On Lattice-Based Signatures with Advanced Functionalities

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor rerum naturalium (Dr. rer. nat.)

von

Nabil Alkeilani Alkadri, M.Sc.

geboren in Dortmund.

Referenten:	Prof. Dr. Johannes Buchmann Prof. Dr. Sebastian Faust
Tag der Einreichung:	10.01.2022
Tag der mündlichen Prüfung:	24.02.2022
Hochschulkennziffer:	D 17

Darmstadt 2022

On Lattice-Based Signatures with Advanced Functionalities
Genehmigte Dissertation von Nabil Alkeilani Alkadri, M.Sc.
Technische Universität Darmstadt, Darmstadt, Germany
Tag der mündlichen Prüfung: 24.02.2022
Jahr der Veröffentlichung der Dissertation auf TUprints: 2022
URN: urn:nbn:de:tuda-tuprints-207938
URL: https://tuprints.ulb.tu-darmstadt.de/id/eprint/20793

Abstract

Lattice-based cryptography is a prominent class of cryptographic systems that has been emerged as one of the main candidates replacing classical cryptography in future computing environments such as quantum computing. Quantum computers exploit quantum mechanical phenomena to solve computational problems, on which the security of currently deployed (classical) cryptographic systems is based. While these computational problems, *e.g.*, factoring integers and computing discrete logarithms, are intractable for conventional (classical) computers, it is meanwhile known that they can be easily solved on quantum computers (Shor 1997). However, lattice problems, such as finding short non-zero vectors, seem to withstand attacks having quantum computing power.

In the last two decades we have seen many cryptographic proposals based on lattices. In particular, lattice-based (ordinary) signature schemes were greatly improved with respect to efficiency and security. This can be observed from the post-quantum standardization process initiated by the *National Institute of Standards and Technology (NIST)*. In fact, from the five signature schemes that have been submitted to this process, there are currently three finalists, where two of them are lattice-based submissions. In this thesis, we are specifically interested in lattice-based signature schemes with advanced functionalities. In addition to the basic security goals that an ordinary signature scheme ensures, *i.e.*, authentication, non-repudiation, and integrity, these schemes provide features that are application-specific. While ordinary signature schemes based on lattices are ready to be deployed in practice, this statement cannot be made for lattice-based signature schemes with advanced functionalities. This thesis makes a significant progress towards deploying the aforementioned type of signature schemes in practice.

With focus on privacy-preserving applications in future computing environments, we particularly facilitate the protection of secret keys in cryptocurrencies such as Bitcoin and Ethereum. We provide practical solutions to anonymous e-cash, anonymous credentials, smart contracts, and e-voting. We believe that our techniques can be used to develop further advanced signature schemes to be deployed in other application scenarios. For instance, in information security systems that perform critical operations such as distributed key generation, anonymization of medical data, and updating reliable routing information.

List of Publications

[A1] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20: 27th Conference on Computer and Communications Security*, pages 1017–1031, Virtual Event, USA, November 9–13, 2020. ACM Press. (cited on pages 23, 33, and 35.)

[A2] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. BLAZE: Practical lattice-based blind signatures for privacy-preserving applications. In Joseph Bonneau and Nadia Heninger, editors, *FC 20: 24th International Conference on Financial Cryptography and Data Security*, volume 12059 of *Lecture Notes in Computer Science*, pages 484–502, Kota Kinabalu, Malaysia, February 10–14, 2020. Springer, Cham. (cited on page 56.)

[A3] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. On lattice-based interactive protocols: An approach with less or no aborts. In Joseph K. Liu and Hui Cui, editors, *ACISP 20: 25th Australasian Conference on Information Security and Privacy*, volume 12248 of *Lecture Notes in Computer Science*, pages 41–61, Perth, WA, Australia, November 30 – December 2, 2020. Springer, Cham. (cited on pages 41 and 56.)

[A4] Nabil Alkeilani Alkadri, Patrick Harasser, and Christian Janson. BlindOR: An efficient lattice-based blind signature scheme from OR-proofs. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *CANS 21: 20th International Conference on Cryptology and Network Security*, *Lecture Notes in Computer Science*, pages 95–115 , Vienna, Austria, December 13–15, 2021. Springer, Cham. (cited on page 56.)

[A5] Nabil Alkeilani Alkadri, Michael Burger, and Giang Nam Nguyen. Optimized implementations of lattice-based blind signature schemes. Work in progress. (cited on page 102.)

[A6] Nabil Alkeilani Alkadri, Johannes Buchmann, Rachid El Bansarkhani, and Juliane Krämer. A framework to select parameters for lattice-based cryptography. Cryptology ePrint Archive, Report 2017/615, 2017. http://eprint.iacr.org/2017/615.

[A7] Johannes Buchmann, Juliane Krämer, Nabil Alkeilani Alkadri, Nina Bindel, Rachid El Bansarkhani, Florian Göpfert, and Thomas Wunderer. Bewertung gitterbasierter kryptografischer Verfahren (Evaluation of lattice-based cryptographic algorithms). Bundesamt für Sicherheit in der Informationstechnik (Federal Office for Information Security), 2019.
https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Gitterbasierte_Verfahren/Gitterbasierte_Verfahren.html.

[A8] Nabil Alkeilani Alkadri. Post-quantum commitment schemes. 25. Krypto-Tag, SAP, Walldorf, Germany, September 2016. https://fg-krypto.gi.de/krypto-tag/.

Contents

1 Introduction

Digital signatures are a fundamental cryptographic ingredient and one of the most critical cryptographic primitives recognized everywhere in our daily digital activities such as online banking, software updates, web authentication, and many others. Basically, a digital signature scheme replaces a hand-written signature to ensure the authenticity of a signer. It allows to generate signatures on documents by using a secret (signing) key, which is only known to the signer. The correctness of these signatures can be verified by any entity using a public (verification) key that corresponds to the secret key. The problem of public key authentication, *i.e.*, the assertion to which entity a public key belongs, is usually solved using certification authorities. In addition to authenticity, digital signatures even ensure non-repudiation and integrity of signed documents. For instance, when using digital signatures to establish a secure connection between two parties, the communication partners can be aware with whom they communicate (authentication). They also know that sending and receiving data cannot be denied (non-repudiation), and the exchanged data has not been altered during interaction (integrity). Specific applications, such as cryptocurrencies, anonymous credentials, smart contracts, anonymous membership authentication, electronic cash, and electronic voting, require digital signature schemes to be endowed with advanced functionalities (additional features).

There is a numerous number of (advanced) signature schemes built from different classes of cryptographic systems. The currently deployed class is called *classical cryptography*. Its security is based on the hardness of number-theoretic assumptions such as the integer factorization and the discrete logarithm problem. While we are convinced that classical cryptography is secure under conventional (classical) computer attacks, it is meanwhile known that alternative classes of secure cryptographic systems must be used in future computing environments such as quantum computing. Quantum computers exploit quantum mechanical phenomena to solve computational problems that are intractable for classical computers. In particular, factoring integers and computing discrete logarithms can be efficiently carried out on large-scale quantum computers via Shor's algorithm [Sho97]. In fact, the enormous efforts that both research and industry are currently making, *e.g.*, by *IBM* [Gam20], indicate that in the near future sufficiently large quantum computers will be constructed. Mosca [Mos18] estimated that quantum computers having the capability of breaking the security of classical cryptographic schemes could exist within the next decade.

In an effort to prepare our information security systems to resist future attacks having quantum computing power, the *National Institute of Standards and Technology (NIST)* has initiated a process to develop and standardize post-quantum (quantum-resistant) cryptographic systems including digital signature schemes, public-key encryption, and key encapsulation mechanisms [Nat17]. Lattice-based cryptography is a prominent class of cryptographic systems that has been emerged as one of the main candidates providing post-quantum security[1]. In fact, among the 64 proposals submitted to the first round of the post-quantum standardization process initiated by the NIST, there were 26 (over 40%) lattice-based submissions. From the 26 candidates left in the second round, there were 12 proposals based on lattices. The finalists of the third round are seven submissions, where five of them are lattice-based schemes.

Lattice-based signature schemes were greatly improved in the last two decades with respect to efficiency and security. This can be observed from the most recent proposals, *e.g.* [DKL+18, ABB+20], which have been submitted to the NIST standardization process. However, the focus has been mainly made on ordinary schemes, *i.e.*, without advanced functionalities. This thesis makes a significant step towards adding application-specific functionalities to lattice-based signature schemes. In other words, the focus of this thesis is on lattice-based signature schemes with advanced functionalities that can be deployed in practice as a future replacement to the (currently used) classical schemes.

1.1 Summary of Results

In the following we give a description of the main contributions and structure of this thesis, in addition to a summary of related work.

Chapter 2: Background

This chapter fixes some notation and presents the basic definitions and tools for this thesis. This includes basic background on lattices and the relevant lattice problems, digital signature schemes and their security notion, and the fundamental tools used in our security proofs including the (quantum) random oracle model and the forking lemma.

Chapter 3: Signature Schemes with Re-Randomizable Keys

A signature scheme with re-randomizable keys is a signature scheme that allows both the public and secret key to be re-randomized in a separate but consistent way. It was first introduced by Fleischhacker *et al.* [FKM+16] in order to construct unlinkable sanitizable signatures [ACdMT05, BFLS10], which can be used, for example, to anonymize medical data and update reliable routing information. Furthermore, signature schemes with re-randomizable keys constitute the main building block for constructing deterministic wallets [DFL19], which are used in cryptocurrencies, such as Bitcoin and Ethereum, to protect secret keys against theft, *i.e.*, to keep the funds of users safe.

[1] The remaining classes of cryptographic systems that are known to provide post-quantum security are hash-based, code-based, isogeny-based, and multivariate cryptography.

We present the first post-quantum signature scheme with re-randomizable keys, which is based on lattices over modules. We propose concrete parameters for our scheme targeting 128 bits of post-quantum security[2]. We show that our construction can be used to build post-quantum deterministic wallets, and further show how it can be instantiated with the ordinary signature scheme qTESLA [ABB+20], which progressed to the second round of the NIST post-quantum standardization process. Moreover, we present further lattice-based approaches for re-randomizing keys including a scheme that has other potential applications such as sanitizable signatures.

In addition to the standard algorithms of an ordinary signature scheme, a signature scheme with re-randomizable keys has two algorithms for re-randomizing the public and secret key. Fleischhacker *et al.* [FKM+16] showed that the secret key of Schnorr's signature scheme [Sch91] can be re-randomized additively. More concretely, the secret key re-randomization algorithm adds some randomness to the secret key, while the public key re-randomization algorithm computes the related re-randomized public key using the same randomness, but without access to any secret key. We show how to realize this approach in the lattice setting using different techniques. This is attained by considering the typical distributions of the secret key used in lattice-based schemes, *i.e.*, the Gaussian distribution and the uniform distribution over some small set.

Chapter 4: Reducing Restarts in Interactive Protocols

Most constructions of lattice-based signatures, including those with advanced functionalities, follow either the hash-and-sign [DH76] or the Fiat-Shamir (with aborts) [FS87, Lyu09] paradigm. All schemes following the latter approach share a crucial procedure when signing messages, *i.e.*, the so-called *rejection sampling* procedure. In general, this procedure is a common tool from statistics that was introduced by von Neumann [vN51]. It allows to sample from an arbitrary target distribution given a bound to some different starting distribution. In the context of Fiat-Shamir signatures based on lattices, rejection sampling is used as a security check. Using a so-called masking term, it allows to verify that a secret (or secret-related) term is concealed and independently distributed from a public term that is computed using both the masking and secret term. For example, it makes sure that computed signatures are distributed independently from the secret key. If this check fails, *i.e.*, if rejection sampling does not accept, the signing protocol is restarted in order to sample a fresh masking term. This is because all computations carried out up to and including the rejection sampling procedure are related to a certain masking term.

While the aforementioned security check does not affect the efficiency of constructions with one rejection sampling procedure, like (non-interactive) ordinary signatures, it has a significant negative impact on the efficiency of interactive protocols with multiple rejection sampling procedures, such as multi-signatures [ES16] and blind signatures [Rüc10]. This is because the number of restarts becomes multiplicative in the number of procedures.

We present a new technique that we call *tree of commitments*. It allows to reduce the number of restarts, or even remove the restarts, inherent in lattice-based protocols. A tree of commitments is a binary hash tree whose leaves are constructed from many masking terms.

[2]Nowadays, 128 bits is a commonly accepted level of security.

This allows a lattice-based protocol to iteratively apply rejection sampling using different masking terms in one execution. In this way, the number of restarts is reduced. Moreover, a large enough number of masking terms even allows to completely eliminate restarts. We present further optimizations that can be exploited when using trees of commitments. This includes saving complete subtrees to use it in the next execution of the protocol.

Using our tree of commitments technique, we construct a lattice-based canonical identification scheme, which is built on top of the scheme by Lyubashevsky [Lyu09]. We show that our protocol has a reduced amount of communication complexity compared to the one by [Lyu09]. Canonical identification schemes [AABN02] constitute the main building block of various types of signature schemes following the Fiat-Shamir approach, *e.g.* [Lyu09, Rüc10, Lyu12, BG14, ES16, DKL+18, BLO18, ABB+20]. In the next chapter we demonstrate the practical relevance of our tree of commitments technique in the context of blind signatures.

We remark that it is possible to sample a masking term from a distribution that outputs large enough elements such that rejection sampling succeeds after a small fixed number of restarts. This approach is already established in previous works as a trade-off between performance and sizes (see, *e.g.* [Lyu12, BG14, DKL+18]), but it does not solve the problem for protocols with multiple rejection sampling procedures. Other works, *e.g.*, the one by Goldwasser *et al.* [GKPV10], suggest to use the so-called *noise flooding* technique. It uses masking terms that are sampled from distributions of a very large size such that rejection sampling accepts with very high probability. However, the negative impact on the efficiency is tremendous as the sizes of the parameters become very large. Furthermore, the concept of binary hash trees has been already suggested, *e.g.*, by Katz *et al.* [KKW18], in order to reduce the communication complexity of (zero-knowledge) proof systems, but not the number of restarts inherent in lattice-based protocols.

Chapter 5: Blind Signature Schemes

A blind signature scheme is an interactive protocol between a signer and a user. It allows the user to generate signatures on messages in a way that prevents the signer from gaining any information about the messages during the interaction. On the other hand, it ensures that the user cannot generate any valid blind signature without interacting with the signer. Blind signatures, first introduced by Chaum [Cha82], are a fundamental building block in privacy-preserving cryptographic applications such anonymous credentials [BL13a], electronic voting [KKS17], and blockchain protocols [HBG16].

We present three new blind signature schemes that we call BLAZE, BLAZE+, and BlindOR. We propose concrete parameters for our schemes targeting 128 bits of security, and provide the corresponding sizes of keys and signatures as well as the communication cost required to generate blind signatures. Furthermore, we show the insecurity of the lattice-based blind signature schemes given in [CCT+11, ZM14, ZH16, GHWX16, GHW+17, ZTZ+17], which follow the hash-and-sign approach and are based on preimage sampleable trapdoor functions [GPV08].

Our first proposal BLAZE improves upon the first lattice-based scheme introduced by Rückert [Rüc10], which follows the Fiat-Shamir approach and involves three security checks (rejection sampling procedures). We identified the main source of its inefficiency and in-

ability to adapt it in practical applications, *i.e.*, the first security check that is carried out at the first user stage. BLAZE removes this check via a new technique that we call *partitioning and permutation*, which may be of independent interest. It allows to hide specific elements so that blindness is satisfied at the first user stage without carrying out any security check. Furthermore, it speeds up the signing process and reduces the sizes of keys and signatures.

Unlike the Fiat-Shamir approach, the signing protocol of BLAZE (and Rückert's scheme) consists of four moves between the signer and user. That is, an extra move is required to confirm the validity of blind signatures. More concretely, this additional move allows the user to request a protocol restart from the signer in case the security check carried out at the second user stage fails. Built upon BLAZE, our second construction BLAZE$^+$ reduces the sizes of keys and signatures as well as the communication complexity of the signing protocol by utilizing the tree of commitments technique presented in Chapter 4. At the first user stage many masking terms are generated at once in order to blind the signature part that will be output from the second user stage without the need to request a protocol restart from the signer. We obtain a three-move scheme similar to the basic structure of the Fiat-Shamir approach.

Using OR-proofs introduced by Cramer *et al.* [CDS94], our third scheme BlindOR makes a significant progress towards a three-move lattice-based construction of blind signatures that is simultaneously practical and provably secure. An OR-proof allows to prove the possession of a witness for one of two (or more) statements, without revealing which one. The design of BlindOR also involves the partitioning and permutation technique as well as the trees of commitments technique in order to reduce/remove the number of restarts induced by applying the rejection sampling procedure. More importantly, using OR-proofs allows to sidestep a subtle security argument, which is pointed out by Hauck *et al.* [HKLN20] and is missing in the security proof of all prior lattice-based blind signature schemes starting from the first one [Rüc10]. We refer to Section 5.5 for more details.

The design of BlindOR is inspired by Abe and Okamoto [AO00], who used OR-proofs to build a partially blind signature scheme [AF96] with security based on the hardness of the discrete logarithm problem. We remark that it is not worthwhile to convert their construction to the lattice setting, as this would result in an inefficient scheme, since we must consider the additional requirements on the parameters that are given in [HKLN20] to obtain a gap-free security reduction. Hauck *et al.* [HKLN20] extended the modular framework for blind signatures from linear functions given in [HKL19] to the lattice setting. However, this framework is mostly of theoretical interest as it entails parameters of huge size, which render the scheme impractical. A further related work is due to Agrawal *et al.* [ASY21], who made a step towards practical two-round lattice-based blind signatures. They improved the construction of Garg *et al.* [GRS$^+$11] which is based on general complexity assumptions. As pointed out by the authors, there are some challenges left before this approach becomes practical. For instance, the scheme requires the homomorphic evaluation of a specific signing algorithm that relies on the random oracle model [BR93]. In practice, this must be instantiated with a cryptographic hash function that can be evaluated homomorphically. Finding such a function is still an open problem, which has not been studied sufficiently. Another challenge that has to be tackled is how to handle the compatibility issue induced by the various formats of the quantities involved in the homomorphic signing.

Chapter 6: Conclusion

This chapter concludes this thesis and discusses possible research directions for future work.

2 Background

In this chapter we recall the mathematical and cryptographic background required throughout this thesis. In Sections 2.1 and 2.2 we define all required notations, basic tools, and relevant rings. Afterwards, we give in Section 2.2 the definition of lattices and the relevant lattice problems. Then, we provide in Section 2.4 a formal definition of digital signature schemes including their security. Finally, we recall the (quantum) random oracle model in Section 2.5, and then the forking lemma in Section 2.6.

2.1 Basic Notation

In this section we fix the notation and define the basic tools that are required throughout this thesis.

We denote by $\mathbb{N}, \mathbb{Z}$, and $\mathbb{R}$ the sets of natural numbers, integers, and real numbers, respectively. If $k \in \mathbb{N}_{>0}$, we let $[k]$ denote the set $\{1, \ldots, k\}$. All logarithms appear in this thesis are to base two.

We denote the security parameter by $\lambda \in \mathbb{N}_{>0}$, and abbreviate probabilistic polynomial-time by PPT and deterministic polynomial-time by DPT. For a probabilistic algorithm A, we write $y \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{O}}(x)$ to denote that A returns y when run on input x and with access to an oracle O. We also write $y \in \mathsf{A}^{\mathsf{O}}(x)$ if y is a possible output of $\mathsf{A}^{\mathsf{O}}(x)$. To make the randomness $r \in \mathcal{R}$ on which A is run explicit, we use the notation $y \leftarrow \mathsf{A}^{\mathsf{O}}(x; r)$, where $\mathcal{R}$ is some randomness space.

We write $x \leftarrow\!\!\$\ D$ to denote that x is sampled randomly according to a distribution D. If S is a finite set, we also write $x \leftarrow\!\!\$\ S$ if x is chosen randomly from the uniform distribution over S. The *statistical distance* between two distributions X and Y over a countable set S is defined by $\Delta(X, Y) := \frac{1}{2} \sum_{s \in S} |\Pr[X = s] - \Pr[Y = s]|$. For $\varepsilon > 0$, we say that X and Y are *ε-statistically close* if $\Delta(X, Y) \leq \varepsilon$.

We denote by Expand a DPT algorithm that, on input $\mathbf{x} \in \{0,1\}^*$, expands $\mathbf{x}$ to an element $\mathbf{y}$ belonging to any desired set, *e.g.*, $\mathbf{y} \in \{0,1\}^\lambda \leftarrow \mathsf{Expand}(\mathbf{x})$. This algorithm is used to save space as it is deterministic. In practice, Expand can be realized by using any extendable output function (XOF) such as SHAKE.

2.2 Rings

In this section we define the required rings and recall their relevant properties.

Let $q \in \mathbb{Z}_{>0}$. We write $\mathbb{Z}_q$ to denote the ring of integers modulo q with representatives in $\left[-\frac{q}{2}, \frac{q}{2}\right) \cap \mathbb{Z}$. Let n be a fixed power of two and consider the polynomial ring $\mathbb{Z}[X]$ in a variable X. We define the rings $R := \mathbb{Z}[X]/\langle X^n + 1\rangle$ and $R_q := \mathbb{Z}_q[X]/\langle X^n + 1\rangle$, *i.e.*, elements in R and R_q are residue classes of polynomials of degree less than n with coefficients in $\mathbb{Z}$ and $\mathbb{Z}_q$, respectively. Elements in R and R_q are denoted by regular font letters. Column vectors and matrices with coefficients in R or R_q are denoted by bold lower-case letters and bold upper-case letters, respectively. The identity matrix of dimension k is denoted by $\mathbf{I}_k$. Note that $\mathbb{Z} \subset R$ and $\mathbb{Z}_q \subset R_q$.

