathcal{K}}^{\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}}(\mathcal{A})}$:
0:	0: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{K}}$
1: $q_H \leftarrow 0$	1: $q_H \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$	2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$	3: $(c^*, \kappa_0^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
4: $\kappa_1^* \leftarrow_{\$} K$	4: $\kappa_1^* \leftarrow_{\$} K$
5: $b \leftarrow_{\$} \{0,1\}$	5: $b \leftarrow_{\$} \{0,1\}$
6: $b' \leftarrow \mathcal{A}(\mathsf{pk}, c^*, \kappa_b^*)$	6: $b' \leftarrow \mathcal{A}^{\mathcal{O}_H^{\mathsf{Z}}}(\mathsf{pk}, c^*, \kappa_b^*)$
7: return $[b = b']$	7: return $[b = b']$

Figure 5.3: Security experiment for $\mathsf{Z}\text{-}\mathsf{IND}\text{-}\mathsf{CPA}$ in the standard model (left) and in the (Q)ROM (right) against a Z adversary $\mathcal{A}$

One-Way Security of KEMs Against Quantum Adversaries. During the one-way security experiment of KEMs, the adversary's task is to fully recover the session key, not just to distinguish it from random as in the indistinguishability notions described earlier. As before, we adapt the experiment against quantum adversaries in both the chosen-plaintext and chosen-ciphertext scenarios. Figure 5.4 shows the security experiments for quantum OW-CPA ($\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CPA}$) and Figure 5.5 shows the security experiments for quantum OW-CCA ($\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CCA}$) in the standard model and (Q)ROM. As before, we assume that the adversary has quantum random oracle access whenever it is quantum as before in the (Q)ROM. We define the corresponding advantages

$$\mathsf{Adv}_{\mathcal{K}}^{\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CPA}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CPA}}(\mathcal{A}) = 1\right], \text{ and}$$

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CCA}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}_{\mathcal{K}}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CCA}}(\mathcal{A}) = 1\right].$$

Given the advantages, the security definitions of $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CPA}$ and $\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-}\mathsf{OW}\text{-}\mathsf{CCA}$ follow easily from the respective definitions in Section 2.4, namely Definition 2.16 and Definition 2.17.

$\underline{\mathrm{Expt}^{\mathsf{Z\text{-}OW\text{-}CPA}}_{\mathcal{K}}(\mathcal{A})}$:	$\underline{\mathrm{Expt}^{\mathsf{Z\text{-}OW\text{-}CPA}}_{\mathcal{K}}(\mathcal{A})}$:
0:	0: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{K}}$
1: $q_H \leftarrow 0$	1: $q_H \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$	2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$	3: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
4: $\kappa' \leftarrow \mathcal{A}(\mathsf{pk}, c^*)$	4: $\kappa' \leftarrow \mathcal{A}^{\mathcal{O}^{\mathsf{Z}}_H}(\mathsf{pk}, c^*)$
5: return $[\kappa^* = \kappa']$	5: return $[\kappa^* = \kappa']$

Figure 5.4: Security experiment for Z-OW-CPA in the standard model (left) and in the (Q)ROM (right) against a Z adversary $\mathcal{A}$

$\underline{\mathrm{Expt}^{\mathsf{X^y Z\text{-}OW\text{-}CCA}}_{\mathcal{K}}(\mathcal{A})}$:	$\underline{\mathrm{Expt}^{\mathsf{X^y Z\text{-}OW\text{-}CCA}}_{\mathcal{K}}(\mathcal{A})}$:
0:	0: $H \leftarrow_{\$} \mathcal{H}_{\mathcal{K}}$
1: $q_D \leftarrow 0$, $q_H \leftarrow 0$	1: $q_D \leftarrow 0$, $q_H \leftarrow 0$
2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$	2: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
3: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$	3: $(c^*, \kappa^*) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
4: $st \leftarrow \mathcal{A}_1^{\mathcal{O}^{\mathsf{y}}_D}(\mathsf{pk}, c^*)$	4: $st \leftarrow \mathcal{A}_1^{\mathcal{O}^{\mathsf{X}}_H, \mathcal{O}^{\mathsf{y}}_D}(\mathsf{pk}, c^*)$
5: $\kappa' \leftarrow \mathcal{A}_2(st)$	5: $\kappa' \leftarrow \mathcal{A}_2^{\mathcal{O}^{\mathsf{Z}}_H}(st)$
6: return $[\kappa^* = \kappa']$	6: return $[\kappa^* = \kappa']$

Figure 5.5: Unified security experiment for $\mathsf{X^y Z}$-OW-CCA in the standard model (left) and in the (Q)ROM (right) against a two-stage adversary $\mathsf{X^y Z}$ adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

The Fujisaki–Okamoto Transform. The Fujisaki–Okamoto (FO) transform [72, 97, 98] constructs an IND-CCA secure PKE or KEM from an IND-CPA secure (or OW-CPA secure) PKE in the ROM; analogues that are secure in the QROM have been given by Targhi and Unruh [210] and in a modular framework by Hofheinz, Hövelmanns, and Kiltz [123]. This can also be applied in our two-stage adversary model. The FO transform converts a C-OW-CPA secure PKE into a $\mathsf{C^c}$C-IND-CCA secure KEM (or PKE), in the ROM. The transforms presented in [123, 210] convert a Q-OW-CPA secure PKE into a $\mathsf{Q^c}$Q-IND-CCA secure KEM (or PKE). It remains an open question how to transform a Q-IND-CPA secure PKE into a $\mathsf{Q^q}$Q-IND-CCA secure KEM (or PKE).

5.1.1.3 Unforgeability of MACs Against Quantum Adversaries

Formally, a MAC is a tuple of algorithms $\mathcal{M} = (\mathsf{MKG}, \mathsf{MAC}, \mathsf{MVf})$ for key generation, MAC tag generation, and MAC tag verification. For our combiners, it suffices to use one-time MACs with multiple verification queries. This means that the adversary can initially choose a message, receives the MAC, and can then make multiple verification attempts for other messages. For our combiners, strong unforgeability is required, i.e., the adversary wins if it creates any new valid message-tag pair, even for the same initial message.

Analogous to before, the two-stage version of One-Time strong Existential Unforgeability (OT-sEUF) with multiple verifications uses an $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary which is of type X while it has access to its verification oracle (of type y) and receives the challenge ciphertext. The adversary is of type Z after it no longer has access to its verification oracle. To capture the strong combiner property of MACs, where the adversary may try to win for a key $k_{\mathsf{mac}} = (k_{\mathsf{mac},1}, k_{\mathsf{mac},2})$ where either $k_{\mathsf{mac},1}$ or $k_{\mathsf{mac},2}$ is chosen by the adversary, we allow the adversary to specify one of the two keys for computing the challenge and for each verification query and in the forgery attempt. We define the unforgeability of MACs formally next.

Definition 5.1 (Two-Stage MAC Unforgeability)**.** *Let $\mathcal{M} = (\mathsf{MKG}, \mathsf{MAC}, \mathsf{MVf})$ be a MAC. We say that $\mathcal{M}$ is a (t, ϵ)-$\mathsf{X}^{\mathsf{y}}\mathsf{Z}$-OT-sEUF secure MAC if for all $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversaries $\mathcal{A}$ interacting with $\mathcal{M}$ in the security experiment given in Figure 5.6 and running in time t the advantage is*

$$\mathrm{Adv}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-OT-sEUF}}_{\mathcal{M}}(\mathcal{A}) = \Pr\left[\mathrm{Expt}^{\mathsf{X}^{\mathsf{y}}\mathsf{Z}\text{-OT-sEUF}}_{\mathcal{M}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

5.1.1.4 PRF Security Against Quantum Adversaries

Two of our combiners, namely the dual-PRF combiner dPRF in Section 5.3.2 and the nested dual-PRF combiner nPRF in Section 5.3.3 are based on the security of PRFs and dual PRFs.

As before, the two-stage versions of the security of PRFs and dual PRFs uses an $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary which is of type X while it has y access to its pseudo-random oracle $\mathcal{O}^{\mathsf{y}}_F$. The adversary is of type Z after it no longer has access to its oracle. We give formal definitions of the respective security notions in Definition 5.2 and 5.3.

Definition 5.2 (Two-Stage PRF Security)**.** *Let $F : \mathsf{Keys} \times \mathsf{In} \to \mathsf{Out}$ be a PRF. Define $\mathsf{Func}[\mathsf{In}, \mathsf{Out}]$ to be the set of all functions $f : \mathsf{In} \to \mathsf{Out}$. Furthermore, let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be a two-stage $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$ adversary interacting with F in the Game $\mathrm{Expt}^{\mathsf{PRF\text{-}SEC}}_F$ given in Figure 5.7.*

$\underline{\text{Expt}^{\mathsf{X^yZ\text{-}OT\text{-}sEUF}}_{\mathcal{M}}(\mathcal{A})}$:
1: $q_V \leftarrow 0$
2: $k = (k_1, k_2) \leftarrow \mathsf{MKG}()$
3: $(m^*, b, k_b^*, st) \leftarrow \mathcal{A}_1()$
4: if $b = 1$ then $k' \leftarrow (k_1^*, k_2)$ else $k^* \leftarrow (k_1, k_2^*)$
5: $\tau^* \leftarrow \mathsf{MAC}_{k^*}(m^*)$
6: $st \leftarrow \mathcal{A}_1^{\mathcal{O}_V^{\mathsf{y}}}(\tau^*)$
7: $(m', \tau', k_b') \leftarrow \mathcal{A}_2(st)$
8: if $b = 1$ then $k' \leftarrow (k_1', k_2)$ else $k' \leftarrow (k_1, k_2')$
9: if $[\mathsf{MVf}_{k'}(m', \tau') = 1] \wedge [(m', \tau') \neq (m^*, \tau^*)]$ then return 1 else return 0

$\underline{\mathcal{O}_V^{\mathsf{c}}(m, \tau, k_b'):}$
1: $q_V \leftarrow q_V + 1$
2: if $b = 1$ then $k' \leftarrow (k_1', k_2)$ else $k' \leftarrow (k_1, k_2')$
3: return $\mathsf{MVf}_{k'}(m, \tau)$

$\underline{\mathcal{O}_V^{\mathsf{q}}(\sum_{m,\tau,b,k_b',t,z} \psi_{m,\tau,b,k_b',t,z} \, |m, \tau, b, k_b', t, z\rangle):}$
1: $q_V \leftarrow q_V + 1$
2: if $b = 1$ then $k' \leftarrow (k_1', k_2)$ else $k' \leftarrow (k_1, k_2')$
3: return $\sum_{m,\tau,b,k_b',t,z} \psi_{m,\tau,b,k_b',t,z} \, |m, \tau, b, k_b', t \oplus \mathsf{MVf}_{k'}(m, \tau), z\rangle$

Figure 5.6: Unified security experiment for $\mathsf{X^yZ\text{-}OT\text{-}sEUF}$ against an $\mathsf{X^yZ}$ adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

We say that F is a $\mathsf{X^yZ}$ (t, ϵ)-secure PRF ($\mathsf{X^yZ\text{-}PRF\text{-}SEC}$) if for all quantum $\mathsf{X^yZ}$ adversaries $\mathcal{A}$ running in time t the advantage is

$$\mathsf{Adv}_F^{\mathsf{X^yZ\text{-}PRF\text{-}SEC}}(\mathcal{A}) = \Pr\left[\text{Expt}_F^{\mathsf{X^yZ\text{-}PRF\text{-}SEC}}(\mathcal{A}) = 1\right] \leq \epsilon.$$

Definition 5.3 (Two-Stage Dual PRF Security)**.** *Let $F : \mathsf{Keys} \times \mathsf{In} \to \mathsf{Out}$ be a PRF. Furthermore, define $F^{\mathsf{swap}} : \mathsf{In} \times \mathsf{Keys} \to \mathsf{Out}$. We say that F is a dual PRF if both F and F^{swap} are PRFs.*

We say that F is a $\mathsf{X^yZ}$ (t, ϵ)-secure dual PRF ($\mathsf{X^yZ\text{-}dPRF\text{-}SEC}$) if for all $\mathsf{X^yZ}$ adversaries $\mathcal{A}$ running in time t the advantage is

$$\mathsf{Adv}_F^{\mathsf{X^yZ\text{-}dPRF\text{-}SEC}}(\mathcal{A}) = \max\{\mathsf{Adv}_F^{\mathsf{X^yZ\text{-}PRF\text{-}SEC}}(\mathcal{A}), \mathsf{Adv}_{F^{\mathsf{swap}}}^{\mathsf{X^yZ\text{-}PRF\text{-}SEC}}(\mathcal{A})\} \leq \epsilon.$$

5.1.2 Separations and Implications

After defining our security model and the security notions for signature schemes and KEM, we now prove the natural hierarchy of our notions $\mathsf{C^cC}, \mathsf{C^cQ}, \mathsf{Q^cQ}, \mathsf{Q^qQ}$ introduced in the beginning of this section.

$\underline{\mathrm{Expt}_F^{\mathsf{X^yZ\text{-}PRF\text{-}SEC}}(\mathcal{A})}$:
1: $k \leftarrow_\$ \mathsf{Keys}_F$
2: $b \leftarrow_\$ \{0,1\}$
3: $f \leftarrow_\$ \mathsf{Func}[\mathsf{In},\mathsf{Out}]$
4: $st \leftarrow \mathcal{A}_1^{\mathcal{O}^{\mathsf{y}}_F}()$
5: $b' \leftarrow \mathcal{A}_2(st)$
6: return $[b = b']$

$\underline{\text{Classical } \mathcal{O}^{\mathsf{y}}_F(x)}$:
1: if $b = 1$:
2: return $f(x)$
3: else
4: return $F(k,x)$

$\underline{\text{Quantum } \mathcal{O}^{\mathsf{y}}_F(\sum_{x,t,z} \psi_{x,t,z} \left|x,t,z\right\rangle)}$:
1: return $\sum_{x,t,z} \psi_{x,t,z} \left|x,t \oplus \mathcal{O}^{\mathsf{y}}_F(x), z\right\rangle$

Figure 5.7: Unified security experiment for $\mathsf{X^yZ\text{-}PRF\text{-}SEC}$ against a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$

It is notation-wise convenient to define an order for the notions, with $\mathsf{Q} \geq \mathsf{C}$ and $\mathsf{q} \geq \mathsf{c}$, consequently implying a partial order $\mathsf{X^yZ} \geq \mathsf{U^vW}$ if $\mathsf{X} \geq \mathsf{U}$, $\mathsf{y} \geq \mathsf{v}$, and $\mathsf{Z} \geq \mathsf{W}$, i.e., $\mathsf{Q^qQ} \geq \mathsf{Q^cQ} \geq \mathsf{C^cQ} \geq \mathsf{C^cC}$. Let $\max S$ (resp., $\min S$) denote the set of maximal (resp., minimal) elements of a set $S \subseteq \{\mathsf{C^cC}, \mathsf{C^cQ}, \mathsf{Q^cQ}, \mathsf{Q^qQ}\}$ according to this partial order. Since we usually have a total order on S, we often simply speak of *the* maximal element. For example, it holds that $\mathsf{C^cQ} = \max\{\mathsf{C^cC}, \mathsf{C^cQ}\}$. It holds that the stronger the security notion the smaller the advantage of an adversary $\mathcal{A}$ breaking a scheme of corresponding security. For example, let a signature scheme $\mathcal{S}$ be $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure. $\mathsf{C^cQ}$ is stronger than $\mathsf{C^cC}$, i.e., $\mathsf{C^cQ} \geq \mathsf{C^cC}$. Hence, $\mathrm{Adv}^{\mathsf{C^cQ\text{-}EUF\text{-}CMA}}_{\mathcal{S}}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{C^cC\text{-}EUF\text{-}CMA}}_{\mathcal{S}}(\mathcal{A})$.

Figure 5.8 shows the implications and separations between these notions for the unforgeability notions of signatures. Figure 5.9 shows the implications and separations between these notions for indistinguishability notions of KEMs. In the figures, "$B \implies A$" denotes an implication, i.e., every B secure scheme is also A secure, and "$A \not\Rightarrow B$" denotes a separation, i.e., there exist a scheme that is A secure but that is not B secure.

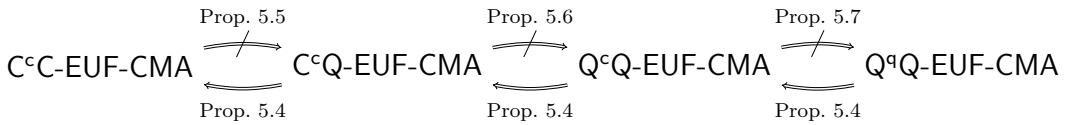


Figure 5.8: Implications and separations between unforgeability notions ($\mathsf{X^yZ\text{-}EUF\text{-}CMA}$) for signature schemes.

The implications in Figure 5.8 and 5.9 are straightforward. Moreover, each of the separations $A \not\Rightarrow B$ follows from a common technique: From an A secure scheme $\mathcal{S}$, construct a (degenerate) A secure scheme $\mathcal{S}'$ that is not B secure. Then the additional power available to a B adversary allows the adversary to recover

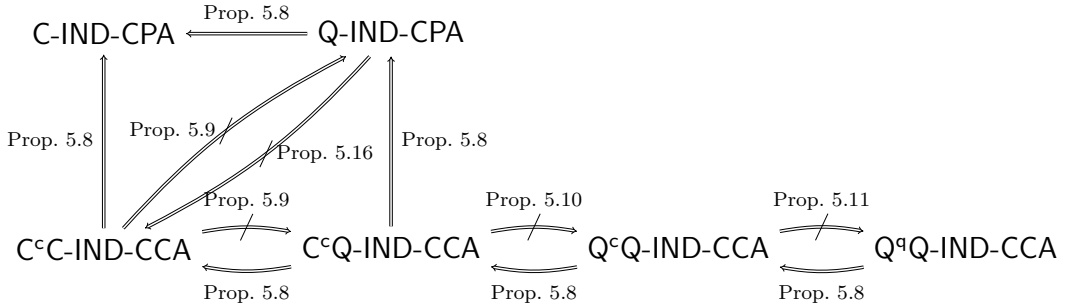


Figure 5.9: Implications and separations between Z-IND-CPA and X^{y}Z-IND-CCA notions; missing arrows can be inferred by transitivity

the secret key of $\mathcal{S}$ that was embedded somewhere in $\mathcal{S}'$. In what follows, we state and prove the above statements formally. We start with the security notions of signature schemes. Afterwards, we turn to KEMs.

5.1.2.1 Implications and Separations for Signature Schemes

First, we prove the implications shown in Figure 5.8.

Proposition 5.4 (Q^{q}Q- $\Rightarrow$ Q^{c}Q- $\Rightarrow$ C^{c}Q- $\Rightarrow$ C^{c}C-EUF-CMA)**.** *If $\mathcal{S}$ is a* Q^{q}Q-EUF-CMA *secure signature scheme, then $\mathcal{S}$ is also* Q^{c}Q-EUF-CMA *secure. If $\mathcal{S}$ is a* Q^{c}Q-EUF-CMA *secure signature scheme, then $\mathcal{S}$ is also* C^{c}Q-EUF-CMA *secure. If $\mathcal{S}$ is a* C^{c}Q-EUF-CMA *secure signature scheme, then $\mathcal{S}$ is also* C^{c}C-EUF-CMA *secure.*

The proof of Proposition 5.4 is straightforward since every classical adversary can be seen as a quantum adversary that forgoes its additional quantum power.

Proof. Suppose $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an adversary against C^{c}C-EUF-CMA, i.e., both $\mathcal{A}_1$ and $\mathcal{A}_2$ are classical. Every C^{c}C-EUF-CMA adversary can be seen as a C^{c}Q-EUF-CMA adversary that does not use its quantum power in the second stage during the C^{c}C-EUF-CMA experiment. Thus, $(\mathcal{A}_1, \mathcal{A}_2)$ wins the C^{c}Q-EUF-CMA experiment with at least the same probability as the adversary wins the C^{c}C-EUF-CMA experiment, i.e.,

$$\mathrm{Adv}_{\mathcal{S}}^{\mathsf{C^cQ\text{-}EUF\text{-}CMA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{S}}^{\mathsf{C^cC\text{-}EUF\text{-}CMA}}(\mathcal{A}).$$

Similarly,

$$\mathrm{Adv}_{\mathcal{S}}^{\mathsf{Q^cQ\text{-}EUF\text{-}CMA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{S}}^{\mathsf{C^cQ\text{-}EUF\text{-}CMA}}(\mathcal{A}).$$

Finally, an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against Q^{c}Q-EUF-CMA can be seen as an adversary against Q^{q}Q-EUF-CMA that simply does not query its signing oracle in superposition and thus,

$$\mathrm{Adv}_{\mathcal{S}}^{\mathsf{Q^qQ\text{-}EUF\text{-}CMA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{S}}^{\mathsf{Q^cQ\text{-}EUF\text{-}CMA}}(\mathcal{A}).$$

□

In the following paragraphs, we show that the implications shown in Figure 5.8 are in fact strict by showing separations between the different notions. We start with Proposition 5.5 which states essentially that there exist signature schemes that are classically secure ($\mathsf{C^cC\text{-}EUF\text{-}CMA}$) but that become insecure once adversaries gain quantum power at a later point in time ($\mathsf{C^cQ\text{-}EUF\text{-}CMA}$).

Proposition 5.5 ($\mathsf{C^cC\text{-}EUF\text{-}CMA} \not\Rightarrow \mathsf{C^cQ\text{-}EUF\text{-}CMA}$)**.** *In the classical ROM, assuming RSA is a one-way function (for classical algorithms), there exists a* $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ *secure signature scheme that is not* $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ *secure.*

The idea of the proof is as follows. From a $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ secure signature scheme $\mathcal{S}$, we construct a degenerate signature scheme $\mathcal{S}'$ which is still $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ secure, but not $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure. In particular, in the public key for $\mathcal{S}'$, we include a copy of the signing secret key encrypted using an RSA-based PKE, i.e., a PKE that is secure as long as the RSA problem is computationally hard. The RSA problem and its relation to the security of RSA schemes and the integer factorization problem is described, e.g., in [158, Section 8.2.2]. If breaking RSA is classically hard then the encrypted signing key of $\mathcal{S}'$ is useless to a $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ adversary. A $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ adversary, however, is able to use Shor's quantum algorithm [206] to break the PKE, recover the signing key, and forge signatures.

Proof. Let $\mathcal{E} = (\mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$ be a PKE that is IND-CPA secure against classical adversaries and whose security relies on the hardness of the RSA problem. Such PKEs are for example [95] or RSA-OAEP [35]. However, a quantum adversary could use Shor's algorithm to factor the modulus and decrypt ciphertexts encrypted using $\mathcal{E}$. We construct a scheme $\mathcal{S}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ that is based on $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, but where the public key of $\mathcal{S}'$ includes a $\mathcal{E}$-encrypted copy of the secret key of $\mathcal{S}$. The scheme $\mathcal{S}'$ is depicted in Figure 5.10.

Next we show that $\mathcal{S}'$ is $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ secure. Since $\mathcal{E}$ is IND-CPA secure, no passive adversary can distinguish $\mathcal{E}$-encryptions of sk from encryptions of $0^{|\mathsf{sk}|}$ with significant advantage. So we can replace c in pk' with an encryption of zeros, while still successfully simulating answers to the signing oracle in the $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ experiment. A $\mathcal{S}'$ forgery is immediately a $\mathcal{S}$ forgery. Hence, this reduction shows that $\mathcal{S}'$ is $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ secure.