Let $a = \sum_{i=0}^{n-1} a_i X^i \in R$. The *rotation matrix* of a is defined by

$$\mathsf{Rot}(a) := \big(\mathbf{a}, \mathsf{rot}(a), \mathsf{rot}^2(a), \ldots, \mathsf{rot}^{n-1}(a)\big) \in \mathbb{Z}^{n\times n},$$

where $\mathbf{a} = (a_0, \ldots, a_{n-1})^\top$, $\mathsf{rot}(a) := (-a_{n-1}, a_0, \ldots, a_{n-2})^\top$, and $\mathsf{rot}^k(a) := \mathsf{rot}\big(\mathsf{rot}^{k-1}(a)\big)$ for all $k \in \{2, \ldots, n-1\}$. If $a = \sum_{i=0}^{n-1} a_i X^i$, $b = \sum_{i=0}^{n-1} b_i X^i \in R_q$, then the polynomial multiplication $ab \in R_q$ corresponds to the matrix-vector product $\mathsf{Rot}(a) \cdot \mathbf{b}$ in $\mathbb{Z}_q$, where $\mathbf{b}$ is the vector representation of the polynomial b, *i.e.*, $\mathbf{b} = (b_0, \ldots, b_{n-1})^\top$.

Let $p \in \mathbb{N}_{>0} \cup \{\infty\}$. The ℓ_p *norm* of any $a \in R$ is defined by the function

$$\|\cdot\|_p \colon R \to \mathbb{R},\ a = \sum_{i=0}^{n-1} a_i X^i \mapsto \begin{cases} \left(\sum_{i=0}^{n-1} |a_i|^p\right)^{1/p} & \text{if } p < \infty \\ \max\{|a_0|, \ldots, |a_{n-1}|\} & \text{if } p = \infty \end{cases}$$

Similarly, the ℓ_p norm of any $\mathbf{b} \in R^k$ is defined by

$$\|\cdot\|_p \colon R^k \to \mathbb{R},\ \mathbf{b} = (b_1, \ldots, b_k)^\top \mapsto \begin{cases} \left(\sum_{i=1}^{k} \|b_i\|_p^p\right)^{1/p} & \text{if } p < \infty \\ \max\left\{\|b_1\|_p, \ldots, \|b_k\|_p\right\} & \text{if } p = \infty \end{cases}$$

In this thesis, we will mostly use the Euclidean norm $\|\cdot\|_2$ and the infinity norm $\|\cdot\|_\infty$. To simplify notations, we abbreviate the Euclidean norm by $\|\cdot\|$.

For any $x \in \mathbb{Z}_{>0}$, we define by S_x the subset of R_q that consists of all polynomials whose infinity norm is bounded by x, *i.e.*, $S_x := \{f \in R_q \mid \|f\|_\infty \le x\}$. Furthermore, we let $\mathbb{T}_\kappa^n$ denote the subset of R_q containing all polynomials with coefficients in $\{-1, 0, 1\}$ and Hamming weight κ, *i.e.*, $\mathbb{T}_\kappa^n := \{f \in R_q \mid (\|f\|_\infty = 1) \wedge (\|f\|_1 = \kappa)\}$. We will use the following result due to Lyubashevsky and Seiler [LS18, Corollary 1.2], which shows that for a suitable choice of the modulus q, all polynomials in R_q with bounded norms are invertible in R_q.

Lemma 2.1. *Let $n \ge p > 1$ be powers of two and $q = 2p + 1 \pmod{4p}$ be a prime. Then, the polynomial $X^n + 1$ factors as*

$$X^n + 1 \equiv \prod_{j=1}^{p} \big(X^{n/p} - r_j\big) \pmod{q}$$

for distinct $r_j \in \mathbb{Z}_q^*$, where $X^{n/p} - r_j$ are irreducible in $\mathbb{Z}_q[X]$. Furthermore, any $y \in R_q$ that satisfies either $0 < \|y\|_\infty < q^{1/p}/\sqrt{p}$ or $0 < \|y\| < q^{1/p}$ has an inverse in R_q.

2.3 Lattices and Cryptography

In this section we first define lattices and their basic properties (Section 2.3.1). After that, we recall the discrete Gaussian distribution and the required results related to it (Section 2.3.2). Finally, we review the definition of the relevant hardness assumptions based on lattices, and describe the hardness estimation of these assumptions (Section 2.3.3).

2.3.1 Basic Definitions

Definition 2.2. Let $m \in \mathbb{N}_{>0}$. An *m-dimensional lattice* is any subset $\mathcal{L}$ of $\mathbb{R}^m$ that satisfies the following properties:

- $\mathcal{L}$ is *discrete*: Every vector $\mathbf{v} \in \mathcal{L}$ has a neighborhood in $\mathbb{R}^m$, in which $\mathbf{v}$ is the only lattice point.
- $\mathcal{L}$ is an *additive subgroup*: $\mathbf{0} \in \mathcal{L}$, and for all $\mathbf{v}, \mathbf{w} \in \mathcal{L}$ we have $-\mathbf{v} \in \mathcal{L}$ and $\mathbf{v}+\mathbf{w} \in \mathcal{L}$.

Any lattice $\mathcal{L}$ can be defined as the integer linear combinations of some k linearly independent vectors from $\mathbb{R}^m$, where $k \leq m$. The integer k is called the *rank* of $\mathcal{L}$.

Definition 2.3. Let $k, m \in \mathbb{N}_{>0}$ with $k \leq m$, and let $\mathbf{B} \in \mathbb{R}^{m \times k}$ be a matrix of rank k. The *m-dimensional lattice of rank k that is generated by the matrix $\mathbf{B}$* is defined by $\mathcal{L}(\mathbf{B}) := \left\{ \mathbf{Bx} \,\middle|\, \mathbf{x} \in \mathbb{Z}^k \right\} \subset \mathbb{R}^m$.

Any matrix $\mathbf{B} \in \mathbb{R}^{m \times k}$ that generates a lattice in $\mathbb{R}^m$ is called a *lattice basis*. Since $\mathbf{B}$ is not unique for $m \geq 1$, we will leave the matrix $\mathbf{B}$ implied and write $\mathcal{L}$ instead of $\mathcal{L}(\mathbf{B})$. A lattice $\mathcal{L} \subset \mathbb{R}^m$ of rank k is called *full-rank* if $k = m$. An *integer lattice* is a lattice whose vectors have integer coefficients.

Definition 2.4. Let $\mathcal{L} \subset \mathbb{R}^m$ be a lattice of rank k, and $\mathbf{B} \in \mathbb{R}^{m \times k}$ is a lattice basis of $\mathcal{L}$. The *determinant* of $\mathcal{L}$ is defined by $\det(\mathcal{L}) := \sqrt{\det(\mathbf{B}^\top \cdot \mathbf{B})}$.

A *q-ary lattice* is an m-dimensional integer lattice $\mathcal{L}$ satisfying $q\mathbb{Z}^m \subseteq \mathcal{L} \subseteq \mathbb{Z}^m$ for some $q \in \mathbb{Z}_{>0}$. In the following we define the most common q-ary lattices used in lattice-based cryptography. Given $k, m, q \in \mathbb{Z}_{>0}$, where $m \geq k$, and $\mathbf{A} \in \mathbb{Z}_q^{k \times m}$, we define the two full-rank lattices

$$\mathcal{L}_{k,m,q}(\mathbf{A}) := \left\{ \mathbf{x} \in \mathbb{Z}^m \,\middle|\, \mathbf{x} = \mathbf{A}^\top \mathbf{s} \pmod q \text{ for some } \mathbf{s} \in \mathbb{Z}^k \right\},$$
$$\mathcal{L}^\perp_{k,m,q}(\mathbf{A}) := \{ \mathbf{x} \in \mathbb{Z}^m \mid \mathbf{0} = \mathbf{Ax} \pmod q \}.$$

Note that the lattice $\mathcal{L}_{k,m,q}(\mathbf{A})$ is generated by the rows of the matrix $\mathbf{A}$ modulo q, while the lattice $\mathcal{L}^\perp_{k,m,q}(\mathbf{A})$ contains all vectors that are orthogonal modulo q to the rows of $\mathbf{A}$. A random instance of both lattices is obtained by choosing the matrix $\mathbf{A}$ according to the uniform distribution over $\mathbb{Z}_q^{k \times m}$. Furthermore, if q is prime and $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{k \times m}$, then we have

$$\Pr\left[\det(\mathcal{L}_{k,m,q}(\mathbf{A})) = q^{m-k}\right] = \Pr\left[\det\left(\mathcal{L}^\perp_{k,m,q}(\mathbf{A})\right) = q^k\right] \geq 1 - \frac{1}{q^{m-k}} \quad \text{(see, \textit{e.g.} [Mic11])}.$$

2.3.2 Gaussian Measures

Definition 2.5. Let $\mathcal{L} \subset \mathbb{R}^m$ be a lattice, $\sigma \in \mathbb{R}_{>0}$, and $\mathbf{c} \in \mathbb{R}^m$. The *discrete Gaussian distribution over $\mathcal{L}$ with standard deviation σ and center $\mathbf{c}$* is the probability distribution $D_{\mathcal{L},\sigma,\mathbf{c}}$, which assigns to every vector $\mathbf{x} \in \mathcal{L}$ the probability of occurrence given by $D_{\mathcal{L},\sigma,\mathbf{c}}(\mathbf{x}) := \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathcal{L})$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right)$ and $\rho_{\sigma,\mathbf{c}}(\mathcal{L}) := \sum_{\mathbf{x}\in\mathcal{L}} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. We will omit the subscript $\mathbf{c}$ when $\mathbf{c} = \mathbf{0}$.

To make the randomness r that is used when sampling $\mathbf{x}$ from $D_{\mathcal{L},\sigma,\mathbf{c}}$ explicit, we write $\mathbf{x} \leftarrow D_{\mathcal{L},\sigma,\mathbf{c}}(r)$. In other words, specifying the randomness r makes a discrete Gaussian sampler a DPT algorithm.

The next two lemmas describe the basic properties of the discrete Gaussian distribution that are required in this thesis. The first one, due to Lyubashevsky [Lyu12, Lemma 4.4], gives a tail bound on Gaussian distributed random variables. The second lemma, given by Boneh and Freeman [BF11, Theorem 9], shows that the sum of Gaussian distributed random variables is also Gaussian distributed.

Lemma 2.6. *Let $\sigma, t, \eta \in \mathbb{R}_{>0}$ and $m \in \mathbb{N}_{>0}$. Then we have:*

1. $\Pr_{x\leftarrow\$D_{\mathbb{Z},\sigma}}[|x| > t\sigma] \leq 2\exp(-t^2/2)$, *and* $\Pr_{\mathbf{x}\leftarrow\$D_{\mathbb{Z}^m,\sigma}}[\|\mathbf{x}\|_\infty > t\sigma] \leq 2m\exp(-t^2/2)$.

2. $\Pr_{\mathbf{x}\leftarrow\$D_{\mathbb{Z}^m,\sigma}}[\|\mathbf{x}\| > \eta\sigma\sqrt{m}] \leq \eta^m \exp\left(\frac{m}{2}(1-\eta^2)\right)$.

Lemma 2.7. *Let $\mathcal{L} \subseteq \mathbb{Z}^m$ be a lattice and $\sigma \in \mathbb{R}_{>0}$. For every $i \in [n]$, let $\mathbf{t}_i \in \mathbb{Z}^m$, and let X_i be mutually independent random variables, where each X_i is sampled from $D_{\mathcal{L}+\mathbf{t}_i,\sigma}$. Let $\mathbf{c} = (c_1, \ldots, c_n) \in \mathbb{Z}^n$, $g = \gcd(c_1, \ldots, c_n)$, and $\mathbf{t} = \sum_{i=1}^n c_i\mathbf{t}_i$. Suppose that $\sigma > \|\mathbf{c}\| \cdot \eta(\mathcal{L})$, where $\eta(\mathcal{L})$ is a lattice parameter called the smoothing parameter [MR04]. Then, $\sum_{i=1}^n c_i X_i$ is ε-statistically close to $D_{g\mathcal{L}+\mathbf{t},\|\mathbf{c}\|\sigma}$, where $\varepsilon \approx 0$.*

Gentry *et al.* [GPV08, Lemma 5.3] showed that the smoothing parameter of the lattice $\mathcal{L}^\perp_{k,m,q}(\mathbf{A})$ can be bounded by $\omega(\sqrt{\ln m})$, *i.e.*, $\eta\left(\mathcal{L}^\perp_{k,m,q}(\mathbf{A})\right) \leq \omega(\sqrt{\ln m})$.

The next lemma [Lyu12, Theorem 4.6] is a version of the rejection sampling lemma that is specified for the discrete Gaussian distribution.

Lemma 2.8. *Let $T \in \mathbb{R}_{>0}$, and define the set $V := \{\mathbf{v} \in \mathbb{Z}^m \mid \|\mathbf{v}\| \leq T\}$. Let $\sigma = \alpha T$ for some $\alpha \in \mathbb{R}_{>0}$, and $h\colon V \to \mathbb{R}$ be a probability distribution. Then, there exists a constant $M \in \mathbb{R}_{>0}$ such that $\exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right) \leq M$, and the following two algorithms are within statistical distance of at most $2^{-100}/M$:*

1. $\mathbf{v} \leftarrow\$ h$, $\mathbf{z} \leftarrow\$ D_{\mathbb{Z}^m,\sigma,\mathbf{v}}$, *output* $(\mathbf{z}, \mathbf{v})$ *with probability* $\frac{D_{\mathbb{Z}^m,\sigma}(\mathbf{z})}{M\cdot D_{\mathbb{Z}^m,\sigma,\mathbf{v}}(\mathbf{z})}$, *and* $\perp$ *otherwise.*

2. $\mathbf{v} \leftarrow\$ h$, $\mathbf{z} \leftarrow\$ D_{\mathbb{Z}^m,\sigma}$, *output* $(\mathbf{z}, \mathbf{v})$ *with probability* $1/M$, *and* $\perp$ *otherwise.*

Moreover, the probability that the first algorithm returns a value different from $\perp$ is at least $\frac{1-2^{-100}}{M}$.

$\mathsf{Rej}(pp, \mathbf{z}, \mathbf{v})$:

1: **parse** $pp = (m, \sigma, M)$
2: $\eta \leftarrow\!\!\$\ [0, 1)$
3: **if** $\eta > \frac{1}{M} \cdot \exp\left(\frac{-2\langle \mathbf{z}, \mathbf{v}\rangle + \|\mathbf{v}\|^2}{2\sigma^2}\right)$ **then**
4: **return** 0
5: **return** 1

Figure 2.1: An implementation of the rejection sampling algorithm Rej.

We let Rej denote an algorithm that carries out rejection sampling on a vector $\mathbf{z}$, where $\mathbf{z} \leftarrow\!\!\$\ D_{\mathbb{Z}^m,\sigma,\mathbf{v}}$, $\mathbf{v}$ is an m-dimensional integer vector such that $\|\mathbf{v}\| \leq T$, and $\sigma = \alpha T$. That is, on input $(pp, \mathbf{z}, \mathbf{v})$, where pp is a set of parameters that includes m, σ, M, Rej returns 1 if $\mathbf{z}$ is accepted and 0 if rejected. By Lemma 2.8, the output 1 indicates that the distribution of $\mathbf{z}$ is within statistical distance of at most $2^{-100}/M$ from the Gaussian distribution $D_{\mathbb{Z}^m,\sigma}$, where $\exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right) \leq M$. The algorithm Rej returns 1 with probability $\approx 1/M$, and hence the expected number of restarts necessary to return 1 is M. A possible implementation of Rej can be found in [DLL+17a], which we recall in Figure 2.1.

2.3.3 Hardness Assumptions

We define the main hard problems underlying the signature schemes considered in this thesis, *i.e.*, the *Module Learning With Errors* (MLWE) problem, the *Module Short Integer Solution* (MSIS) problem, and the SelfTargetMSIS problem. For each problem, we assume that there exists an algorithm that, on input 1^λ, generates a set of public parameters pp.

Definition 2.9. Let $pp = (n, k_1, k_2, q, \chi, \mathbf{A})$, where $n, k_1, k_2, q \in \mathbb{Z}_{>0}$, χ is a probability distribution over R (typically $D_{\mathbb{Z}^n,\sigma}$ for some $\sigma > 0$, or the uniform distribution over a small subset of R), and $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$. We say that the module learning with errors (MLWE) problem is (t, ε)-*hard w.r.t.* pp if, for every algorithm D^* running in time at most t, we have

$$\mathbf{Adv}^{\mathsf{MLWE}}_{\mathsf{D}^*}(pp) := 2 \cdot \left|\Pr\left[\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}(pp) = 1\right] - \frac{1}{2}\right| \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}$ is depicted in Figure 2.2.

Definition 2.10. Let $pp = (n, k_1, k_2, q, \beta)$, where $n, k_1, k_2, q \in \mathbb{Z}_{>0}$ and $\beta \in \mathbb{R}_{>0}$. We say that the Hermite normal form of the module short integer solution (MSIS) problem is (t, ε)-*hard w.r.t.* pp if, for every algorithm A^* running in time at most t, we have

$$\mathbf{Adv}^{\mathsf{MSIS}}_{\mathsf{A}^*}(pp) := \Pr\left[\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}$ is depicted in Figure 2.2.

$\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}(pp)$:

1: **parse** $pp = (n, k_1, k_2, q, \chi, \mathbf{A})$
2: $b \leftarrow\!\!\$\ \{0,1\}$
3: **if** $b = 0$ **then**
4: $\quad \mathbf{b} \leftarrow\!\!\$\ R_q^{k_1}$
5: **else**
6: $\quad \mathbf{s} \leftarrow\!\!\$\ \chi^{k_1+k_2}$
7: $\quad \mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
8: $b^* \leftarrow\!\!\$\ \mathsf{D}^*(pp, \mathbf{b})$
9: **if** $b = b^*$ **then**
10: $\quad$ **return** 1
11: **return** 0

$\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}(pp)$:

11: **parse** $pp = (n, k_1, k_2, q, \beta)$
12: $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$
13: $\mathbf{x} \leftarrow\!\!\$\ \mathsf{A}^*(pp, \mathbf{A})$
14: **if** $\left(\mathbf{x} \in R^{k_1+k_2} \backslash \{\mathbf{0}\}\right) \wedge \left(\|\mathbf{x}\|_p \leq \beta\right) \wedge$
$\quad (\mathbf{0} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} \pmod q)$ **then**
15: $\quad$ **return** 1
16: **return** 0

Figure 2.2: Definition of the experiments $\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}$ and $\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}$.

The MLWE problem, defined by Langlois and Stehlé [LS15], generalizes the Learning With Errors (LWE) problem [Reg05] and its ring variant RLWE [LPR10]. More precisely, by setting $k_1 = 1$ in Definition 2.9 we obtain the ring variant RLWE, while setting $k_1 > 1$ and $R_q = \mathbb{Z}_q$ yields a definition of the LWE problem. The same applies for MSIS [LS15] (Definition 2.10) and its special variants SIS [Ajt96] and RSIS [Mic02].

We remark that MLWE has two versions called the *Decision* and the *Search* version. In this thesis, we will use the decision version, which is given in Definition 2.9. Informally, the search version of MLWE asks to find the vector $\mathbf{s}$, given the pair $(pp, \mathbf{b})$. Being able to find $\mathbf{s}$ obviously implies the ability to win the experiment $\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}$ given in Figure 2.2, *i.e.*, there is a reduction from the hardness of the decision version of MLWE to the hardness of the search version. The opposite direction of this reduction has been established, *e.g.*, by Regev [Reg05], so that the hardness of both versions are equivalent.

Langlois and Stehlé [LS15] showed that both MLWE and MSIS are as hard as the module variant of the *Shortest Independent Vectors Problem* (SIVP) in the worst-case. This lattice problem is defined as follows: Given a basis $\mathbf{B} \in \mathbb{R}^{m \times k}$ of a lattice $\mathcal{L}$ and an approximation factor $\gamma \in \mathbb{R}_{>0}$, the SIVP problem asks to find m linearly independent lattice vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$ such that $\max\{\|\mathbf{x}_1\|_p, \ldots, \|\mathbf{x}_m\|_p\}$ is within a factor γ of its optimal value. For any $\gamma \in O(1)$, SIVP is NP-hard [BS99], and it is conjectured that there is no polynomial-time (quantum) algorithm that achieves an approximation factor γ bounded by any polynomial in m (see, *e.g.* [LS15]).

In the following we give a further relation of MLWE and MSIS to lattices. We show that both lattices $\mathcal{L}_{k,m,q}(\mathbf{A})$ and $\mathcal{L}^{\perp}_{k,m,q}(\mathbf{A})$ appear implicitly in MLWE and MSIS, respectively. More concretely, both MLWE and MSIS are connected to the following well-known (average-case) lattice problems in the q-ary lattices $\mathcal{L}_{k,m,q}(\mathbf{A})$ and $\mathcal{L}^{\perp}_{k,m,q}(\mathbf{A})$, respectively:

- MLWE and $\mathcal{L}_{k,m,q}(\mathbf{A})$: Solving MLWE w.r.t. $pp = (n, k_1, k_2, q, \chi, \mathbf{A})$ amounts to solving the module variant of the *Bounded Distance Decoding* (BDD) problem in the

$\mathbf{Exp}^{\mathsf{SelfTargetMSIS}}_{\mathsf{A}^*}(pp)$:

1: **parse** $pp = (n, k_1, k_2, q, \kappa, \beta, \mathsf{H})$
2: $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$
3: $(\mathbf{x} = [\mathbf{z} \mid c]^\top, m) \leftarrow\!\!\$\ \mathsf{A}^{*\mathsf{H}}(pp, \mathbf{A})$ // $(\mathbf{x}, m) \in R^{k_1+k_2+1} \times \{0,1\}^*$
4: **if** $(\|\mathbf{x}\|_p \leq \beta) \wedge (c = \mathsf{H}([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} \pmod q, m))$ **then**
5: **return** 1
6: **return** 0

Figure 2.3: Definition of the experiment $\mathbf{Exp}^{\mathsf{SelfTargetMSIS}}_{\mathsf{A}^*}$.

q-ary lattice $\mathcal{L}_{k_1+k_2,k_1,q}\left([\mathbf{I}_{k_1} \mid \mathbf{A}]^\top\right)$. Given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{k \times m}$ and a vector $\mathbf{b} \in \mathbb{Z}_q^m$, the BDD problem asks to distinguish between the case that $\mathbf{b}$ is chosen uniformly from $\mathbb{Z}_q^m$ and the case in which $\mathbf{b}$ is the result of perturbing each coefficient of a random lattice vector in $\mathcal{L}_{k,m,q}(\mathbf{A})$ via the distribution χ.

- MSIS and $\mathcal{L}^\perp_{k,m,q}(\mathbf{A})$: Solving MSIS w.r.t. $pp = (n, k_1, k_2, q, \beta)$ amounts to solving the module variant of the *Shortest Vector Problem* (SVP) in the q-ary lattice $\mathcal{L}^\perp_{k_1,k_1+k_2,q}([\mathbf{I}_{k_1} \mid \mathbf{A}])$. Given a basis $\mathbf{B} \in \mathbb{R}^{m \times k}$ of a lattice $\mathcal{L}$ and an approximation factor $\gamma \in \mathbb{R}_{>0}$, the SVP problem asks to find a non-zero vector $\mathbf{x} \in \mathcal{L}$ such that $\|\mathbf{x}\|_p$ is within a factor γ of the length of a shortest lattice vector, *i.e.*, within a factor γ of the minimum distance between any two distinct lattice vectors.

Definition 2.11. Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ be a cryptographic hash function. Furthermore, let $pp = (n, k_1, k_2, q, \kappa, \beta, \mathsf{H})$, where $n, k_1, k_2, q, \kappa \in \mathbb{Z}_{>0}$ and $\beta \in \mathbb{R}_{>0}$. We say that the SelfTargetMSIS problem is (t, ε)-*hard w.r.t.* pp if, for every algorithm A^* running in time at most t, we have

$$\mathbf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathsf{A}^*}(pp) := \Pr\left[\mathbf{Exp}^{\mathsf{SelfTargetMSIS}}_{\mathsf{A}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathsf{SelfTargetMSIS}}_{\mathsf{A}^*}$ is depicted in Figure 2.3.

The SelfTargetMSIS problem, defined by Kiltz *et al.* [KLS18], can be seen as a combination of the function H and the MSIS problem. Indeed, the following lemma shows that there is a reduction from the hardness of MSIS to the hardness of SelfTargetMSIS in the random oracle model (ROM) (see Section 2.5). This reduction uses the forking lemma (see Section 2.6) and it is implicit in the security proof of many lattice-based signature schemes.

Lemma 2.12. *Let $n, k_1, k_2, q, \kappa \in \mathbb{Z}_{>0}$ and $\beta \in \mathbb{R}_{>0}$. Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ be a cryptographic hash function modeled as a random oracle, and q_H be the maximum number of hash queries that are allowed to be made to this random oracle. The* SelfTargetMSIS *problem is (t, ε)-hard w.r.t. $pp = (n, k_1, k_2, q, \kappa, \beta, \mathsf{H})$ in the ROM if* MSIS *is (t', ε')-hard w.r.t. pp', where*

$$pp' = (n, k_1, k_2, q, 2\beta),\ t' \approx 2t,\ \text{and } \varepsilon' \approx \left(\varepsilon - \frac{1}{|\mathbb{T}_\kappa^n|}\right) \cdot \left(\frac{\varepsilon - \frac{1}{|\mathbb{T}_\kappa^n|}}{q_\mathsf{H}} - \frac{1}{|\mathbb{T}_\kappa^n|}\right).$$

Moreover, Kiltz *et al.* [KLS18] gave reasonable arguments to assume that as long as H is second-preimage resistant against quantum attackers and the algebraic structures of H and the matrix $\mathbf{A}$ are completely independent, the hardness of $\mathsf{SelfTargetMSIS}$ and MSIS is equivalent, *i.e.*, $\varepsilon' \approx \varepsilon$, even if the adversary is given quantum access to H.

We remark that in the definitions of MSIS and $\mathsf{SelfTargetMSIS}$, the solution is measured in the ℓ_p norm. If necessary, we will write MSIS^p and $\mathsf{SelfTargetMSIS}^p$ to explicitly specify the value of p, where $p \in \{2, \infty\}$. Otherwise, we write MSIS and $\mathsf{SelfTargetMSIS}$ to indicate that the statement applies to any norm. Similarly, we write MLWE^2 if the probability distribution χ given in Definition 2.9 is set to $\chi = D_{\mathbb{Z}^n,\sigma}$ for some $\sigma > 0$, and MLWE^∞ if $\chi = S_x$ for some $x \in \mathbb{Z}_{>0}$. Furthermore, we write $pp = (n, k_1, k_2, q, \sigma, \mathbf{A})$ if $\chi = D_{\mathbb{Z}^n,\sigma}$, and $pp = (n, k_1, k_2, q, x, \mathbf{A})$ if $\chi = S_x$.

Next, we explain the methodology that we follow in this thesis to estimate the hardness of MLWE, MSIS, and $\mathsf{SelfTargetMSIS}$ with respect to a given set of public parameters. First, we remark that all known algorithms solving MLWE, MSIS, and $\mathsf{SelfTargetMSIS}$ do not exploit their algebraic structure.

Estimating the hardness of MLWE w.r.t. $pp = (n, k_1, k_2, q, \chi, \mathbf{A})$ is carried out by using the well-known and widely used LWE-*Estimator* presented by Albrecht *et al.* [APS15]. This estimator is a Sage module, which determines an upper bound on the number of operations required to be carried out by the currently fastest classical and quantum LWE solvers.

Given $pp = (n, k_1, k_2, q, \beta)$ and $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$, the hardness of solving MSIS w.r.t. pp is equivalent to finding a non-trivial vector, whose ℓ_p norm is bounded by β, in the q-ary lattice $\mathcal{L}^{\perp}_{k,m,q}([\mathbf{I}_k \mid \mathbf{A}'])$, where $k = k_1 n$, $m = (k_1 + k_2)n$, and $\mathbf{A}'$ is the matrix obtained by computing the rotation matrix of each entry of $\mathbf{A}$, *i.e.*,

$$\mathbf{A}' = \begin{bmatrix} \mathsf{Rot}(a_{1,1}) & \dots & \mathsf{Rot}(a_{1,k_2}) \\ \vdots & \ddots & \vdots \\ \mathsf{Rot}(a_{k_1,1}) & \dots & \mathsf{Rot}(a_{k_1,k_2}) \end{bmatrix} \in \mathbb{Z}_q^{k_1 n \times k_2 n}, \text{ where } \mathbf{A} = \begin{bmatrix} a_{1,1} & \dots & a_{1,k_2} \\ \vdots & \ddots & \vdots \\ a_{k_1,1} & \dots & a_{k_1,k_2} \end{bmatrix}.$$

The best known (classical and quantum) algorithm for finding short non-trivial vectors is due to Schnorr and Euchner [SE94]. It is called the Block-Korkine-Zolotarev algorithm (BKZ), and was improved in practice by Chen and Nguyen [CN11]. As a subroutine, BKZ uses an SVP solver in lattices of dimension b, where b is called the *block size*. The best known classical algorithm for SVP with no memory restrictions is due to Becker *et al.* [BDGL16], and it takes time $\approx 2^{0.292\,b}$, while the best known quantum SVP solver is due to Laarhoven [Laa16, Section 14.2.10], and it takes time $\approx 2^{0.265\,b}$. The time required by BKZ to run with block size b on an m-dimensional lattice $\mathcal{L}$ is given by (see, *e.g.* [BDGL16])

$$8m\, 2^{\xi b + 16.4}, \tag{2.1}$$

where $\xi \in \{0.265, 0.292\}$. The output of BKZ is a vector of length $\delta^m \det(\mathcal{L})^{1/m}$, where δ is called the *Hermite delta* and it is given by (see, *e.g.* [CN11, Che13])

$$\delta = \left(b\,(\pi b)^{\frac{1}{b}} / (2\pi e) \right)^{\frac{1}{2(b-1)}}. \tag{2.2}$$

Therefore, in order to compute the time required by BKZ to solve MSIS w.r.t. pp, we first determine δ by setting $\beta = \delta^m \det(\mathcal{L})^{1/m}$, where $m = (k_1 + k_2)n$. After that, we

compute the minimum block size b required to achieve δ by using Equation (2.2). The resulted b is put in Equation (2.1) to obtain the time required by BKZ to solve MSIS w.r.t. $pp = (n, k_1, k_2, q, \beta)$.

Finally, the hardness of solving SelfTargetMSIS w.r.t. $pp = (n, k_1, k_2, q, \kappa, \beta, \mathsf{H})$ is estimated by following Lemma 2.12, *i.e.*, by estimating the hardness of solving MSIS w.r.t. pp' as described above, where $pp' = (n, k_1, k_2, q, 2\beta)$.

2.4 Digital Signature Schemes

In this section we recall the syntax and security of ordinary signature schemes.

Definition 2.13. An *(ordinary) signature scheme* is a tuple of polynomial-time algorithms

$$\mathsf{Sig} = (\mathsf{Sig.PGen}, \mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify}),$$

where:

$\mathsf{Sig.PGen}(1^\lambda)$ is a PPT parameter generation algorithm that, on input the security parameter λ, returns a set of public parameters pp, which implicitly contains λ in unary, *i.e.*, $pp \leftarrow\!\!\$\ \mathsf{Sig.PGen}(1^\lambda)$. We assume that pp identifies the message space $\mathcal{M}$ of Sig.

$\mathsf{Sig.KGen}(pp)$ is a PPT key generation algorithm that, on input a set of public parameters pp, returns a key pair (pk, sk), where pk is a public (verification) key and sk is a secret (signing) key, *i.e.*, $(pk, sk) \leftarrow\!\!\$\ \mathsf{Sig.KGen}(pp)$.

$\mathsf{Sig.Sign}(pp, sk, m)$ is a PPT signing algorithm that, on input a set of public parameters pp, a secret key sk, and a message m from the message space $\mathcal{M}$, returns a signature sig, *i.e.*, $sig \leftarrow\!\!\$\ \mathsf{Sig.Sign}(pp, sk, m)$. We write $sig = \perp$ to denote failure.

$\mathsf{Sig.Verify}(pp, pk, m, sig)$ is a DPT verification algorithm that, on input a set of public parameters pp, a public key pk, a message m, and a signature sig, returns 1 if sig is valid and 0 otherwise, *i.e.*, $b \leftarrow \mathsf{Sig.Verify}(pp, pk, m, sig)$, where $b \in \{0, 1\}$.

Signature schemes are required to satisfy the correctness property given in the following definition:

Definition 2.14. Let Sig be a signature scheme, and $pp \in \mathsf{Sig.PGen}(1^\lambda)$. We say that Sig is $\mathrm{corr}_{\mathsf{Sig}}$-*correct* w.r.t. pp if, for every key pair $(pk, sk) \in \mathsf{Sig.KGen}(pp)$ and every message $m \in \mathcal{M}$, we have

$$\Pr_{sig \leftarrow\!\!\$ \mathsf{Sig.Sign}(pp, sk, m)}[\mathsf{Sig.Verify}(pp, pk, m, sig) \neq 1] \leq \mathrm{corr}_{\mathsf{Sig}}.$$

The standard security of signature schemes, dates back to Goldwasser *et al.* [GMR88], is captured by the security notion of *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA). In this notion, the adversary is given access to a signing oracle $\mathsf{O}_{\mathsf{Sig}}$, from which the adversary is allowed to obtain up to q_{Sign} signatures on messages of its own choice. The adversary must return one valid signature on a message not queried to $\mathsf{O}_{\mathsf{Sig}}$.

$\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}(pp)$:

1: $(pk, sk) \leftarrow\!\!\$\ \mathsf{Sig.KGen}(pp)$
2: $\mathrm{L}_m \leftarrow \varnothing$
3: $(m^*, sig^*) \leftarrow\!\!\$\ \mathsf{A}^{*\mathsf{O}_{\mathsf{Sig}}}(pp, pk)$
4: **if** $m^* \in \mathrm{L}_m$ **then**
5: **return** 0
6: **return** $\mathsf{Sig.Verify}(pp, pk, m^*, sig^*)$

$\mathsf{O}_{\mathsf{Sig}}(m)$:

11: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
12: $sig \leftarrow\!\!\$\ \mathsf{Sig.Sign}(pp, sk, m)$
13: **return** sig

Figure 2.4: Definition of the experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}$.

Definition 2.15. Let Sig be a signature scheme, and $pp \in \mathsf{Sig.PGen}(1^\lambda)$. We say that Sig is $(t, q_{\mathsf{Sign}}, \varepsilon)$-EUF-CMA w.r.t. pp if, for every adversary A^* running in time at most t and making at most q_{Sign} signing queries to the oracle $\mathsf{O}_{\mathsf{Sig}}$, we have

$$\mathbf{Adv}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}(pp) = \Pr\left[\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}$ is defined in Figure 2.4.

2.5 The (Quantum) Random Oracle Model

In this thesis we deal with (advanced) signature schemes that are secure in the classical random oracle model or in the quantum random oracle model. This section gives a brief description of both models.

The *random oracle model* (ROM) is a model of computation introduced by Bellare and Rogaway [BR93]. It allows to prove the security of cryptographic schemes that are otherwise hard or even impossible to prove in the *standard* (or *plain*) model. In the ROM, every party has access to an oracle O_{H} implementing a random function H, *i.e.*, a function that is randomly chosen from the set of all functions $\left\{\mathsf{H}\colon \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{H}}}\right\}$ for some $\ell_{\mathsf{H}} \in \mathbb{N}_{>0}$. Upon being queried on some input $\mathbf{x} \in \{0,1\}^*$, the oracle O_{H} answers with a random output $\mathbf{y} \in \{0,1\}^{\ell_{\mathsf{H}}}$. Every further invocation on input $\mathbf{x}$, even by other parties, results in the same $\mathbf{y}$. The oracle O_{H} can be queried up to q_{H} times. In the security proofs of cryptographic schemes, hash functions are often modeled as random oracles. This allows the reduction algorithm to (adaptively) program the input-output behavior of H outside of the view of the remaining algorithms.

Since hash functions are public, Boneh *et al.* [BDF+11] observed that diverse techniques to prove security in the ROM are generally not applicable in the post-quantum setting. In the real world, an adversary equipped with a quantum computer is able to implement the hash function and evaluate it in superposition. Therefore, Boneh *et al.* introduced the *quantum random oracle model* (QROM). In this model, parties with quantum computing power are allowed to query the oracle O_{H} in superposition.

We remark that the QROM is nowadays considered the de facto standard for post-quantum security proofs of cryptographic schemes relying on random oracles. This post-quantum security model considers the adversary to be quantum while the challenger -

representing the honest user in real-world applications - remains classical. Consequently, every oracle provided by the challenger can be accessed only classically, while oracles that can be accessed by the adversary directly can be accessed using quantum computing power, *i.e.*, in superposition. This describes a threat model where an adversary can use its quantum power to locally access the random oracle, while it observes signatures created by a user on a classical machine. This standard post-quantum security model is meaningful, since it indicates that the cryptographic scheme can still be used on classical computers. This is in contrast to the (fully) quantum setting, where the scheme itself is implemented on quantum computers as well. While this is a stronger security model, it is more of theoretical interest as it requires users to have access to quantum computers as well.

2.6 The Forking Lemma

In this section we recall the general forking lemma due to Bellare and Neven [BN06], which constitutes a crucial tool for proving the security of cryptographic schemes, in particular signature schemes, in the ROM.

Let $\mathcal{C}$ be some finite set and $\mathcal{R}$ be some randomness space. Let IGen be a PPT algorithm, and consider an algorithm A that, on input an instance $x \in \mathsf{IGen}$ and random values $h_1, \ldots, h_q \in \mathcal{C}$, returns a pair (idx, out), where $0 \leq idx \leq q$ and out is a side output related to h_{idx}. The index $idx = 0$ indicates that A has failed to compute a side output out related to any of the values $h_1, \ldots, h_q$. The general forking lemma gives a lower bound on the probability of the forking experiment in which A, if run twice on the same instance x and randomness $r \in \mathcal{R}$, but partially different values from $\mathcal{C}$, will return the same index idx and two side outputs out and out', which are related to the values h_{idx} and h'_{idx}, respectively. The experiment fails if both runs of A return two different indices, or if $h_{idx} = h'_{idx}$.

In this thesis we will need a minor version of the general forking lemma [BN06]. We consider an algorithm A that further returns a second index as part of the output, *i.e.*, A returns a tuple (idx_1, idx_2, out), where idx_1, out are as before, and $0 \leq idx_2 < \ell$ for $\ell \in \mathbb{N}_{>0}$. In this version, the forking experiment succeeds only if both runs of A return the same pair of indices (idx_1, idx_2) and $h_{idx} \neq h'_{idx}$. The proof of this version of the lemma is almost identical to the proof given in [BN06]. In the following we state our minor version of the general forking lemma and provide its proof for completeness. First, we recall two supporting lemmas given in [BN06].

Lemma 2.16. *Let X be a random variable with values from $\mathbb{R}$. Then, we have*

$$\mathrm{E}\big[X^2\big] \geq \mathrm{E}[X]^2,$$

where $\mathrm{E}[X]$ is the expectation of X.

Lemma 2.17. *Let $q \in \mathbb{N}_{>0}$ and $x_1, \ldots, x_q \in \mathbb{R}_{\geq 0}$. Then, we have*

$$\sum_{i=1}^{q} x_i^2 \geq \frac{1}{q} \cdot \left(\sum_{i=1}^{q} x_i \right)^2.$$

$\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$:

1: $x \leftarrow\!\!\$\ \mathsf{IGen}$
2: $h_1, \ldots, h_q \leftarrow\!\!\$\ \mathcal{C}$
3: $(idx_1, idx_2, out) \leftarrow\!\!\$\ \mathsf{A}(x, h_1, \ldots, h_q)$
4: **if** $1 \le idx_1 \le q$ **then**
5: **return** 1
6: **return** 0

$\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$:

11: $x \leftarrow\!\!\$\ \mathsf{IGen}$
12: $(b, out, out') \leftarrow\!\!\$\ \mathsf{Frk}_{\mathcal{C},\mathsf{A}}(x)$
13: **return** b

$\mathsf{Frk}_{\mathcal{C},\mathsf{A}}(x)$:

21: $r \leftarrow\!\!\$\ \mathcal{R}$
22: $h_1, \ldots, h_q \leftarrow\!\!\$\ \mathcal{C}$
23: $(idx_1, idx_2, out) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; r)$
24: **if** $idx_1 = 0$ **then**
25: **return** $(0, \bot, \bot)$
26: $h'_{idx_1}, \ldots, h'_q \leftarrow\!\!\$\ \mathcal{C}$
27: $(idx'_1, idx'_2, out') \leftarrow \mathsf{A}(x, h_1, \ldots, h_{idx_1-1}, h'_{idx_1}, \ldots, h'_q; r)$
28: **if** $(idx_1 = idx'_1) \wedge (idx_2 = idx'_2) \wedge (h_{idx_1} \neq h'_{idx_1})$ **then**
29: **return** $(1, out, out')$
30: **return** $(0, \bot, \bot)$

Figure 2.5: Definition of the experiments $\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$, $\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$ and forking algorithm $\mathsf{Frk}_{\mathcal{C},\mathsf{A}}$.