Finally, we prove that $\mathcal{S}'$ is not $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure. Given the verification key $\mathsf{pk}' = (\mathsf{pk}, \mathsf{ek}, c)$ of $\mathcal{S}'$, run the quantum adversary $\mathcal{A}_2$ on ek and c to recover sk. We can now forge signatures in $\mathcal{S}'$ by using the signing algorithm with sk. □

Next, we show that there exist signature schemes that are secure in the future quantum setting ($\mathsf{C^cQ\text{-}EUF\text{-}CMA}$) but that become insecure in the post-quantum setting ($\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$).

Algorithm 5.1 KeyGen'

Require: -
Ensure: $(\mathsf{sk}', \mathsf{pk}')$

1: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
2: $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}()$
3: $c \leftarrow \mathsf{Enc}_{\mathcal{E}}(\mathsf{ek}, \mathsf{sk})$
4: $\mathsf{pk}' \leftarrow (\mathsf{pk}, \mathsf{ek}, c)$
5: **return** $(\mathsf{sk}, \mathsf{pk}')$

Algorithm 5.2 Sign'

Require: sk, m
Ensure: s

1: **return** $\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$

Algorithm 5.3 Verify'

Require: $(\mathsf{pk}, \mathsf{ek}, c), \mathsf{m}, \mathsf{s}$
Ensure: $\{0, -1\}$

1: **return** $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \mathsf{s})$

Figure 5.10: Description of the separating signature scheme $\mathcal{S}'$ which is $\mathsf{C^cC\text{-}EUF\text{-}CMA}$ secure but not $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure

Proposition 5.6 ($\mathsf{C^cQ\text{-}EUF\text{-}CMA} \not\Longrightarrow \mathsf{Q^cQ\text{-}EUF\text{-}CMA}$)**.** *In the ROM, assuming RSA is a one-way function (for classical algorithms), there exists a* $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ *secure signature scheme that is not* $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ *secure.*

The idea of the proof is as follows. As before we construct a degenerate signature scheme $\mathcal{S}'$. This time the public key of the signature scheme $\mathcal{S}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ includes an RSA encrypted random challenge string. Furthermore, we define Sign' so that, if the adversary queries the signing oracle on the random challenge string, the signing key is returned. If breaking the RSA problem is computationally hard for classical algorithms, a $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ adversary will not be able to recover the challenge while it has access to the signing oracle, and thus cannot make use of the degeneracy to recover the signing key. A $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ adversary, however, is able to recover the secret key.

Proof. Let $\mathcal{E} = (\mathsf{KeyGen}_{\mathcal{E}}, \mathsf{Enc}_{\mathcal{E}}, \mathsf{Dec}_{\mathcal{E}})$ be a PKE that is IND-CPA secure against classical adversaries and whose security relies on the hardness of the RSA problem as in the proof of Proposition 5.5. However, a quantum adversary can use Shor's algorithm to factor the modulus and decrypt ciphertexts encrypted using $\mathcal{E}$. Our signature scheme $\mathcal{S}' = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$ is designed such that the signing oracle can be used by a quantum adversary that has access to the signing oracle to learn the signing key. A quantum adversary *without* access to the signing oracle, however, cannot learn the signing key. The construction idea of the separating signature scheme is as follows. We put an encrypted random challenge in the public verification key, and if the adversary asks for that challenge to be signed, we have the signing oracle return the signing key. Intuitively, only an adversary that can break the challenge while it has access to the signing oracle (i.e., a quantum first-stage adversary) can solve the challenge. The signature scheme $\mathcal{S}'$ is depicted in Figure 5.11.

Let λ be the security parameter of the signature scheme $\mathcal{S}'$. By construction of $\mathcal{S}'$ it holds that if $\mathcal{S}$ is ϵ-correct, then $\mathcal{S}'$ is $(\epsilon + \frac{1}{2^{2\lambda}})$-correct since $s^* \in \{0,1\}^{2\lambda}$.

We start by showing the $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ security of $\mathcal{S}'$. Since $\mathcal{E}$ is IND-CPA secure, no passive, classical adversary can distinguish $\mathcal{E}$-encryptions of s^* from encryptions of $0^{2\lambda}$ with significant advantage. An adversary could guess s^* with a probability of $\frac{q_S}{2^{2\lambda}}$, where q_S is the number of queries to the signing oracle. Therefore, we can replace ch in pk' with an encryption of zeros, while still successfully simulating answers to the (classical) signing oracle in the $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ experiment. An $\mathcal{S}'$ forgery is immediately an $\mathcal{S}$ forgery. Hence, this reduction shows that $\mathcal{S}'$ is $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure.

Next we show that $\mathcal{S}'$ is not $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ secure. Suppose $\mathcal{A}$ breaks the message recovery of $\mathcal{E}$, i.e., decrypts ciphertexts encrypted using $\mathcal{E}$. Consider an algorithm $\mathcal{B}_1$ which uses $\mathcal{A}$ to decrypt ch and recovers the correct solution s^*. Afterwards, $\mathcal{B}_1$ queries s^* to its signing oracle to obtain the signing key sk. $\mathcal{B}_1$ will return a valid forgery. Taking $\mathcal{B}_2$ as the identity function, $(\mathcal{B}_1, \mathcal{B}_2)$ is a forger for $\mathcal{S}'$ if $\mathcal{A}$ can decrypt ciphertexts of $\mathcal{E}$. As mentioned at the beginning of the proof, $\mathcal{A}$ could be an adversary using Shor's quantum algorithm to factor the modulus and decrypt ciphertexts encrypted using $\mathcal{E}$, for example. □

Algorithm 5.4 KeyGen'

Require: -
Ensure: $(\mathsf{sk}', \mathsf{pk}')$

1: $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}()$
2: $(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}_{\mathcal{E}}()$
3: $s^* \leftarrow_{\$} \{0,1\}^{2\lambda}$
4: $ch \leftarrow \mathsf{Enc}_{\mathcal{E}}(\mathsf{ek}, s^*)$
5: $\mathsf{pk}' \leftarrow (\mathsf{pk}, \mathsf{ek}, ch)$
6: **return** $(\mathsf{sk}, \mathsf{pk}')$

Algorithm 5.5 Sign'

Require: $(\mathsf{sk}, s^*), \mathsf{m}$
Ensure: s

1: **if** $\mathsf{m} = s^*$ **then**
2: **return** sk
3: **else**
4: **return** $\mathsf{Sign}(\mathsf{sk}, \mathsf{m})$

Algorithm 5.6 Verify'

Require: $(\mathsf{pk}, \mathsf{ek}, ch), \mathsf{m}, \mathsf{s}$
Ensure: $\{0, -1\}$

1: **return** $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \mathsf{s})$

Figure 5.11: Description of the separating signature scheme $\mathcal{S}'$ which is $\mathsf{C^cQ\text{-}EUF\text{-}CMA}$ secure but not $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ secure

We now state that post-quantum secure signature schemes ($\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$) are not necessarily secure in the fully quantum setting where the adversary has quantum access to the signing oracle ($\mathsf{Q^qQ\text{-}EUF\text{-}CMA}$).

Proposition 5.7 ($\mathsf{Q^cQ} \not\Rightarrow \mathsf{Q^qQ}$)**.** *Assuming there exists a quantum secure pseudo-random family of permutations and let $\mathcal{S}$ be a signature scheme that is* $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ *secure, then there exists a signature scheme $\mathcal{S}'$ that is* $\mathsf{Q^cQ\text{-}EUF\text{-}CMA}$ *secure but not* $\mathsf{Q^qQ\text{-}EUF\text{-}CMA}$ *secure.*

The formal proof by McKague can be found in [B10]. The idea of the proof is as follows. The secret is hidden using a query-complexity problem that can be solved with just a few queries in superposition by a quantum algorithm, but takes exponential many queries when asking classical queries. The specific problem we use is a variant of the hidden linear structure problem [70]. The proof of the corresponding statement for IND-CCA security of KEMs is similar (see the proof of Proposition 5.11).

5.1.2.2 Implications and Separations for KEMs

Similarly to Section 5.1.2.1, and as described in Figure 5.9, the various indistinguishability notions for KEMs are related to each other through a series of implications and separations as we show next. As before, we prove the implications shown in Figure 5.9 first.

Proposition 5.8 ($\mathsf{Q^qQ\text{-}} \Rightarrow \mathsf{Q^cQ\text{-}} \Rightarrow \mathsf{C^cQ\text{-}} \Rightarrow \mathsf{C^cC\text{-}IND\text{-}CCA}$, $\mathsf{Q\text{-}} \Rightarrow \mathsf{C\text{-}IND\text{-}CPA}$)**.** *Let $\mathcal{K}$ be a KEM. If $\mathcal{K}$ is* $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ *secure, then $\mathcal{K}$ is also* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *secure. If $\mathcal{K}$ is* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *secure, then $\mathcal{K}$ is also* $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ *secure. If $\mathcal{K}$ is* $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ *secure, then $\mathcal{K}$ is also* $\mathsf{C^cC\text{-}IND\text{-}CCA}$ *secure and* $\mathsf{Q\text{-}IND\text{-}CPA}$ *secure. If $\mathcal{K}$ is* $\mathsf{Q\text{-}IND\text{-}CPA}$ *secure or* $\mathsf{C^cC\text{-}IND\text{-}CCA}$ *secure, then $\mathcal{K}$ is also* $\mathsf{C\text{-}IND\text{-}CPA}$ *secure.*

The proof of Proposition 5.8 is straightforward since every classical adversary can be seen as a quantum adversary that forgoes its additional quantum power.

Proof. Suppose $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is an adversary against $\mathsf{C^cC\text{-}IND\text{-}CCA}$, i.e., both $\mathcal{A}_1$ and $\mathcal{A}_2$ are classical. Every $\mathsf{C^cC\text{-}IND\text{-}CCA}$ adversary can be seen as a $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ adversary that does not use its quantum power in the second stage during the $\mathsf{C^cC\text{-}IND\text{-}CCA}$ experiment. Thus, it wins the $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ experiment with at least the same probability as it wins the $\mathsf{C^cC\text{-}IND\text{-}CCA}$ experiment, i.e.,

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{C^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{C^cC\text{-}IND\text{-}CCA}}(\mathcal{A}).$$

Similarly,

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{C^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}).$$

Finally, an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ against $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ can be seen as an adversary against $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ that does not query its decryption oracle in superposition. Thus,

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q^qQ\text{-}IND\text{-}CCA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}).$$

Using similar arguments, for a $\mathsf{C\text{-}IND\text{-}CPA}$ adversary $\mathcal{A}$ it holds that

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q\text{-}IND\text{-}CPA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{C\text{-}IND\text{-}CPA}}(\mathcal{A}).$$

Moreover, it trivially holds for a C-IND-CPA adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ that

$$\mathrm{Adv}_{\mathcal{K}}^{\mathsf{C^cC\text{-}IND\text{-}CCA}}(\mathcal{A}) \geq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{C^cC\text{-}IND\text{-}CPA}}(\mathcal{A}) = \mathrm{Adv}_{\mathcal{K}}^{\mathsf{C\text{-}IND\text{-}CPA}}(\mathcal{A}_2).$$

□

Moving forward, we now show that these implications are in fact strict by showing separations between the different notions. We start with Proposition 5.9 which states essentially that there exist KEMs that are classically secure (C^cC-IND-CCA), but that become insecure once adversaries gain quantum power at a later point in time (C^cQ-IND-CCA).

Proposition 5.9 (C^cC-IND-CCA $\not\Longrightarrow$ Q-IND-CPA, C^cQ-IND-CCA)**.** *In the ROM, assuming RSA is a one-way function (for classical algorithms), there exists a* C^cC-IND-CCA *secure KEM in the ROM that is neither* Q-IND-CPA *secure nor* C^cQ-IND-CCA *secure.*

The proof follows easily from the existence of the RSA-OAEP encryption [35].

Proof. The proposition follows immediately from the following facts: It is known that the KEM based on RSA-OAEP is C^cC-IND-CCA secure in the ROM [35]. However, an adversary with local access to quantum computing power in the second stage can run Shor's quantum algorithm to factor the RSA modulus. Hence, the adversary can recover the decapsulation key to win the Q-IND-CPA or C^cQ-IND-CCA experiments. This implies that RSA-OAEP is not Q-IND-CPA or C^cQ-IND-CCA secure. □

Next, we show that there exist KEMs that are secure as long as only classical adversaries interact with the decapsulation oracle (C^cQ-IND-CCA) but that become insecure in the post-quantum setting (Q^cQ-IND-CCA).

Proposition 5.10 (C^cQ-IND-CCA $\not\Longrightarrow$ Q^cQ-IND-CCA)**.** *Let $\mathcal{K}$ be a* C^cQ-IND-CCA *secure KEM and $\mathcal{K}_{\mathcal{BD}}$ be a* C-IND-CPA *secure KEM that is not* Q-OW-CPA*. Then there exists a KEM $\mathcal{K}'$ that is* C^cQ-IND-CCA *secure but not* Q^cQ-IND-CCA *secure.*

The idea of this separation is to include a backdoor in a KEM which is only available if the first-stage adversary has access to local quantum computing power. In particular, we encapsulate a secret value in a ciphertext attached to the public key and modify the decapsulation oracle to return the secret key if queried on the secret value. A quantum adversary can easily recover the secret, while it is hidden to classical adversaries due to the OW-CPA security of the extra ciphertext. If the adversary becomes quantum only in the second stage, it can not recover the secret, since the decapsulation oracle is no longer available.

Proof. Similarly to the proofs in Section 5.1.2.1, we construct a degenerated KEM $\mathcal{K}' = (\mathsf{KeyGen}', \mathsf{Encaps}', \mathsf{Decaps}')$, which is $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ secure, but not $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure. $\mathcal{K}'$ is described in Figure 5.12. Moreover, we define $\mathcal{K}_{\mathcal{BD}} = (\mathsf{KeyGen}_{\mathcal{BD}}, \mathsf{Encaps}_{\mathcal{BD}}, \mathsf{Decaps}_{\mathcal{BD}})$ to be a $\mathsf{C\text{-}OW\text{-}CPA}$ secure KEM which can be broken with local quantum power. An example is again the RSA-OAEP based KEM.

We start by showing that $\mathcal{K}'$ is $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ secure. Assume it is not, i.e., there exists an efficient $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ adversary $\mathcal{A}$ that can break the $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}'$. Then there exist an adversary $\mathcal{B}$ that can break the $\mathsf{C^cQ\text{-}IND\text{-}CCA}$-security of $\mathcal{K}$ as we show next. The adversary $\mathcal{B}$ receives its challenge, say, $(\mathsf{pk}, c^*, \kappa_b^*)$. It runs steps 2-3 of KeyGen' by itself, and sends $((\mathsf{pk}, \mathsf{pk}_{\mathcal{BD}}, c_{\mathcal{BD}}), c^*, \kappa_b^*)$ as input to $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$. Whenever $\mathcal{A}$ (in its first stage $\mathcal{A}_1$) queries the decapsulation oracle $\mathcal{O}_{D'}$ on some ciphertext $c \neq k_{\mathcal{BD}}$, algorithm $\mathcal{B}$ forwards the query to its own decapsulation oracle $\mathcal{O}_D$. If the adversary queries the oracle on $k_{\mathcal{B}}$, then $\mathcal{B}$ returns $\perp$. Since the KEM $\mathcal{K}_{\mathcal{BD}}$ is OW-CPA and we are still in the first phase, any query of $\mathcal{A}$ about $k_{\mathcal{BD}}$ would immediately refute the one-wayness via a black-box reduction. Hence, $\mathcal{B}$'s simulation is correct, except if $\mathcal{A}$ breaks the one-wayness of $\mathcal{K}_{\mathcal{BD}}$. This implies that $\mathcal{K}'$ is $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ secure.

Algorithm 5.7 KeyGen'

Require: -
Ensure: $(\mathsf{sk}', \mathsf{pk}')$

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
2: $(\mathsf{pk}_{\mathcal{BD}}, \mathsf{sk}_{\mathcal{BD}}) \leftarrow \mathsf{KeyGen}_{\mathcal{BD}}()$
3: $(c_{\mathcal{BD}}, k_{\mathcal{BD}}) \leftarrow \mathsf{Encaps}_{\mathcal{BD}}(\mathsf{pk}_{\mathcal{BD}})$
4: $\mathsf{pk}' \leftarrow (\mathsf{pk}, \mathsf{pk}_{\mathcal{BD}}, c_{\mathcal{BD}})$
5: $\mathsf{sk}' \leftarrow (\mathsf{sk}, k_{\mathcal{BD}})$
6: **return** $(\mathsf{sk}', \mathsf{pk}')$

Algorithm 5.8 Encaps'

Require: $(\mathsf{pk}, \mathsf{pk}_{\mathcal{BD}}, c_{\mathcal{BD}})$
Ensure: (c, k)

1: **return** $\mathsf{Encaps}(\mathsf{pk})$

Algorithm 5.9 Decaps'

Require: $(\mathsf{sk}, k_{\mathcal{BD}}), c$
Ensure: k

1: **if** $c = k_{\mathcal{BD}}$ **then**
2: **return** sk
3: **else**
4: **return** $\mathsf{Decaps}(\mathsf{sk}, c)$

Figure 5.12: Description of the separating KEM $\mathcal{K}'$ which is $\mathsf{C^cQ\text{-}IND\text{-}CCA}$ secure but not $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure

Finally, we show that $\mathcal{K}'$ is not $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure. The first-stage adversary has access to local quantum computing power. With this it can break the classically secure KEM $\mathcal{K}_{\mathcal{BD}}$ to obtain $k_{\mathcal{BD}}$ from $c_{\mathcal{BD}}$ attached to the public key. By construction of $\mathcal{K}'$, the decapsulation oracle queried on $k_{\mathcal{BD}}$ returns the secret key sk of $\mathcal{K}$, allowing

it to recover the encapsulated key. □

Moving forward, we show that post-quantum secure KEMs ($\mathsf{Q^cQ\text{-}IND\text{-}CCA}$) are not necessarily secure in the fully quantum setting where the adversary has quantum access to its decapsulation oracle ($\mathsf{Q^qQ\text{-}IND\text{-}CCA}$).

Proposition 5.11 ($\mathsf{Q^cQ\text{-}IND\text{-}CCA} \not\Rightarrow \mathsf{Q^qQ\text{-}IND\text{-}CCA}$)**.** *Assume that there exists a quantum secure family of pseudo-random permutations. Furthermore, assume there exists a* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *secure KEM* $\mathcal{K}$ *whose ciphertexts are at least* 3λ *bits long where* λ *is the security parameter of the KEM* $\mathcal{K}$*. Then there exists a KEM* $\mathcal{K}'$ *that is* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *secure but not* $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ *secure.*

Similar to the proof of Proposition 5.10, we construct a KEM where the secret key is hidden behind a problem that is hard for adversaries with classical query access and easy with quantum query access. To do so, we build a trapdoor involving the quantum-safe hidden linear structure problem that was first defined by McKague in [B10] and is based on the hidden linear structure problem [70]. This problem has constant query complexity with quantum oracle access, and exponential query complexity with classical oracle access. During our proof we split the bit representation of ciphertexts up into three parts each of size at least λ bits to hide and access the secret. Hence, ciphertexts of the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure KEM $\mathcal{K}$ have to be at least 3λ bits long.

We introduce the (quantum-safe) hidden linear structure problem and recall statements necessary for the proof of Proposition 5.11 next.

Definition 5.12 ([70])**.** *Let* $Perm(S)$ *denote the set of all permutations on a set* S*. Given oracle access to* $\mathcal{B}_{s,\pi}(x,y) = (x, \pi(y \oplus sx))$*, where* x, y, s *are elements in the Galois field with* 2^n *elements, i.e.,* $x, y, s \in GF(2^n)$*, and* $\pi \in Perm(\{0,1\}^n)$ *with* s *and* π *chosen uniformly at random. The hidden linear structure problem is to determine* s*.*

The hidden linear structure problem has constant query complexity with quantum oracle access for a $\mathsf{Q^qQ}$ adversary but it is hard for a $\mathsf{Q^cQ}$ adversary as it is required for our proof.

Theorem 5.13 ([70],[B10])**.** *The hidden linear structure problem has query complexity* $\Omega(2^{n/2})$ *for classical queries, and* 1 *for quantum queries. More specifically, there exists a quantum algorithm which solves the hidden linear structure problem with 1 query and probability 1, while any algorithm which queries the oracle classically and uses* 2^b *queries with* $2b \leq n-2$ *outputs the correct* s *with probability at most* 2^{2b-n+1}*.*

Following the idea by McKague, we use a restricted version of the hidden linear structure problem which replaces π with a pseudo-random permutation: The quantum-safe hidden linear structure problem is a hidden linear structure problem (see Definition 5.14). It is indistinguishable from the hidden linear structure problem in time d with advantage greater than δ if there exists a (d, δ)-quantum indistinguishable family of secure pseudo-random permutations as defined next. Let $\mathcal{P} = \{\pi_t : t \in \{0,1\}^k\}$ be a family of pseudo-random permutations on $\{0,1\}^{|\mathsf{sk}|}$. We say a set $\mathcal{P} = \{\pi_t : t \in \{0,1\}^k\}$ of pseudo-random permutations on the set $\{0,1\}^l$ is $(t_{\mathcal{P}}, \epsilon_{\mathcal{P}})$-quantum-indistinguishable (PRP-IND) if no quantum algorithm with run-time less than $t_{\mathcal{P}}$ can win the indistinguishability game depicted in Figure 5.13 with advantage more than $\epsilon_{\mathcal{P}}$ when using $t_{\mathcal{P}}$ quantum oracle queries.

$\text{Expt}^{\mathsf{PRP\text{-}IND}}_{\mathcal{P}}(\mathcal{A})$:
1: $a \leftarrow\!\!\$\ \{0,1\}$
2: if $a = 0$:
3: $\quad \pi \leftarrow\!\!\$\ \mathcal{P}$
4: else
5: $\quad \pi \leftarrow\!\!\$\ Perm(\{0,1\}^l)$
6: $a' \leftarrow\!\!\$\ \mathcal{A}^{\pi}$
7: return $[a' = a]$

Figure 5.13: Quantum indistinguishability experiment for a family pseudo-random permutations $\mathcal{P}$ against adversary $\mathcal{A}$

Definition 5.14. *The quantum-safe hidden linear structure problem is a hidden linear structure problem where π is drawn from a set $\mathcal{P}$ of quantum-indistinguishable pseudo-random permutations.*

Finally, we can prove the separation of $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ and $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$.

Proof of Proposition 5.11. Suppose that there exists a quantum secure pseudo-random family of permutations. Furthermore, assume there exists a $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure KEM $\mathcal{K} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ whose ciphertext c is at least 3λ bits long. We define $c.x$ to be the first λ bit, $c.y$ to be the second λ bit, and $c.z$ to be the remaining bits of c's bit representation. As mentioned before, we assume that the computation of 2^{λ} is infeasible and hence, that a probability of $2^{-\lambda}$ is negligible.

Let $\mathcal{B}_{s,t}$ be an oracle for the quantum safe hidden linear structure problem that is trying to guess s. Moreover, let $\mathcal{B}_{s,t} : GF(2^k) \times GF(2^k) \to GF(2^k) \times GF(2^k)$ be a family of functions such that, given oracle access to $\mathcal{B}_{s,t}$, at least $t_{\mathcal{B}}$ classical queries are required to determine s with probability greater than $\epsilon_{\mathcal{B}}$, whereas a single query

suffices when given access to $\mathcal{B}_{s,t}$ via a quantum oracle. In our construction, s is a secret which unlocks access to the decapsulation key sk'. The second parameter, t, needs to be secret but is otherwise not important for our application. Now we construct a degenerate KEM $\mathcal{K}' = (\mathsf{KeyGen}', \mathsf{Encaps}', \mathsf{Decaps}')$ that is $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ but not $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ secure. The KEM $\mathcal{K}'$ is defined in Figure 5.14.

First, we show the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}'$. Suppose that it takes at least $q_{\mathcal{B}}$ queries to $\mathcal{B}_{s,t}$ to determine s with probability $\epsilon_{\mathcal{B}}$, and that it takes at least t_K time for an adversary $\mathcal{A}$ to break the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}$ with probability p_K. It is important to note that $\mathcal{K}$ and s are unrelated. Hence, knowledge of the public key pk and access to the decapsulation oracle $\mathcal{O}_D^c$ does not reduce the complexity of guessing s. Likewise, access to an oracle for $\mathcal{B}_{s,t}$ does not increase the advantage of an adversary during the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ experiment of $\mathcal{K}$ as we explain next. Assume the contrary, i.e., the advantage of an adversary $\mathcal{A}$ against $\mathcal{K}$ would increase when having access to an oracle for $\mathcal{B}_{s,t}$. Then $\mathcal{A}$ could choose s, t uniformly at random, simulate $\mathcal{B}_{s,t}$, and increase the advantage against the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}$. This, however, would contradict our assumption on $\mathcal{K}$.

The above arguments imply, that the only relation between s and sk is the element $w = \mathsf{sk} \cdot \delta_{s,c.z}$ during the decapsulation of $\mathcal{K}'$, where $\delta_{s,c.z}$ is the *Kronecker delta* over s and the least bits of the cipher c. However, $\mathcal{A}$ only learns information from w, if $w \neq 0$ and thus, $w = \mathsf{sk}$. By definition of $\delta_{s,c.z}$ this happens if and only if the input c to $\mathcal{O}_{D'}$ is such that $c.z = s$. But this is only possible if $\mathcal{A}$ solves the quantum safe hidden linear structure problem or guesses s directly. By Lemma 5.15, the following equation holds

$$\mathrm{Adv}_{\mathcal{K}'}^{\mathsf{Q^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}_1, \mathcal{A}_2) \leq \mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}_1, \mathcal{A}_2) + \epsilon_{\mathcal{B}} + min\{q_{\mathcal{B}}, t_{\mathcal{K}}\} \cdot 2^{-\lambda}.$$

Assuming the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}$, the advantage $\mathrm{Adv}_{\mathcal{K}}^{\mathsf{Q^cQ\text{-}IND\text{-}CCA}}(\mathcal{A}_1, \mathcal{A}_2)$ is small. Furthermore, since $\mathcal{A}$ only has classical access $\epsilon_{\mathcal{B}}$ and because of Theorem 5.13, $\epsilon_{\mathcal{B}}$ is small as well. Hence, $\mathcal{K}'$ is $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure.

We move on to show that $\mathcal{K}'$ is not $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ secure. By Theorem 5.13, the quantum safe hidden linear structure problem is in fact solvable in one quantum query to a $\mathcal{B}_{s,t}$ oracle. We construct a suitable quantum oracle that gives access to $\mathcal{B}_{s,t}$ using one call to the decapsulation oracle. For convenience we call this oracle $\mathcal{B}_{s,t}(|c, 0, 0, 0, 0, 0, \theta, \eta, z\rangle)$ on input of the form $c' = |c, 0, 0, 0, 0, 0, \theta, \eta\rangle$. It is given in Figure 5.15. The oracle returns $|c, 0, 0, 0, 0, 0, \theta \oplus u, \eta \oplus v\rangle$, where by construction of $\mathcal{O}_{D'}^{\mathsf{q}}$, it holds that $(u, v) = \mathcal{B}_{s,t}(c.x, c.y)$. Hence, the decapsulation oracle can be turned into a $\mathcal{B}_{s,t}$ oracle. Thus, an adversary with quantum access can determine the secret s by solving the quantum safe hidden linear structure problem. Afterwards the adversary asks its decapsulation on a ciphertext c with $c.z = s$ and hence, recovers the decapsulation key sk. Thus, the adversary can then win the $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ game, i.e., $\mathcal{K}'$ is not $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ secure. □

Algorithm 5.10 KeyGen'

Require: -
Ensure: $\mathsf{pk}', \mathsf{sk}'$

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
2: $s \leftarrow_\$ \{0,1\}^{2\lambda}$
3: $t \leftarrow_\$ \{0,1\}^{2\lambda}$
4: $\mathsf{sk}' \leftarrow (\mathsf{sk}, s, t)$
5: **return** $(\mathsf{pk}, \mathsf{sk}')$

Algorithm 5.11 Encaps'

Require: pk
Ensure: (c, k)

1: $(c, k) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
2: **return** (c, k)

Algorithm 5.12 Decaps'

Require: $(\mathsf{sk}, s, t), c$
Ensure: $(k, (u, v), w)$

1: $k \leftarrow \mathsf{Decaps}(\mathsf{sk}, c)$
2: $(u, v) \leftarrow \mathcal{B}_{s,t}(c.x, c.y)$
3: $w \leftarrow \mathsf{sk} \cdot \delta_{s,c.z}$
4: **return** $(k, (u, v), w)$

Figure 5.14: Description of the separating KEM $\mathcal{K}'$ which is $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure but not $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ secure

Quantum oracle for $\mathcal{B}_{s,t}(|c,0,0,0,0,\theta,\eta,z\rangle)$:
1: $|c,k,u,v,w,z\rangle \leftarrow \mathcal{O}^{\mathsf{q}}_{D'}(|c,0,0,0,0,0\rangle)$
2: return $|c,0,0,0,0,\theta \oplus u, \eta \oplus v, z\rangle$

$\mathsf{Decaps}'^{\perp}(\mathsf{sk}, c, c^*)$:
1: if $c = c^*$: return $\perp$
2: else: return $(k,(u,v),w) \leftarrow \mathsf{Decaps}'(\mathsf{sk}, c)$

$\mathcal{O}^{\mathsf{q}}_{D'}(|c,\alpha,\beta,\gamma,\epsilon\rangle)$:
1: $q_D \leftarrow q_D + 1$
2: return $\sum_{c,t,z} \psi_{c,t,z} |c, t \oplus \mathsf{Decaps}'^{\perp}(\mathsf{sk}, c, c^*), z\rangle$

$\mathcal{O}^{\mathsf{c}}_{D'}(c)$:
1: $q_D \leftarrow q_D + 1$
2: return $\mathsf{Decaps}'^{\perp}(\mathsf{sk}, c, c^*)$

Figure 5.15: Description of classical and quantum oracles for the separating KEM $\mathcal{K}'$ that is $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ secure, but not $\mathsf{Q^qQ\text{-}IND\text{-}CCA}$ secure

Lemma 5.15. *Suppose that it takes at least $q_{\mathcal{B}}$ queries to $\mathcal{B}_{s,t}$ to determine s with probability $\epsilon_{\mathcal{B}}$, and that it takes at least $t_{\mathcal{K}}$ time for an adversary $\mathcal{A}$ to break the* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *security of $\mathcal{K}$ with probability $\epsilon_{\mathcal{K}}$. If $\mathcal{A}$ has access to a classical oracle $\mathcal{O}^{\mathsf{q}}_{D'}(c)$, knows pk', and runs for time $t < min\{q_{\mathcal{B}}, t_{\mathcal{K}}\}$, then $\mathcal{A}$ breaks the* $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ *security of $\mathcal{K}'$ with probability at most $\epsilon \leq \epsilon_{\mathcal{B}} + \epsilon_{\mathcal{K}} + 2^{-\lambda}t$.*

Proof. Suppose that it takes at least $q_{\mathcal{B}}$ queries to $\mathcal{B}_{s,t}$ to determine s with probability $\epsilon_{\mathcal{B}}$. Assume furthermore that it takes at least $t_{\mathcal{K}}$ time to break the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ security of $\mathcal{K}$ with probability ϵ_K. Moreover, assume an adversary $\mathcal{A}$ has access to a classical oracle $\mathcal{O}^{\mathsf{c}}_{D'}(c)$, knows pk', and runs for time $t < min\{q_{\mathcal{B}}, t_{\mathcal{K}}\}$. Since $t < q_{\mathcal{B}}$, $\mathcal{A}$ has to have made fewer than $q_{\mathcal{B}}$ queries. Hence, $\mathcal{A}$ learns s with probability at most $\epsilon_{\mathcal{B}}$ by assumption. The probability of learning s by guessing is at most $2^{-\lambda}t$. So the decapsulation oracle $\mathcal{O}^{\mathsf{c}}_{D'}(c)$ returns an answer of the form $(\cdot,\cdot,\cdot,\mathsf{sk})$ (and hence $\mathcal{A}$ breaks $\mathcal{K}$) with no more than $\epsilon_{\mathcal{B}} + 2^{-\lambda}t$. Since $t < t_K$, the probability that $\mathcal{A}$ distinguishes between a real and random key to win the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ game for $\mathcal{K}$ is at most $\epsilon_{\mathcal{K}}$, unless one of the above cases applies. Distinguishing a real or random key for $\mathcal{K}'$ implies distinguishing a real or random

key for $\mathcal{K}$. Thus the probability that $\mathcal{A}$ wins the $\mathsf{Q^cQ\text{-}IND\text{-}CCA}$ game for $\mathcal{K}'$ is at most $\epsilon \leq \epsilon_{\mathcal{B}} + \epsilon_{\mathcal{K}} + 2^{-\lambda}t$. □

Lastly, we show that IND-CPA security in the quantum setting is not necessarily enough to show IND-CCA security in the fully classical setting.

Proposition 5.16 ($\mathsf{Q\text{-}IND\text{-}CPA} \not\Longrightarrow \mathsf{C^cC\text{-}IND\text{-}CCA}$)**.** *Assume there exists a* $\mathsf{Q\text{-}IND\text{-}CPA}$ *secure KEM* $\mathcal{K}$*. Then there exists a KEM* $\mathcal{K}'$ *that is* $\mathsf{Q\text{-}IND\text{-}CPA}$ *secure but not* $\mathsf{C^cC\text{-}IND\text{-}CCA}$ *secure.*

Proposition 5.16 is proven by constructing a degenerated KEM $\mathcal{K}'$ from a $\mathsf{Q\text{-}IND\text{-}CPA}$ secure KEM $\mathcal{K}$ where the decapsulation of $\mathcal{K}'$ returns the secret key when given the public key as input. This is clearly still secure in the $\mathsf{Q\text{-}IND\text{-}CPA}$ setting but not $\mathsf{C^cC\text{-}IND\text{-}CCA}$ secure.

Proof. Let $\mathcal{K} = (\mathsf{KeyGen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a $\mathsf{Q\text{-}IND\text{-}CPA}$ secure KEM, then we can construct a KEM $\mathcal{K}'$ that is $\mathsf{Q\text{-}IND\text{-}CPA}$ but not $\mathsf{C^cC\text{-}IND\text{-}CCA}$ secure. Let the KEM $\mathcal{K}' = (\mathsf{KeyGen}', \mathsf{Encaps}', \mathsf{Decaps}')$ be defined as in Figure 5.16.

Clearly $\mathcal{K}'$ is not $\mathsf{C^cC\text{-}IND\text{-}CCA}$ secure since it is broken as soon as the public key is asked as a ciphertext to the decapsulation oracle of $\mathcal{K}'$. However, as long as no queries are allowed to the decapsulation oracle, an adversary cannot distinguish $\mathcal{K}$ and $\mathcal{K}'$. Hence, $\mathcal{K}'$ is $\mathsf{Q\text{-}IND\text{-}CPA}$ secure. □

Algorithm 5.13 KeyGen'

Require: -
Ensure: pk, sk

1: $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}()$
2: **return** $(\mathsf{pk}, \mathsf{sk})$

Algorithm 5.14 Encaps'

Require: pk
Ensure: c, k

1: $(c, k) \leftarrow \mathsf{Encaps}(\mathsf{pk})$
2: **return** (c, k)

Algorithm 5.15 Decaps'

Require: sk, c
Ensure: k

1: **if** $c = \mathsf{pk}$ **then**
2: **return** sk
3: **else**
4: **return** $\mathsf{Decaps}(sk, c)$

Figure 5.16: Description of the separating KEM $\mathcal{K}'$ which is $\mathsf{Q\text{-}IND\text{-}CPA}$ secure but not $\mathsf{C^cC\text{-}IND\text{-}CCA}$ secure

5.2 Hybrid Signature Schemes

We now discuss the use of robust combiners to construct hybrid signature schemes. Informally, we call a signature combiner *robust* if the resulting signature scheme is ($\mathsf{X^yZ\text{-}EUF\text{-}CMA}$) secure if at least one of the candidate signature schemes is ($\mathsf{X^yZ\text{-}EUF\text{-}CMA}$) secure. In 2005, Harnik et al. [119] introduced so-called (k, n)-robust

combiners, that, for an arbitrary primitive $\mathcal{P}$ combine n input candidate schemes in such a manner that the combined scheme $\mathcal{C}$ is a secure and efficient implementation of $\mathcal{P}$ as long as k out of the n schemes are secure implementations.

In this section, we present three different signature combiners. The first combiner is a fairly natural construction; the second and third combiner are motivated by practical applications of hybrid signatures. The first combiner, Con, simply concatenates the two signatures generated by each of the two combined signature schemes. The second combiner, sNest, concatenates again two signatures were the second one signs the message and the first signature. It is motivated to be used as a backwards compatible hybrid certificate in S/MIME. The third combiner, dNest, concatenates nested signatures of two different messages. It is motivated to be used as backwards compatible hybrid certificate in X.509v3. We summarize all combiners and their security properties depending on the security of the combined schemes in Table 5.1. It is important to recall that $\max\{\mathsf{X}^{\mathsf{y}}\mathsf{Z}, \mathsf{U}^{\mathsf{v}}\mathsf{W}\}$ denotes the stronger unforgeability notion with respect to the natural hierarchy of security notions shown in Figure 5.8.

Combiner	Hybrid signature $\mathsf{s} = (\mathsf{s}_1 \| \mathsf{s}_2)$	Unforgeability	Application
Single-message combiners			
Con	$\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m})$; $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, \mathsf{m})$	$\max\{\mathsf{X}^{\mathsf{y}}\mathsf{Z}, \mathsf{U}^{\mathsf{v}}\mathsf{W}\}$	X.509v3, TLS 1.3, S/MIME
sNest	$\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m})$; $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}, \mathsf{s}_1))$	$\max\{\mathsf{X}^{\mathsf{c}}\mathsf{Z}, \mathsf{U}^{\mathsf{c}}\mathsf{W}\}$	S/MIME
Dual-message combiners			
dNest	$\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m}_1)$; $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2))$	$\max\{\mathsf{X}^{\mathsf{c}}\mathsf{Z}, \mathsf{U}^{\mathsf{c}}\mathsf{W}\}$	X.509v3

Unforgeability: If $\mathcal{S}_1$ is $\mathsf{X}^{\mathsf{y}}\mathsf{Z}$-EUF-CMA and $\mathcal{S}_2$ is $\mathsf{U}^{\mathsf{v}}\mathsf{W}$-EUF-CMA, then $\mathcal{C}[\mathcal{S}_1, \mathcal{S}_2]$ is ...-EUF-CMA.

Table 5.1: Signature combiners using signature schemes $\mathcal{S}_1$ and $\mathcal{S}_2$

Throughout this section we let $\mathcal{S}_1 = (\mathsf{KeyGen}_1, \mathsf{Sign}_1, \mathsf{Verify}_1)$ and $\mathcal{S}_2 = (\mathsf{KeyGen}_2, \mathsf{Sign}_2, \mathsf{Verify}_2)$ be two signature schemes. Furthermore, we denote signature schemes that are constructed by one of our three proposals $\mathcal{C} \in \{\mathsf{Con}, \mathsf{sNest}, \mathsf{dNest}\}$ as $\mathcal{C}[\mathcal{S}_1, \mathcal{S}_2] = (\mathsf{KeyGen}_{\mathcal{C}}, \mathsf{Sign}_{\mathcal{C}}, \mathsf{Verify}_{\mathcal{C}})$. In all our schemes, $\mathsf{KeyGen}_{\mathcal{C}}$ simply returns the concatenation of the two public keys ($\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$) and the two secret keys ($\mathsf{sk} \leftarrow (\mathsf{sk}_1, \mathsf{sk}_2)$). The verification in each case is defined in the natural way. That is, the hybrid signature s is accepted if both s_1 and s_2 are accepted. Otherwise, s is rejected.

5.2.1 Con: Concatenation Combiner

We start with the concatenation combiner.

5.2.1.1 Description and Security of Con

The combiner Con is the trivial combiner which just places independent signatures from the two schemes side-by-side as depicted in Algorithm 5.16.

Algorithm 5.16 Signature generation of $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$

Require: Message m and secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$
Ensure: Signature s

1: $\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m})$
2: $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, \mathsf{m})$
3: **return** $\mathsf{s} \leftarrow (\mathsf{s}_1 \| \mathsf{s}_2)$

We can now show that the concatenation combiner is a robust signature combiner, in the sense that the resulting signature is as secure as the strongest of the two input signature schemes. In particular, we show in Theorem 5.17 that $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$ is EUF-CMA secure in the fully quantum setting ($\mathsf{Q^qQ}$) if at least one of the two signature schemes is secure against fully quantum adversaries.

Theorem 5.17 (Unforgeability of Con)**.** *Let $\mathcal{S}_1$ be $\mathsf{X^yZ}$-EUF-CMA secure, $\mathcal{S}_2$ be $\mathsf{U^vW}$-EUF-CMA secure, and $\mathsf{R^sT} = \max\{\mathsf{X^yZ}, \mathsf{U^vW}\}$. Then $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$ is $\mathsf{R^sT}$-EUF-CMA secure. More precisely, for any EUF-CMA adversary $\mathcal{A}$ of type $\mathsf{R^sT}$ against the combiner $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$, we derive efficient adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathsf{Con}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \;\leq\; \min\left\{\mathsf{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\right\},$$

while the run-times of $\mathcal{B}_1$ and $\mathcal{B}_2$ are approximately the same as the run-time of $\mathcal{A}$.

Proof. Suppose $\mathcal{A}$ is an $\mathsf{R^sT}$-EUF-CMA adversary that finds a forgery in $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$– in other words, it outputs $q_S + 1$ valid signatures under $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$ on distinct messages. We can construct an $\mathsf{R^sT}$-EUF-CMA algorithm $\mathcal{B}_1$ that finds a forgery in $\mathcal{S}_1$. $\mathcal{B}_1$ interacts with an $\mathsf{R^sT}$-EUF-CMA challenger for $\mathcal{S}_1$ which provides a public key pk_1. $\mathcal{B}_1$ generates a key pair $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{KeyGen}_2()$ and sets the public key for $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$ to be $(\mathsf{pk}_1, \mathsf{pk}_2)$. When $\mathcal{A}$ asks for $\sum_{\mathsf{m},t,z} \psi_{\mathsf{m},t,z} |\mathsf{m}, t, z\rangle$ to be signed using $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$, we treat t as consisting of two registers $t_1 \| t_2$. $\mathcal{B}_1$ proceeds by passing the m, t_1, and z registers to its signing oracle for $\mathcal{S}_1$. Moreover, $\mathcal{B}_1$ runs the quantum signing operation from Figure 2.3 for Sign_2 on the m, t_2, and z registers.

There is a one-to-one correspondence between $\mathcal{A}$'s queries to its signing oracle and $\mathcal{B}_1$'s queries to its signing oracle.

If $\mathcal{S}_1$ is proven to be secure in the (Q)ROM (rather than the standard), then this proof of $\mathsf{Con}[\mathcal{S}_1, \mathcal{S}_2]$ also proceeds in the (Q)ROM: $\mathcal{B}_1$ relays $\mathcal{A}$'s hash oracle queries directly to its oracle, giving a one-to-one correspondence between $\mathcal{A}$'s queries to its hash oracle and $\mathcal{B}_1$'s queries to its hash oracle. This holds in either the ROM or QROM.

If $\mathcal{A}$ wins the $\mathsf{R^sT\text{-}EUF\text{-}CMA}$ game, then it has returned $q_S + 1$ valid signatures $\mathsf{s}_i = (\mathsf{s}_{i,1}, \mathsf{s}_{i,2})$ on distinct messages m_i such that $\mathsf{Verify}_1(\mathsf{pk}_1, \mathsf{m}_i, \mathsf{s}_{i,1}) = 0$ and $\mathsf{Verify}_2(\mathsf{pk}_2, \mathsf{m}_i, \mathsf{s}_{i,2}) = 0$. $\mathcal{B}_1$ can extract from this $q_S + 1$ valid signatures under $\mathcal{S}_1$ on distinct messages. Thus,

$$\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathsf{Con}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1).$$

Similarly it holds for $\mathcal{S}_2$ that

$$\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathsf{Con}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2).$$

It, hence, follows that

$$\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathsf{Con}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \min\{\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\}.$$

Thus, if either $\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1)$ or $\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)$ is small, then so too is $\mathrm{Adv}^{\mathsf{R^sT\text{-}EUF\text{-}CMA}}_{\mathsf{Con}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A})$. □

5.2.1.2 Application of Con

After describing the combiner Con and proving its unforgeability, we elaborate on possible applications next. According to Herath and Stebila, it can be applied in the standard X.509 standard version 3, TLS 1.3, and S/MIME.

Dual X.509v3 Certificates. The concatenation signature combiner provides the simplest approach to use hybrid X.509v3 certificates by creating separate certificates, e.g., one for the traditional algorithm and the other for the post-quantum algorithms. This approach leaves the task of conveying the "hybrid" certificate (actually, two certificates) to the application, which will suffice in some settings (e.g., in S/MIME and some TLS settings that we explain below), but is unsatisfactory in others. Hence, it is not backwards compatible in general.

Hybrid Certificates in TLS. The concatenation signature combiner could be used in a straightforward way in the *post-handshake authentication* mode of TLS 1.3 [194] described briefly next. In the current draft of TLS 1.3, a *post-handshake*

authentication mode for clients [194, Section 4.5.2], where clients can be requested to (further) authenticate using a certificate for a given algorithm, is proposed. This would allow client authentication using two (or more) signature schemes, e.g., a classical and a post-quantum signature. Each client signature in this draft is over the same handshake context data structure. Hence, this approach is not backwards compatible but relies on the proposal for TLS 1.3.

Parallel `SignerInfos` in S/MIME. CMS [124] is the main cryptographic component of S/MIME [189], which enables PKEs and digital signatures for e-mail. An S/MIME signed e-mail consists of a header and the body. The header is used to specify the algorithms used. The body of the e-mail is divided into the body to be signed and an encoding of a CMS `SignedData` object. The `SignedData` object in turn contains several fields, including a set of certificates and a set of `SignerInfo` objects. To construct a hybrid signature in S/MIME with the Con combiner, the certificate for each algorithm can be added in the `SignedData` object's set of certificates (with no need for hybrid certificates), and then include `SignerInfo` objects for the signature from each algorithm. As before, this approach including the Con combiner is not backwards compatible with software that is not able to process post-quantum signatures.