Lemma 2.18. *Let $q, \ell \in \mathbb{N}_{>0}$, $\mathcal{C}$ be a finite set of size $|\mathcal{C}| \ge 2$, and $\mathcal{R}$ be a randomness space. Let IGen be a PPT algorithm, and A be a PPT algorithm that, on input $x \in \mathsf{IGen}$ and $h_1, \ldots, h_q \in \mathcal{C}$, outputs a tuple (idx_1, idx_2, out), where $0 \le idx_1 \le q$ and $0 \le idx_2 < \ell$. Define the* accepting probability *and the* forking probability *of A by*

$$acc = \Pr\left[\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}} = 1\right] \quad \textit{and} \quad frk = \Pr\left[\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}} = 1\right],$$

where the experiments $\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$ and $\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$ are depicted in Figure 2.5. Then we have

$$frk \ge acc \cdot \left(\frac{acc}{q\ell} - \frac{1}{|\mathcal{C}|}\right).$$

Proof. For any instance $x \in \mathsf{IGen}$ we define

$$acc(x) := \Pr\left[\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}} = 1 \;\middle|\; x \in \mathsf{IGen}\right] \quad \text{and}$$
$$frk(x) := \Pr\left[\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}} = 1 \;\middle|\; x \in \mathsf{IGen}\right].$$

First, we show that

$$frk(x) \ge acc(x) \cdot \left(\frac{acc(x)}{q\ell} - \frac{1}{|\mathcal{C}|}\right).$$

Observe that

$$\begin{aligned} frk(x) &= \Pr\left[(idx_1 \ge 1) \wedge (idx_1 = idx'_1) \wedge (idx_2 = idx'_2) \wedge \left(h_{idx_1} \neq h'_{idx_1}\right)\right] \\ &\ge \Pr[(idx_1 \ge 1) \wedge (idx_1 = idx'_1) \wedge (idx_2 = idx'_2)] - \Pr\left[(idx_1 \ge 1) \wedge \left(h_{idx_1} = h'_{idx_1}\right)\right] \\ &= \Pr[(idx_1 \ge 1) \wedge (idx_1 = idx'_1) \wedge (idx_2 = idx'_2)] - \frac{\Pr[idx_1 \ge 1]}{|\mathcal{C}|} \end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \Pr[(idx_1 = i) \wedge (idx'_1 = i) \wedge (idx_2 = j) \wedge (idx'_2 = j)] - \frac{acc(x)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \sum_{\bar{r},\bar{h}_1,\ldots,\bar{h}_{i-1}} \Pr\left[\begin{array}{l|l}(idx_1 = i) \wedge (idx'_1 = i) \wedge & (r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \\ (idx_2 = j) \wedge (idx'_2 = j) & \cdots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\end{array}\right] \\
&\qquad \cdot \Pr\big[(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \cdots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\big] - \frac{acc(x)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \sum_{\bar{r},\bar{h}_1,\ldots,\bar{h}_{i-1}} \Pr\left[(idx_1 = i) \wedge (idx_2 = j) \;\middle|\; \begin{array}{l}(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \\ \cdots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\end{array}\right]^2 \\
&\qquad \cdot \Pr\big[(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \cdots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\big] - \frac{acc(x)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \mathop{\mathrm{E}}_{r,h_1,\ldots,h_{i-1}} \big[\Pr[(idx_1 = i) \wedge (idx_2 = j) \mid r \wedge h_1 \wedge \cdots \wedge h_{i-1}]^2\big] - \frac{acc(x)}{|\mathcal{C}|} \\
&\geq \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \mathop{\mathrm{E}}_{r,h_1,\ldots,h_{i-1}} [\Pr[(idx_1 = i) \wedge (idx_2 = j) \mid r \wedge h_1 \wedge \cdots \wedge h_{i-1}]]^2 - \frac{acc(x)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \Pr[(idx_1 = i) \wedge (idx_2 = j)]^2 - \frac{acc(x)}{|\mathcal{C}|} \\
&\geq \frac{1}{q\ell}\left(\sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \Pr[(idx_1 = i) \wedge (idx_2 = j)]\right)^2 - \frac{acc(x)}{|\mathcal{C}|} \\
&\geq \frac{acc(x)^2}{q\ell} - \frac{acc(x)}{|\mathcal{C}|}.
\end{aligned}
$$

Note that the critical (in)equalities given above are obtained by the conditional independence given $r, h_1, \ldots, h_{i-1}$ and by applying Lemma 2.16 once and Lemma 2.17 twice.

Finally, by taking the expectation over $x \in \mathsf{IGen}$ and applying Lemma 2.16 in addition to the fact that $\mathrm{E}[acc(x)] = acc$, we conclude the proof as follows:

$$
\begin{aligned}
frk &= \mathrm{E}[frk(x)] \\
&\geq \mathrm{E}\left[acc(x) \cdot \left(\frac{acc(x)}{q\ell} - \frac{1}{|\mathcal{C}|}\right)\right] \\
&= \frac{\mathrm{E}\big[acc(x)^2\big]}{q\ell} - \frac{\mathrm{E}[acc(x)]}{|\mathcal{C}|} \\
&\geq \frac{\mathrm{E}[acc(x)]^2}{q\ell} - \frac{\mathrm{E}[acc(x)]}{|\mathcal{C}|} \\
&= acc \cdot \left(\frac{acc}{q\ell} - \frac{1}{|\mathcal{C}|}\right).
\end{aligned}
$$

□

3

Signature Schemes with Re-Randomizable Keys

A signature scheme with re-randomizable keys is an advanced type of signature schemes that allows both the public and secret key to be re-randomized in a separate but consistent way. It was first introduced by Fleischhacker *et al.* [FKM+16] as the main building block for constructing efficient unlinkable sanitizable signatures [ACdMT05,BFLS10], which have various applications such as the anonymization of medical data and updating reliable routing information. As suggested in [FKM+16], signature schemes with re-randomizable keys can also be used as a simple solution to the stealth addresses problem [Fra15] in cryptocurrencies such as Bitcoin and Ethereum. A further application of this cryptographic primitive in cryptocurrencies was shown by Das *et al.* [DFL19]. More concretely, a generic construction of a deterministic wallet scheme from signature schemes with re-randomizable keys was introduced, and an instantiation based on a concrete scheme presented in [FKM+16] was analyzed. Deterministic wallets offer an elegant solution to protect secret keys, *i.e.*, users' funds, against theft. Therefore, it is crucial that they enjoy strong security guarantees, especially, when considering attacks by adversaries with quantum computing power. The same also applies to signature schemes with re-randomizable keys. The constructions introduced in [FKM+16] do not offer security under quantum computer attacks, since their security is based on the hardness of assumptions that are known to be efficiently solvable using quantum computers, e.g., the discrete logarithm problem. Hence, there is a lack of efficient and post-quantum secure signature schemes with re-randomizable keys that can be used within the above mentioned applications in the post-quantum era.

Contributions

In this chapter we present the first signature scheme with re-randomizable keys that provides security in the post-quantum setting. More concretely, we present a generic construction of signature schemes with re-randomizable keys that uses lattice-based Fiat-Shamir signature schemes as a building block. We prove the security of our construction in the QROM. Our design also considers security under related key attacks [MSM+16]. We pro-

pose concrete parameters for the scheme targeting 128 bits of post-quantum security. Furthermore, we show how our construction can be instantiated with the ordinary signature scheme qTESLA [ABB+20]. Since qTESLA is an optimized variant of the lattice-based generic construction of Fiat-Shamir signatures, we prove that the resulted instantiation is also secure in the QROM.

At a high level besides the standard algorithms of an ordinary signature scheme (parameter generation, key generation, signing, and verification), a signature scheme with re-randomizable keys, denoted by RSig, has two additional algorithms, namely RSig.RandSK and RSig.RandPK. These algorithms take as input the secret key sk, respectively public key pk and a randomness ρ, and return fresh (re-randomized) keys sk', respectively pk'. Fleischhacker *et al.* [FKM+16] showed that in Schnorr's signature scheme [Sch91], a secret key sk can be re-randomized by computing $sk' = sk + \rho$ such that the sum is distributed identically to sk, which is selected according to the uniform distribution over the underlying group. Inspired by this work, we show how the secret key of Fiat-Shamir signatures based on lattices can also be re-randomized additively by considering the typical distributions of sk in the lattice setting, *i.e.*, the discrete Gaussian distribution and the uniform distribution over some small subset of the underlying ring. The re-randomized public key pk' related to sk' is computed separately, *i.e.*, without access to any secret key. We argue why the Gaussian distribution is the most suitable distribution of sk in the context of lattice-based deterministic wallets. Basically, this is because the sum of two Gaussian distributed variables is also Gaussian distributed (*cf.* Lemma 2.7), but with a slightly different standard deviation. This is in contrast to the uniform distribution over some small integer set S, where further checks must be carried out in order to ensure that sk' follows the uniform distribution over a subset of S. In both cases we obtain a scheme, in which the distribution of pk' is identical to the distribution of pk, while this does not hold for sk' and sk. We formally define such relaxed notion and call it a *signature scheme with re-randomizable public keys*. We then show that this notion is sufficient for at least the case of deterministic wallets. Moreover, we describe alternative approaches for re-randomizing keys in the lattice setting. For instance, we describe a scheme that can use either Gaussian or uniformly distributed secret keys, and allows RSig.RandSK and RSig.RandPK to synchronize on some state. While this approach cannot be used in deterministic wallets, as we will show, it may be utilized, for example, in the construction of sanitizable signatures given in [FKM+16].

Organization

We formally define signature schemes with re-randomizable (public) keys in addition to their security properties in Section 3.1. We recall in Section 3.2 a generic construction of Fiat-Shamir signatures based on lattice assumptions. In Section 3.3 we describe our signature scheme with re-randomizable public keys including its security analysis in the QROM, and propose concrete parameters for the scheme. We show in Section 3.4 how our scheme can be used to build post-quantum deterministic wallets. In Section 3.5 we provide the alternative approaches for building lattice-based signature schemes with re-randomizable keys and show potential applications. Finally, we show in Section 3.6 how to instantiate our scheme with qTESLA while preserving its security in the QROM.

Publications

The contributions of this chapter are part of the publication [A1], which was presented at ACM CCS'20. This chapter extends these contributions to include concrete parameters of our scheme with a description of their selection.

3.1 Signature Schemes with Re-Randomizable Keys

In this section we first recall the definition and security of signature schemes with re-randomizable keys [FKM+16]. Then, we define a relaxation of this cryptographic notion.

Definition 3.1. A *signature scheme with perfectly re-randomizable keys* is a tuple of polynomial-time algorithms:

$$\mathsf{RSig} = (\mathsf{RSig.PGen}, \mathsf{RSig.KGen}, \mathsf{RSig.RandSK}, \mathsf{RSig.RandPK}, \mathsf{RSig.Sign}, \mathsf{RSig.Verify}),$$

where:

$\mathsf{RSig.PGen}(1^\lambda)$ is a PPT parameter generation algorithm that, on input the security parameter λ, returns a set of public parameters pp, which implicitly contains λ in unary, *i.e.*, $pp \leftarrow\!\!\$\ \mathsf{RSig.PGen}(1^\lambda)$. We assume that pp identifies the message space $\mathcal{M}$ of the scheme RSig.

$\mathsf{RSig.KGen}(pp)$ is a PPT key generation algorithm that, on input a set of public parameters pp, returns a key pair (pk, sk), where pk is a public key and sk is a secret key, *i.e.*, $(pk, sk) \leftarrow\!\!\$\ \mathsf{RSig.KGen}(pp)$.

$\mathsf{RSig.RandSK}(pp, sk, \rho)$ is a DPT secret key re-randomization algorithm that, on input a set of public parameters pp, a secret key sk, and a randomness ρ, returns a randomized secret key sk', *i.e.*, $sk' \leftarrow \mathsf{RSig.RandSK}(pp, sk, \rho)$. The randomness ρ is selected from the randomness space $\mathcal{R}$ of RSig, where $\mathcal{R}$ is assumed to be identified by pp.

$\mathsf{RSig.RandPK}(pp, pk, \rho)$ is a DPT public key re-randomization algorithm that, on input a set of public parameters pp, a public key pk, and a randomness $\rho \in \mathcal{R}$, returns a randomized public key pk', *i.e.*, $pk' \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$.

$\mathsf{RSig.Sign}(pp, sk, m)$ is a PPT signing algorithm that, on input a set of public parameters pp, a secret key sk, and a message m from the message space $\mathcal{M}$, returns a signature sig, *i.e.*, $sig \leftarrow\!\!\$\ \mathsf{RSig.Sign}(pp, sk, m)$. We write $sig = \perp$ to denote failure.

$\mathsf{RSig.Verify}(pp, pk, m, sig)$ is a DPT verification algorithm that, on input a set of public parameters pp, a public key pk, a message m, and a signature sig, returns 1 if sig is valid and 0 otherwise, *i.e.*, $b \leftarrow \mathsf{RSig.Verify}(pp, pk, m, sig)$, where $b \in \{0, 1\}$.

Signature schemes with perfectly re-randomizable keys are required to satisfy the correctness property defined as follows.

Definition 3.2. Let RSig be a signature scheme with perfectly re-randomizable keys, and $pp \in \mathsf{RSig.PGen}(1^\lambda)$. We say that RSig is $\mathrm{corr}_{\mathsf{RSig}}$-*correct* w.r.t. pp if and only if the following two conditions are satisfied:

1. For all randomness $\rho \in \mathcal{R} \cup \{null\}$, all messages $m \in \mathcal{M}$, every (re-randomized) key pair (pk, sk), where $(pk, sk) \in \mathsf{RSig.KGen}(pp)$ or $pk \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$ and $sk \leftarrow \mathsf{RSig.RandSK}(pp, sk, \rho)$, we have

$$\Pr_{sig \leftarrow\!\$ \mathsf{RSig.Sign}(pp,sk,m)}[\mathsf{RSig.Verify}(pp, pk, m, sig) \neq 1] \leq \mathrm{corr}_{\mathsf{RSig}}.$$

2. For all key pairs $(pk, sk) \in \mathsf{RSig.KGen}(pp)$ and a randomness $\rho \leftarrow\!\$ \mathcal{R}$, the distribution of the re-randomized key pair (pk', sk') is identical to the distribution of the key pair (pk'', sk''), where $pk' \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$, $sk' \leftarrow \mathsf{RSig.RandSK}(pp, sk, \rho)$, and $(pk'', sk'') \leftarrow\!\$ \mathsf{RSig.KGen}(pp)$.

In this thesis we consider a relaxed version of Definition 3.2, where its second condition holds only for public keys, but not in case of secret keys. We call a scheme RSig that satisfies this relaxation a *signature scheme with perfectly re-randomizable public keys*. More formally, RSig is defined as in Definition 3.1, but it is required to satisfy the following correctness definition:

Definition 3.3. Let RSig be a signature scheme with perfectly re-randomizable public keys, and $pp \in \mathsf{RSig.PGen}(1^\lambda)$. We say that RSig is $\mathrm{corr}_{\mathsf{RSig}}$-*correct* w.r.t. pp if and only if the following two conditions are satisfied:

1. This condition is defined similar to the first one of Definition 3.2.

2. For all public keys $(pk, \cdot) \in \mathsf{RSig.KGen}(pp)$ and a randomness $\rho \leftarrow\!\$ \mathcal{R}$, the distribution of public keys pk' and pk'' is identical, where $pk' \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$ and $(pk'', \cdot) \leftarrow\!\$ \mathsf{RSig.KGen}(pp)$.

The security of signature schemes with re-randomizable (public) keys is defined by the security notion EUF-CMA-RK given by Fleischhacker *et al.* [FKM+16]. This notion captures the EUF-CMA security (*cf.* Definition 2.15) under re-randomized keys. In the EUF-CMA-RK experiment, an adversary A^* gets access to a signing oracle $\mathsf{O}_{\mathsf{RSig}}$ that, on input a message $m \in \mathcal{M}$ and a randomness $\rho \in \mathcal{R} \cup \{null\}$, returns a signature on m either under the original key pair or under the re-randomized key pair that is derived from ρ. Observe that the randomness can be chosen by A^*. The adversary A^* wins the experiment if A^* is able to produce a valid forgery either under the original key, *i.e.*, $\rho = null$, or under a re-randomized key of its choice.

In the following we recall the security notion EUF-CMA-HRK due to Das *et al.* [DFL19], which captures the EUF-CMA security under honestly re-randomized keys. It is similar to the notion EUF-CMA-RK, but restricts the capabilities of A^* in the following way: In addition to the signing oracle $\mathsf{O}_{\mathsf{RSig}}$, in the EUF-CMA-HRK experiment A^* is given access to the oracle O_ρ, which returns a fresh randomness ρ. This randomness can be used to query $\mathsf{O}_{\mathsf{RSig}}$ and obtain a signature under the re-randomized key that is derived from ρ.

$\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}(pp)$:

1: $(pk, sk) \leftarrow\!\!\$\ \mathsf{RSig.KGen}(pp)$
2: $\mathrm{L}_m \leftarrow \varnothing$
3: $\mathrm{L}_\rho \leftarrow \varnothing$
4: $(m^*, sig^*, \rho^*) \leftarrow\!\!\$\ \mathsf{A}^{*\mathsf{O}_\rho, \mathsf{O}_{\mathsf{RSig}}}(pp, pk)$
5: **if** $m^* \in \mathrm{L}_m$ **then**
6: **return** 0
7: **if** $\rho^* \neq null$ **then**
8: **if** $\rho^* \notin \mathrm{L}_\rho$ **then**
9: **return** 0
10: $pk \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho^*)$
11: **return** $\mathsf{RSig.Verify}(pp, pk, m^*, sig^*)$

$\mathsf{O}_\rho()$:

21: $\rho \leftarrow\!\!\$\ \mathcal{R}$
22: $\mathrm{L}_\rho \leftarrow \mathrm{L}_\rho \cup \{\rho\}$
23: **return** ρ

$\mathsf{O}_{\mathsf{RSig}}(m, \rho)$:

31: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
32: **if** $\rho \neq null$ **then**
33: **if** $\rho \notin \mathrm{L}_\rho$ **then**
34: **return** $\perp$
35: $sk \leftarrow \mathsf{RSig.RandSK}(pp, sk, \rho)$
36: $sig \leftarrow\!\!\$\ \mathsf{RSig.Sign}(pp, sk, m)$
37: **return** sig

Figure 3.1: Definition of the experiment $\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}$.

The adversary A^* can only win the EUF-CMA-HRK experiment if A^* is able to return a valid forgery either under the original key or under a re-randomized key, where the related randomness was obtained honestly by querying the oracle O_ρ. We formally define the EUF-CMA-HRK security.

Definition 3.4. Let RSig be a signature scheme with honestly re-randomizable (public) keys, and $pp \in \mathsf{RSig.PGen}(1^\lambda)$. We say that RSig is $(t, q_{\mathsf{Sign}}, \varepsilon)$-EUF-CMA-HRK w.r.t. pp if, for every adversary A^* running in time at most t and making at most q_{Sign} signing queries to the oracle $\mathsf{O}_{\mathsf{RSig}}$, we have

$$\mathbf{Adv}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}(pp) = \Pr\left[\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}$ is depicted in Figure 3.1.

3.2 Lattice-Based Fiat-Shamir Signatures

In this section we review a generic construction of lattice-based Fiat-Shamir signatures, which is EUF-CMA in the QROM. Fiat-Shamir signatures are obtained by converting any canonical identification scheme [AABN02] together with a cryptographic hash function into a signature scheme via the Fiat-Shamir transformation [FS87]. For further details about canonical identification and the transformation we refer to Chapter 4.

Let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^n_\kappa$ be a cryptographic hash function. The signature scheme is formally described in Figure 3.2. The parameter generation algorithm $\mathsf{Sig.PGen}(1^\lambda)$ generates a set of public parameters given by

$$pp = \left(1^\lambda, n, k_1, k_2, q, \chi, \kappa, y, z, \ell_{\mathsf{H}}, \mathbf{A}\right),$$

Sig.KGen(pp):

1: $\mathbf{s} \leftarrow\!\!\$\ \chi^{k_1+k_2}$
2: $\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
3: $pk \leftarrow \mathbf{b}$
4: $sk \leftarrow \mathbf{s}$
5: **return** (pk, sk)

Sig.Verify(pp, pk, m, sig):

21: **parse** $sig = (c, \mathbf{z})$
22: **if** $\mathbf{z} \notin S_z^{k_1+k_2}$ **then**
23: **return** 0
24: **parse** $pk = \mathbf{b}$
25: $\mathbf{w} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$
26: **if** $c \neq \mathsf{H}(\mathbf{w}, m)$ **then**
27: **return** 0
28: **return** 1

Sig.Sign(pp, sk, m):

11: **parse** $sk = \mathbf{s}$
12: $\mathbf{y} \leftarrow\!\!\$\ S_y^{k_1+k_2}$
13: $\mathbf{v} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y} \pmod q$
14: $c \leftarrow \mathsf{H}(\mathbf{v}, m)$
15: $\mathbf{z} \leftarrow \mathbf{y} + \mathbf{s}c$
16: **if** $\mathbf{z} \notin S_z^{k_1+k_2}$ **then**
17: **restart** Sig.Sign
18: $sig \leftarrow (c, \mathbf{z})$
19: **return** sig

Figure 3.2: A formal description of a generic (non-optimized) construction of lattice-based signatures following the Fiat-Shamir approach. The algorithm Sig.PGen is explained in the text.

where $\mathbf{A}$ is a matrix that is chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. It can be shared among all users in a multi-user setting. In order to save bandwidth, $\mathbf{A}$ can be generated in practice by expanding a random seed using the function Expand, and including the seed in the public and secret key rather than storing the whole matrix. The parameter ℓ_{H} defines the length of the hash values output by H, which are then encoded as polynomials from $\mathbb{T}_\kappa^n$. The remaining parameters included in pp are explained below.

Basically, the key generation algorithm Sig.KGen generates an instance of MLWE (or a special variant of it such as RLWE) with respect to the parameters $(n, k_1, k_2, q, \chi, \mathbf{A})$. The secret vector $\mathbf{s}$ of this instance is chosen according to the distribution χ. In the state-of-the-art lattice-based signature schemes, *e.g.*, Dilithium [DKL+18] and qTESLA [ABB+20], the distribution χ of $\mathbf{s}$ is either the Gaussian distribution $D_{\mathbb{Z}^n,\sigma'}$ for some $\sigma' > 0$, or the uniform distribution over the set S_s, where $s \in \mathbb{Z}_{\geq 1}$. Note that the set of public parameters pp includes the description of the distribution χ, but not the distribution itself. That is, if $\chi = D_{\mathbb{Z}^n,\sigma'}$, then pp will include σ' rather than $D_{\mathbb{Z}^n,\sigma'}$. Similarly, if $\chi = S_s$, then pp will include s rather than S_s.

A signature on a message m consists of the pair $(c, \mathbf{z}) \in \mathbb{T}_\kappa^n \times S_z^{k_1+k_2}$. The polynomial c is the result of evaluating the hash function H on input $(\mathbf{v}, m)$, where $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y} \pmod q$ and $\mathbf{y}$ is a masking vector chosen randomly from the uniform distribution over $S_y^{k_1+k_2}$, for some integer y chosen as specified below. The vector $\mathbf{z}$ is generated by adding $\mathbf{y}$ to the secret-related vector $\mathbf{s}c$, *i.e.*, $\mathbf{z} = \mathbf{y} + \mathbf{s}c$. The signature is only output after verifying that $\mathbf{z}$ lies in the set $S_z^{k_1+k_2}$, where the integer bound z is defined depending on the distribution χ of the secret key $\mathbf{s}$. This check ensures that $\mathbf{z}$ follows the uniform distribution over $S_z^{k_1+k_2}$,

and that it does not leak any information about the secret key. If this is not the case, the signing algorithm is restarted. The probability that $\mathbf{z} \in S_z^{k_1+k_2}$ is given by

$$\left(\frac{2z+1}{2y+1}\right)^{(k_1+k_2)n}, \text{ where } z < y.$$

The integer y is usually selected such that the above probability is at least $1/M$, where $M \geq 1$ denotes the average number of restarts. We note that the vector $\mathbf{y}$ can also be chosen from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ for some $\sigma > 0$. In this case, the rejection sampling procedure Rej is applied on $(pp, \mathbf{z}, \mathbf{s}c)$ in order to ensure that $\mathbf{z}$ masks $\mathbf{s}c$ and that it follows the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ as well. We further remark that this generic construction can be optimized by either following the technique due to Bai and Galbraith [BG14] (adopted in qTESLA) or the approach used in Dilithium. The first one optimizes the signature size, while the second one optimizes the total size of public key and signature.