5.2.2 sNest: Strong Nesting Combiner

After explaining the concatenation combiner, we now elaborate on the so-called *strong nesting combiner*.

5.2.2.1 Description and Security of sNest

For this combiner, the second signature scheme signs both the message and the signature from the first signature scheme as depicted in Algorithm 5.17.

It is important to note that in order to construct a robust combiner in the sense that the resulting signature is as secure as the strongest of the two input signature schemes, the second signature should be computed over the message $(\mathsf{m}, \mathsf{s}_1)$. We explain why this is necessary next. In the following explanation, we consider the case where $\mathcal{S}_1$ is broken. Let $(\mathsf{s}_1, \mathsf{s}_2)$ be a signature for m with $\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m})$ and $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, \mathsf{s}_1)$. Then, an adversary might be able to exchange m for another different message m' such that $\mathsf{s}_1 = \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m}')$, since $\mathcal{S}_1$ is broken. However, $(\mathsf{s}_1, \mathsf{s}_2)$ would be accepted as a hybrid signature for $\mathsf{m}' \neq \mathsf{m}$ by the verification algorithm. This scenario is not possible if $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}, \mathsf{s}_1))$ as in Algorithm 5.17.

We can now show that the strong nesting combiner is a robust signature combiner. In particular, we show in Theorem 5.18 that $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ is EUF-CMA secure in

Algorithm 5.17 Signature generation of $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$

Require: Message m and secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$
Ensure: Signature s

1: $\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m})$
2: $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}, \mathsf{s}_1))$
3: **return** $\mathsf{s} \leftarrow (\mathsf{s}_1 \| \mathsf{s}_2)$

the post-quantum setting ($\mathsf{Q^cQ}$) if at least one of the two signature schemes is secure against post-quantum adversaries.

Theorem 5.18 (Unforgeability of sNest)**.** *Let* $\mathcal{S}_1$ *be* $\mathsf{X^cZ\text{-}EUF\text{-}CMA}$ *secure,* $\mathcal{S}_2$ *be* $\mathsf{U^cW\text{-}EUF\text{-}CMA}$ *secure, and* $\mathsf{R^cT} = \max\{\mathsf{X^cZ}, \mathsf{U^cW}\}$*. Then* $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ *is* $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ *secure. More precisely, for any EUF-CMA adversary* $\mathcal{A}$ *of type* $\mathsf{R^cT}$ *against* $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$*, we derive efficient adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{sNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \;\leq\; \min\left\{\mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\right\},$$

while the run-times of $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *are approximately the same as the run-time of* $\mathcal{A}$*.*

The proof follows the same approach as the proof of Theorem 5.17: An $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ adversary that finds a forgery in $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ is also an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ forger for $\mathcal{S}_1$ and $\mathcal{S}_2$.

Proof. Suppose $\mathcal{A}$ is an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ adversary that finds a forgery in $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$– in other words, it outputs $q_S + 1$ valid signatures under $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ on distinct messages. We can construct an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ algorithm $\mathcal{B}_1$ that finds a forgery in $\mathcal{S}_1$ and an $\mathsf{R^cT}$ algorithm $\mathcal{B}_2$ that finds a forgery in $\mathcal{S}_2$. We prove the case for $\mathcal{B}_1$ next, the case for $\mathcal{B}_2$ follows the same approach.

$\mathcal{B}_1$ interacts with an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ challenger for $\mathcal{S}_1$ which provides a public key pk_1. $\mathcal{B}_1$ generates a key pair $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{KeyGen}_2()$ and sets the public key for $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ to be $(\mathsf{pk}_1, \mathsf{pk}_2)$. When $\mathcal{A}$ asks for m to be signed using $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$, $\mathcal{B}_1$ proceeds by passing m to its signing oracle for $\mathcal{S}_1$ which returns a signature s_1 for m_1. Afterwards, $\mathcal{B}_1$ runs the signing operation for Sign_2 on the message $(\mathsf{m}, \mathsf{s}_1)$. There is a one-to-one correspondence between $\mathcal{A}$'s queries to its signing oracle and $\mathcal{B}_1$'s queries to its signing oracle.

As before in the proof of Theorem 5.17, if $\mathcal{S}_1$ is proven to be secure in the (Q)ROM, then this proof of $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$ also proceeds in the (Q)ROM: $\mathcal{B}_1$ relays $\mathcal{A}$'s hash oracle queries directly to its random oracle, giving a one-to-one correspondence between $\mathcal{A}$'s queries to its hash oracle and $\mathcal{B}_1$'s queries to its hash oracle.

If $\mathcal{A}$ wins the $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ game, then it has returned $q_S + 1$ valid signatures $\mathsf{s}_i = (\mathsf{s}_{i,1}, \mathsf{s}_{i,2})$ on distinct messages m_i such that $\mathsf{Verify}_1(\mathsf{pk}_1, \mathsf{m}_i, \mathsf{s}_{i,1}) = 0$ and

$\mathsf{Verify}_2(\mathsf{pk}_2, (\mathsf{m}_i, \mathsf{s}_{i,1}), \mathsf{s}_{i,2}) = 0$. $\mathcal{B}_1$ can extract from this $q_S + 1$ valid signatures under $\mathcal{S}_1$ on distinct messages. Thus,

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{sNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1).$$

All steps described above can similarly be done by $\mathcal{B}_2$. Only the simulation of $\mathcal{A}$'s signing queries is slightly different: When $\mathcal{A}$ asks for m to be signed using $\mathsf{sNest}[\mathcal{S}_1, \mathcal{S}_2]$, $\mathcal{B}_2$ first runs the signing operation for Sign_1 on the message m using its secret key sk_1. Afterwards, $\mathcal{B}_2$ passes $(\mathsf{m}, \mathsf{s}_1)$ to its signing oracle for $\mathcal{S}_2$ which returns a signature s_2. As previously for $\mathcal{B}_1$, $\mathcal{B}_2$ can extract $q+1$ signature-message pairs with distinct messages $(\mathsf{m}_i, \mathsf{s}_{i,1})$. Hence, it holds for $\mathcal{S}_2$:

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{sNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2).$$

It follows that

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{sNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \min\left\{\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\right\}.$$

Thus, if at least one of $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1)$ and $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)$ is small, so too is $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{sNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A})$. □

5.2.2.2 Application of sNest in S/MIME

We now move on to explain Herath and Stebila's approach to apply the combiner sNest in S/MIME. The goal of the approach is to include a hybrid signature such that the implementation is backwards compatible.

As explained in Section 5.2.1.2, the `SignedData` object of an e-mail signed with S/MIME contains a set of `SignerInfo` objects. Each `SignerInfo` object contains a signer identifier, algorithm identifier, signature, and optional signed and unsigned attributes. To utilize the nested signature combiner, these optional attributes in the `SignerInfo` object are used to embed a second signature. There exists three different approaches based on the description of the standard:

2.a) Put a second certificate in the set of certificates, and put a second `SignerInfo` in an attribute of the first `SignerInfo`.
2.b) Put a hybrid certificate in the set of certificates, and put a second `SignerInfo` in an attribute of the first `SignerInfo`.
2.c) Put a second `SignedData` in an attribute of the first `SignerInfo`.

These approaches require defining a new attribute type, but this is easily done. If the extra data is put in the *signed* attribute of the first `SignerInfo` then the strong nesting combiner is used (if the extra data is put in the *unsigned* attribute of the first `SignerInfo`, then the concatenation combiner Con is used).

For honest parties that can recognize the optional attributes, the security relies on the classical and post-quantum security. Another advantage lies in its backwards compatibility: The CMS standard indicates that verifiers can accept signatures with unrecognized attributes, so this approach results in backwards-compatible signatures that should be accepted by existing software. Herath and Stebila tested five S/MIME libraries/applications for acceptance of S/MIME messages from the three approaches above. The tested libraries were Apple Mail 10.2, BouncyCastle 1.56 with Java SE 1.8.0_131, Microsoft Outlook 2016, Mozilla Thunderbird 45.7.1, and OpenSSL 1.0.2k.

They conclude that approach 1 is not fully backwards-compatible since only Apple Mail accepted e-mails in this case. Moreover, they observed that all the tested libraries support approaches 2.a–2.c. However, Thunderbird struggled with very large attributes. This implies that qTESLA might be a candidate to be used in hybrid certificates. TESLA, in contrast, might not be suitable due to its large public keys.

5.2.3 dNest: Dual Message Combiner Using Nesting

Moving forward, we now introduce the dual message combiner using nesting.

5.2.3.1 Description and Security of dNest

Some applications require a combiner for two (possibly related) messages signed with two signature schemes. For example, in an application of X.509 certificates one certificate is signed with $\mathcal{S}_1$ and then embedded as an extension inside a second certificate signed with $\mathcal{S}_2$. We depict the signature generation of our nesting combiner dNest using two messages $\mathsf{m}_1, \mathsf{m}_2$ in Algorithm 5.18.

Similarly to the case of our strong nesting combiner sNest described in Section 5.2.2, it is important to note that in order to construct a robust combiner, the second signature should be computed over the message $(\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2)$. In particular, is is important that m_1 is part of the message-to-be signed by $\mathcal{S}_2$ as explained next. As before, we consider the case where $\mathcal{S}_1$ is broken. Let $(\mathsf{s}_1, \mathsf{s}_2)$ be a signature for $(\mathsf{m}_1, \mathsf{m}_2)$ with $\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m}_1)$ and $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{s}_1, \mathsf{m}_2))$. Then, an adversary might be able to exchange m_1 for another different message m'_1 such that $\mathsf{s}_1 = \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m}'_1)$, since $\mathcal{S}_1$ is broken. However, $(\mathsf{s}_1, \mathsf{s}_2)$ would be accepted as a hybrid signature for $(\mathsf{m}'_1, \mathsf{m}_2) \neq (\mathsf{m}_1, \mathsf{m}_2)$ by the verification algorithm. This is not possible if $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2))$ as stated in Algorithm 5.18.

Algorithm 5.18 Signature generation of $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$

Require: Messages m_1 and m_2, and secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$
Ensure: Signature s

1: $\mathsf{s}_1 \leftarrow \mathsf{Sign}_1(\mathsf{sk}_1, \mathsf{m}_1)$
2: $\mathsf{s}_2 \leftarrow \mathsf{Sign}_2(\mathsf{sk}_2, (\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2))$
3: **return** $\mathsf{s} \leftarrow (\mathsf{s}_1 \| \mathsf{s}_2)$

We can now show that the nesting combiner of dual messages is a robust signature combiner in the sense that the resulting signature is as secure as the strongest of the two input signature schemes. In particular, we show in Theorem 5.19 that $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ is EUF-CMA secure in the post-quantum setting ($\mathsf{Q^cQ}$) if at least one of the two signature schemes is secure against post-quantum adversaries.

Theorem 5.19 (Unforgeability of dNest)**.** *Let* $\mathcal{S}_1$ *be* $\mathsf{X^cZ}$-EUF-CMA *secure,* $\mathcal{S}_2$ *be* $\mathsf{U^cW}$-EUF-CMA *secure, and* $\mathsf{R^cT} = \max\{\mathsf{X^cZ}, \mathsf{U^cW}\}$*, then* $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ *is* $\mathsf{R^cT}$-EUF-CMA *secure. More precisely, for any EUF-CMA adversary* $\mathcal{A}$ *of type* $\mathsf{R^cT}$ *against* $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$*, we derive efficient adversaries* $\mathcal{B}_1$ *and* $\mathcal{B}_2$ *such that*

$$\mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{dNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \;\leq\; \min\left\{\mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\right\},$$

while $\mathcal{B}_1$*'s and* $\mathcal{B}_2$*'s run-times are approximately the same as* $\mathcal{A}$*'s run-time.*

The proof follows the same approach as the proof of Theorem 5.17: An $\mathsf{R^cT}$-EUF-CMA adversary that finds a forgery in $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ is also an $\mathsf{R^cT}$-EUF-CMA forger for $\mathcal{S}_1$ and $\mathcal{S}_2$.

Proof. To prove the statement, we construct adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that break the unforgeability of $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively, using an adversary $\mathcal{A}$ against the unforgeability of $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$.

We start with the construction of $\mathcal{B}_1$. Suppose $\mathcal{A}$ is a $\mathsf{R^cT}$-EUF-CMA algorithm that outputs a forgery for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$–in other words, it outputs $q_S + 1$ valid signatures under $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ on distinct messages. We can construct an $\mathsf{R^cT}$-EUF-CMA algorithm $\mathcal{B}_1$ that finds a forgery in $\mathcal{S}_1$. $\mathcal{B}_1$ interacts with an $\mathsf{R^cT}$-EUF-CMA challenger for $\mathcal{S}_1$ which provides a public key pk_1. $\mathcal{B}_1$ generates a key pair $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{KeyGen}_2()$ and sets the public key for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ to be $(\mathsf{pk}_1, \mathsf{pk}_2)$. When $\mathcal{A}$ asks classically for $(\mathsf{m}_1, \mathsf{m}_2)$ to be signed using $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$, $\mathcal{B}_1$ proceeds by passing the m_1 to its signing oracle for $\mathcal{S}_1$, receiving the signature s_1 of m_1. Then $\mathcal{B}_1$ runs Sign_2 on the message $(\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2)$ using its secret key sk_2. There is a one-to-one correspondence between $\mathcal{A}$'s queries to its signing oracle for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ and $\mathcal{B}_1$'s queries to its signing oracle for $\mathcal{S}_1$.

As before in the proof of Theorem 5.17, if $\mathcal{S}_1$ is proven to be secure in the (Q)ROM, then this proof of $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ also proceeds in the (Q)ROM: $\mathcal{B}_1$ relays $\mathcal{A}$'s random oracle queries directly to its random oracle, giving a one-to-one correspondence between $\mathcal{A}$'s queries to its hash oracle and $\mathcal{B}_1$'s queries to its hash oracle.

If $\mathcal{A}$ wins the $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ game, then it has returned $q_S + 1$ distinct tuples $(\mathsf{m}_{1,i}, \mathsf{m}_{2,i}, \mathsf{s}_{1,i}, \mathsf{s}_{2,i})$ such that the following holds true $\mathsf{Verify}_1(\mathsf{pk}_1, \mathsf{m}_{1,i}, \mathsf{s}_{1,i}) = 0$ and $\mathsf{Verify}_2(\mathsf{pk}_2, (\mathsf{m}_{1,i}, \mathsf{s}_{1,i}, \mathsf{m}_{2,i}), \mathsf{s}_{2,i}) = 0$. Hence, $\mathcal{B}_1$ can extract $q_S + 1$ valid signatures under $\mathcal{S}_1$ with distinct messages $\mathsf{m}_{1,i}$ and thus, it holds that

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{dNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1).$$

Now we construct the adversary $\mathcal{B}_2$ against the unforgeability of $\mathcal{S}_2$. As before, suppose $\mathcal{A}$ is an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ algorithm that outputs a forgery for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$– in other words, it outputs $q_S + 1$ valid signatures under $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ on distinct messages. We can construct an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ algorithm $\mathcal{B}_2$ that finds a forgery in $\mathcal{S}_2$. $\mathcal{B}_2$ interacts with an $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ challenger for $\mathcal{S}_2$ which provides a public key pk_2. $\mathcal{B}_2$ generates a key pair $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{KeyGen}_1()$ and sets the public key for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ to be $(\mathsf{pk}_1, \mathsf{pk}_2)$. When $\mathcal{A}$ asks for the message $(\mathsf{m}_1, \mathsf{m}_2)$ to be signed using $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$, $\mathcal{B}_2$ proceeds computing the signature s_1 of m_1 with Sign_1 and the secret key sk_1. Then $\mathcal{B}_2$ queries the message $(\mathsf{m}_1, \mathsf{s}_1, \mathsf{m}_2)$ to its signing oracle for $\mathcal{S}_2$. There is a one-to-one correspondence between $\mathcal{A}$'s queries to its signing oracle for $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ and $\mathcal{B}_2$'s queries to its oracle for $\mathcal{S}_2$.

As before, if $\mathcal{S}_2$ is proven to be secure in the (Q)ROM, then this proof of $\mathsf{dNest}[\mathcal{S}_1, \mathcal{S}_2]$ also proceeds in the (Q)ROM: $\mathcal{B}_2$ relays $\mathcal{A}$'s random oracle queries directly to its random oracle, giving a one-to-one correspondence between $\mathcal{A}$'s queries to its hash oracle and $\mathcal{B}_2$'s queries to its hash oracle.

If $\mathcal{A}$ wins the $\mathsf{R^cT\text{-}EUF\text{-}CMA}$ game, then it has returned $q_S + 1$ distinct tuples $(\mathsf{m}_{1,i}, \mathsf{m}_{2,i}, \mathsf{s}_{1,i}, \mathsf{s}_{2,i})$ such that the following holds true $\mathsf{Verify}_1(\mathsf{pk}_1, \mathsf{m}_{1,i}, \mathsf{s}_{1,i}) = 0$ and $\mathsf{Verify}_2(\mathsf{pk}_2, (\mathsf{m}_{1,i}, \mathsf{s}_{1,i}, \mathsf{m}_{2,i}), \mathsf{s}_{2,i}) = 0$. Hence, $\mathcal{B}_2$ can extract $q_S + 1$ valid signatures under $\mathcal{S}_2$ with distinct messages $(\mathsf{m}_{1,i}, \mathsf{s}_{1,i}, \mathsf{m}_{2,i})$ and thus, it holds that

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{dNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2).$$

This implies finally that

$$\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{dNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A}) \leq \min\{\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1), \mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)\}.$$

Thus, if at least one of $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_1}(\mathcal{B}_1)$ and $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathcal{S}_2}(\mathcal{B}_2)$ is small, so too is $\mathrm{Adv}^{\mathsf{R^cT\text{-}EUF\text{-}CMA}}_{\mathsf{dNest}[\mathcal{S}_1,\mathcal{S}_2]}(\mathcal{A})$. □

5.2.3.2 Application of dNest in X.509

This application follows that same idea as in the case of the combiner sNest and the optional attributes of S/MIME. In case of the combiner dNest and X.509, Stebila

and Herath use the standard's extension mechanism as explained next. Let c_1 be the certificate obtained by the Certificate Authority (CA) signing `tbsCertificate` m_1 (containing subject public key $\mathsf{pk}^{\text{Sub}}_{\text{PQ}}$) using signature scheme $\mathcal{S}_1$. Construct certificate c_2 by the CA signing `tbsCertificate` m_2, which contains the subject's public key $\mathsf{pk}^{\text{Sub}}_{\text{RSA}}$ as well as (an encoding of) c_1 as an extension in m_2, using signature scheme $\mathcal{S}_2$. We depict a simplified structure of the hybrid X.509 certificate in Figure 5.17. The extension containing c_1 would use a distinct extension identifier saying "this is an additional certificate". The extension is marked as non-critical and hence, existing software (not aware of the hybrid structure) should ignore the unrecognized extension, continue validating the certificate, and using it in applications without change. However, software that recognizes the extension correctly *must* process it [65, Section 4.2]. Hence, this approach adds post-quantum security using the dNest combiner for honest users and is backwards compatible as long as the size of the additional certificate is not too large as explained next.

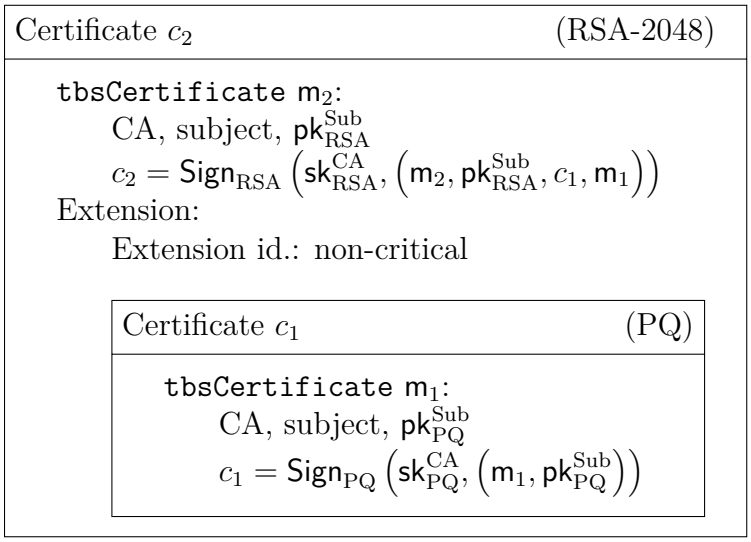


Figure 5.17: Simplified structure of the hybrid X.509v3 certificate using the nested signature combiner dNest with the classical RSA-2048 and a Post-Quantum (PQ) signature scheme

The backwards compatibility of this approach depends on whether the size of the extension is accepted by the respective libraries. Herath and Stebila evaluated the backwards compatibility of this approach using command-line certificate verification programs in various libraries, namely GnuTLS 3.5.11, Java SE 1.8.0_131, mbedTLS 2.4.2, NSS 3.29.1, and OpenSSL 1.0.2k. They concluded that all tested libraries were able to parse and verify X.509v3 certificates containing unrecognized extensions of all extension sizes that range from 1.5 KB until 1333.0 KB. In particular that means that all parameter sets of qTESLA can be used as an instantiation of the

post-quantum certificate; sizes that are as large as the public keys of TESLA are most likely not compatible with current software. We refer to Section 3.3.3 for the concrete instantiations of qTESLA and TESLA.

In addition to the command-line certificate verification programs, Herath and Stebila also tested the compatibility of various TLS implementations using the above-described hybrid X.509 certificate. In particular, they tested whether the libraries GnuTLS 3.5.11, Java SE 1.8.0_131, mbedTLS 2.4.2, NSS 3.29.1, and OpenSSL 1.0.2k can be used to establish a TLS 1.2 connection using the hybrid certificate with an extension of a certain size. They concluded that only Java completes connections with extensions the size of 1333.0 KB and mbedTLS cannot even handle certificates with extensions of the size of 43 KB. Moreover, they also investigated whether popular web browsers such Apple Safari 10.1, Google Chrome, Microsoft Edge, Microsoft IE, Mozilla Firefox, or Opera can be used to establish a TLS 1.2 connection to a TLS server using a hybrid certificate with an extension of a certain size. Their results have shown that the Microsoft browsers on Windows 10 cannot handle hybrid certificates with extensions the size of 43 KB. Furthermore, no browser except Safari could handle extensions of size 1333.0 KB. Hence, according to these experiments it is reasonable to expect that all heuristic qTESLA parameters (namely, qTESLA-h-I, qTESLA-h-III-size, qTESLA-h-III-speed) can be used to instantiate the hybrid certificate. Moreover, also the provable secure parameter set qTESLA-p-I can be used in hybrid certificates depending on the application. However, the public key and signature sizes of qTESLA-p-III and all TESLA parameters seem too large to be used in a hybrid certificate.

5.3 Hybrid KEMs

Now we turn to discuss the use of robust combiners to construct hybrid KEMs. We propose three combiners motivated by practical applications of hybrid KEMs. Informally, we call a KEM combiner robust if the resulting KEM is ($\mathsf{X}^{y}\mathsf{Z}$-IND-ATK) secure if at least one of the candidate signature schemes is ($\mathsf{X}^{y}\mathsf{Z}$-IND-ATK) secure with IND-ATK $\in \{\mathsf{IND\text{-}CCA}, \mathsf{IND\text{-}CPA}\}$.

The study of such hybrid KEMs dates back to work by Zhang et al. [223] and Dodis and Katz [76] who examined the security of using multiple IND-CCA secure public key encryption schemes. Recently, Giacon et al. [105] presented various KEM combiners. While their work is an important first step towards constructing hybrid KEMs, their solutions focus solely on classical adversaries. Since most of the constructions of [105] use idealized assumptions such as random oracles, the security reductions might not immediately transfer to the quantum setting [45].

Overall this section presents three combiners. The first combiner, called the *XOR-then-MAC* combiner XtM, uses a simple exclusive-or of the two keys k_1, k_2

of the single KEMs but adds a message authentication over the ciphertexts (with a key derived as part of the exclusive-or of the keys). Hence, this solution relies solely on the additional assumption of a secure one-time MAC $\mathcal{M}$ which, in turn, can be instantiated unconditionally. The second combiner, dPRF, relies on the existence of PRFs $\mathcal{P}$ and dual PRFs $\mathcal{DP}$ [31–33] which provide security if either the key material or the input carries entropy. The Hashed Message Authentication Code (HMAC)-based Key Derivation Function (HKDF) is, for example, based on this dual principle [146]. The third combiner, nPRF, is a nested variant of the dual-PRF combiner inspired by the key derivation procedure in TLS 1.3 and the proposal how to augment it for hybrid schemes in [201]. Its security relies on an additional second PRF. We summarize our three combiners with their respective building blocks and applications in Table 5.2.

Throughout this section we let $\mathcal{K}_1 = (\mathsf{KeyGen}_1, \mathsf{Encaps}_1, \mathsf{Decaps}_1)$ and $\mathcal{K}_2 = (\mathsf{KeyGen}_2, \mathsf{Encaps}_2, \mathsf{Decaps}_2)$ be two KEMs. Furthermore, we write $\mathcal{C}[\mathcal{K}_1, \mathcal{K}_2] = (\mathsf{KeyGen}_\mathcal{C}, \mathsf{Encaps}_\mathcal{C}, \mathsf{Decaps}_\mathcal{C})$ for the hybrid KEM constructed by one of the three proposals $\mathcal{C} \in \{\mathsf{XtM}, \mathsf{dPRF}, \mathsf{nPRF}\}$. In all our schemes, $\mathsf{KeyGen}_\mathcal{C}$ simply returns the concatenation of the two public keys ($\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$) and the two secret keys ($\mathsf{sk} \leftarrow (\mathsf{sk}_1, \mathsf{sk}_2)$).

Combiner	Building blocks	Indistinguishability	Application
XtM	$\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}$	$\max\{\mathsf{X}^\mathsf{c}\mathsf{Z}, \mathsf{U}^\mathsf{c}\mathsf{W}\}$	–
dPRF	$\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}$	$\max\{\mathsf{X}^\mathsf{c}\mathsf{Z}, \mathsf{U}^\mathsf{c}\mathsf{W}\}$	TLS 1.3 [216]
nPRF	$\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}_1, \mathcal{P}_2$	$\max\{\mathsf{X}^\mathsf{c}\mathsf{Z}, \mathsf{U}^\mathsf{c}\mathsf{W}\}$	TLS 1.3 [201]

Indistinguishability: If $\mathcal{K}_1$ is $\mathsf{X}^\mathsf{c}\mathsf{Z}$-IND-CCA and $\mathcal{K}_2$ is $\mathsf{U}^\mathsf{c}\mathsf{W}$-IND-CCA, then $\mathcal{C}[\mathcal{K}_1, \mathcal{K}_2]$ is ...-IND-CCA.

Table 5.2: KEM combiners using KEMS $\mathcal{K}_1$ and $\mathcal{K}_2$

Hybrid KEMs are designed to secure the transitional phase until quantum computers become available. The eventually following widespread deployment of quantum computers and cryptography, and thus security against $\mathsf{Q}^\mathsf{q}\mathsf{Q}$ adversaries, is outside the scope of the post-quantum setting. Hence, we focus on proving security against $\mathsf{C}^\mathsf{c}\mathsf{C}$, $\mathsf{C}^\mathsf{c}\mathsf{Q}$, and $\mathsf{Q}^\mathsf{c}\mathsf{Q}$ adversaries and omit $\mathsf{Q}^\mathsf{q}\mathsf{Q}$-IND-CCA security in what follows.

5.3.1 XtM: XOR-then-MAC Combiner

We first elaborate on the XOR-then-MAC combiner.

5.3.1.1 Description of the XOR-then-MAC Combiner

Giacon et al. [105] show that the plain XOR-combiner, which concatenates the ciphertexts and XORs the individual keys, preserves IND-CPA security. Moreover, they show that it does not preserve IND-CCA security in general, e.g., the combiner may become insecure if one of the KEMs is insecure. We note that it is easy to see that this is even true if both KEMs are IND-CCA secure: Given a challenge ciphertext (c_1^*, c_2^*), the adversary can make two decapsulation requests for (c_1^*, c_2) and (c_1, c_2^*) with fresh ciphertexts $c_1 \neq c_1^*$, $c_2 \neq c_2^*$ for which it knows the encapsulated keys. This allows the adversary to easily recover the challenge key from the answers.

Our approach is to prevent the adversary from mix-and-match attacks by computing a MAC over the ciphertexts and attaching it to the encapsulation. For this we require a strongly robust MAC combiner which takes two keys $k_{\mathsf{mac},1}, k_{\mathsf{mac},2}$ as input and provides one-time unforgeability, even if one of the keys is chosen adversarially. The combined KEM key is derived as an exclusive or of the leading parts of the two encapsulated keys, $k \leftarrow k_1 \oplus k_2$, and the MAC key $k_{\mathsf{mac}} = (k_{\mathsf{mac},1}, k_{\mathsf{mac},2})$ consisting of the remaining parts of both encapsulated keys. If necessary, the encapsulated keys can be stretched pseudorandomly by the underlying encapsulation schemes first to achieve the desired output length. We depict encapsulation and decapsulation of the resulting hybrid KEM in Algorithm 5.19 and 5.20, respectively. In both the algorithms let $\mathcal{M} = (\mathsf{MKG}, \mathsf{MAC}, \mathsf{MVf})$ be a MAC for some key k_{mac}.

Algorithm 5.19 Encapsulation of $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$

Require: Public key $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$
Ensure: Ciphertext (c, τ) and key k

1: $(c_1, k_1 || k_{\mathsf{mac},1}) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_1)$
2: $(c_2, k_2 || k_{\mathsf{mac},2}) \leftarrow \mathsf{Encaps}_2(\mathsf{pk}_2)$
3: $k_{\mathsf{kem}} \leftarrow k_1 \oplus k_2$
4: $k_{\mathsf{mac}} \leftarrow (k_{\mathsf{mac},1}, k_{\mathsf{mac},2})$
5: $c \leftarrow (c_1, c_2)$
6: $\tau \leftarrow \mathsf{MAC}_{k_{\mathsf{mac}}}(c)$
7: **return** $((c, \tau), k_{\mathsf{kem}})$

Algorithm 5.20 Decapsulation of $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$

Require: Secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$, ciphertext $((c_1, c_2), \tau)$
Ensure: Key k

1: $k'_1 || k'_{\mathsf{mac},1} \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$
2: $k'_2 || k'_{\mathsf{mac},2} \leftarrow \mathsf{Decaps}_2(\mathsf{sk}_2, c_2)$
3: $k'_{\mathsf{kem}} || k'_{\mathsf{mac}} \leftarrow k'_1 \oplus k'_2$
4: $k'_{\mathsf{mac}} \leftarrow (k'_{\mathsf{mac},1}, k'_{\mathsf{mac},2})$
5: **if** $\mathsf{MVf}_{k'_{\mathsf{mac}}}((c_1, c_2), \tau) = 0$ **then**
6: **return** $\perp$
7: **else**
8: **return** k'_{kem}

5.3.1.2 Security of the XOR-then-MAC Combiner

We can now show that the XOR-then-MAC combiner is a robust KEM combiner, in the sense that the resulting KEM is as secure as the strongest of the two input KEMs (assuming the MAC is also equally secure). In particular, we show in Theorem 5.20 that $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$ is IND-CCA secure in the post-quantum setting ($\mathsf{Q^cQ}$) if the MAC $\mathcal{M}$ is $\mathsf{X^yZ\text{-}OT\text{-}sEUF}$ and at least one of the two KEMs is post-quantum IND-CCA secure. In fact, the security offered by the MAC is only required in case of IND-CCA attacks, yielding an even better bound for the IND-CPA case. We recall that $\mathsf{X^cZ\text{-}IND\text{-}ATK} \in \{\mathsf{X^cZ\text{-}IND\text{-}CCA}, \mathsf{X^cZ\text{-}IND\text{-}CPA}\}$. The security experiment for $\mathsf{X^yZ\text{-}OT\text{-}sEUF}$ of $\mathcal{M}$ is given in Figure 5.6.

Theorem 5.20 (Indistinguishability of XtM)**.** *Let $\mathcal{K}_1$ be a $\mathsf{X^cZ\text{-}IND\text{-}ATK}$ secure KEM, $\mathcal{K}_2$ a $\mathsf{U^cW\text{-}IND\text{-}ATK}$ secure KEM, and $\mathcal{M}$ be an $\mathsf{R^cT\text{-}OT\text{-}sEUF}$ secure MAC, where $\mathsf{R^cT} = \max\{\mathsf{X^cZ}, \mathsf{U^cW}\}$. Then $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$ is also $\mathsf{R^cT\text{-}IND\text{-}ATK}$ secure.*

More precisely, for any efficient adversary $\mathsf{R^cT\text{-}IND\text{-}ATK}$ $\mathcal{A}$ against $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$, there exist efficient algorithms $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) \;\leq\; & 2 \cdot \min\left\{\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_2}(\mathcal{B}_2)\right\} \\ & + \; \mathsf{Adv}^{\mathsf{R^cT\text{-}OT\text{-}sEUF}}_{\mathcal{M}}(\mathcal{B}_3).\end{aligned}$$

Moreover, the run-times of $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ are approximately the same as that of $\mathcal{A}$, and $\mathcal{B}_3$ makes at most as many verification queries as $\mathcal{A}$ makes decapsulation queries.

Proof. Assume there exists an adversary $\mathcal{A}$ that breaks the $\mathsf{R^cT\text{-}IND\text{-}ATK}$ security of $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$. We show that this yields an adversary that then breaks either the $\mathsf{R^cT\text{-}IND\text{-}ATK}$ security of $\mathcal{K}_1$ or $\mathcal{K}_2$, or the $\mathsf{R^cT\text{-}OT\text{-}sEUF}$ security of $\mathcal{M}$. Because of symmetry it suffices to consider the case of $\mathcal{K}_1$. The following proof holds

analogously for $\mathcal{K}_2$ being $\mathsf{R^cT\text{-}IND\text{-}ATK}$ secure. We focus on the IND-CCA case in the following proof; the IND-CPA case follows easily from this.

We prove the theorem by applying the common technique of game hopping, bounding the adversary's advantage introduced with each game hop until the adversary cannot win beyond the guessing probability.

Game 0. This is the original $\mathsf{R^cT\text{-}IND\text{-}ATK}$ game against $\mathsf{XtM}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{M}]$.

Game 1. We now replace the key $k_1^*||k_{\mathsf{mac},1}^*$ of $\mathcal{K}_1$ returned with the challenge ciphertext part c_1^* with a uniformly random and independent value $r_1^*\mathsf{Con}r_{\mathsf{mac},1}^*$ from the same keyspace K. This means that we first create $(c_1^*, k_1^*||k_{\mathsf{mac},1}^*)$ and then use $(c_1^*, r_1^*||r_{\mathsf{mac},1}^*)$ immediately from then on. This is done consistently in the challenge value for deriving the challenge key portion and the MAC, as well as in all decapsulation requests involving the ciphertext portion c_1^*. More precisely, we replace the step "$k_1||k_{\mathsf{mac},1} \leftarrow\!\!\$\ \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$" in the decapsulation procedure with the step "if $c_1 = c_1^*$ then $k_1||k_{\mathsf{mac},1} \leftarrow r_1^*||r_{\mathsf{mac},1}^*$ else $k_1||k_{\mathsf{mac},1} \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$".

We show that if $\mathcal{A}$ can efficiently distinguish Game 1 from Game 0, then there exists an adversary $\mathcal{B}_1$ against the $\mathsf{R^cT\text{-}IND\text{-}ATK}$ security of $\mathcal{K}_1$. Algorithm $\mathcal{B}_1$ receives as input a public key pk_1 and a challenge ciphertext c_1^* of $\mathcal{K}_1$, as well as the challenge key $k_1^*||k_{\mathsf{mac},1}^*$. This challenge key is either the actual key or random. Algorithm $\mathcal{B}_1$ simulates the environment for $\mathcal{A}$ as follows.

First, $\mathcal{B}_1$ generates the key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$ for $\mathcal{K}_2\mathsf{Con}k_{\mathsf{mac},2}$ and sets $\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$. Furthermore, $\mathcal{B}_1$ chooses the second challenge ciphertext portion c_2^* and key share $k_2^*||k_{\mathsf{mac},2}^*$ itself. It computes $k_{\mathsf{kem}}^* \leftarrow k_1^* \oplus k_2^*$ and $k_{\mathsf{mac}}^* \leftarrow (k_{\mathsf{mac},1}^*, k_{\mathsf{mac},2}^*)$, and assembles the challenge ciphertext (c_1^*, c_2^*, τ^*) where $\tau^* \leftarrow \mathsf{MAC}_{k_{\mathsf{mac}}^*}((c_1^*, c_2^*))$. It also picks a challenge bit and replaces k_{kem}^* by a random value if this bit is 1. Adversary $\mathcal{B}_1$ then runs $\mathcal{A}$ on input $(\mathsf{pk}, (c_1^*, c_2^*, \tau^*), k_{\mathsf{kem}}^*)$.

If $\mathcal{A}$ is an active adversary, mounting an IND-CCA attack, decapsulation queries for ciphertexts $c = (c_1, c_2)$ with $c_1 \neq c_1^*$ and some MAC tag τ are answered as follows: c_1 is decapsulated using $\mathcal{B}_1$'s decapsulation oracle for $\mathcal{K}_1$, c_2 is decapsulated using sk_2, and the response k_{kem} is then computed as the appropriately truncated XOR of these decapsulations after verifying the MAC tag. If $c = (c_1^*, c_2)$ then $\mathcal{B}_1$ uses $k_1^*||k_{\mathsf{mac},1}^*$ as the decapsulation of c_1^*, and then continues as in the previous case. For passive IND-CPA adversaries $\mathcal{A}$, algorithm $\mathcal{B}_1$ does not need to provide any simulation of decapsulation queries. At some point, the distinguisher $\mathcal{A}$ terminates and outputs a guess bit b'. Adversary $\mathcal{B}_1$ outputs the same bit b'.

Clearly, $\mathcal{B}_1$ perfectly simulates the environments for $\mathcal{A}$ corresponding to Game 0 if the challenge key $k_1^*||k_{\mathsf{mac},1}^*$ is the actual key, and perfectly simulates Game 1 if $k_1^*||k_{\mathsf{mac},1}^*$ is random. Furthermore, $\mathcal{B}_1$ is of the same type as $\mathcal{A}$. Hence, it holds

that

$$\mathsf{Adv}^{G_0}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) \leq \mathsf{Adv}^{G_1}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_1}(\mathcal{B}_1),$$

where the factor 2 is owned to the transition from the prediction-based $\mathsf{R^cT\text{-}IND\text{-}ATK}$ attack with a random challenge bit b to an indistinguishability-based comparison between fixed games here.

Game 2. In a syntactical change we replace the now random value r_1^* by $r_1^* \oplus k_2^*$ where k_2^* is the encapsulated key in $\mathcal{K}_2$ in the challenge ciphertext. This is done consistently in the challenge value and MAC, as well as in all decapsulation requests involving the ciphertext portion c_1^*. We leave $r^*_{\mathsf{mac},1}$ unaltered.

Effectively, the modification means that the encapsulated key in the challenge ciphertext is now $r_1^* = (r_1^* \oplus k_2^*) \oplus k_2^*$. Since r_1^* and k_2^* are independent, the distributions of r_1^* and $r_1^* \oplus k_2^*$ are identical, so the adversary's advantage does not change, i.e.,

$$\mathsf{Adv}^{G_1}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) = \mathsf{Adv}^{G_2}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}).$$

In Game 2, the adversary now receives a random value as the challenge key and the MAC is also computed over a random key part $r^*_{\mathsf{mac},1}$, independently of the challenge bit b. To complete the argument we only need to show that the adversary in an $\mathsf{R^cT\text{-}IND\text{-}CCA}$ attack does not gain any advantage via the decapsulation oracle (in which $r_1^* \oplus k_2^*$ from the challenge key is used for inputs of the form $(c_1^*, *, *)$). We next argue that the difference is negligible, though, because in the actual attack this can only happen if the adversary forges a MAC.

Game 3. In this game we change the decapsulation oracle in that we let it immediately reject with output $\perp$ if it is queried on a ciphertext of the form $(c_1^*, *, *)$ for the challenge ciphertext c_1^*.

An adversary is only able to notice the difference between Games 2 and 3 if it queries about a fresh ciphertext $(c_1^*, c_2, \tau) \neq (c_1^*, c_2^*, \tau^*)$ with c_2 being a $\mathcal{K}_2$ ciphertext of the adversary's choice, and τ being a valid MAC tag. If τ is not a valid MAC tag or the adversary queries exactly the challenge ciphertext, then our decapsulation oracle would also return $\perp$.

We show that if an adversary $\mathcal{A}$ distinguishes between the games, we can build an adversary $\mathcal{B}_3$ against the MAC. Adversary $\mathcal{B}_3$ runs $\mathcal{A}$ according to Game 2, choosing all components $(\mathsf{sk}_1, \mathsf{pk}_1)$ and $(\mathsf{sk}_2, \mathsf{pk}_2)$ and c_1^*, c_2^* of $\mathcal{K}_1$ and $\mathcal{K}_2$ itself. To create the challenge ciphertext, adversary $\mathcal{B}_3$ makes its one-time MAC request with message (c_1^*, c_2^*) and receives τ^*. It runs $\mathcal{A}$ on (c_1^*, c_2^*, τ^*) and a random string k^*_{kem}. For an $\mathsf{R^cT\text{-}IND\text{-}CCA}$ attack, if $\mathcal{A}$ makes a decapsulation query about (c_1, c_2, τ) for $c_1 \neq c_1^*$, then $\mathcal{B}_3$ uses its knowledge of its decapsulation keys to compute the answer.

For $c_1 = c_1^*$, adversary $\mathcal{B}_3$ calls its verification oracle with $(c_1, c_2, \tau, 2, k_2)$, where $k_2||k_{\mathsf{mac},2} \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_2, c_2)$, and returns $\perp$ to $\mathcal{A}$ to continue the simulation.

For the analysis it is important to note that Game 2 uses as the challenge key either an independent random string r_1^* (if $b = 0$) or a random key (if $b = 1$). In both cases, the KEM part of the key is a uniform key independent of bit b–as is $\mathcal{B}_3$'s choice k_{kem}^*–and the MAC key part is also independent and uniform. The latter holds in $\mathcal{B}_3$'s simulation as well, since the OT-sEUF game chooses a random MAC key. In other words, the simulation is perfect up to the step where, potentially, $\mathcal{A}$ makes a query for a fresh ciphertext with a valid MAC which would yield a reply different from $\perp$. But then $\mathcal{B}_3$ would find a forgery against the MAC in one of its multiple verification attempts. Since $\mathcal{B}_3$ is of the same type $\mathsf{R^cT}$ as $\mathcal{A}$, it, thus, holds that

$$\mathsf{Adv}^{G_2}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) \leq \mathsf{Adv}^{G_3}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) + \mathsf{Adv}^{\mathsf{R^cT\text{-}OT\text{-}sEUF}}_{\mathcal{M}}(\mathcal{B}_3).$$

The claim now follows, noting that in the final game the secret bit b is perfectly hidden from $\mathcal{A}$. The challenge key is an independent string in either case $b = 0$ or $b = 1$, and the decapsulation queries are now also independent of the bit b, since the change in the oracle's answers only depends on the public value c_1^*. Hence, $\mathcal{A}$'s output is independent of the secret bit, and thus

$$\mathsf{Adv}^{G_3}_{\mathsf{XtM}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{M}]}(\mathcal{A}) = 0.$$

□

5.3.1.3 Using the XOR-then-MAC Combiner in Protocols

It may seem that it would be preferable to protect against the above-mentioned mix-and-match attacks by protecting the derived key directly (by making key derivation depend on both keys and both ciphertexts), as opposed to the approach in the XOR-then-MAC combiner, which appends a MAC to the ciphertext to protect the ciphertext from modification. However, in many practical protocols, the parties compute a MAC over the transcript to provide integrity and authenticity. For example, this is done in the `Finished` message in the TLS protocol. The key for the MAC is usually derived from the session key, and the transcript includes the data for establishing the key, such as the KEM ciphertexts. In these cases, it may be possible to apply the XOR-then-MAC approach without an additional MAC over the ciphertext, instead relying on the MAC that is already present.

5.3.2 dPRF: Dual-PRF Combiner

Our second combiner is based on dual PRFs [31–33]. The definitions of (dual) PRF security in our two-stage model can be found in Section 5.1.1.4. Informally, a dual

PRF $\mathcal{DP}(k, x)$ is a PRF when either the key material k is random (i.e., $\mathcal{DP}(k, \cdot)$ is a PRF), or alternatively when the input x is random (i.e., $\mathcal{DP}(\cdot, x)$ is a PRF). HMAC has been shown to be a secure MAC under the assumption that it is a dual PRF. Moreover, Bellare and Lysyanskaya have given a generic validation of the dual PRF assumption for HMAC [33] and therefore HKDF.

5.3.2.1 Description of the Dual-PRF Combiner

To construct a hybrid KEM from a dual PRF, the naive approach of directly using a dual PRF to compute the session key of the combined KEM as $\mathcal{DP}(k_1, k_2)$ is not sufficient. For example, if $\mathcal{K}_1$ is secure but $\mathcal{K}_2$ is broken then an adversary might be able to transform the challenge ciphertext (c_1^*, c_2^*) into (c_1^*, c_2), where $c_2 \neq c_2^*$ but c_2 encapsulates the same key k_2 as c_2^*. With a single decapsulation query the adversary would be able to recover the key $\mathcal{DP}(k_1, k_2)$ and distinguish it from random. Our approach, shown in Algorithm 5.21 and 5.22, is to apply another PRF with the output of the dual PRF as the PRF key and the ciphertexts as the input label: $\mathcal{P}(\mathcal{DP}(k_1, k_2), (c_1, c_2))$.