Finally, the EUF-CMA security of lattice-based Fiat-Shamir signatures in the QROM was analyzed in several works, *e.g.*, in [Unr17,DKL+18,KLS18,LZ19,DFMS19,ABB+20]. In particular, when modeling the hash function H as a quantum random oracle, the EUF-CMA security of the construction described above is based on the hardness of both MLWE w.r.t. $pp' = (n, k_1, k_2, q, \chi, \mathbf{A})$ and MSIS^∞ w.r.t. $pp'' = (n, k_1, k_2 + 1, q, 2z)$.

3.3 A Signature Scheme with Honestly Re-Randomizable Public Keys

In this section we propose a lattice-based construction of a signature scheme with honestly re-randomizable public keys. In such a scheme, the distribution of honestly re-randomized public keys is identical to the distribution of the original public key, while honestly re-randomized secret keys are allowed to be distributed differently from the original secret key (*cf.* Definition 3.3). The scheme extends the generic construction of lattice-based Fiat-Shamir signatures described in Section 3.2. The scheme is presented in Section 3.3.1. Its EUF-CMA-HRK security in the QROM is analyzed in Section 3.3.2. We propose concrete parameters of the scheme in Section 3.3.3.

3.3.1 Description of the Scheme

For the remainder of this section we let $\mathsf{Sig} = (\mathsf{Sig.PGen}, \mathsf{Sig.KGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ be the lattice-based signature scheme given in Figure 3.2. We define the following functions and algorithms:

Max_j is a function that, on input a polynomial $f \in R$, returns the j^{th} largest absolute coefficient of f. This function is used for bounding the secret-related vectors of the form $\mathbf{s}c \in R^{k_1+k_2}$, and hence the vector $\mathbf{z} \in S_z^{k_1+k_2}$ included in the signatures that are generated by the algorithm $\mathsf{Sig.Sign}$ (*cf.* Figure 3.2).

GenGauss is a DPT algorithm that, on input a tuple (k, σ, ρ, s), returns the k-dimensional integer vector $\mathbf{e} = (e_1, \ldots, e_k)$ that is sampled from the Gaussian distribution $D^k_{\mathbb{Z}^n,\sigma}$ using the randomness ρ, and such that each Gaussian polynomial e_i satisfies the condition $\sum_{j=1}^{\kappa} \mathsf{Max}_j(e_i) \leq s$ for $i \in [k]$, where $s > \kappa\sigma$ and κ specifies the set $\mathbb{T}^n_\kappa$. The algorithm is given in Figure 3.3.

G is a collision resistant hash function of the form $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{G}}$, where $\ell_\mathsf{G} \geq 2\lambda$. It is used to hash the public key in order to prevent related key attacks [MSM+16].

Our signature scheme with honestly re-randomizable public keys requires the distribution χ of the secret key to be the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$. More precisely, we assume that the key generation algorithm Sig.KGen of the signature scheme depicted in Figure 3.2 samples a random string str from the uniform distribution over $\{0,1\}^{\ell_{str}}$ and returns a key pair $(pk, sk) = (\mathbf{b}, \mathbf{s})$, where

$$\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q \text{ and } \mathbf{s} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, str, s/2).$$

The modified algorithm Sig.KGen is given in Figure 3.3. This changes the public parameters generated by the algorithm $\mathsf{Sig.PGen}(1^\lambda)$ to the following set:

$$pp = \big(1^\lambda, n, k_1, k_2, q, \sigma, s, \kappa, y, z, \ell_{str}, \ell_\mathsf{H}, \ell_\mathsf{G}, \mathbf{A}\big).$$

Setting $\chi = D_{\mathbb{Z}^n,\sigma}$ is crucial for re-randomizing the secret key in our construction because the sum of two Gaussian distributed random variables with standard deviation σ is also Gaussian distributed with standard deviation $\sqrt{2}\sigma$ (see Lemma 2.7).

Next, we describe our signature scheme with honestly re-randomizable public keys. The respective algorithms are formalized in Figure 3.3.

Parameter generation. Given 1^λ, RSig.PGen returns the set of pubic parameters obtained by running Sig.PGen on input 1^λ.

Key generation. On input a set of public parameters pp, RSig.KGen runs Sig.KGen on input pp to obtain a key pair (pk, sk) as described above, where $pk = \mathbf{b} \in R_q^{k_1}$ and $sk = \mathbf{s} \in D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$. Then, it computes $hpk = \mathsf{G}(pk)$, prepends hpk to sk, and returns the updated key pair (pk, sk).

Secret key re-randomization. Given a set of public parameters pp, a secret key sk of the form $(hpk, \mathbf{s}) \in \{0,1\}^{\ell_\mathsf{G}} \times D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$, and an honestly generated randomness ρ, the algorithm RSig.RandSK generates the Gaussian vector $\mathbf{e} \in D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$ via the algorithm GenGauss on input $(k_1 + k_2, \sigma, \rho, s/2)$. Then, it computes the vector $\mathbf{s}' = \mathbf{s} + \mathbf{e}$. Note that by Lemma 2.7, the vector $\mathbf{s}'$ follows the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Finally, the algorithm computes $\mathbf{b}' = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}' \pmod q$, $hpk' = \mathsf{G}(\mathbf{b}')$, and returns the honestly re-randomized secret key $sk' = (hpk', \mathbf{s}')$.

Public key re-randomization. Given a set of public parameters pp, a public key pk of the form $\mathbf{b} \in R_q^{k_1}$, and an honestly generated randomness ρ, the algorithm RSig.RandPK runs GenGauss on input $(k_1 + k_2, \sigma, \rho, s/2)$ to obtain the Gaussian vector $\mathbf{e} \in D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$. Then, it computes the vector $\mathbf{b}' = \mathbf{b} + [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} \pmod q$ and returns the honestly re-randomized public key $pk' = \mathbf{b}'$.

GenGauss(k, σ, ρ, s):

1: **for** $i = 1$ **to** k **do**
2: $\quad e_i \leftarrow \perp$
3: $\quad ctr \leftarrow 1$
4: $\quad$ **while** $e_i = \perp$ **do**
5: $\quad\quad \rho_i \leftarrow \mathsf{Expand}(\rho, i, ctr)$
6: $\quad\quad e_i \leftarrow D_{\mathbb{Z}^n,\sigma}(\rho_i)$
7: $\quad\quad$ **if** $\sum_{j=1}^{\kappa} \mathsf{Max}_j(e_i) > s$ **then**
8: $\quad\quad\quad ctr \leftarrow ctr + 1$
9: $\quad\quad\quad e_i \leftarrow \perp$
10: $\mathbf{e} \leftarrow (e_1, \ldots, e_k)$
11: **return** $\mathbf{e}$

Sig.KGen(pp):

21: $str \leftarrow\!\!\$\ \{0,1\}^{\ell_{str}}$ // $\ell_{str} \geq 2\lambda$
22: $\mathbf{s} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, str, s/2)$
23: $\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
24: $pk \leftarrow \mathbf{b}$
25: $sk \leftarrow \mathbf{s}$
26: **return** (pk, sk)

RSig.PGen(1^λ):

31: $pp \leftarrow\!\!\$\ \mathsf{Sig.PGen}(1^\lambda)$
32: **return** pp

RSig.KGen(pp):

41: $(pk, sk) \leftarrow\!\!\$\ \mathsf{Sig.KGen}(pp)$
42: $hpk \leftarrow \mathsf{G}(pk)$
43: $sk \leftarrow (hpk, sk)$
44: **return** (pk, sk)

RSig.RandSK(pp, sk, ρ):

51: **parse** $sk = (hpk, \mathbf{s})$
52: $\mathbf{e} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, \rho, s/2)$
53: $\mathbf{s}' \in D^{k_1+k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma} \leftarrow \mathbf{s} + \mathbf{e}$
54: $\mathbf{b}' \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}' \pmod q$
55: $hpk' \leftarrow \mathsf{G}(\mathbf{b}')$
56: $sk' \leftarrow (hpk', \mathbf{s}')$
57: **return** sk'

RSig.RandPK(pp, pk, ρ):

61: **parse** $pk = \mathbf{b}$
62: $\mathbf{e} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, \rho, s/2)$
63: $\mathbf{b}' \leftarrow \mathbf{b} + [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} \pmod q$
64: $pk' \leftarrow \mathbf{b}'$
65: **return** pk'

RSig.Sign(pp, sk, m):

71: **parse** $sk = (hpk, \mathbf{s})$
72: $\mu \leftarrow (m, hpk)$
73: $sig \leftarrow\!\!\$\ \mathsf{Sig.Sign}(pp, sk, \mu)$
74: **return** sig

RSig.Verify(pp, pk, m, sig):

81: $\mu \leftarrow (m, \mathsf{G}(pk))$
82: **return** $\mathsf{Sig.Verify}(pp, pk, \mu, sig)$

Figure 3.3: Description of our signature scheme with honestly re-randomizable public keys.

Signing. Given a set of public parameters pp, a secret key $sk = (hpk, \mathbf{s})$, and a message m, the algorithm RSig.Sign returns the signature obtained by calling the algorithm Sig.Sign on input (pp, sk, μ), where the message μ is given by $\mu = (m, hpk)$. Signing messages together with the hash value of (honestly re-randomized) public keys ensures security under related key attacks [MSM+16].

Verification. On input (pp, pk, m, sig), the verification algorithm $\mathsf{RSig.Verify}$ returns the bit obtained by running $\mathsf{Sig.Verify}$ on input (pp, pk, μ, sig), where the message μ is given by $\mu = (m, \mathsf{G}(pk))$.

We remark that re-randomizing the secret key must be carried out only with the original secret key sk, *i.e.*, the secret key generated by the algorithm $\mathsf{RSig.KGen}$. This means that a re-randomized secret key sk' cannot be used to generate a new re-randomized one. This ensures that all honestly re-randomized secret keys have identical distribution, *i.e.*, the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Furthermore, signatures generated using honestly re-randomized secret keys have different distribution from signatures generated using the original secret key. More precisely, the vector $\mathbf{z}$ included in the signature follows the uniform distribution over $S^{k_1+k_2}_z$, where

$$z = \begin{cases} y - s/2 & \text{if } (\cdot, sk) \leftarrow\!\!\$\ \mathsf{RSig.KGen}(\cdot) \\ y - s & \text{if } sk \leftarrow \mathsf{RSig.RandSK}(\cdot) \end{cases}$$

The integer bound y is chosen such that the probability of generating valid signatures for both forms of z is at least $1/M$, *i.e.*,

$$\left(\frac{2z+1}{2y+1}\right)^{(k_1+k_2)n} \geq 1/M, \tag{3.1}$$

where $M \geq 1$ is the expected number of restarting the algorithm $\mathsf{Sig.Sign}$ (see Section 3.2).

Finally, we observe that the scheme RSig described above satisfies both correctness conditions given in Definition 3.3. In particular, the first condition of Definition 3.3 follows from the correctness of the underlying Fiat-Shamir signature scheme Sig, while the second condition follows from the MLWE assumption. That is, for any public key $\mathbf{b}$ and any honestly re-randomized public key $\mathbf{b}'$, both pairs $(\mathbf{A}, \mathbf{b})$, $(\mathbf{A}, \mathbf{b}')$ are indistinguishable from the uniform distribution over $R_q^{k_1 \times k_2} \times R_q^{k_1}$.

3.3.2 Security Analysis

In this section we analyze the EUF-CMA-HRK security of the construction introduced in Section 3.3.1 in the post-quantum setting. More precisely, we give a reduction in the QROM from the EUF-CMA security of the underlying Fiat-Shamir signature scheme to the EUF-CMA-HRK security of our signature scheme with honestly re-randomized public keys.

Theorem 3.5. *Let* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^n_\kappa$ *be a cryptographic hash function modeled as a quantum random oracle. Furthermore, let* Sig *be the Fiat-Shamir signature scheme described in Figure 3.2. The signature scheme with honestly re-randomizable public keys* RSig *depicted in Figure 3.3 is* $(t, q_{\mathsf{Sign}}, q_{\mathsf{H}}, \varepsilon)$-EUF-CMA-HRK *w.r.t.* $pp \in \mathsf{RSig.PGen}(1^\lambda)$ *in the QROM if* Sig *is* $(t', M \cdot q_{\mathsf{Sign}}, q_{\mathsf{H}}, \varepsilon')$-EUF-CMA *w.r.t.* $pp \in \mathsf{Sig.PGen}(1^\lambda)$ *in the QROM, where* $t' \approx t$ *and* $\varepsilon' = \varepsilon$.

Proof. Let A^* be an adversary that is able to forge signatures under the signature scheme with honestly re-randomizable public keys RSig, *i.e.*, A^* runs in time t and wins the experiment $\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}$ given in Definition 3.4 with probability ε. We construct a reduction algorithm R that runs in time $t' \approx t$ and uses A^* in a black-box manner to win the experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}$ (see Definition 2.15) against the scheme Sig with probability $\varepsilon' = \varepsilon$.

Recall that since we consider the post-quantum security model, the oracles provided by the challenger are run on a classical computer. Hence, even if the adversary has quantum computing power, it gets only classical access to these oracles. However, the adversary has quantum access to the random oracle that implements the hash function H, *i.e.*, it can query this oracle in superposition. More precisely, A^* has classical access to the random oracle O_ρ to obtain randomness for re-randomizing keys, classical access to the signing oracle $\mathsf{O}_{\mathsf{RSig}}$ to get signatures that are generated by the scheme RSig, and quantum access to the random oracle O'_{H} that implements the hash function H. The reduction R has quantum access to the random oracle O_{H} that implements H and classical access to the signing oracle $\mathsf{O}_{\mathsf{Sig}}$, which returns to R signatures that are generated by the scheme Sig. Furthermore, R initializes two empty lists $\mathrm{L}_m, \mathrm{L}_\rho$ in order to store queries to $\mathsf{O}_{\mathsf{RSig}}, \mathsf{O}_\rho$, respectively.

The description of the algorithm R including the simulation of the oracles $\mathsf{O}_\rho, \mathsf{O}'_{\mathsf{H}}, \mathsf{O}_{\mathsf{RSig}}$ is given in Figure 3.4.

Let (m, sig, ρ) be a valid forgery output by A^*, where $sig = (c, \mathbf{z})$. This means that $m \notin \mathrm{L}_m$ and $\mathsf{RSig.Verify}(pp, pk, m, sig) = 1$. Moreover, $\rho \in \mathrm{L}_\rho$ in case $\rho \neq null$.

We first analyze the case that $\rho = null$. In this case we have

$$\mathbf{z} \in S^{k_1+k_2}_{y-s/2} \text{ and } c = \mathsf{O}'_{\mathsf{H}}(\mathbf{w}, m, \mathsf{G}(pk)) = \mathsf{O}_{\mathsf{H}}(\mathbf{w}, m, \mathsf{G}(pk)),$$

where $\mathbf{w} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$. Hence, this forgery constitutes a valid signature under the scheme Sig on message $\mu = (m, \mathsf{G}(pk))$. Note that if c was not queried by some input, then A^* produces such c only with probability $1/|\mathbb{T}^n_\kappa|$. Thus, with probability $1 - 1/|\mathbb{T}^n_\kappa|$, c must be a random oracle answer to a query made by A^*, where $|\mathbb{T}^n_\kappa| = 2^\kappa \binom{n}{\kappa}$ and κ is chosen such that $|\mathbb{T}^n_\kappa| \geq 2^{2\lambda}$. This ensures that the probability of mapping two different hash values to the same output of H is at most $2^{-2\lambda}$.

Next, we assume that $\rho \neq null$, *i.e.*, A^* returns a valid forgery under honestly re-randomized public key $pk' = \mathbf{b}'$. This means that $\mathbf{z} \in S^{k_1+k_2}_{y-s}$. In this case, R transforms this signature into a forgery under the original public key $pk = \mathbf{b}$. This is carried out as follows: R runs the algorithm $\mathsf{GenGauss}$ on input $(k_1 + k_2, \sigma, \rho, s/2)$ to obtain the vector $\mathbf{e}$ that follows the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$. Then, R computes the vector $\mathbf{z}' = \mathbf{z} - \mathbf{e}c$, which satisfies

$$\|\mathbf{z}'\|_\infty \leq \|\mathbf{z}\|_\infty + \|\mathbf{e}c\|_\infty \leq y - s + s/2 = y - s/2.$$

Furthermore, we have

$$\mathbf{w} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}' - \mathbf{b}c = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{z} - \mathbf{e}c) - \mathbf{b}c = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}'c \pmod q.$$

Hence, it holds that

$$\mathbf{z}' \in S^{k_1+k_2}_{y-s/2} \text{ and } c = \mathsf{O}'_{\mathsf{H}}(\mathbf{w}, m, \mathsf{G}(pk')) = \mathsf{O}_{\mathsf{H}}(\mathbf{w}, m, \mathsf{G}(pk')).$$

$\mathsf{R}(pp, pk)$:

1: $\mathrm{L}_m \leftarrow \varnothing$
2: $\mathrm{L}_\rho \leftarrow \varnothing$
3: $(m, sig, \rho) \leftarrow\!\!\$\ \mathsf{A}^{*\mathsf{O}_\rho, \mathsf{O}'_\mathsf{H}, \mathsf{O}_{\mathsf{RSig}}}(pp, pk)$
4: **if** $m \in \mathrm{L}_m$ **then**
5: **return** 0
6: **if** $\rho = null$ **then**
7: $\mu \leftarrow (m, \mathsf{G}(pk))$
8: **return** $\mathsf{Sig.Verify}(pp, pk, \mu, sig)$
9: **if** $\rho \neq null$ **then**
10: **if** $\rho \notin \mathrm{L}_\rho$ **then**
11: **return** 0
12: $pk' \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$
13: $\mu' \leftarrow (m, \mathsf{G}(pk'))$
14: $\mathbf{e} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, \rho, s/2)$
15: **parse** $sig = (c, \mathbf{z})$
16: $\mathbf{z}' \leftarrow \mathbf{z} - \mathbf{e}c$
17: $sig' \leftarrow (c, \mathbf{z}')$
18: **return** $\mathsf{Sig.Verify}(pp, pk', \mu', sig')$

$\mathsf{O}_\rho()$:

21: $\rho \leftarrow\!\!\$\ \mathcal{R}$
22: $\mathrm{L}_\rho \leftarrow \mathrm{L}_\rho \cup \{\rho\}$
23: **return** ρ

$\mathsf{O}'_\mathsf{H}(\cdot)$:

31: **return** $\mathsf{O}_\mathsf{H}(\cdot)$

$\mathsf{RSig.Sim}(pp, pk, m, \rho)$:

41: **if** $\rho = null$ **then**
42: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
43: $\mu \leftarrow (m, \mathsf{G}(pk))$
44: $sig \leftarrow\!\!\$\ \mathsf{O}_{\mathsf{Sig}}(\mu)$
45: **return** sig
46: **if** $\rho \neq null$ **then**
47: **if** $\rho \notin \mathrm{L}_\rho$ **then**
48: **return** $\perp$
49: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
50: $pk' \leftarrow \mathsf{RSig.RandPK}(pp, pk, \rho)$
51: $\mu' \leftarrow (m, \mathsf{G}(pk'))$
52: $\mathbf{e} \leftarrow \mathsf{GenGauss}(k_1 + k_2, \sigma, \rho, s/2)$
53: $sig \leftarrow \perp$
54: **while** $sig = \perp$ **do**
55: $sig' \leftarrow\!\!\$\ \mathsf{O}_{\mathsf{Sig}}(\mu')$
56: **parse** $sig' = (c, \mathbf{z}')$
57: $\mathbf{z} \leftarrow \mathbf{z}' + \mathbf{e}c$
58: **if** $\mathbf{z} \notin S_{y-s}^{k_1+k_2}$ **then**
59: $sig \leftarrow \perp$
60: $sig \leftarrow (c, \mathbf{z})$
61: **return** sig

Figure 3.4: Reduction from the EUF-CMA security of the Fiat-Shamir signature scheme Sig given in Figure 3.2 to the EUF-CMA-HRK security of our signature scheme with honestly re-randomizable public keys (*cf.* Figure 3.3). Queries to O'_H made by A^* are redirected to the oracle O_H, to which R has access. Queries to O_ρ are answered locally by R.

Therefore, the forgery output by A^* can be turned into a valid forgery sig' under the original public key $pk = \mathbf{b}$ for message $\mu' = (m, \mathsf{G}(pk'))$, *i.e.*, the signature sig' is a valid forgery under the scheme Sig.

We observe that the environment of A^* is perfectly simulated, and whenever A^* wins the experiment $\mathbf{Exp}^{\text{EUF-CMA-HRK}}_{\mathsf{RSig},\mathsf{A}^*}$, R wins the experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}$, *i.e.*, $\varepsilon' = \varepsilon$.