Algorithm 5.21 Encapsulation of $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$

Require: Public key $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$
Ensure: Ciphertext c and key k

1: $(c_1, k_1) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_1)$
2: $(c_2, k_2) \leftarrow \mathsf{Encaps}_2(\mathsf{pk}_2)$
3: $c \leftarrow (c_1, c_2)$
4: $k_d \leftarrow \mathcal{DP}(k_1, k_2)$
5: $k \leftarrow \mathcal{P}(k_d, c)$
6: **return** (c, k)

Algorithm 5.22 Decapsulation of $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$

Require: Secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$, ciphertext (c_1, c_2)
Ensure: Key k

1: $k_1' \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$
2: $k_2' \leftarrow \mathsf{Decaps}_2(\mathsf{sk}_2, c_2)$
3: $k_d' \leftarrow \mathcal{DP}(k_1', k_2')$
4: **return** $\mathcal{P}(k_d', (c_1, c_2))$

5.3.2.2 Security of the Dual-PRF Combiner

In the following we show that the dual-PRF combiner is a robust KEM combiner, in the sense that the resulting KEM has the security of the strongest of the two input KEMs (assuming the PRF and dual PRF are also sufficiently secure). In particular, we show that $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$ is IND-CCA secure in the post-quantum setting ($\mathsf{Q^cQ}$) if $\mathcal{DP}$ is a post-quantum secure dual PRF, $\mathcal{P}$ is a post-quantum secure PRF, and at least one of the two KEMs is post-quantum IND-CCA secure.

Theorem 5.21 (Indistinguishability of dPRF)**.** *Let $\mathcal{K}_1$ be an $\mathsf{X^cZ\text{-}IND\text{-}ATK}$ secure KEM, $\mathcal{K}_2$ be a $\mathsf{U^cW\text{-}IND\text{-}ATK}$ secure KEM, and $\mathsf{R^cT} = \max\{\mathsf{X^cZ}, \mathsf{U^cW}\}$. Moreover, let $\mathcal{DP} : K_1 \times K_2 \to K'$ be an $\mathsf{R^cT}$ secure dual PRF, and $\mathcal{P} : K' \times \{0,1\}^* \to K_{\mathsf{dPRF}}$ be an $\mathsf{R^cT}$ secure PRF. Then $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$ is $\mathsf{R^cT\text{-}IND\text{-}ATK}$ secure.*

More precisely, for any $\mathsf{IND\text{-}ATK}$ adversary $\mathcal{A}$ of type $\mathsf{R^cT}$ against the combiner $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$, we derive efficient adversaries $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ with

$$\begin{aligned}\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) \;\leq\; & \min\left\{\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_2}(\mathcal{B}_2)\right\} \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}dPRF\text{-}SEC}}_{\mathcal{DP}}(\mathcal{B}_3) + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}PRF\text{-}SEC}}_{\mathcal{P}}(\mathcal{B}_4),\end{aligned}$$

while the run-times of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, and $\mathcal{B}_4$ are approximately the same as that of $\mathcal{A}$.

Proof. As before, we prove the theorem by considering a sequence of game hops. We focus on the case that $\mathcal{K}_1$ is secure, and mention the necessary modifications in the proof for $\mathcal{K}_2$ being secure as we progress through the games.

Game 0. This is the original $\mathsf{R^cT\text{-}IND\text{-}ATK}$ game for $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$.

Game 1. We replace the value k_1^* computed in the challenge ciphertext by a uniformly random value r_1^* of equal length and compute the final key in the challenge value as $\mathcal{P}(\mathcal{DP}(r_1^*, k_2^*), (c_1^*, c_2^*))$; it is important to note that for $b = 1$ this value is eventually replaced with a random value. Decapsulation requests (c_1, c_2) are also answered by using r_1^* instead of k_1^* in case $c_1^* = c_1$. That is, instead of computing $k_1 \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$ we compute "if $c_1 = c_1^*$ then $k_1 \leftarrow r_1^*$ else $k_1 \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$".

An adversary distinguishing Game 0 from Game 1 would immediately yield an efficient adversary $\mathcal{B}_1$ against the indistinguishability of $\mathcal{K}_1$ as explained next. Adversary $\mathcal{B}_1$ receives as input pk_1 and a challenge (c_1^*, k_1^*), and simulates the environment for $\mathcal{A}$ as follows: First, $\mathcal{B}_1$ generates the key pair $(\mathsf{pk}_2, \mathsf{sk}_2)$ for $\mathcal{K}_2$ and sets $\mathsf{pk} \leftarrow (\mathsf{pk}_1, \mathsf{pk}_2)$. Furthermore, $\mathcal{B}_1$ generates the second challenge ciphertext portion c_2^* and key share k_2^* itself. It assembles the challenge ciphertext as (c_1^*, c_2^*) and computes $k^* = \mathcal{P}(\mathcal{DP}(k_1^*, k_2^*), (c_1^*, c_2^*))$. It also picks a challenge bit and

replaces k^* by a random value if this bit is 1. Adversary $\mathcal{B}_1$ then runs $\mathcal{A}$ on input $(\mathsf{pk}, (c_1^*, c_2^*), k^*)$.

If $\mathcal{A}$ is an active adversary in the IND-CCA case, decapsulation queries for ciphertexts $c = (c_1, c_2)$ with $c_1 \neq c_1^*$ are answered by relaying c_1 to the corresponding decapsulation oracle for $\mathcal{K}_1$ and decapsulating c_2 with the help of sk_2. The final response is computed according to the protocol description. If $c = (c_1^*, c_2)$ then $\mathcal{B}_1$ simply substitutes the evaluation and response of $\mathcal{K}_1$'s decapsulation oracle with k_1^* and computes the answer accordingly. For passive adversaries $\mathcal{A}$ in an IND-CPA attack, algorithm $\mathcal{B}_1$ does not need to provide any simulation of decapsulation queries. At some point, the distinguisher $\mathcal{A}$ terminates and outputs a guess bit b'. Adversary $\mathcal{B}_1$ outputs the same bit b'.

For the analysis note that the difference between the two games lies exactly in the distinction between the actual key k_1^* and a random value. Hence, we have

$$\mathsf{Adv}^{G_0}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) \leq \mathsf{Adv}^{G_1}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]} + 2 \cdot \mathsf{Adv}^{\mathsf{X^cZ\text{-}IND\text{-}ATK}}_{\mathcal{K}_1}(\mathcal{B}_1),$$

where the factor 2 is owned to the transition from the prediction-based $\mathsf{R^cT\text{-}IND\text{-}ATK}$ attack with a random challenge bit b to an indistinguishability-based comparison between fixed games here. For $\mathcal{K}_2$ being secure the proof applies analogously, yielding an adversary $\mathcal{B}_2$.

Game 2. Next, we replace the value $\mathcal{DP}(r_1^*, k_2^*)$ by a uniformly random value r^* in the computation of the challenge ciphertext. We make the following additional modification to the current decapsulation procedure: We use r^* in decapsulation requests (instead of $\mathcal{DP}(r_1^*, k_2^*)$) for any request of the form (c_1^*, c_2) for which $\mathsf{Decaps}_2(\mathsf{sk}_2, c_2) = k_2^*$. It is important to note that we still use r_1^* and $\mathcal{DP}(r_1^*, k_2)$ for queries of the form $c_1 = c_1^*$ but $k_2 = \mathsf{Decaps}_2(\mathsf{sk}_2, c_2) \neq k_2^*$.

Distinguishing Game 1 from Game 2 would immediately yield an efficient adversary $\mathcal{B}_3$ against the PRF-security of $\mathcal{DP}$. The reduction is straightforward. Algorithm $\mathcal{B}_3$, in its first query, can ask about some input value and either receives the PRF value or a random reply, and from then on $\mathcal{B}_3$ can ask about other inputs to learn further PRF values.

Algorithm $\mathcal{B}_3$ creates keys $(\mathsf{pk}_1, \mathsf{sk}_1)$, $(\mathsf{pk}_2, \mathsf{sk}_2)$, as well as c_1^*, c_2^* (with encapsulated keys k_1^*, k_2^*). It then queries its PRF oracle about (r_1^*, k_2^*) to receive a value r^*. It starts a simulation of $\mathcal{A}$ on input $(\mathsf{pk}, (c_1^*, c_2^*), \mathcal{P}(r^*, (c_1^*, c_2^*))$. Each decapsulation query for (c_1, c_2) with $c_1 \neq c_1^*$ is answered with the help of $\mathsf{sk}_1, \mathsf{sk}_2$. Each query with $c_1 = c_1^*$ is answered by decapsulating k_2 from c_2 with sk_2. If now $k_2 = k_2^*$ then we use r^* to compute the final answer, else we query the PRF oracle about k_2 and use the reply to complete the computation. It is important to note that this is admissible since k_2 is different from the input k_2^* in $\mathcal{B}_3$'s first query. Finally, $\mathcal{B}_3$ outputs the final answer of adversary $\mathcal{A}$.

If $\mathcal{B}_3$ receives a pseudo-random value r^* then we perfectly simulate Game 1. If, on the other hand, r^* is random, then we perfectly simulate Game 2. Thus,

$$\mathsf{Adv}^{G_1}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) \leq \mathsf{Adv}^{G_2}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}dPRF\text{-}SEC}}_{\mathcal{DP}}(\mathcal{B}_3).$$

Here the factor 2 in the distinguishing advantage against $\mathcal{DP}$ comes from the fact that we use the variant of having a real-or-random challenge and then communicating with the actual function on different inputs.

For $\mathcal{K}_2$ being secure the same line of reasoning applies, because $\mathcal{DP}$ is a dual PRF and hence, $\mathcal{DP}(\cdot, r_2^*)$ is also pseudo-random.

Game 3. Finally, we replace the value $\mathcal{P}(r^*, (c_1^*, c_2^*))$ by a uniformly random value R^* in the computation of the challenge ciphertext. The decapsulation procedure remains unchanged.

We show security by a reduction to the security of the PRF. Algorithm $\mathcal{B}_4$ again creates keys $(\mathsf{pk}_1, \mathsf{sk}_1)$, $(\mathsf{pk}_2, \mathsf{sk}_2)$ and the challenge ciphertext (c_1^*, c_2^*). It queries the PRF oracle about $(r^*, (c_1^*, c_2^*))$ to obtain a value R^* and runs $\mathcal{A}$ on $(\mathsf{pk}, (c_1^*, c_2^*), R^*)$. Each decapsulation query (c_1, c_2) is answered as follows: If $c_1 \neq c_1^*$ we answer with the help of the decapsulation keys $\mathsf{sk}_1, \mathsf{sk}_2$, computing the same reply as the original decapsulation oracle. In case $c_1 = c_1^*$ (and consequently $c_2 \neq c_2^*$) and where $k_2 = k_2^*$ we call the external oracle about (c_1^*, c_2) to derive the answer. For $c_1 = c_1^*$ but $k_2 \neq k_2^*$ we use $\mathcal{DP}(r_1^*, k_2)$ to compute the answer.

We again have that if $\mathcal{B}_4$ receives a pseudo-random value R^* then we perfectly simulate Game 2. If R^* is random, then we perfectly simulate Game 3. Hence,

$$\mathsf{Adv}^{G_2}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) \leq \mathsf{Adv}^{G_3}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}PRF\text{-}SEC}}_{\mathcal{P}}(\mathcal{B}_4).$$

The analogous applies when $\mathcal{K}_2$ is secure.

In the final game neither the challenge ciphertext nor the decapsulation oracle carries any information about the secret challenge bit b. That is, the oracle does not depend on R^* in case $b = 0$, so this case is indistinguishable from the $b = 1$ case. We thus arrive at the final bound:

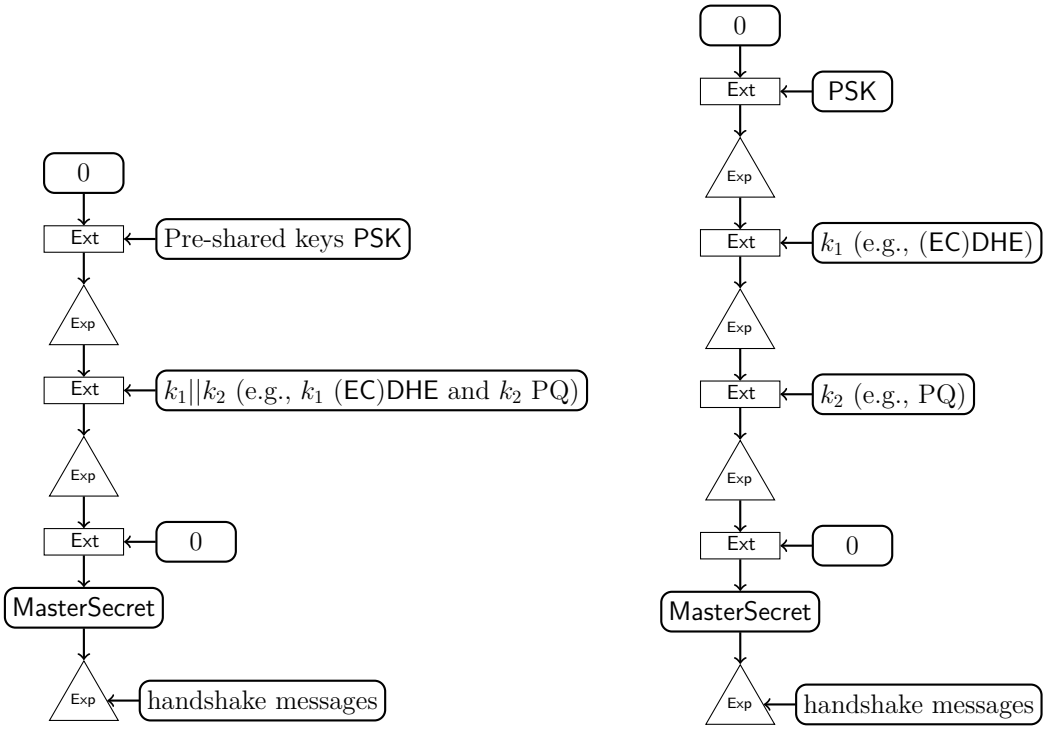
$$\mathsf{Adv}^{G_3}_{\mathsf{dPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P}]}(\mathcal{A}) = 0$$

This concludes the proof that $\mathsf{dPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}]$ is a hybrid KEM with $\mathsf{R^cT\text{-}IND\text{-}ATK}$ security. □

5.3.2.3 Application of the Dual-PRF Combiner

After proving the security of the combiner dPRF, we now move on to explain a possible application of the dual-PRF KEM combiner.

Our dPRF combiner is inspired by the key derivation in the current draft of TLS 1.3 [195] and models Whyte et al.'s proposal for supporting hybrid key exchange in TLS 1.3 [216]. In TLS 1.3, HKDF's extract function is applied to the raw ECDH shared secret; the result is then fed through HKDF's expand function with the (hash of the) transcript as (part of) the label. In Whyte et al.'s hybrid proposal, the session keys from multiple KEMs are concatenated as a single shared secret input to HKDF extract. The dPRF combiner models this by taking $\mathcal{DP}$ as HKDF extract and $\mathcal{P}$ as HKDF expand, as depicted in Figure 5.18a.



(a) Following [216] using dPRF

(b) Following [201] using nPRF

Figure 5.18: Excerpt from altered TLS 1.3 key schedule as proposed in [216] (left) and [201] (right) to incorporate an additional PQ secret k_2

5.3.3 nPRF: Nested Dual-PRF Combiner

After explaining the dual-PRF combiner, we now turn to explain the nested dual-PRF combiner.

5.3.3.1 Description of the Nested Dual-PRF combiner

We augment the dPRF combiner in the Section 5.3.2 by an extra preprocessing step for the key k_1: $k_e \leftarrow \mathcal{EX}(0, k_1)$, where $\mathcal{EX}$ is another PRF. This is the nested dual-PRF combiner nPRF shown in Algorithm 5.23 and 5.24.

Algorithm 5.23 Encapsulation of $\mathsf{nPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}, \mathcal{EX}]$

Require: Public key $\mathsf{pk} = (\mathsf{pk}_1, \mathsf{pk}_2)$
Ensure: Ciphertext c and key k

1: $(c_1, k_1) \leftarrow \mathsf{Encaps}_1(\mathsf{pk}_1)$
2: $(c_2, k_2) \leftarrow \mathsf{Encaps}_2(\mathsf{pk}_2)$
3: $c = (c_1, c_2)$
4: $k_e = \mathcal{EX}(0, k_1)$
5: $k_d = \mathcal{DP}(k_e, k_2)$
6: $k = \mathcal{P}(k_d, c)$
7: **return** (c, k)

Algorithm 5.24 Decapsulation of $\mathsf{nPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}, \mathcal{EX}]$

Require: Secret key $\mathsf{sk} = (\mathsf{sk}_1, \mathsf{sk}_2)$, ciphertext (c_1, c_2)
Ensure: Key k

1: $k'_1 \leftarrow \mathsf{Decaps}_1(\mathsf{sk}_1, c_1)$
2: $k'_2 \leftarrow \mathsf{Decaps}_2(\mathsf{sk}_2, c_2)$
3: $k'_e = \mathcal{EX}(0, k_1)$
4: $k'_d = \mathcal{DP}(k'_e, k_2)$
5: **return** $\mathcal{P}(k'_d, (c_1, c_2))$

5.3.3.2 Security of the Nested Dual-PRF Combiner

The nested dual-PRF combiner nPRF is a robust KEM combiner, in the sense that the resulting KEM has the security of the strongest of the two input KEMs (assuming the PRFs are sufficiently secure). Informally, the theorem shows that $\mathsf{nPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}, \mathcal{EX}]$ is IND-CCA secure in the post-quantum setting if $\mathcal{DP}$ is a post-quantum secure dual PRF, $\mathcal{P}$ and $\mathcal{EX}$ are post-quantum secure PRFs, and at least one of the two KEMs is post-quantum IND-CCA secure.

Theorem 5.22 (Indistinguishability of nPRF)**.** *Let* $\mathcal{K}_1$ *be an* $\mathsf{X^cZ\text{-}IND\text{-}ATK}$ *secure KEM,* $\mathcal{K}_2$ *be a* $\mathsf{U^cW\text{-}IND\text{-}ATK}$ *secure KEM,* $\mathcal{DP} : K' \times K_2 \to K''$ *be an* $\mathsf{R^cT} = \max\{\mathsf{X^cZ}, \mathsf{U^cW}\}$ *secure dual PRF,* $\mathcal{P} : K'' \times \{0,1\}^* \to K_{\mathsf{nPRF}}$ *be an* $\mathsf{R^cT}$ *secure PRF, and* $\mathcal{EX} : \{0,1\}^* \times K_1 \to K'$ *be an* $\mathsf{R^cT}$ *secure PRF. Then the combiner* $\mathsf{nPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}, \mathcal{EX}]$ *is* $\mathsf{R^cT\text{-}IND\text{-}ATK}$ *secure.*

More precisely, for any IND-ATK *adversary* $\mathcal{A}$ *of type* $\mathsf{R^cT}$ *against the combined KEM* $\mathsf{nPRF}[\mathcal{K}_1, \mathcal{K}_2, \mathcal{DP}, \mathcal{P}, \mathcal{EX}]$*, we derive efficient adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$*,* $\mathcal{B}_4$*, and* $\mathcal{B}_5$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathsf{nPRF}[\mathcal{K}_1,\mathcal{K}_2,\mathcal{DP},\mathcal{P},\mathcal{EX}]}(\mathcal{A}) \;\leq\; & \min\left\{\mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_1}(\mathcal{B}_1), \mathsf{Adv}^{\mathsf{R^cT\text{-}IND\text{-}ATK}}_{\mathcal{K}_2}(\mathcal{B}_2)\right\} \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}dPRF\text{-}SEC}}_{\mathcal{DP}}(\mathcal{B}_3) + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}PRF\text{-}SEC}}_{\mathcal{P}}(\mathcal{B}_4) \\ & + 2 \cdot \mathsf{Adv}^{\mathsf{R^cT\text{-}PRF\text{-}SEC}}_{\mathcal{EX}}(\mathcal{B}_5),\end{aligned}$$

while the run-times of $\mathcal{B}_1$*,* $\mathcal{B}_2$*,* $\mathcal{B}_3$*,* $\mathcal{B}_4$*, and* $\mathcal{B}_5$ *are approximately the same as that of* $\mathcal{A}$*.*

The proof follows easily from the proof of the dPRF combiner. In particular, the proof of Theorem 5.22 would also consist of the sequence of game hops Game 1, Game 2, and Game 3 in the proof of Theorem 5.21. Additionally, one more intermediate step in which we use the pseudo-randomness of $\mathcal{EX}$ to argue that the output of $\mathcal{EX}(0, k_1)$ is pseudo-random, would be made. Namely, the first game hop would be to exchange k_1 with a uniformly random r_1 from the same key space. With the same arguments as given in the proof of Theorem 5.21, it would follow that an adversary that is able to detect this change, would also be able to break the $\mathsf{R^cT\text{-}PRF\text{-}SEC}$ security of $\mathcal{EX}$. The following game hop would then be to substitute k_e by a random value which corresponds exactly to Game 1 in the proof of Theorem 5.21.

5.3.3.3 Application of the Nested Dual-PRF Combiner

Lastly, we explain an application of the nested dual-PRF KEM combiner. Our combiner nPRF models Schanck and Stebila's proposal [201] for hybrid key exchange in TLS 1.3. In their proposal, as depicted in Figure 5.18b, one stage of the TLS 1.3 key schedule is applied for each of the constituent KEMs in the hybrid KEM construction: Each stage in the key schedule applies the HKDF extract function with one input being the output from the previous stage of the key schedule and the other input being the shared secret from this stage's KEM. Finally, HKDF expand incorporates the (hash of the) transcript, including all KEMs' ciphertexts. Modeling the extraction function $\mathcal{EX}$ as a PRF, our nested combiner nPRF captures this scenario. This approach is not backwards compatible with non-aware TLS 1.3

implementations. The advantage of this is that the combiner is robust in the sense that it is secure as long as one of the candidate KEMs is secure.

In this chapter we elaborated on hybrid combiners that offer a secure way to combine classical and post-quantum secure signature schemes and KEMs. The next chapter presents a summary of the results of this thesis.

6 Conclusion

We, in this thesis, have presented post-quantum secure signature schemes that can be offered as an alternative for classical schemes. Additionally, we have also presented an approach on how to integrate post-quantum schemes to our current PKI to guarantee security within the Internet, in spite the potential presence of large-scale quantum computers. In particular, on the one hand, we constructed and analyzed lattice-based digital signature schemes. On the other hand, we constructed hybrid signature and KEM schemes that base their security on the security of the classical as well as, e.g., the lattice-based schemes. Based on these, we now summarize our results and provide directions for future research.

Our presented signature schemes TESLA and qTESLA have been proven to be secure against post-quantum adversaries as long as the LWE problem remains computationally hard. The tightness of our security reductions enabled choosing efficient quantum-resistant instantiations that are provably secure. The experimental results obtained from the software implementations of TESLA and qTESLA demonstrated their practicability. For instance, TESLA's run-time is 3 to 7 times faster and its signature size is about 12 times smaller than the corresponding values of GPV, which is the only other standard-lattice-based signature scheme with parameters chosen according to its quantum security reduction. Our findings indicate that qTESLA is much more efficient than TESLA. In particular, qTESLA's signing algorithm is faster than that of the other three lattice-based signature schemes submitted to NIST [164] for the same security level, while qTESLA's verification is faster than that of two of the other NIST submissions. Additionally, qTESLA's secret key is smallest among these four signature schemes. Furthermore, run-times of the provably secure instantiation qTESLA-p-III for signing are a factor 1.7 faster than RSA-3072. Moreover, the heuristic instantiation qTESLA-h-III-speed is 12 times faster for signing than and verification is as fast as RSA-3072. However, the key and signature sizes of qTESLA are larger than of those of RSA. For example, the sizes of qTESLA-h-III-size are 5 to 8 times larger than those of RSA-3072. For most applications, however, the larger key sizes of qTESLA do not seem to cause irreconcilable problems as exemplified next. The experiments by Herath and Stebila (see Section 5.2) or by Kampanakis et al. [131] indicate that standards such as X.509

and most implementations of the TLS or the IKEv2 protocols are able to handle key and signature sizes as they occur in qTESLA. Additionally, authentication of e-mails or software updates do not depend on small key or signature sizes and hence, the sizes of qTESLA should not pose an obstacle in such applications. Furthermore, as shown by Deng, Szefer, Tian, and Wang, it is possible to compile and run qTESLA on the VexRiscV platform, demonstrating its suitability for FPGAs. Nevertheless, some application rely on key and signature sizes that are smaller than RSA sizes and hence, qTESLA sizes. For examples, signature schemes used to instantiate aggregate network attestation [17] are required to be very small since the memory of some low-end embedded devices is only in the order of kilo-bytes. This is well below the overall qTESLA communication cost of public key and signatures.

Moving forwards, we also showed the vulnerability of lattice-based signatures against implementation attacks, which are summarized as follows. We first detected possible vulnerabilities against cache side channels in four subroutines of ring-TESLA–the predecessor of qTESLA. The potential vulnerabilities detected are during the rejection sampling, a signature validity check, and the sparse polynomial multiplication. We then proposed mitigations for all the vulnerabilities that we found in the ring-TESLA implementation. Our analyses demonstrated that rejection sampling and sparse multiplication should be implemented with particular carefulness. While these techniques are not very common in classical cryptography like RSA, they are common building blocks in lattice-based cryptographic schemes. To exemplify, rejection sampling occurs in most of the efficient lattice-based signature schemes such as [24, 81, 82] and in authenticated key exchange protocols such as [222]. Moreover, sparse multiplication also occurs in many lattice-based schemes [55, 81, 82]. Afterwards, we turned to fault attacks and presented a careful analysis of the signature schemes BLISS, ring-TESLA, and GLP with respect to randomizing, skipping, and zeroing faults. Our analysis showed that for 9 out of 15 considered attacks, at least one of the three schemes was vulnerable. All three schemes were vulnerable against zeroing faults during the sign algorithm, against zeroing faults during the verification, against skipping faults during the key generation, and against a skipping fault during the verification algorithm. Particularly interesting was that not only different ways of implementing led to different vulnerabilities, but also the different instantiations. As exemplified, BLISS and GLP were more vulnerable to the randomizing attack R-S-Sec during the key generation because of their sparse secrets than ring-TESLA which is instantiated with Gaussian distributed secrets. Due to the similarities of ring-TESLA and qTESLA, our implementation attack analysis and countermeasures can be applied to qTESLA as well.

Lastly in this thesis, we also showed how to build hybrid signature schemes and KEMs offering post-quantum security while preserving today's classical security

guarantees. As a result we presented three signature and three KEM combiners. Our analysis has shown that all of our combiners are robust in the sense that the resulting signature or KEM is as secure as the strongest of the two input signature schemes or KEMs. Hence, no matter if the classical scheme (e.g., because of existing large-scale quantum computers) or the post-quantum scheme (e.g., because of a sudden cryptanalytic break) becomes insecure, a hybrid scheme constructed using one of our combiners is still secure as long as at least one of the schemes is secure. Additionally, the applicability of our proposed combiners was demonstrated. For example, two of our KEM and one signature combiners, namely dPRF, nPRF, and Con, can be used in proposals for TLS 1.3. Other applications of our signature combiners are in X.509v3 or CMS of S/MIME. All our hybrid schemes have been proven to be secure in our novel two-stage adversary model that accounts for a fine-grained distinction of the adversary's quantum power. This family of security notions is thus, a unification and extension of existing quantum security models and it is easily transferable to other cryptographic primitives.

Future Work. Besides the potential improvements mentioned in the previous chapters, we consider the following research directions interesting and important.

The research area of protecting implementations of lattice-based schemes against physical attacks offers several important open questions. For instance, general power and electromagnetic attacks on lattice-based signatures have not yet been presented[15]. Moreover, this work is a starting point for fault analysis of lattice-based cryptographic schemes. As this work is not comprehensive, further detailed analyses can be carried out. For example, we did not analyze fault attacks targeting underlying computations such as the NTT. Additionally, the effects of zeroing attacks on the most (or least) significant bits of polynomial coefficients or the modulus is still an open question. In addition to these ideas for deeper side-channel and fault analysis, we also recommend investigating the effectiveness of proposed measures by software simulations for future research. This holds particular interest if countermeasures against several vulnerabilities are implemented. As a consequence of these future findings, the qTESLA implementation should be adapted. The current implementation of qTESLA is optimized for speed while it is also protected against the most common side-channels and a very powerful fault attack. To prevent more advanced (cache-, power-, and electromagnetic-) side-channel and fault attacks, the countermeasures discussed in this thesis and mitigations of potential future attacks should be implemented as a next step.

While it is important to guarantee the security of post-quantum secure schemes and their implementations, an interesting direction is to take one step further as

[15]The attack presented in [91] targets subroutines of the signature scheme BLISS that are rarely used in other lattice-based signature schemes.

explained next. This thesis mostly focused on constructions and analyses in the post-quantum world ($\mathsf{Q}^{\mathsf{c}}\mathsf{Q}$) where only the adversary accesses quantum computers. However, an interesting direction is to consider schemes and their security in a fully quantum world ($\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$) where all parties access quantum computers. This scenario has already been considered in our two-stage adversary model. Moreover, there exists an approach to construct $\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$-EUF-CMA secure signatures in the literature: Boneh and Zhandry [49] proposed a $\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$-EUF-CMA secure construction based on a $\mathsf{C}^{\mathsf{c}}\mathsf{C}$-EUF-CMA secure signature scheme. However, so far this approach does not seem to yield practical lattice-based $\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$-EUF-CMA secure signature schemes. To ensure authenticity, integrity, and non-repudiation of data in the Internet also in the future when quantum computers are widely deployed, it is important to provide practical $\mathsf{Q}^{\mathsf{q}}\mathsf{Q}$-EUF-CMA secure signature schemes.

Bibliography

[1] M. Abdalla, P.-A. Fouque, V. Lyubashevsky, and M. Tibouchi. Tightly-secure signatures from lossy identification schemes. In EUROCRYPT 2012, volume 7237 of LNCS, pages 572–590. Springer, 2012.

[2] A. Abel and J. Reineke. Reverse engineering of cache replacement policies in Intel microprocessors and their evaluation. In ISPASS, pages 141–142. IEEE Computer Society, 2014.

[3] O. Aciiçmez and Çetin Kaya. Koç. Trace-driven cache attacks on AES (short paper). In ICICS 06, volume 4307 of LNCS, pages 112–121. Springer, Heidelberg, 2006.

[4] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The EM Side-Channel(s). In CHES 2002, volume 2523 of LNCS, pages 29–45. Springer, 2003.

[5] M. R. Albrecht. On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL. In EUROCRYPT 2017, Part II, volume 10211 of LNCS, pages 103–129. Springer, Heidelberg, 2017.

[6] M. R. Albrecht, C. Cid, J. Faugère, R. Fitzpatrick, and L. Perret. Algebraic algorithms for LWE problems. ACM Comm. Computer Algebra, 49(2):62, 2015.

[7] M. R. Albrecht, C. Cid, J.-C. Faugère, R. Fitzpatrick, and L. Perret. On the complexity of the BKW algorithm on LWE. Designs, Codes and Cryptography, 74(2):325–354, 2015.

[8] M. R. Albrecht, B. R. Curtis, A. Deo, A. Davidson, R. Player, E. W. Postlethwaite, F. Virdia, and T. Wunderer. Estimate all the LWE, NTRU schemes! In SCN 2018, LNCS. Springer, 2018.

[9] M. R. Albrecht, R. Fitzpatrick, and F. Göpfert. On the efficacy of solving LWE by reduction to unique-SVP. In ICISC 13, volume 8565 of LNCS, pages 293–310. Springer, Heidelberg, 2014.

[10] M. R. Albrecht, F. Göpfert, F. Virdia, and T. Wunderer. Revisiting the expected cost of solving uSVP and applications to LWE. In ASIACRYPT 2017, Part I, volume 10624 of LNCS, pages 297–322. Springer, Heidelberg, 2017.

[11] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. Journal of Mathematical Cryptology, 9(3):169–203, 2015.

[12] A. Alim Kamal and A. M. Youssef. Strengthening hardware implementations of NTRUEncrypt against fault analysis attacks. Journal of Cryptographic Engineering, 3(4):227–240, 2013.

[13] E. Alkim, R. Avanzi, J. Bos, L. D. Ducas, A. de la Piedra, T. Pöppelmann, P. Schwabe, and D. Stebila. NewHope. NIST Post-Quantum Standardization [164], 2017. `https://newhopecrypto.org/`. Accessed: 2018-07-23.

[14] E. Alkim, J. W. Bos, L. Ducas, P. Longa, I. Mironov, M. Naehrig, V. Nikolaenko, C. Peikert, A. Raghunathan, D. Stebila, K. Easterbrook, and B. LaMacchia. FrodoKEM–Learning With Errors Key Encapsulation. NIST Post-Quantum Standardization [164], 2017. `https://frodokem.org/`. Accessed: 2018-07-23.

[15] E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange - A new hope. In USENIX Security 16, pages 327–343. USENIX Association, 2016.

[16] N. Allen, M. Anvari, E. Crockett, N. Drucker, V. Gheorghiu, S. Gueron, C. Paquin, T. Lepoint, S. Mishra, and D. Stebila. liboqs – nist-branch: C library for quantum-resistant cryptographic algorithms, 2018. GitHub at `https://github.com/open-quantum-safe/liboqs`, commit-id: 86c6ab1.

[17] M. Ambrosin, M. Conti, A. Ibrahim, G. Neven, A.-R. Sadeghi, and M. Schunter. SANA: Secure and scalable aggregate network attestation. In ACM CCS 16, pages 731–742. ACM Press, 2016.

[18] H. An, S. Kim, J. Lee, R. Choi, and K. Kim. Timing and Fault Attacks on Lattice-based Cryptographic Libraries. In SCIS 2017. The Institute of Electronics, Information and Communication Engineers, 2017.

[19] Y. Aono, P. Q. Nguyen, and Y. Shen. Quantum Lattice Enumeration and Tweaking Discrete Pruning. Cryptology ePrint Archive, Report 2018/546, 2018.

[20] Y. Aono, Y. Wang, T. Hayashi, and T. Takagi. Improved progressive BKZ algorithms and their precise cost estimation by sharp simulator. In EUROCRYPT 2016, Part I, volume 9665 of LNCS, pages 789–819. Springer, Heidelberg, 2016.

[21] S. Arora and R. Ge. New algorithms for learning in presence of errors. In ICALP 2011, Part I, volume 6755 of LNCS, pages 403–415. Springer, Heidelberg, 2011.

[22] A. Aysu, Y. Tobah, M. Tiwari, A. Gerstlauer, , and M. Orshansky. Horizontal Side-Channel Vulnerabilities of Post-Quantum Key Exchange Protocols. In HOST, pages 81–88. IEEE Computer Society, 2018.

[23] L. Babai. A las vegas-NC algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues. In 27th FOCS, pages 303–312. IEEE Computer Society Press, 1986.

[24] S. Bai and S. D. Galbraith. An improved compression technique for signatures based on learning with errors. In CT-RSA 2014, volume 8366 of LNCS, pages 28–47. Springer, Heidelberg, 2014.

[25] S. Bai and S. D. Galbraith. Lattice decoding attacks on binary LWE. In ACISP 14, volume 8544 of LNCS, pages 322–337. Springer, Heidelberg, 2014.

[26] R. E. Bansarkhani. LARA - A Design Concept for Lattice-based Encryption. Cryptology ePrint Archive, Report 2017/049, 2017.

[27] R. E. Bansarkhani and J. Buchmann. Improvement and efficient implementation of a lattice-based signature scheme. In SAC 2013, volume 8282 of LNCS, pages 48–67. Springer, Heidelberg, 2014.

[28] F. Bao, R. H. Deng, Y. Han, A. Jeng, A. D. Narasimhalu, and T. Ngair. Breaking public key cryptosystems on tamper resistant devices in the presence of transient faults. In Security Protocols: 5th International Workshop Paris, pages 115–124. Springer, 1998.

[29] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan. The sorcerer's apprentice guide to fault attacks. Proceedings of the IEEE, 94(2):370–382, 2006.

[30] P. S. L. M. Barreto, P. Longa, M. Naehrig, J. E. Ricardini, and G. Zanon. Sharper ring-LWE signatures. Cryptology ePrint Archive, Report 2016/1026, 2016.

[31] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In CRYPTO 2006, volume 4117 of LNCS, pages 602–619. Springer, Heidelberg, 2006.

[32] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In CRYPTO'96, volume 1109 of LNCS, pages 1–15. Springer, Heidelberg, 1996.

[33] M. Bellare and A. Lysyanskaya. Symmetric and dual PRFs from standard assumptions: A generic validation of an HMAC assumption. Cryptology ePrint Archive, Report 2015/1198, 2015.

[34] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In ACM CCS 93, pages 62–73. ACM Press, 1993.

[35] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In EUROCRYPT'94, volume 950 of LNCS, pages 92–111. Springer, Heidelberg, 1995.

[36] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani. Strengths and Weaknesses of Quantum Computing. SIAM Journal on Computing, 26(5), 1997.

[37] D. J. Bernstein. Cache-timing attacks on AES. Technical report, University of Illinois at Chicago, 2005.

[38] D. J. Bernstein. ChaCha, a variant of Salsa20. In SASC 2008, 2008.

[39] D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. Post-quantum cryptography. Mathematics and Statistics Springer-11649; ZDB-2-SMA. Springer, 2009.

[40] D. J. Bernstein, C. Dobraunig, M. Eichlseder, S. Fluhrer, S.-L. Gazdag, A. Hülsing, P. Kampanakis, S. Kölbl, T. Lange, M. M. Lauridsen, F. Mendel, R. Niederhagen, C. Rechberger, J. Rijneveld, and P. Schwabe. SPHINCS+. NIST Post-Quantum Standardization [164], 2017. `https://sphincs.org`. Accessed: 2018-07-23.

[41] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang. High-speed high-security signatures. In CHES 2011, volume 6917 of LNCS, pages 124–142. Springer, Heidelberg, 2011.

[42] J.-F. Biasse and F. Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In 27th SODA, pages 893–902. ACM-SIAM, 2016.

[43] J. Blömer, R. G. da Silva, P. Günther, J. Krämer, and J. Seifert. A Practical Second-Order Fault Attack against a Real-World Pairing Implementation. In FDTC 2014, pages 123–136. IEEE Computer Society, 2014.

[44] A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. In 32nd ACM STOC, pages 435–440. ACM Press, 2000.

[45] D. Boneh, Ö. Dagdelen, M. Fischlin, A. Lehmann, C. Schaffner, and M. Zhandry. Random oracles in a quantum world. In ASIACRYPT 2011, volume 7073 of LNCS, pages 41–69. Springer, Heidelberg, 2011.

[46] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults (extended abstract). In EUROCRYPT'97, volume 1233 of LNCS, pages 37–51. Springer, Heidelberg, 1997.

[47] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. Journal of Cryptology, 14(2):101–119, 2001.

[48] D. Boneh and M. Zhandry. Quantum-secure message authentication codes. In EUROCRYPT 2013, volume 7881 of LNCS, pages 592–608. Springer, Heidelberg, 2013.

[49] D. Boneh and M. Zhandry. Secure signatures and chosen ciphertext security in a quantum computing world. In CRYPTO 2013, Part II, volume 8043 of LNCS, pages 361–379. Springer, Heidelberg, 2013.

[50] J. Bornecrantz, Dolu1990, T. Kao, and T. Verbeure. VexRISCV–A FPGA friendly 32 bit RISC-V CPU implementation, 2017. GitHub at `https://github.com/SpinalHDL/VexRiscv`, commit-id: 7ab04a1.

[51] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, and D. Stehlé. CRYSTALS–Kyber: a CCA-secure module-lattice-based KEM. NIST Post-Quantum Standardization [164], 2017. `https://pq-crystals.org/kyber/index.shtml`. Accessed: 2018-07-23.

[52] J. W. Bos, C. Costello, L. Ducas, I. Mironov, M. Naehrig, V. Nikolaenko, A. Raghunathan, and D. Stebila. Frodo: Take off the Ring! Practical, Quantum-Secure Key Exchange from LWE. In CCS 2016. ACM, 2016.

[53] X. Boyen and Q. Li. Towards tightly secure lattice short signature and id-based encryption. In ASIACRYPT 2016, Part II, volume 2501, pages 404–434. Springer, 2016.

[54] M. Boyer, G. Brassard, P. Hoeyer, and A. Tapp. Tight bounds on quantum searching, 1996. `https://arxiv.org/abs/quant-ph/9605034`.

[55] J. Buchmann, F. Göpfert, T. Güneysu, T. Oder, and T. Pöppelmann. High-Performance and Lightweight Lattice-Based Public-Key Encryption. In IoTPTS@AsiaCCS, pages 2–9. ACM, 2016.

[56] bushing, marcan, and sven. Console hacking 2010 – ps3 epic fail. 27th Chaos Communication Congress, 2010. `https://events.ccc.de/congress/2010/Fahrplan/events/4087.en.html`. Accessed: 2018-07-23.

[57] P. Campbell, M. Groves, and D. Shepherd. SOLILOQUY: A cautionary tale. In ETSI 2nd Quantum-Safe Crypto Workshop. 2014.

[58] V. Chandrasekaran, B. Recht, P. A. Parrilo, and A. S. Willsky. The Convex Geometry of Linear Inverse Problems. Foundations of Computational Mathematics, 12(6):805–849, 2012.

[59] S. Chatterjee, N. Koblitz, A. Menezes, and P. Sarkar. Another Look at Tightness II: Practical Issues in Cryptography. In Mycrypt, volume 10311 of LNCS, pages 21–55. Springer, 2016.

[60] S. Chatterjee, A. Menezes, and P. Sarkar. Another Look at Tightness. In SAC 2011, volume 7118 of LNCS, pages 293–319. Springer, Heidelberg, 2012.

[61] C. Chen, J. Hoffstein, and W. W. Z. Zhang. pqNTRUSign–A modular lattice signature scheme. NIST Post-Quantum Standardization [164], 2017. `https://www.onboardsecurity.com/nist-post-quantum-crypto-submission` . Accessed: 2018-07-23.

[62] M.-S. Chen, A. Hülsing, J. Rijneveld, S. Samardjiska, and P. Schwabe. From 5-pass MQ-based identification to MQ-based signatures. In ASIACRYPT 2016, Part II, volume 10032 of LNCS, pages 135–165. Springer, Heidelberg, 2016.

[63] Y. Chen. Réduction de réseau et sécurité concréte du chiffrement complétement homomorphe. PhD thesis, École normale supérieure, Paris, 2013.

[64] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In ASIACRYPT 2011, volume 7073 of LNCS, pages 1–20. Springer, Heidelberg, 2011.

[65] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), 2008.

[66] C. Costello, K. Easterbrook, B. LaMacchia, P. Longa, and M. Naehrig. Lattice Cryptography Library, 2016. `https://www.microsoft.com/en-us/research/project/lattice-cryptography-library/`. Accessed: 2018-07-23.

[67] R. Cramer, L. Ducas, C. Peikert, and O. Regev. Recovering short generators of principal ideals in cyclotomic rings. In EUROCRYPT 2016, Part II, volume 9666 of LNCS, pages 559–585. Springer, Heidelberg, 2016.

[68] R. Cramer, L. Ducas, and B. Wesolowski. Short stickelberger class relations and application to ideal-SVP. In EUROCRYPT 2017, Part I, volume 10210 of LNCS, pages 324–348. Springer, Heidelberg, 2017.

[69] Ö. Dagdelen, R. E. Bansarkhani, F. Göpfert, T. Güneysu, T. Oder, T. Pöppelmann, A. H. Sánchez, and P. Schwabe. High-speed signatures from standard lattices. In LATINCRYPT 2014, volume 8895 of LNCS, pages 84–103. Springer, 2015.

[70] N. de Beaudrap, R. Cleve, and J. Watrous. Sharp Quantum versus Classical Query Complexity Separations. Algorithmica, 34(4):449–461, 2002.

[71] A. de Touzalin, C. Marcus, F. Heijman, I. Cirac, R. Murray, and T. Calarco. Quantum Manifesto–A New Era of Technology. `http://qurope.eu/manifesto`, 2016. Accessed: 2018-07-23.

[72] A. W. Dent. A designer's guide to KEMs. In 9th IMA International Conference on Cryptography and Coding, volume 2898 of LNCS, pages 133–151. Springer, Heidelberg, 2003.

[73] Des. Data Encryption Standard. In In FIPS PUB 46-3, Federal Information Processing Standards Publication, 1977.

[74] F. Dewald, H. Mantel, and A. Weber. AVR processors as a platform for language-based security. In ESORICS 2017, Part I, volume 10492 of LNCS, pages 427–445. Springer, Heidelberg, 2017.

[75] W. Diffie and M. E. Hellman. New directions in cryptography. IEEE Transactions on Information Theory, 22(6):644–654, 1976.

[76] Y. Dodis and J. Katz. Chosen-ciphertext security of multiple encryption. In TCC 2005, volume 3378 of LNCS, pages 188–209. Springer, Heidelberg, 2005.

[77] E. Dottax, C. Giraud, M. Rivain, and Y. Sierra. On second-order fault analysis resistance for CRT-RSA implementations. In WISTP 2009, volume 5746 of LNCS, pages 68–83. Springer, 2009.

[78] G. Doychev, B. Köpf, L. Mauborgne, and J. Reineke. CacheAudit: A Tool for the Static Analysis of Cache Side Channels. ACM TISSEC, 18(1):4:1–4:32, 2015.

[79] A. Duc, F. Tramèr, and S. Vaudenay. Better algorithms for LWE and LWR. In E. Oswald and M. Fischlin, editors, EUROCRYPT 2015, volume 9056 of LNCS, pages 173–202. Springer, 2015.

[80] L. Ducas. Accelerating bliss: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874, 2014.

[81] L. Ducas, A. Durmus, T. Lepoint, and V. Lyubashevsky. Lattice signatures and bimodal Gaussians. In CRYPTO 2013, Part I, volume 8042 of LNCS, pages 40–56. Springer, Heidelberg, 2013.

[82] L. Ducas, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé. CRYSTALS – Dilithium: Digital Signatures from Module Lattices. Cryptology ePrint Archive, Report 2017/633, 2017.

[83] L. Ducas and D. Micciancio. FHEW–a fully homomorphic encryption library, May 2017. GitHub at `https://github.com/lducas/FHEW`, commit-id: f53cd4b.

[84] L. Ducas and P. Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In ASIACRYPT 2012, volume 7658 of LNCS, pages 433–450. Springer, Heidelberg, 2012.

[85] M. Dürmuth. Novel classes of side channels and covert channels. PhD thesis, Saarland University, 2009.

[86] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In Proceedings of the 13th Internet Measurement Conference, 2013.