Finally, we show that the number of signing queries made by R to the signing oracle $\mathsf{O}_{\mathsf{Sig}}$ is at most Mq_{Sign}, where M is the average number of restarting the signing algorithm $\mathsf{Sig.Sign}$ and q_{Sign} is the maximum number of signing queries made by A^* to the signing oracle $\mathsf{O}_{\mathsf{RSig}}$. Let (m, ρ) be a signing query made by A^* to the oracle $\mathsf{O}_{\mathsf{RSig}}$. We assume that A^* sends q_1 queries to $\mathsf{O}_{\mathsf{RSig}}$ of the form $(m, \rho = null)$ and q_2 queries of the form $(m, \rho \neq null)$ so

that we have $q_1 + q_2 \leq q_{\mathsf{Sign}}$. If $\rho = \mathit{null}$, then R queries its signing oracle $\mathsf{O}_{\mathsf{Sig}}$ only once. However, in case $\rho \neq \mathit{null}$, R queries $\mathsf{O}_{\mathsf{Sig}}$ at most M times (see Line 54 to Line 59 in Figure 3.4). This is because R transforms the signature $\mathit{sig}' = (c, \mathbf{z}') \in \mathbb{T}_\kappa^n \times S_{y-s/2}^{k_1+k_2}$ into a signature $\mathit{sig} = (c, \mathbf{z}) \in \mathbb{T}_\kappa^n \times S_{y-s}^{k_1+k_2}$ by computing $\mathbf{z} = \mathbf{z}' + \mathbf{e}c$, where $\mathbf{e}$ is obtained by running $\mathsf{GenGauss}(k_1 + k_2, \sigma, \rho, s/2)$. The probability that $\mathbf{z} \in S_{y-s}^{k_1+k_2}$ when $\mathbf{z}' \in S_{y-s/2}^{k_1+k_2}$ is given by

$$\left(\frac{2(y-s)+1}{2(y-s/2)+1}\right)^{(k_1+k_2)n} > \left(\frac{2(y-s)+1}{2y+1}\right)^{(k_1+k_2)n} \geq 1/M,$$

where Equation (3.1) is used to obtain the bound $1/M$. Therefore, after at most M queries to $\mathsf{O}_{\mathsf{Sig}}$, R obtains a valid signature sig, and hence can answer the query of A^*. Hence, the total queries made by R to the signing oracle $\mathsf{O}_{\mathsf{Sig}}$ is given by $q_1 + Mq_2 \leq Mq_{\mathsf{Sign}}$. □

3.3.3 Concrete Parameters

In this section we propose concrete parameters for our signature scheme with honestly re-randomizable public keys introduced in Section 3.3.1. The parameters target 128 bits of post-quantum security.

We review the parameter description of the scheme in Table 3.1. The table also gives the theoretical sizes of (re-randomized) keys and signatures. Our concrete parameters are summarized in Table 3.2.

In the following we highlight some key points that we considered when selecting these parameters. For simplicity, we assume that the original key pair created by the algorithm $\mathsf{RSig.KGen}$ is only used for re-randomization, and signatures are generated and verified using honestly re-randomized key pairs. This means, we set $z = y - s$. We remark that this assumption is already the case in the setting of deterministic wallet schemes [DFL19, A1].

The modulus q can be any 25-bit prime number that satisfies $q = 1 \pmod{2n}$ and $q > 2y$. The first condition is for efficient polynomial multiplication, while the latter is for security. We set $\kappa = 60$ so that $|\mathbb{T}_\kappa^n| \geq 2^{256}$, and hence the probability of mapping two different hash values to the same output of H is at most 2^{-256}. For the parameter $\sigma = 4$, the instance of MLWE^2 with respect to the parameters $(n, k_1, k_2, q, \sigma, \mathbf{A})$, which underlies the public key, is hard enough for the target security level. According to the LWE-Estimator [APS15], the number of operations required to solve this instance is approximately 2^{280}.

The parameter y is chosen as follows: We fix the average number of restarting the signing algorithm $\mathsf{RSig.Sign}$ to $M = 4$. Then, we choose y satisfying Equation (3.1). Furthermore, y is chosen to be a power of two in order to facilitate the uniform sampling over $S_y^{k_1+k_2}$. By following the methodology described in Section 2.3.3, the algorithm BKZ requires at least 2^{131} operations in order to solve the instance of MSIS^∞ with respect to the parameters $(n, k_1, k_2 + 1, q, 2z)$, which underlies our scheme.

Finally, we note that the size of the vector $\mathbf{s}'$ included in a re-randomized secret key is computed according to Lemma 2.6. More concretely, we set $t = 11.3$ so that $\|\mathbf{s}'\|_\infty$ is bounded by $t\sqrt{2}\sigma$ with probability at least $1 - 2^{-80}$, *i.e.*,

$$\Pr\left[\|\mathbf{s}'\|_\infty > t\sqrt{2}\sigma\right] \leq 2^{-80}.$$

Table 3.1: A review of the parameters and sizes of keys and signatures of the signature scheme with honestly re-randomizable public keys presented in Section 3.3.1. We note that in practice, the public matrix $\mathbf{A}$ is deterministically generated by expanding a uniformly random seed from $\{0,1\}^{\ell_{seed}}$ via the function Expand. This seed is then included in the public and secret key. Therefore, the sizes of keys are given accordingly. Furthermore, we assume that the function H outputs hash values of length ℓ_{H}, which are then encoded as polynomials from $\mathbb{T}_\kappa^n$.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 1 \pmod{2n}$, $q > 2y$
σ	Standard deviation of $\mathbf{s}$	$\sigma > 0$
s	Bound of $\|\mathbf{s}c\|_\infty$	$s = 2\lceil \eta_s \kappa \sigma \rceil$, $\eta_s > 0$
κ	Specifies the set $\mathbb{T}_\kappa^n$	$2^\kappa \binom{n}{\kappa} \geq 2^{2\lambda}$
M	No. restarts of $\mathsf{RSig.Sign}$	$M \geq 1$
y	Bound of $\|\mathbf{y}\|_\infty$	$\left(\frac{2z+1}{2y+1}\right)^{(k_1+k_2)n} \geq 1/M$
z	Bound of $\|\mathbf{z}\|_\infty$	$z = \begin{cases} y - s/2 & \text{if } (\cdot, sk) \leftarrow\!\!\$\ \mathsf{RSig.KGen}(\cdot) \\ y - s & \text{if } sk \leftarrow \mathsf{RSig.RandSK}(\cdot) \end{cases}$
$\ell_{str}, \ell_{\mathsf{H}}, \ell_{\mathsf{G}}, \ell_{seed}$	length of randomness and in-/output of $\mathsf{H}, \mathsf{G}, \mathsf{Expand}$	$\ell_{str}, \ell_{\mathsf{H}}, \ell_{\mathsf{G}}, \ell_{seed} \geq 2\lambda$
(Re-randomized) public key size (bit)		$\ell_{seed} + k_1 n \lceil \log q \rceil$
Secret key size (bit)		$\ell_{seed} + (k_1 + k_2) n \lceil \log(t\sigma + 1) \rceil$, $t > 0$ (see Lemma 2.6)
Re-randomized secret key size (bit)		$\ell_{seed} + (k_1 + k_2) n \lceil \log(t\sqrt{2}\sigma + 1) \rceil$
Signature size (bit)		$\ell_{\mathsf{H}} + (k_1 + k_2) n \lceil \log(2z + 1) \rceil$

3.4 Application to Post-Quantum Deterministic Wallets

In Section 3.1 we defined the notions of signature schemes with re-randomizable public keys and EUF-CMA-HRK security. While these definitions deviate from the notions given in previous works [FKM+16, DFL19], it turns out that they are sufficient for at least building post-quantum deterministic wallets as we explain below. In other words, we show in this section that our signature scheme with honestly re-randomizable public keys presented in Section 3.3 can be used to construct post-quantum deterministic wallets. To this end, we first give a brief description of deterministic wallet schemes.

Deterministic wallets are used in many cryptocurrencies, such as Bitcoin and Ethereum, in order to protect secret keys against theft. A deterministic wallet scheme consists of two components called *hot wallet* and *cold wallet*. The hot wallet is permanently connected to the Internet, while the cold wallet only comes online when a specified amount of money has to be transferred. In other words, it is crucial for security to keep both components disconnected from each other. At a high level, a deterministic wallet scheme works as follows. In

Table 3.2: Concrete parameters of the scheme RSig depicted in Figure 3.3, and corresponding sizes (in bytes) of re-randomized keys and signatures. The parameters target 128 bits of post-quantum security. We set $\ell_{str} = \ell_{\mathsf{H}} = \ell_{\mathsf{G}} = \ell_{seed} = 256$ (see Table 3.1).

n	k_1	k_2	q	σ	s	κ	M	y	z	pk'	sk'	sig
256	5	4	$\approx 2^{25}$	4	1200	60	4	2^{21}	2095952	4032	2048	6368

an initialization phase, a pair of master keys (mpk, msk) is generated. The master public key mpk is stored on the hot wallet, while the cold wallet keeps the master secret key msk. The main building block of the scheme is a deterministic key derivation procedure, which is realized by a signature scheme with re-randomizable keys. It allows both the hot and cold wallet to derive matching public and secret session keys without interacting with each other. To this end, they share a state, which allows to derive the corresponding session key by combining the master key with a randomness that can be deterministically derived via this state. The (post-quantum) security of deterministic wallets due to [DFL19, A1] considers two properties. The first one is called *wallet unforgeability*, which states that funds sent to the cold wallet must remain secure even if the hot wallet is corrupted. The second property is called *wallet unlinkability*. It guarantees that individual transactions that are sent to the same wallet are unlinkable despite being publicly available on the blockchain.

Next, we explain the relevance of our relaxed notions of signature schemes with re-randomizable public keys and EUF-CMA-HRK security. These are the notions satisfied by our construction depicted in Figure 3.3, and considered in [A1, Section 3] to build post-quantum deterministic wallets.

Re-randomizable public keys. One of the main benefits of using deterministic wallets is that individual payments to the wallet are unlinkable. In order to satisfy unlinkability, Das *et al.* [DFL19] required that the underlying signature scheme must have re-randomizable public and secret keys. However, the adversary, playing the unlinkability experiment, gets access only to the public key, while the secret key is never revealed to the adversary (as revealing it would trivially break the security of the scheme). Hence, it is sufficient to use our relaxed notion of re-randomizable public keys in order to achieve the unlinkability property.

EUF-CMA-HRK security. Similar to Das *et al.* [DFL19], we use the security notion of EUF-CMA under honestly re-randomizable keys, where unforgeability only holds if the randomness used to derive the keys is *honestly* generated. This is in contrast to the stronger notion of EUF-CMA security under re-randomizable keys defined by Fleischhacker *et al.* [FKM+16], where unforgeability must hold even if the randomness is chosen by the adversary. However, deterministic wallets derive the randomness deterministically from a state, which is initially generated during a trusted setup when the master keys are created. Therefore, the adversary has no influence on the randomness used during the re-randomization procedures. Hence, the security notion of EUF-CMA-HRK is not only suitable but also sufficient in the context of deterministic wallets.

3.5 Alternative Methods for Re-Randomizing Keys

In this section we describe alternative approaches for re-randomizing keys in the lattice setting and show why the construction introduced in Section 3.3 is the most suitable approach, particularly, in the context of deterministic wallets.

First, we recall that the scheme given in Section 3.3 assumes that the distribution χ of the secret key that is generated by the algorithm RSig.KGen is the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$. This allows to use Lemma 2.7 in order to obtain re-randomized secret keys that are also Gaussian distributed, but with a slightly different standard deviation, *i.e.*, $D_{\mathbb{Z}^n,\sqrt{2}\sigma}$. Our scheme cannot use secret keys that are sampled from the uniform distribution over a set $S_s \subset R_q$, for some integer s. This is due to the fact that the sum of two random polynomials chosen from the uniform distribution over S_s does not yield a polynomial that follows the uniform distribution over some subset $S \subseteq S_s$. Therefore, using a uniformly random secret key for re-randomization would yield re-randomized secret keys with unknown distribution. This raises, at least, security issues as the hardness assumption underlying the re-randomized key pairs may be unclear.

Let us now discuss the alternative approaches.

Re-randomizing Gaussian distributed secret keys. In theory, we can re-randomize key pairs such that the re-randomized secret keys have the same distribution as the original secret key. More precisely, assume that the secret key sk is sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$. Given a randomness ρ, a re-randomized secret key is computed as $sk' = sk + \rho$. Goldwasser *et al.* [GKPV10] showed that the statistical distance between the distribution of sk and sk' is given by $\Delta = \frac{\|\rho\|_\infty}{\sqrt{2\pi}\sigma}$. Therefore, choosing σ large enough such that $\Delta \approx 0$, ensures that sk' follows the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$ as well. For instance, if $\|\rho\|_\infty = 1$, then we have to set $\sigma \approx 2^{79}$ in order to obtain a statistical distance $\Delta \approx 2^{-80}$. This value of σ yields secret keys and signatures of huge size. This makes the aforementioned method of theoretical interest only and rules out using the resulting scheme in practice, although it achieves the original (not relaxed) correctness definition of signature schemes with re-randomizable keys due to Fleischhacker *et al.* [FKM+16] (*cf.* Definition 3.2).

Re-randomizing uniformly distributed secret keys. Let $s \in \mathbb{Z}_{>1}$. It is theoretically possible to sample the secret key from the uniform distribution over S_s rather than the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$, but still under the relaxed correctness definition of signature schemes with re-randomizable keys (*cf.* Definition 3.3). This can be realized as follows: Assume without loss of generality that $sk \in S_s^{k_1+k_2}$ and $\rho \in S_1^{k_1+k_2}$. Then, the re-randomized secret key $sk' = sk + \rho$ follows the uniform distribution over $S_{s-1}^{k_1+k_2}$ with probability

$$\left(\frac{2(s-1)+1}{2s+1}\right)^{(k_1+k_2)n} = \left(\frac{2s-1}{2s+1}\right)^{(k_1+k_2)n}. \tag{3.2}$$

For a very large value of s, this probability would be approximately one. For example, by considering the recommended set of parameters proposed for the signature scheme Dilithium [DKL+18], we have $k_1 = 5$, $k_2 = 4$, and $n = 256$. Hence, we have to set

$s \approx 2^{92}$ in order to make the probability stated in Equation (3.2) at least $1 - 2^{-80}$. This example shows that this approach is merely of theoretical interest only, and it is not suitable for practical applications as it requires huge sizes of secret keys, and hence signatures.

Allowing re-randomization algorithms to communicate. Consider a cryptographic application, in which the algorithms RSig.RandSK and RSig.RandPK synchronize after each invocation of RSig.RandSK. In this case, we can re-randomize key pairs as follows: Given a secret key sk and a randomness ρ, the algorithm RSig.RandSK uses ρ together with a counter ctr, initialized by one, in order to deterministically generate a randomness ρ', *e.g.*, by using the function Expand on input (ρ, ctr). Then, it computes $sk' = sk + \rho'$ and returns the re-randomized secret key sk' only after verifying that it has the correct distribution. If this is not the case, the algorithm RSig.RandSK increases ctr by one and repeats the process until sk' follows the target distribution. More precisely, if sk is chosen from the uniform distribution over $S_s^{k_1+k_2}$ for some $s \in \mathbb{Z}_{>1}$, then RSig.RandSK verifies that $sk' \in S_{s-1}^{k_1+k_2}$, where we assume without loss of generality that $\|\rho'\|_\infty = 1$. If sk is sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$, then RSig.RandSK runs the rejection sampling algorithm Rej on input (pp, sk', ρ') to verify that sk' follows the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$ as well. The algorithm RSig.RandPK needs to receive the corresponding ctr from RSig.RandSK in order to generate the re-randomized public key related to sk'. Note that if sk is Gaussian distributed, then we even obtain a scheme with re-randomizable public and secret keys as defined by Fleischhacker *et al.* [FKM+16], *i.e.*, the original (not relaxed) correctness definition of signature schemes with re-randomizable keys (*cf.* Definition 3.2).

Observe that this method is practical and may be applicable, *e.g.*, in the construction of sanitizable signatures proposed in [FKM+16]. However, we note that it cannot be used in the setting of deterministic wallets due to the fact that in each signing process, the algorithm RSig.RandPK must obtain the correct ctr that were used to generate sk'. This synchronization requirement undermines the main concept of deterministic wallets, namely the fact that hot and cold wallets do not communicate with each other after the initialization process.

3.6 An Instantiation with the Signature Scheme qTESLA

In this section we show how our signature scheme with honestly re-randomizable public keys introduced in Section 3.3 can be instantiated with the ordinary signature scheme qTESLA [ABB+20].

First, we remark that the most recent lattice-based signature schemes following the Fiat-Shamir approach are Dilithium [DKL+18] and qTESLA. We consider the latter scheme, since its secret key follows the Gaussian distribution. This is crucial for re-randomizing the secret key in our setting, and hence sufficient for our scheme with honestly re-randomizable public keys presented in Section 3.3. On the other hand, the secret key of Dilithium follows the uniform distribution over a small subset of R_q, which is not suitable in our setting. Using the Gaussian distribution in the key generation algorithm of Dilithium requires to adjust its

parameters and security analysis. The resulting scheme would be similar to qTESLA, with slight differences in how signatures are compressed. We choose not to modify Dilithium's original design but stick to qTESLA, which does not need any modification for our setting and is well-studied in comparison to a modified version of Dilithium.

The design of our signature scheme with honestly re-randomizable public keys is based on lattices over modules. In order to employ qTESLA in our construction we set $k_2 = 1$ to obtain a variant based on lattices over ideals and security based on the hardness of RLWE. The original secret key created by the key generation algorithm RSig.KGen is the vector $\mathbf{s}$, which is sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+1}$. The corresponding public key is given by the vector $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{a}] \cdot \mathbf{s} \pmod q$, where $\mathbf{a} = (a_1, \ldots, a_{k_1}) \leftarrow\!\!\$\ R_q^{k_1}$. For the remainder of this section we write $\mathbf{s} = (\mathbf{s}_1, s_2) \in D_{\mathbb{Z}^n,\sigma}^{k_1} \times D_{\mathbb{Z}^n,\sigma}$ and $\mathbf{s}_1 = (s_{1,1}, \ldots, s_{k_1,1})$. The vector $\mathbf{s}$ satisfies

$$\mathsf{Max}_j(s_{i,1}) \leq s/2 \text{ for all } i \in [k_1], \text{ and } \mathsf{Max}_j(s_2) \leq s/2,$$

where Max_j is defined in Section 3.3.

In comparison to the generic construction of Fiat-Shamir signatures given in Section 3.2, qTESLA compresses signatures by employing the technique introduced by Bai and Galbraith [BG14]. When using this technique, the signature $(c, \mathbf{z}) \in \mathbb{T}_\kappa^n \times S_z^{k_1+1}$ is compressed to $(c, z_2) \in \mathbb{T}_\kappa^n \times S_z$, where the polynomial z_2 is given by $z_2 = y + s_2 c$, and y is a masking polynomial chosen by the signer from the uniform distribution over S_y.

Finally, we observe that the compression technique due to [BG14] does not affect the EUF-CMA-HRK security of our signature scheme with honestly re-randomizable public keys. More precisely, the security reduction depicted in Figure 3.4 remains the same, and only the simulation of the signing oracle $\mathsf{O}_{\mathsf{RSig}}$ requires to include an additional check to ensure the correctness of simulated signatures. More concretely, we extend the while-loop (*cf.* Line 54) of the algorithm RSig.Sim to include the last for-loop of qTESLA's signature generation algorithm (see [ABB+20, Algorithm 4]).

4

Reducing Restarts in Interactive Protocols

A canonical identification (CID) scheme [AABN02] is an interactive protocol that allows to prove the possession of some secret key in three moves. These moves are called the *commitment* (initiated by the prover), *challenge* (generated by the verifier), and *response*. The response allows, together with the commitment, to verify the authenticity of the prover while not leaking any information about the secret key (zero-knowledge). Lattice-based CID is a fundamental building block of many cryptographic constructions, *e.g.*, ordinary signature schemes [Lyu12, BG14, DKL+18, ABB+20], and even signature schemes with advanced functionalities such as ring signatures [BLO18, TSS+18], blind signatures [Rüc10], and multi-signatures [ES16].

In number-theoretic constructions like Schnorr's CID scheme [Sch91], the response already hides the secret key, since it is uniformly distributed over the underlying group. In the lattice setting however, the prover must carry out a security check before sending the response to the verifier. This security check is realized by the so-called rejection sampling procedure, which allows to make sure that the response does not leak any information about the secret key, and that it follows a target distribution. If this is not the case, the prover aborts and repeats the identification protocol until the response becomes independently distributed from the secret key. While this does not affect the efficiency of constructions with one rejection sampling procedure like (non-interactive) ordinary signatures, it has a significant negative impact on the efficiency of interactive protocols with multiple rejection sampling procedures. Consider a lattice-based protocol with $N > 1$ rejection sampling procedures, where each one is repeated x_i times on average, for $i = 1, \ldots, N$. Then, the total average number of restarts in such a protocol is given by $\prod_{i=1}^{N} x_i$. For instance, in the blind signature scheme introduced in [Rüc10] not only signers have to carry out rejection sampling and repeat the signing process, say x_1 times, until the secret key is concealed, but for maintaining the blindness property of the scheme even users have to apply rejection sampling, say x_2 times, and request a protocol restart in case of failure. This imposes a multiplicative number of restarts, *i.e.*, $x_1 \cdot x_2$, and an additional communication step due to the possibility of failures, *i.e.*, the user sends a proof of failure to the signer. This proof allows the signer to verify that the user was not able to obtain a valid blind signature.

This additional step increases the communication complexity required to generate blind signatures and forces the use of commitment schemes to retain security (see Chapter 5). Another example is the multi-signature scheme proposed in [ES16], which entails a number of restarts that grows in the number of users participating in the signing protocol.

Therefore, it is of great importance for lattice-based protocols with multiple rejection sampling procedures to mask secrets (or secret-related terms) such that restarts occur as little as possible while maintaining efficiency and security. This reduces the total amount of communication required to successfully complete the protocol, and can even improve its overall performance.

Contributions

In this chapter we show how to reduce the number of restarts, or even remove the restarts, inherent in lattice-based protocols by means of a tool that we call *tree of commitments*. A tree of commitments is an (unbalanced) binary hash tree, whose leaves are the hash values of ℓ commitments. These commitments are generated by the prover all at once, and then aggregated in one tree whose root represents the new commitment of the CID protocol. The new response consists of two elements. The first one is the response rp_k for which the rejection sampling procedure succeeds for the first time after k unsuccessful trials, and the second element is the authentication path of the leaf that has the index k in the tree, where $0 \leq k < \ell$. This leaf is the hash value of the commitment related to the response rp_k. More precisely, the rejection sampling procedure is iteratively applied on each potential response related to the ℓ commitments and on the secret-related term until it succeeds on some response rp_k. The number ℓ can be selected such that rejection sampling succeeds at least once with a probability that is chosen as desired. Hence, a large enough value of ℓ allows to completely remove the protocol restarts. Interestingly, even trees with small heights can be used so that restarts are triggered with a very small probability. For instance, when using a tree of commitments of height 3, a restart can occur with a probability of at most 2^{-10}. We also present further optimizations that can be exploited when using trees of commitments. This includes saving complete subtrees to use it in the next execution of the protocol.

We demonstrate the effectiveness of using our method in interactive protocols. More concretely, we construct in this chapter a new CID scheme using trees of commitments. It is based on a lattice-based CID scheme originally introduced by Lyubashevsky [Lyu09]. We show that the new scheme has a reduced communication complexity. In Chapter 5, we show the practical relevance of the tree of commitments technique by employing it in the design of two blind signature schemes called BLAZE$^+$ and BlindOR. In both schemes a user, interacting with the signer, constructs a tree of commitments such that with a probability very close to 1, *e.g.*, $1 - 2^{-80}$, the blindness property of the scheme is achieved without the need to request a protocol restart from the signer. This allows to safely remove the additional communication step of the signing protocol that is required to trigger protocol restarts, and hence to remove the proof of failures and the use of commitment schemes. We obtain a three-move protocol with reduced sizes of keys and signatures as well as communication complexity.

Organization

In Section 4.1 we recall the definition of canonical identification schemes. Afterwards, we describe in Section 4.2 the tree of commitments technique. Then, we present in Section 4.3 the canonical identification scheme that uses trees of commitments together with a proof of its security. Finally, we introduce in Section 4.4 further optimizations that can be exploited when using trees of commitments.

Publications

The contributions of this chapter are based on the publication [A3], which was presented at ACISP'20. This chapter extends the published contributions by providing an improvement and simplification of the security proof of our canonical identification scheme, which employs the tree of commitment technique.

4.1 Canonical Identification Schemes

In this section we review the definition and security of canonical identification schemes.

A canonical identification (CID) scheme [AABN02] is a three-move interactive protocol of the following form: A prover P initiates the protocol by sending a commitment cm to a verifier V. Upon receiving cm, V sends a random challenge ch to P. Afterwards, P computes a response rp and sends it to V. Together with cm, this response allows V to make a deterministic decision about P's authenticity. The tuple (cm, ch, rp) represents a protocol transcript. A formal definition follows.

Definition 4.1. A *canonical identification scheme* is a tuple of polynomial-time algorithms

$$\mathsf{CID} = (\mathsf{CID.PGen}, \mathsf{CID.KGen}, \mathsf{CID.P}, \mathsf{CID.V}, \mathsf{CID.Rec}),$$

where:

$\mathsf{CID.PGen}(1^\lambda)$ is a PPT parameter generation algorithm that, on input the security parameter λ, returns a set of public parameters pp, which implicitly contains λ in unary, *i.e.*, $pp \leftarrow\!\!\$\ \mathsf{CID.PGen}(1^\lambda)$.

$\mathsf{CID.KGen}(pp)$ is a PPT key generation algorithm that, on input a set of public parameters pp, returns a key pair (pk, sk), where pk is a public key and sk is a secret key, *i.e.*, $(pk, sk) \leftarrow\!\!\$\ \mathsf{CID.KGen}(pp)$.

$\mathsf{CID.P}(pp, sk)$ is an interactive algorithm that is called *prover* and consists of two algorithms $\mathsf{CID.P} = (\mathsf{CID.P}_1, \mathsf{CID.P}_2)$, where:

- $\mathsf{CID.P}_1$ is a PPT algorithm that, on input a set of public parameters pp and a secret key sk, returns a message cm, called the *commitment*, and a state st_P, *i.e.*, $(cm, st_\mathsf{P}) \leftarrow\!\!\$\ \mathsf{CID.P}_1(pp, sk)$.
- $\mathsf{CID.P}_2$ is a DPT algorithm that, on input a set of public parameters pp, a secret key sk, a state st_P, and a verifier message ch, returns a message rp, called the *response*, *i.e.*, $rp \leftarrow \mathsf{CID.P}_2(pp, sk, st_\mathsf{P}, ch)$. We write $rp = \perp$ to denote failure.

$\mathsf{CID.V}(pp, pk)$ is an interactive algorithm that is called *verifier* and consists of two algorithms $\mathsf{CID.V} = (\mathsf{CID.V}_1, \mathsf{CID.V}_2)$, where:

- $\mathsf{CID.V}_1$ is a PPT algorithm that, on input a set of public parameters pp, a public key pk, and a commitment cm, returns a message ch, called the *challenge*, and a state $st_{\mathsf{V}} = (cm, ch)$, *i.e.*, $(ch, st_{\mathsf{V}}) \leftarrow\!\!\$\ \mathsf{CID.V}_1(pp, pk, cm)$. The challenge ch is sampled from the uniform distribution over a finite set $\mathcal{C}$ called the *challenge space*. We assume that pp implicitly defines $\mathcal{C}$.
- $\mathsf{CID.V}_2$ is a DPT algorithm that, on input a set of public parameters pp, a public key pk, a state $st_{\mathsf{V}} = (cm, ch)$, and a response rp, returns a decision $b \in \{0, 1\}$, where $b = 1$ indicates that the transcript (cm, ch, rp) is accepted and $b = 0$ means it is rejected, *i.e.*, $b \leftarrow \mathsf{CID.V}_2(pp, pk, st_{\mathsf{V}}, rp)$.

$\mathsf{CID.Rec}(pp, pk, ch, rp)$ is a DPT algorithm called the *commitment recovering* algorithm. On input a set of public parameters pp, a public key pk, a challenge ch, and a response rp, the algorithm returns a commitment cm, *i.e.*, $cm \leftarrow \mathsf{CID.Rec}(pp, pk, ch, rp)$. We write $cm = \perp$ to denote failure.

Any CID scheme is required to satisfy the correctness property. It states that the algorithm $\mathsf{CID.V}_2$ validates honestly generated transcripts (cm, ch, rp) with probability at least $1 - \mathrm{corr}_{\mathsf{CID}}$, where $\mathrm{corr}_{\mathsf{CID}} \geq 0$ is called the *correctness error*. If $rp = \perp$, then the prover restarts the identification protocol. This case occurs with some probability bound, which depends on the set of the public parameters pp. Furthermore, if the algorithm $\mathsf{CID.Rec}$ returns a commitment $cm \neq \perp$ on input (pp, pk, ch, rp), then it is required that the transcript (cm, ch, rp) can be correctly verified with some probability that depends on pp.

The standard security notion of CID schemes is called *impersonation under the active or passive attack model*. In the active attack model, any adversary interacting with a prover as a verifier must not be able to extract any useful information (zero-knowledge). Passive attacks correspond to eavesdropping, *i.e.*, the adversary is in possession of transcripts generated by interactions between the real prover and the verifier. According to Abdalla *et al.* [AABN02], impersonation under active attacks is the model that is usually desired in practice for the purpose of identification. However, impersonation under passive attacks is the model that is tightly connected to the EUF-CMA security of the signature scheme that is derived from a CID scheme (see below).

Since our focus in this thesis is on advanced signature schemes, we omit the formal definitions of correctness and security of CID schemes as well as other properties such as honest-verifier zero-knowledge. For further details and formal definitions, we refer, for example, to [AABN02, KMP16, KLS18].

Finally, we recall how to build a signature scheme with security in the ROM from a CID scheme and a cryptographic hash function via the Fiat-Shamir transformation [FS87]. Let CID be a CID scheme and $\mathsf{H}\colon \{0,1\}^* \to \mathcal{C}$ be a cryptographic hash function modeled as a random oracle. In order to sign a message m, the signer acts as a prover of the CID scheme CID. Moreover, rather than obtaining the challenge ch from a verifier, ch is also generated by the signer. This is performed by calling the function H on input (cm, m). The signature is given by $sig = (ch, rp)$, which can be verified by first recovering a commitment cm via the algorithm $\mathsf{CID.Rec}$ on input (pp, pk, ch, rp), and then checking if $ch = \mathsf{H}(cm, m)$.

$\mathsf{Sig.PGen}(1^\lambda)$:

1: $pp \leftarrow\!\!\$\ \mathsf{CID.PGen}(1^\lambda)$
2: **return** pp

$\mathsf{Sig.KGen}(pp)$:

11: $(pk, sk) \leftarrow\!\!\$\ \mathsf{CID.KGen}(pp)$
12: **return** (pk, sk)

$\mathsf{Sig.Sign}(pp, sk, m)$:

21: $(cm, st_\mathsf{P}) \leftarrow\!\!\$\ \mathsf{CID.P}_1(pp, sk)$
22: $ch \leftarrow \mathsf{H}(cm, m)$
23: $rp \leftarrow \mathsf{CID.P}_2(pp, sk, st_\mathsf{P}, ch)$
24: **if** $rp = \bot$ **then**
25: **restart** Sig.Sign
26: $sig \leftarrow (ch, rp)$
27: **return** sig

$\mathsf{Sig.Verify}(pp, pk, m, sig)$:

31: **parse** $sig = (ch, rp)$
32: $cm \leftarrow \mathsf{CID.Rec}(pp, pk, ch, rp)$
33: **if** $cm = \bot$ **then**
34: **return** 0
35: **if** $ch \neq \mathsf{H}(cm, m)$ **then**
36: **return** 0
37: **return** 1

Figure 4.1: A formal description of the Fiat-Shamir transformation. It converts a canonical identification scheme CID into a signature scheme $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$ using a cryptographic hash function $\mathsf{H}\colon \{0,1\}^* \to \mathcal{C}$, where $\mathcal{C}$ is the challenge space of CID.

We denote this signature scheme by $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$ and formally describe its respective algorithms in Figure 4.1.

4.2 Trees of Commitments

In this section we show how the number of restarts inherent in lattice-based protocols can be reduced or even be removed at all.

First, we give a brief description of the CID scheme introduced in [Lyu09], which underlies many lattice-based constructions, in particular different types of signature schemes. Let $\mathbf{A}$ be a public matrix that is chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. Given the public key $pk = \mathbf{b} \in R_q^{k_1}$, the goal of the prover is to prove to a verifier the possession of a secret key $sk = \mathbf{s} \in R^{k_1+k_2}$, where $\mathbf{s}$ has small entries in R and satisfies $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. The commitment message is a vector $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y} \pmod q$, where $\mathbf{y} \in R^{k_1+k_2}$ is sampled from some distribution on R that outputs elements with small entries. Typically, this distribution is the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$ for some $\sigma > 0$, or the uniform distribution over a subset $S_y^{k_1+k_2}$ for some $y \in \mathbb{Z}_{>0}$. For a challenge c chosen randomly by the verifier from the uniform distribution over the challenge space $\mathbb{T}_\kappa^n$, the response is given by $\mathbf{z} = \mathbf{y} + \mathbf{s}c$, which is only sent to the verifier after checking that it follows a target distribution on R, which depends on the distribution of the vector $\mathbf{y}$. This check is performed by using the rejection sampling procedure, which makes sure that $\mathbf{y}$ masks the secret-related term $\mathbf{s}c$. If $\mathbf{z}$ does not follow the desired distribution, then the prover restarts the protocol. The verifier accepts if and only if $\|\mathbf{z}\|_p$ is bounded by some predefined value, and if $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$, where p and the bound of $\|\mathbf{z}\|_p$ depends on the distribution of $\mathbf{z}$.

Consider a lattice-based protocol with $N > 1$ rejection sampling procedures. Furthermore, assume that each procedure is repeated $x_i \geq 1$ times on average, where $i = 1, \ldots, N$. In this case, it is easy to deduce that the total average number of the restarts induced by these rejection sampling procedures is given by $\prod_{i=1}^{N} x_i$, and a large amount of communication is required to successfully complete the protocol. The main goal of this chapter is to reduce this number.

We first remark that one can sample the masking vectors $\mathbf{y}$ from a distribution that outputs elements with large enough entries such that rejection sampling succeeds after a fixed number of restarts M, *e.g.*, $M \leq 2$. This approach is already established in previous works as a trade-off between performance and sizes (see, *e.g.* [Lyu12, BG14, DKL+18]), but it does not solve the problem for protocols with multiple rejection sampling procedures. Some works, *e.g.*, the one due to Goldwasser *et al.* [GKPV10], suggest the so-called "noise flooding" technique, which uses masking terms sampled from distributions of a very large size such that rejection sampling always succeeds. However, the negative impact on the efficiency is tremendous as the sizes of the parameters become very large.

Our first attempt towards solving the aforementioned problem is the following: Rather than sampling one masking vector and repeating this process until rejection sampling accepts, the prover generates ℓ masking vectors $\mathbf{y}_0, \ldots, \mathbf{y}_{\ell-1}$ all at once, and then sends their related commitments $\mathbf{v}_0, \ldots, \mathbf{v}_{\ell-1}$ to the verifier, where $\mathbf{v}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod{q}$ for all $j \in \{0, \ldots, \ell - 1\}$. The response is then the vector $\mathbf{z}_k$ for which the rejection sampling procedure accepts for the first time after k unsuccessful trials, where $0 \leq k < \ell$. That is, the rejection sampling procedure is iteratively applied on the pairs $(\mathbf{z}_0, \mathbf{s}c), \ldots, (\mathbf{z}_k, \mathbf{s}c)$, where $\mathbf{z}_j = \mathbf{y}_j + \mathbf{s}c$, until it accepts for the first time at index k. If all masking vectors are consumed without success, then the protocol is restarted. Observe that this approach reduces the number of restarts, or even removes restarts for a suitable choice of the parameter ℓ. However, the amount of exchanged data grows in ℓ, because the first prover message consists of ℓ commitments. We can improve the communication complexity by using a cryptographic hash function F, and instead of sending the commitments $\mathbf{v}_0, \ldots, \mathbf{v}_{\ell-1}$, we simply send their hash values $\mathsf{F}(\mathbf{v}_0), \ldots, \mathsf{F}(\mathbf{v}_{\ell-1})$. But this is still not satisfactory, and in fact, we can do better to solve this issue as we explain next.

Our final solution is to use a new technique that we call *tree of commitments*. A tree of commitments is an (unbalanced) binary hash tree of height $h = \lceil \log(\ell) \rceil$. The leaves of this tree are the hash values $\mathsf{F}(\mathbf{v}_j)$, for all $j \in \{0, \ldots, \ell - 1\}$. The commitment message that the prover sends is simply the root *root* of this tree, and the response is the pair $(\mathbf{z}_k, \mathit{auth})$, where $\mathbf{z}_k$ is as in the first attempt and *auth* is the authentication path of the leaf with index k. Verifying the transcript $(\mathit{root}, c, (\mathbf{z}_k, \mathit{auth}))$ is carried out by checking that $\|\mathbf{z}_k\|_p$ has a specified bound, and that *root* is equal to the root of the tree associated to the leaf $\mathsf{F}([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod{q})$ and its given authentication path *auth*.

Note that trees of commitments realize the concept of allowing different commitments to belong to one challenge in an aggregated form and only the valid response and its related commitment will be revealed. The masking vectors for which rejection sampling rejects, *i.e.*, $\mathbf{y}_0, \ldots, \mathbf{y}_{k-1}$, together with the vectors that were not consumed, *i.e.*, $\mathbf{y}_{k+1}, \ldots, \mathbf{y}_{\ell-1}$, remain hidden and will never be revealed. Using a tree of commitments not only reduces the number of restarts, it also reduces the communication complexity. Moreover, it can improve the performance of interactive protocols with multiple rejection sampling procedures.

In the following we formally define trees of commitments.

Definition 4.2. Let $cm_0, \ldots, cm_{\ell-1}$ be commitments to ℓ masking terms $msk_0, \ldots, msk_{\ell-1}$. Furthermore, let $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ be a cryptographic hash function, where $\ell_\mathsf{F} \geq 2\lambda$. A *tree of commitments* is a tuple of algorithms

$$\mathsf{ToC} = (\mathsf{HashTree}, \mathsf{BuildAuth}, \mathsf{RootCalc}),$$

where:

$\mathsf{HashTree}(cm_0, \ldots, cm_{\ell-1})$ is a DPT algorithm that, on input $cm_0, \ldots, cm_{\ell-1}$, returns a pair $(root, tree)$, where $root$ is the root of the (unbalanced) binary hash tree of height $h = \lceil \log(\ell) \rceil$ whose leaves are the hash values $\mathsf{F}(cm_j)$, for all $j \in \{0 \ldots, \ell - 1\}$, and $tree$ is the sequence that consists of all leaves and inner nodes of the tree, *i.e.*, $(root, tree) \leftarrow \mathsf{HashTree}(cm_0, \ldots, cm_{\ell-1})$.

$\mathsf{BuildAuth}(k, tree, h)$ is a DPT algorithm that, on input an index k, a sequence of nodes $tree$, and a height h, returns the authentication path $auth = (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$ of the index k, where $0 \leq k < \ell$, $\mathbf{a}_i \in \{0,1\}^{\ell_\mathsf{F}}$, and $0 \leq i < h$, *i.e.*, $auth \leftarrow \mathsf{BuildAuth}(k, tree, h)$. This output represents the authentication path of the response rp_k for which the rejection sampling procedure accepts using the masking term msk_k.

$\mathsf{RootCalc}(cm, auth)$ is a DPT algorithm that, on input a commitment cm and its authentication path $auth$, where $cm \in \{cm_0, \ldots, cm_{\ell-1}\}$ and $auth = (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$, returns the root of the hash tree that includes the leaf $\mathsf{F}(cm)$ at index k and the inner nodes $\mathbf{a}_0, \ldots, \mathbf{a}_{h-1}$, *i.e.*, $root \leftarrow \mathsf{RootCalc}(cm, auth)$.

In Figure 4.2 we formally give one possible implementation of the algorithms that build a tree of commitments ToC, *i.e.*, HashTree, BuildAuth, and RootCalc. This implementation uses the following notation: The leaves of the tree are the hash values of commitments $cm_0, \ldots, cm_{\ell-1}$, *i.e.*, $v_0[j] = \mathsf{F}(cm_j)$ for $0 \leq j < \ell$. The inner nodes of height i are denoted by $v_i[j]$, where $0 < i < h$ and $0 \leq j < 2^{h-i}$. They are typically computed as $v_i[j] = \mathsf{F}(v_{i-1}[2j], v_{i-1}[2j+1])$. The root is the only node of height h, *i.e.*, $v_h[0] = root$. It constitutes an aggregated commitment to the ℓ masking terms msk_j. Figure 4.3 illustrates a tree of commitments of height $h = 3$.

4.3 Canonical Identification Using Trees of Commitments

In this section we formally present a canonical identification scheme that employs the tree of commitments technique in order to reduce the number of protocol restarts or even remove the restarts induced by applying the rejection sampling procedure.