[87] M. J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. NIST, 2015.

[88] K. Eisenträger, S. Hallgren, A. Kitaev, and F. Song. A quantum algorithm for computing the unit group of an arbitrary degree number field. In 46th ACM STOC, pages 293–302. ACM Press, 2014.

[89] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In CRYPTO'84, volume 196 of LNCS, pages 10–18. Springer, Heidelberg, 1984.

[90] Y. Elias, K. E. Lauter, E. Ozman, and K. E. Stange. Provably weak instances of ring-LWE. In CRYPTO 2015, Part I, volume 9215 of LNCS, pages 63–92. Springer, Heidelberg, 2015.

[91] T. Espitau, P. Fouque, B. Gérard, and M. Tibouchi. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In CCS 2017, pages 1857–1874. ACM, 2017.

[92] T. Espitau, P.-A. Fouque, B. Gérard, and M. Tibouchi. Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. In SAC 2016, volume 10532 of LNCS, pages 140–158. Springer, Heidelberg, 2016.

[93] L. D. Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. Journal of Mathematical Cryptology, 8(3):209–247, 2014.

[94] A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In CRYPTO'86, volume 263 of LNCS, pages 186–194. Springer, Heidelberg, 1987.

[95] R. Fischlin and C.-P. Schnorr. Stronger security proofs for RSA and Rabin bits. Journal of Cryptology, 13(2):221–244, 2000.

[96] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU. NIST Post-Quantum Standardization [164], 2017. `https://falcon-sign.info/`. Accessed: 2018-07-23.

[97] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In CRYPTO'99, volume 1666 of LNCS, pages 537–554, 1999.

[98] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. Journal of Cryptology, 26(1):80–101, Jan. 2013.

[99] T. Gagliardoni. Quantum Security of Cryptographic Primitives. PhD thesis, Technische Universität Darmstadt, 2017.

[100] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In EUROCRYPT 2010, volume 6110 of LNCS, pages 257–278. Springer, Heidelberg, 2010.

[101] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In CHES 2001, volume 2162 of LNCS, pages 251–261. Springer, Heidelberg, 2001.

[102] S. Garg, C. Gentry, and S. Halevi. Candidate multilinear maps from ideal lattices. In EUROCRYPT 2013, volume 7881 of LNCS, pages 1–17. Springer, Heidelberg, 2013.

[103] Q. Ge, Y. Yarom, D. Cock, and G. Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. Journal of Cryptographic Engineering, 8(1):1–27, 2018.

[104] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In 40th ACM STOC, pages 197–206. ACM Press, 2008.

[105] F. Giacon, F. Heuer, and B. Poettering. KEM combiners. In PKC 2018, Part I, volume 10769 of LNCS, pages 190–218. Springer, Heidelberg, 2018.

[106] C. Giraud and E. W. Knudsen. Fault attacks on signature schemes. In ACISP 04, volume 3108 of LNCS, pages 478–491. Springer, Heidelberg, 2004.

[107] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing, 17(2):281–308, 1988.

[108] F. Göpfert. Securely Instantiating Cryptographic Schemes Based on the Learning with Errors Assumption. PhD thesis, Technische Universität Darmstadt, Germany, 2016.

[109] F. Göpfert, C. van Vredendaal, and T. Wunderer. A hybrid lattice basis reduction and quantum search attack on LWE. In PQCrypto 2017, volume 10346 of LNCS, pages 184–202. Springer, 2017.

[110] L. Groot Bruinderink, A. Hülsing, T. Lange, and Y. Yarom. Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme. In CHES 2016, volume 9813 of LNCS, pages 323–345. Springer, Heidelberg, 2016.

[111] L. Groot Bruinderink and P. Pessl. Differential Fault Attacks on Deterministic Lattice Signatures. IACR Transaction on Cryptographic Hardware and Embedded Systems, 2018(3), 2018.

[112] L. K. Grover. A fast quantum mechanical algorithm for database search. In 28th ACM STOC, pages 212–219. ACM Press, 1996.

[113] S. Gueron and F. Schlieker. Optimized implementation of ring-TESLA. GitHub at `https://github.com/fschlieker/ring-TESLA`, commit-id: b09d812, 2016.

[114] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann. Practical lattice-based cryptography: A signature scheme for embedded systems. In CHES 2012, volume 7428 of LNCS, pages 530–547. Springer, Heidelberg, 2012.

[115] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe. Software speed records for lattice-based signatures. In PQCrypto 2013, volume 7932 of LNCS, pages 67–82. Springer, 2013.

[116] Q. Guo, T. Johansson, and P. Stankovski. Coded-BKW: Solving LWE using lattice codes. In CRYPTO 2015, Part I, volume 9215 of LNCS, pages 23–42. Springer, Heidelberg, 2015.

[117] S. Halevi and V. Shoup. Algorithms in HElib. In CRYPTO 2014, Part I, volume 8616 of LNCS, pages 554–571. Springer, Heidelberg, 2014.

[118] S. Halevi and V. Shoup. HElib, 2014. GitHub at `https://github.com/shaih/HElib`, commit-id: f905e95.

[119] D. Harnik, J. Kilian, M. Naor, O. Reingold, and A. Rosen. On robust combiners for oblivious transfer and other primitives. In EUROCRYPT 2005, volume 3494 of LNCS, pages 96–113. Springer, Heidelberg, 2005.

[120] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices. In USENIX Security, pages 205–220. USENIX Association, 2012.

[121] J. Hoffstein, N. Howgrave-Graham, J. Pipher, and W. Whyte. Practical lattice-based cryptography: NTRUEncrypt and NTRUSign. ISC, pages 349–390. Springer, Heidelberg, 2010.

[122] J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, and W. Whyte. Practical signatures from the partial fourier recovery problem. In ACNS 14, volume 8479 of LNCS, pages 476–493. Springer, Heidelberg, 2014.

[123] D. Hofheinz, K. Hövelmanns, and E. Kiltz. A modular analysis of the Fujisaki-Okamoto transformation. In TCC 2017, Part I, volume 10677 of LNCS, pages 341–371. Springer, Heidelberg, 2017.

[124] R. Housley. Cryptographic Message Syntax (CMS). RFC 5652 (Internet standard), 2009.

[125] IBM Press Release. IBM Announces Advances to IBM Quantum Systems & Ecosystem. `https://www-03.ibm.com/press/us/en/pressrelease/53374.wss`, 2017. Accessed: 2018-05-21.

[126] Intel Corporation. Intel® 64 and IA-32 Architectures Optimization Reference Manual. Order Number: 248966-032, 2016.

[127] Internet Security Research Group. Let's encrypt stats. `https://letsencrypt.org/stats/`, 2018. Accessed: 2018-06-06.

[128] M. Jackson. 6 Things Quantum Computers Will Be Incredibly Useful For. `https://singularityhub.com/2017/06/25/6-things-quantum-computers-will-be-incredibly-useful-for/`, 2017. Accessed: 2018-07-23.

[129] S. Jana and V. Shmatikov. Memento: Learning secrets from process footprints. In 2012 IEEE Symposium on Security and Privacy, pages 143–157. IEEE Computer Society Press, 2012.

[130] A. A. Kamal and A. Youssef. Fault Analysis of the NTRUEncrypt Cryptosystem. IEICE Transactions, E94.A(4):1156–1158, 2011.

[131] P. Kampanakis, P. Panburana, E. Daw, and D. V. Geest. The Viability of Post-quantum X.509 Certificates. Cryptology ePrint Archive, Report 2018/063, 2018.

[132] M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen. pqm4–Post-quantum crypto library for the ARM Cortex-M4, 2018. GitHub at `https://github.com/mupq/pqm4`, commit-id: 133c0e8.

[133] A. Karmakar, S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Constant-time Discrete Gaussian Sampling. IEEE Trans. Computers, 2018. To be published.

[134] J. Katz and Y. Lindell. Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC, 2007.

[135] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In ACM CCS 03, pages 155–164. ACM Press, 2003.

[136] J. Kelly. A Preview of Bristlecone, Google's New Quantum Processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`, 2018. Accessed: 2018-07-23.

[137] J. Kelsey. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash. NIST Special Publication, 800:185, 2016.

[138] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Side channel cryptanalysis of product ciphers. In ESORICS'98, volume 1485 of LNCS, pages 97–110. Springer, Heidelberg, 1998.

[139] E. Kiltz, V. Lyubashevsky, and C. Schaffner. A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model. In EUROCRYPT 2018, Part III, volume 10822 of LNCS, pages 552–586. Springer, 2018.

[140] P. Kirchner and P.-A. Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In CRYPTO 2015, Part I, volume 9215 of LNCS, pages 43–62. Springer, Heidelberg, 2015.

[141] N. Koblitz and A. Menezes. Another Look at "Provable Security" II. In INDOCRYPT 2006, volume 4329 of LNCS. Springer, 2006.

[142] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In CRYPTO'96, volume 1109 of LNCS, pages 104–113. Springer, Heidelberg, 1996.

[143] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In CRYPTO'99, volume 1666 of LNCS, pages 388–397. Springer, Heidelberg, 1999.

[144] J. Krämer. Why cryptography should not rely on physical attack complexity. PhD thesis, Berlin Institute of Technology, 2015.

[145] H. Krawczyk. SIGMA: The "SIGn-and-MAc" approach to authenticated Diffie-Hellman and its use in the IKE protocols. In CRYPTO 2003, volume 2729 of LNCS, pages 400–425. Springer, Heidelberg, 2003.

[146] H. Krawczyk and P. Eronen. HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (Informational), 2010.

[147] T. Laarhoven. Search problems in cryptography. PhD thesis, Eindhoven University of Technology, 2016.

[148] T. Laarhoven, M. Mosca, and J. Pol. Solving the Shortest Vector Problem in Lattices Faster Using Quantum Search. In PQCrypto 2013, volume 7932 of LNCS, pages 83–101. Springer, 2013.

[149] L. Lamport. Constructing digital signatures from a one-way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory, 1979.

[150] M. Lee, J. E. Song, D. Choi, and D. Han. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. IEICE Transactions, 93-A(1):153–163, 2010.

[151] R. Lindner and C. Peikert. Better key sizes (and attacks) for LWE-based encryption. In CT-RSA 2011, volume 6558 of LNCS, pages 319–339. Springer, Heidelberg, 2011.

[152] M. Liu and P. Q. Nguyen. Solving BDD by enumeration: An update. In CT-RSA 2013, volume 7779 of LNCS, pages 293–309. Springer, Heidelberg, 2013.

[153] V. Lomné, E. Prouff, M. Rivain, T. Roche, and A. Thillard. How to estimate the success rate of higher-order side-channel attacks. In CHES 2014, volume 8731 of LNCS, pages 35–54. Springer, 2014.

[154] V. Lyubashevsky. Lattice signatures without trapdoors. In EUROCRYPT 2012, volume 7237 of LNCS, pages 738–755. Springer, Heidelberg, 2012.

[155] V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. In EUROCRYPT 2010, volume 6110 of LNCS, pages 1–23. Springer, Heidelberg, 2010.

[156] S. Mangard, E. Oswald, and T. Popp. Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security). Springer, 2007.

[157] T. C. May and M. H. Woods. A new physical mechanism for soft errors in dynamic memories. In 16th International Reliability Physics Symposium, pages 33–40, 1978.

[158] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[159] R. C. Merkle. A certified digital signature. In CRYPTO'89, volume 435 of LNCS, pages 218–238. Springer, Heidelberg, 1990.

[160] M. Mosca. Cybersecurity in an era with quantum computers: Will we be ready? Cryptology ePrint Archive, Report 2015/1075, 2015.

[161] M. Mosca and D. Stebila. Open quantum safe – software for prototyping quantum-resistant cryptography, 2018. `https://openquantumsafe.org/`. Accessed: 2018-07-26.

[162] D. M'Raïhi, D. Naccache, D. Pointcheval, and S. Vaudenay. Computational alternatives to random number generators. In SAC 1998, volume 1556 of LNCS, pages 72–80. Springer, Heidelberg, 1999.

[163] D. Naccache, P. Q. Nguyen, M. Tunstall, and C. Whelan. Experimenting with faults, lattices and the DSA. In PKC 2005, volume 3386 of LNCS, pages 16–28. Springer, Heidelberg, 2005.

[164] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography Standardization. `https://csrc.nist.gov/projects/post-quantum-cryptography`, 2017. Accessed: 2018-07-23.

[165] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography: Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process. `https://csrc.nist.gov/CSRC/media/Projects/Post-Quantum-Cryptography/documents/call-for-proposals-final-dec-2016.pdf`, 2017. Accessed: 2018-07-23.

[166] P. Q. Nguyen and O. Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In EUROCRYPT 2006, volume 4004 of LNCS, pages 271–288. Springer, Heidelberg, 2006.

[167] M. A. Nielsen and I. L. Chuang. Quantum Computation and Quantum Information. Cambridge University Press, 2000.

[168] T. Oder, T. Schneider, T. Pöppelmann, and T. Güneysu. Practical CCA2-secure and masked ring-LWE implementation. Cryptology ePrint Archive, Report 2016/1109, 2016.

[169] D. A. Osvik, A. Shamir, and E. Tromer. Cache Attacks and Countermeasures: The Case of AES. In CT-RSA, pages 1–20, 2006.

[170] D. Page. Theoretical use of cache memory as a cryptanalytic side-channel. Cryptology ePrint Archive, Report 2002/169, 2002.

[171] A. Park and D. Han. Chosen ciphertext Simple Power Analysis on software 8-bit implementation of ring-LWE encryption. In AsianHOST 2016, pages 1–6. IEEE Computer Society, 2016.

[172] E. Peeters. Side-channel cryptanalysis: A brief survey. In Advanced DPA Theory and Practice: Towards the Security Limits of Secure Embedded Circuits, chapter 2, pages 11–19. Springer Science+Business Media, 2013.

[173] C. Peikert. A decade of lattice cryptography. Foundations and Trends in Theoretical Computer Science, 10(4):283–424, 2016.

[174] C. Peikert and B. Waters. Lossy trapdoor functions and their applications. In 40th ACM STOC, pages 187–196. ACM Press, 2008.

[175] S. Perez. Amazon shipped over 5 billion items with prime in 2017. `https://techcrunch.com/2018/01/02/amazon-shipped-over-5-billion-items-with-prime-in-2017/`, 2018. Accessed: 2018-06-05.

[176] P. Pessl. Analyzing the shuffling side-channel countermeasure for lattice-based signatures. In INDOCRYPT 2016, pages 153–170. Springer, 2016.

[177] P. Pessl, L. G. Bruinderink, and Y. Yarom. To BLISS-B or not to be – Attacking strongSwan's Implementation of Post-Quantum Signatures. In CCS 2017, pages 1843–1855. ACM, 2017.

[178] T. Plantard, A. Sipasseuth, C. Dumondelle, and W. Susilo. DRS: Diagonal dominant Reduction for lattice-based Signature. NIST Post-Quantum Standardization [164], 2017. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions` . Accessed: 2018-07-23.

[179] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler. Attacking Deterministic Signature Schemes Using Fault Attacks. In EuroS&P 2018, pages 338–352, 2018.

[180] D. Pointcheval and J. Stern. Security proofs for signature schemes. In EUROCRYPT'96, volume 1070 of LNCS, pages 387–398. Springer, Heidelberg, 1996.

[181] T. Pöppelmann and T. Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. In SAC 2013, volume 8282 of LNCS, pages 68–85. Springer, Heidelberg, 2014.

[182] T. Pöppelmann, T. Oder, and T. Güneysu. High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers. In

LATINCRYPT 2015, volume 9230 of LNCS, pages 346–365. Springer, Heidelberg, 2015.

[183] T. Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979 (Informational), 2013.

[184] R. Primas, P. Pessl, and S. Mangard. Single-trace side-channel attacks on masked lattice-based encryption. In CHES 2017, volume 10529 of LNCS, pages 513–533. Springer, 2017.

[185] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. Quantum Information & Computation, 3(4):317–344, 2003.

[186] J.-J. Quisquater. A new tool for non-intrusive analysis of smart cards based on electro-magnetic emissions: the SEMA and DEMA methods. Rump session EUROCRYPT 2000, 2000.

[187] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. In Smart Card Programming and Security, pages 200–210. Springer, 2001.

[188] M. O. Rabin. Digital signatures and public key functions as intractable as factorization. Technical Report MIT/LCS/TR-212, Massachusetts Institute of Technology, 1979.

[189] B. Ramsdell and S. Turner. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification. RFC 5751 (Proposed Standard), 2010.

[190] P. Rauzy and S. Guilley. Countermeasures against High-Order Fault-Injection Attacks on CRT-RSA. In FDTC 2014, pages 68–82. IEEE Computer Society, 2014.

[191] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In 37th ACM STOC, pages 84–93. ACM Press, 2005.

[192] O. Reparaz, R. de Clercq, S. S. Roy, F. Vercauteren, and I. Verbauwhede. Additively Homomorphic Ring-LWE Masking. In PQCrypto 2016, volume 9606 of LNCS, pages 233–244. Springer, 2016.

[193] O. Reparaz, S. S. Roy, F. Vercauteren, and I. Verbauwhede. A masked ring-LWE implementation. In CHES 2015, volume 9293 of LNCS, pages 683–702. Springer, Heidelberg, 2015.

[194] E. Rescorla. The Transport Layer Security (TLS) protocol version 1.3, draft 19, 2017. `https://tools.ietf.org/html/draft-ietf-tls-tls13-19`.

[195] E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3 – draft-ietf-tls-tls13-23. `https://tools.ietf.org/html/draft-ietf-tls-tls13-23`, 2018.

[196] R. L. Rivest, A. Shamir, and L. M. Adleman. A method for obtaining digital signature and public-key cryptosystems. Communications of the Association for Computing Machinery, 21(2):120–126, 1978.

[197] P. Rohatgi. Electromagnetic attacks and countermeasures. In Cryptographic Engineering, pages 407–430. Springer, 2009.

[198] S. S. Roy, O. Reparaz, F. Vercauteren, and I. Verbauwhede. Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014.

[199] M.-J. O. Saarinen. Arithmetic coding and blinding countermeasures for lattice signatures. Journal of Cryptographic Engineering, 2017.

[200] SafeCrypto. NIST Software Analysis–Signatures. `https://www.safecrypto.eu/pqclounge/software-analysis-signatures/`. Accessed: 2018-07-05.

[201] J. Schank and D. Stebila. A Transport Layer Security (TLS) Extension For Establishing An Additional Shared Secret draft-schanck-tls-additional-keyshare-00. `https://tools.ietf.org/html/draft-schanck-tls-additional-keyshare-00`, 2017.

[202] C. Schnorr. Lattice reduction by random sampling and birthday methods. In STACS 2003, volume 2607 of LNCS, pages 145–156. Springer, 2003.

[203] C.-P. Schnorr. Efficient identification and signatures for smart cards. In CRYPTO'89, volume 435 of LNCS, pages 239–252. Springer, Heidelberg, 1990.

[204] J.-P. Seifert. On authenticated computing and RSA-based authentication. In ACM CCS 05, pages 122–127. ACM Press, 2005.

[205] H. Seo, Z. Liu, T. Park, H. Kwon, S. Lee, and H. Kim. Secure Number Theoretic Transform and Speed Record for Ring-LWE Encryption on Embedded Processors. In ICISC 2017–Revised Selected Papers, volume 10779 of LNCS, pages 175–188. Springer, 2017.

[206] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing, 26:1484–1509, 1997.

[207] J. H. Silverman and W. Whyte. Timing attacks on NTRUEncrypt via variation in the number of hash calls. In CT-RSA 2007, volume 4377 of LNCS, pages 208–224. Springer, Heidelberg, 2007.

[208] F. Song. A Note on Quantum Security for Post-Quantum Cryptography, volume 8772 of LNCS, pages 246–265. Springer, 2014.

[209] R. Spreitzer, V. Moonsamy, T. Korak, and S. Mangard. Sok: Systematic classification of side-channel attacks on mobile devices. CoRR, abs/1611.03748, 2016.

[210] E. E. Targhi and D. Unruh. Post-quantum security of the Fujisaki-Okamoto and OAEP transforms. In TCC 2016-B, Part II, volume 9986 of LNCS, pages 192–216. Springer, Heidelberg, 2016.

[211] W. van Eck. Electromagnetic radiation from video display units: An eavesdropping risk? Comput. Secur., 4(4):269–286, 1985.

[212] I. Verbauwhede, D. Karaklajic, and J. Schmidt. The Fault Attack Jungle - A Classification Model to Guide You. In FDTC 2011, pages 3–8, 2011.

[213] N. V. Vizev. Side Channel Attacks on NTRUEncrypt. Bachelor thesis, Technische Universität Darmstadt, 2007.

[214] J. von Neumann. Various techniques used in connection with random digits. In Monte Carlo Method, volume 12 of National Bureau of Standards Applied Mathematics Series, pages 36–38. 1951.

[215] A. Wang, X. Zheng, and Z. Wang. Power Analysis Attacks and Countermeasures on NTRU-Based Wireless Body Area Networks. Transactions on Internet and Information Systems, 7(5):1094–1107, 2013.

[216] W. Whyte, S. Fluhrer, Z. Zhang, and O. Garcia-Morchon. Quantum-Safe Hybrid (QSH) Key Exchange for Transport Layer Security (TLS) version 1.3 draft-whyte-qsh-tls13-06. `https://tools.ietf.org/html/draft-whyte-qsh-tls13-06`, 2017.

[217] T. Wunderer. Revisiting the hybrid attack: Improved analysis and refined security estimates. Cryptology ePrint Archive, Report 2016/733, 2016.

[218] Y. Yarom and K. Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In USENIX Security, pages 719–732, 2014.

[219] Y. Yu and L. Ducas. Learning strikes again: the case of the DRS signature scheme. Cryptology ePrint Archive, Report 2018/294, 2018.

[220] M. Zhandry. How to construct quantum random functions. In 53rd FOCS, pages 679–687. IEEE Computer Society Press, 2012.

[221] M. Zhandry. Secure identity-based encryption in the quantum random oracle model. In CRYPTO 2012, volume 7417 of LNCS, pages 758–775. Springer, Heidelberg, 2012.

[222] J. Zhang, Z. Zhang, J. Ding, M. Snook, and Ö. Dagdelen. Authenticated key exchange from ideal lattices. In EUROCRYPT 2015, Part II, volume 9057 of LNCS, pages 719–751. Springer, Heidelberg, 2015.

[223] R. Zhang, G. Hanaoka, J. Shikata, and H. Imai. On the security of multiple encryption or CCA-security+CCA-security=CCA-security? In PKC 2004, volume 2947 of LNCS, pages 360–374. Springer, Heidelberg, 2004.

[224] X. Zheng, A. Wang, and W. Wei. First-order collision attack on protected NTRU cryptosystem. Microprocessors and Microsystems - Embedded Hardware Design, 37(6-7):601–609, 2013.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, August 2018 _____

Wissenschaftlicher Werdegang

Juni 2014 - heute

Wissenschaftliche Mitarbeiterin in der Arbeitsgruppe von Prof. Dr. Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt

Oktober 2011 - April 2014

Studium im Studiengang „Mathematics International“(Master of Science) an der Technischen Universität Kaiserslautern

Oktober 2008 - September 2011

Studium im Studiengang „Mathematik mit Nebenfach Informatik“(Bachelor of Science) an der Technischen Universität Kaiserslautern