The scheme was briefly explained in Section 4.2. It is a variant of the one given in [Lyu09]. The parameter generation algorithm $\mathsf{CID.PGen}(1^\lambda)$ generates a set of public parameters given by

$$pp = \left(1^\lambda, n, k_1, k_2, q, \kappa, \sigma', \ell, \sigma, M, h, B_z, \ell_\mathsf{F}, \mathbf{A}\right), \tag{4.1}$$

where $\mathbf{A}$ is a matrix chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. The description of the remaining parameters is given in Table 4.1. In the following we give a

$\mathsf{HashTree}(cm_0, \ldots, cm_{\ell-1})$:

1: $h \leftarrow \lceil \log(\ell) \rceil$
2: $tree \leftarrow \varnothing$
3: **for** $j = 0$ **to** $\ell - 1$ **do**
4: $\quad v_0[j] \leftarrow \mathsf{F}(cm_j)$
5: $\quad tree \leftarrow tree \cup \{v_0[j]\}$
6: **for** $j = \ell$ **to** $2^h - 1$ **do** // fill remaining leaves
7: $\quad v_0[j] \leftarrow \mathsf{F}(j)$
8: $\quad tree \leftarrow tree \cup \{v_0[j]\}$
9: **for** $i = 1$ **to** $h - 1$ **do**
10: $\quad$ **for** $j = 0$ **to** $2^{h-1-i} - 1$ **do**
11: $\quad\quad v_i[j] = \mathsf{F}(v_{i-1}[2j], v_{i-1}[2j+1])$
12: $\quad\quad tree \leftarrow tree \cup \{v_i[j]\}$
13: $root \leftarrow \mathsf{F}(v_{h-1}[0], v_{h-1}[1])$
14: **return** $(root, tree)$

$\mathsf{BuildAuth}(k, tree, h)$:

21: **parse** $tree = (v_i[j])_{i,j}$ // $0 \le i < h \wedge 0 \le j < 2^{h-i}$
22: **for** $i = 0$ **to** $h - 1$ **do**
23: $\quad s \leftarrow \lfloor k/2^i \rfloor$
24: $\quad b \leftarrow s \bmod 2$
25: $\quad$ **if** $b = 1$ **then**
26: $\quad\quad \mathbf{a}_i \leftarrow v_i[s-1]$
27: $\quad$ **else**
28: $\quad\quad \mathbf{a}_i \leftarrow v_i[s+1]$
29: $auth \leftarrow (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$
30: **return** $auth$

$\mathsf{RootCalc}(cm, auth)$:

31: **parse** $auth = (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$
32: $\mathbf{b}_0 \leftarrow \mathsf{F}(cm)$
33: **for** $i = 1$ **to** h **do**
34: $\quad s \leftarrow \lfloor k/2^{i-1} \rfloor$
35: $\quad b \leftarrow s \bmod 2$
36: $\quad$ **if** $b = 1$ **then**
37: $\quad\quad \mathbf{b}_i \leftarrow \mathsf{F}(\mathbf{a}_{i-1}, \mathbf{b}_{i-1})$
38: $\quad$ **else**
39: $\quad\quad \mathbf{b}_i \leftarrow \mathsf{F}(\mathbf{b}_{i-1}, \mathbf{a}_{i-1})$
40: $root \leftarrow \mathbf{b}_h$
41: **return** $root$

Figure 4.2: A description of the algorithms HashTree, BuildAuth, and RootCalc, which define a tree of commitments ToC (*cf.* Definition 4.2).

description of the remaining algorithms CID.KGen, CID.P, CID.V, and CID.Rec, which are formalized in Figure 4.4. The secret key $sk = \mathbf{s}$ is chosen from the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma'}$, while the masking vectors $\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\ell-1)}$ that are sampled by the prover to hide the secret-related term $\mathbf{s}c$ are distributed according to $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$, where $c \in \mathbb{T}^n_\kappa$ is the challenge generated by the verifier. If Rej accepts on input $(pp, \mathbf{z}, \mathbf{s}c)$ for some masking vector $\mathbf{y}^{(k)}$, then $\mathbf{z}$ follows the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$. We point out that the index k of the masking vectors is selected from the uniform distribution over the set $T \subseteq \{0, \ldots, \ell - 1\}$ (see the algorithm IterateRej in Figure 4.4). The random choice of the index k is for security purposes. It ensures that when simulating the prover, the simulator returns a

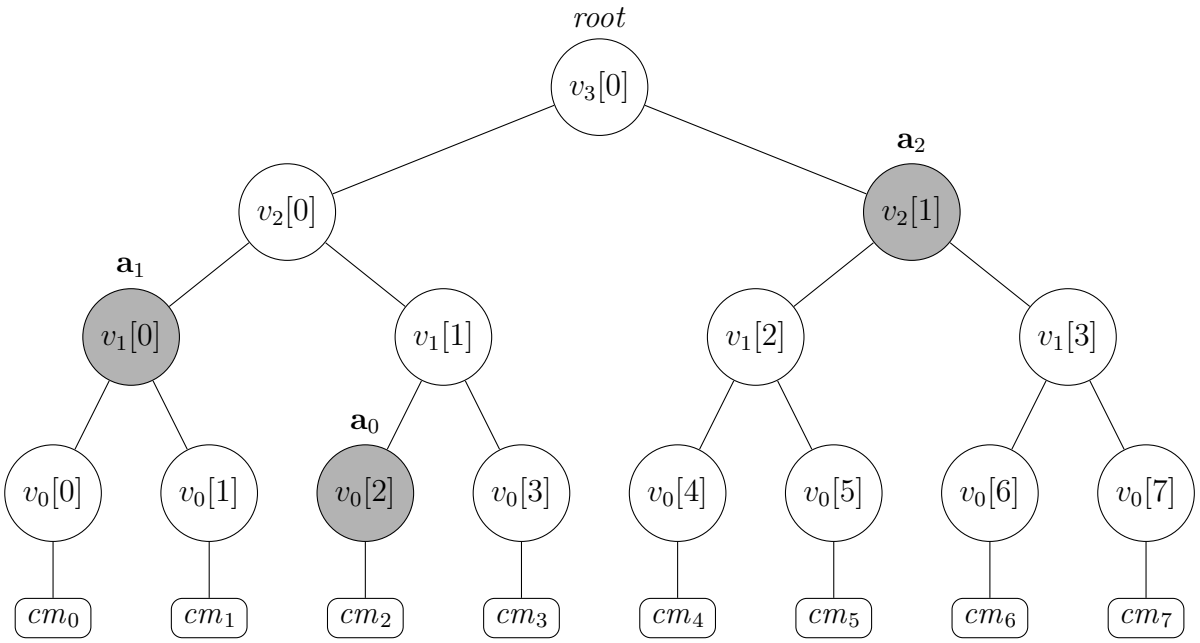

Figure 4.3: A tree of commitments of height $h = 3$ and $\ell = 8$ commitments. Assume that the rejection sampling procedure accepts for the first time after 3 unsuccessful trials, *i.e.*, at index $k = 3$, where $0 \leq k < \ell$. Then, the gray colored nodes together with the integer 3 represent the authentication path of index $k = 3$, *i.e.*, $auth = (3, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2)$. This authentication path together with the commitment cm_3 are required to compute the root of the tree *root*.

response with the same probability as the real prover. We note that it is also possible to use other distributions for the secret key and the masking vectors in both algorithms $\mathsf{CID.KGen}$ and $\mathsf{CID.P}$, *e.g.*, the uniform distribution over a small subset of $R^{k_1+k_2}$.

The correctness of the CID scheme depicted in Figure 4.4 is standard and directly follows from the base CID scheme introduced in [Lyu09] together with the correctness of the algorithms $\mathsf{HashTree}, \mathsf{BuildAuth}$, and $\mathsf{RootCalc}$. A possible approach to establish its security is to rely on the equivalence result shown by Abdalla *et al.* [AABN02]. It states that a CID scheme CID is secure against impersonation under passive attacks if and only if the signature scheme $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$ is existentially unforgeable under adaptive chosen-message attacks (EUF-CMA) in the ROM, where Sig is obtained by applying the Fiat-Shamir transformation (*cf.* Figure 4.1) on the scheme CID and the cryptographic hash function H. We follow this approach, since it is related to signature schemes.

Theorem 4.3. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be two cryptographic hash functions modeled as random oracles. Let* $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$*, where* CID *is the CID scheme depicted in Figure 4.4. Then,* Sig *is* $(t, q_{\mathsf{Sign}}, q_\mathsf{F}, q_\mathsf{H}, \varepsilon)$-EUF-CMA *w.r.t.* $pp \in \mathsf{Sig.PGen}(1^\lambda)$ *in the ROM if* MLWE^2 *is* (t', ε')*-hard w.r.t.* $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$ *and* MSIS^2 *is* (t'', ε'')*-hard w.r.t.* $pp'' = (n, k_1, k_2 + 1, q, \beta)$*, where* $\beta = 2\sqrt{B_z^2 + \kappa}$ *and*

$$t' \approx t \;\wedge\; \varepsilon' \approx \varepsilon \;\wedge\; t'' \approx 2t \;\wedge\; \varepsilon'' \approx \left(\varepsilon - \frac{q_\mathsf{F}^2 + q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{1}{|\mathbb{T}_\kappa^n|}\right) \cdot \left(\frac{\varepsilon - \frac{q_\mathsf{F}^2+q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{1}{|\mathbb{T}_\kappa^n|}}{q_\mathsf{H}\ell} - \frac{1}{|\mathbb{T}_\kappa^n|}\right).$$

Table 4.1: A review of the parameters of the canonical identification scheme given in Figure 4.4.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 1 \pmod{2n}$
σ'	Standard deviation of $\mathbf{s}$	$\sigma' > 0$
κ	Specifies the set $\mathbb{T}_\kappa^n$	$2^\kappa \binom{n}{\kappa} \geq 2^\lambda$
σ	Standard deviation of $\mathbf{z}$	$\sigma = \alpha\|\mathbf{s}c\| = \alpha\sqrt{\kappa}B_s$, $\alpha > 0$ (Lemma 2.8)
ℓ	No. masking vectors $\mathbf{y}^{(k)}$	$\ell \in \mathbb{N}_{>1}$
h	Tree height	$h = \lceil \log(\ell) \rceil$
M	No. restarts of $\mathsf{CID.P}$	$M = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$ (Lemma 2.8)
B_s	Bound of $\|\mathbf{s}\|$	$B_s = \eta\sigma'\sqrt{(k_1+k_2)n}$, $\eta > 0$ (Lemma 2.6)
B_z	Bound of $\|\mathbf{z}\|$	$B_z = \eta\sigma\sqrt{(k_1+k_2)n}$ (Lemma 2.6)
ℓ_F	Output length of F	$\ell_\mathsf{F} \geq 2\lambda$

Proof. Let A^* be an adversary that is able to forge signatures under the signature scheme $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$. That is, A^* runs in time t and is able to win the experiment $\mathbf{Exp}^{\text{EUF-CMA}}_{\mathsf{Sig},\mathsf{A}^*}$ given in Definition 2.15 with probability ε. We first remark that the hardness of MLWE^2 w.r.t. $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$ is required to protect against key recovery attacks, whose success directly allows to forge signatures. Therefore, in what follows we assume the hardness of MLWE^2 w.r.t. pp' and construct a reduction algorithm R that solves MSIS^2 w.r.t. $pp'' = (n, k_1, k_2 + 1, q, \beta)$, where $\beta = 2\sqrt{B_z^2 + \kappa}$.

Given pp'' and a uniformly random matrix $\mathbf{A}' \in R_q^{k_1 \times (k_2+1)}$, the reduction R writes $\mathbf{A}' = [\mathbf{A} \mid \mathbf{b}] \in R_q^{k_1 \times k_2} \times R_q^{k_1}$. Afterwards, R generates the remaining public parameters of $\mathsf{Sig} = [\mathsf{CID}, \mathsf{H}]$ as described in Table 4.1 to obtain a set of public parameters pp of the form given in Equation (4.1). Then, R sets $pk = \mathbf{b}$, and runs A^* on input (pp, pk). The random oracle queries and signing queries that are made by A^* are answered by R as follows:

Random oracle query. The reduction algorithm R maintains a list L_H, which is initialized by the empty set. It stores pairs of queries to the random oracle O_H and their answers. If O_H was previously queried by A^* on some input, then R looks up its entry in L_H and returns its answer $c \in \mathbb{T}_\kappa^n$. Otherwise, R selects a new polynomial c according to the uniform distribution over $\mathbb{T}_\kappa^n$ and updates the list L_H. Furthermore, R initializes an empty list L_F in order to store pairs of queries to the random oracle O_F and their answers. The queries to O_F are answered by R in a way that excludes collisions and chains. Excluding collisions rules out queries $\mathbf{x} \neq \mathbf{x}'$ such that $\mathsf{O}_\mathsf{F}(\mathbf{x}) = \mathsf{O}_\mathsf{F}(\mathbf{x}')$, while excluding chains guarantees that the query $\mathsf{O}_\mathsf{F}(\mathsf{O}_\mathsf{F}(\mathbf{x}))$ will not be made before the query $\mathsf{O}_\mathsf{F}(\mathbf{x})$. This ensures that each node output by the algorithm $\mathsf{HashTree}$ has a unique preimage, and prevents spanning hash trees with cycles. Simulating O_F this way is within statistical distance of at most $\frac{q_\mathsf{F}^2 + q_\mathsf{F}}{2^{\ell_\mathsf{F}}}$ from an oracle that allows the existence of collisions and chains. The description of O_H and O_F is given in Figure 4.5.

$\mathsf{CID.KGen}(pp)$:

1: $\mathbf{s} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma'}^{k_1+k_2}$
2: $\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
3: $pk \leftarrow \mathbf{b}$
4: $sk \leftarrow \mathbf{s}$
5: **return** (pk, sk)

$\mathsf{CID.P}_1(pp, sk)$:

11: **for** $k = 0$ **to** $\ell - 1$ **do**
12: $\quad \mathbf{y}^{(k)} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$
13: $\quad \mathbf{v}^{(k)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}^{(k)} \pmod q$
14: $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\ell-1)})$
15: $\mathbf{y} \leftarrow (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\ell-1)})$
16: $st_{\mathsf{P}} \leftarrow (\mathbf{y}, tree)$
17: **return** $(root, st_{\mathsf{P}})$

$\mathsf{CID.V}_1(pp, pk, root)$:

21: $c \leftarrow\!\!\$\ \mathbb{T}_\kappa^n$
22: $st_{\mathsf{V}} \leftarrow (root, c)$
23: **return** (c, st_{V})

$\mathsf{CID.P}_2(pp, sk, st_{\mathsf{P}}, c)$:

31: **parse** $sk = \mathbf{s}$
32: **parse** $st_{\mathsf{P}} = (\mathbf{y}, tree)$
33: $(\mathbf{z}, k) \leftarrow \mathsf{IterateRej}(pp, \mathbf{y}, \mathbf{s}c)$
34: **if** $(\mathbf{z}, k) = (\perp, \perp)$ **then**
35: $\quad$ **return** $(\perp, \perp)$
36: $auth \leftarrow \mathsf{BuildAuth}(k, tree, h)$
37: **return** $(\mathbf{z}, auth)$

$\mathsf{IterateRej}(pp, \mathbf{y}, \mathbf{s}c)$:

41: **parse** $\mathbf{y} = (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\ell-1)})$
42: $T \leftarrow \{0, \ldots, \ell - 1\}$
43: **while** $T \neq \emptyset$ **do**
44: $\quad k \leftarrow\!\!\$\ T$
45: $\quad \mathbf{z} \leftarrow \mathbf{y}^{(k)} + \mathbf{s}c$
46: $\quad$ **if** $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{s}c) = 1$ **then**
47: $\quad\quad$ **return** $(\mathbf{z}, k)$
48: $\quad T \leftarrow T \setminus \{k\}$
49: **return** $(\perp, \perp)$

$\mathsf{CID.V}_2(pp, pk, st_{\mathsf{V}}, (\mathbf{z}, auth))$:

51: **parse** $st_{\mathsf{V}} = (root, c)$
52: $root' \leftarrow \mathsf{CID.Rec}(pp, pk, c, (\mathbf{z}, auth))$
53: **if** $root' \neq root$ **then**
54: $\quad$ **return** 0
55: **return** 1

$\mathsf{CID.Rec}(pp, pk, c, (\mathbf{z}, auth))$:

61: **if** $\|\mathbf{z}\| > B_z$ **then**
62: $\quad$ **return** $\perp$
63: **parse** $pk = \mathbf{b}$
64: $\mathbf{w} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$
65: $root \leftarrow \mathsf{RootCalc}(\mathbf{w}, auth)$
66: **return** $root$

Figure 4.4: A formal description of a canonical identification scheme CID that uses a tree of commitments $\mathsf{ToC} = (\mathsf{HashTree}, \mathsf{BuildAuth}, \mathsf{RootCalc})$. The parameter generation algorithm $\mathsf{CID.PGen}$ is explained in the text. The prover restarts the protocol if $\mathsf{CID.P}_2$ returns $(\perp, \perp)$.

Signature query. Upon receiving a signature query from A^*, R runs the algorithm Sig.Sim depicted in Figure 4.5 in order to generate a signature and sends it back to A^*. Furthermore, R updates both lists L_{F} and L_{H} accordingly. By Lemma 2.8, simulating the computation of the vector $\mathbf{z}$, *i.e.*, without a secret key, is statistically indistinguishable from generating it as in a real execution of the signing algorithm.

$\mathsf{Sig.Sim}(pp, \mathbf{b}, m)$:

1: **return** $\perp$ with probability $1 - 1/M$
2: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
3: $c \leftarrow\!\!\$\ \mathbb{T}_\kappa^n$
4: $k \leftarrow\!\!\$\ \{0, \ldots, \ell - 1\}$
5: $\mathbf{z} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$
6: $\mathbf{v}^{(k)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$
7: **for** $i = 0$ **to** $\ell - 1$ **do**
8: **if** $i = k$ **then**
9: **continue**
10: $\mathbf{y}^{(i)} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$
11: $\mathbf{v}^{(i)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}^{(i)} \pmod q$
12: $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\ell-1)})$
13: $\mathsf{O_H}(root, m) \leftarrow c$
14: $\mathrm{L_H} \leftarrow \mathrm{L_H} \cup \{((root, m), \mathsf{O_H}(root, m))\}$
15: $auth \leftarrow \mathsf{BuildAuth}(k, tree, h)$
16: $sig \leftarrow (c, \mathbf{z}, auth)$
17: **return** sig with probability $1/M$

$\mathsf{O_F}(\mathbf{x})$:

21: **if** $\exists(\mathbf{x}, \mathsf{O_F}(\mathbf{x})) \in \mathrm{L_F}$ **then**
22: **return** $\mathsf{O_F}(\mathbf{x})$
23: $\mathsf{O_F}(\mathbf{x}) \leftarrow\!\!\$\ \{0,1\}^{\ell_\mathsf{F}}$
24: **if** $\exists(\mathbf{x}', \mathsf{O_F}(\mathbf{x}')) \in \mathrm{L_F}$: $(\mathbf{x} \neq \mathbf{x}') \wedge (\mathsf{O_F}(\mathbf{x}) = \mathsf{O_F}(\mathbf{x}'))$ **then**
25: **return** $\perp$
26: **if** $\exists(\mathbf{y}, \mathsf{O_F}(\mathbf{y})) \in \mathrm{L_F} : \mathbf{y} = \mathsf{O_F}(\mathbf{x})$ **then**
27: **return** $\perp$
28: $\mathrm{L_F} \leftarrow \mathrm{L_F} \cup \{(\mathbf{x}, \mathsf{O_F}(\mathbf{x}))\}$
29: **return** $\mathsf{O_F}(\mathbf{x})$

$\mathsf{O_H}(root, m)$:

31: **if** $\exists((root, m), \mathsf{O_H}(root, m)) \in \mathrm{L_H}$ **then**
32: **return** $\mathsf{O_H}(root, m)$
33: $\mathsf{O_H}(root, m) \leftarrow\!\!\$\ \mathbb{T}_\kappa^n$
34: $\mathrm{L_H} \leftarrow \mathrm{L_H} \cup \{((root, m), \mathsf{O_H}(root, m))\}$
35: **return** $\mathsf{O_H}(root, m)$

Figure 4.5: The algorithms that are used in the proof of Theorem 4.3. The algorithm $\mathsf{Sig.Sim}$ simulates the signing queries that are made by the adversary A^*. Random oracle queries made by A^* are answered as described in $\mathsf{O_F}$ and $\mathsf{O_H}$.

By the hardness of MLWE^2 w.r.t. pp', both the matrix $\mathbf{A}$ and any public key generated by the algorithm $\mathsf{Sig.KGen}$ are indistinguishable from the uniform distribution over $R_q^{k_1 \times k_2} \times R_q^{k_1}$. Therefore, A^* returns a valid signature $sig = (c, \mathbf{z}, auth)$ on a message m with probability ε. If $\mathsf{O_H}$ was not programmed or queried during the invocation of A^*, then A^* produces a $c \in \mathbb{T}_\kappa^n$ that validates correctly with probability $1/|\mathbb{T}_\kappa^n|$. Therefore, the probability that A^* succeeds in a forgery $sig = (c, \mathbf{z}, auth)$, where c corresponds to one of the random oracle queries made by A^* is at least $\varepsilon - 1/|\mathbb{T}_\kappa^n|$. Then, one of the following two cases applies: c was programmed during a signing query or it was an answer to a hash query to $\mathsf{O_H}$. Suppose that c was programmed during a signing query made by A^* to sign a message m', *i.e.*, $c = \mathsf{O_H}(root', m')$. Then, we have

$$\mathsf{O_H}(root, m) = c = \mathsf{O_H}(root', m').$$

If $m \neq m'$ or $root \neq root'$, then a second preimage of c has been found by A^*. This occurs with probability at most $1/|\mathbb{T}_\kappa^n|$. Therefore, we may assume that $m = m'$ and $root = root'$. In this case, A^* was not able to produce a forgery because $m = m'$ (*cf.* Definition 2.15). Hence, we may assume in the following that c was an answer to a hash query to $\mathsf{O_H}$ that was not part of a signing query. In this case, R records the pair $(m, sig = (c, \mathbf{z}, auth))$ and invokes A^* again with the same random tape and the random

oracle queries $\{c_1, \ldots, c_{j-1}, c'_j, \ldots, c'_{q_\mathsf{H}}\}$, where $c'_j, \ldots, c'_{q_\mathsf{H}} \in \mathbb{T}_\kappa^n$ are freshly generated by R. By the forking lemma (Lemma 2.18), the forger A^* returns a signature $sig' = (c', \mathbf{z}', auth')$ on a message m' with the probability ε'' that is given in the theorem statement such that

$$(c \neq c') \wedge (root = root') \wedge (m = m') \wedge (k = k'),$$

where $k, k' \in \{0, \ldots, \ell - 1\}$ are the indices included in $auth, auth'$, respectively. Furthermore, answering the hash queries to the random oracle O_F as described in Figure 4.5 ensures that both $auth, auth'$ include the same sequence of hash values. Therefore, we have $auth = auth'$ and $\mathbf{w} = \mathbf{w}'$, where

$$\mathbf{w} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q \text{ and } \mathbf{w}' = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}' - \mathbf{b}c' \pmod q.$$

This implies that

$$[\mathbf{I}_{k_1} \mid \mathbf{A}'] \cdot \mathbf{x} = \mathbf{0} \pmod q,$$

where $\mathbf{x} = [\mathbf{z} - \mathbf{z}', c' - c]^\top$. Since $c \neq c'$, we have $\mathbf{x} \neq \mathbf{0}$ and $\|\mathbf{x}\| \leq 2\sqrt{B_z^2 + \kappa} = \beta$. Hence, R returns $\mathbf{x}$ as a non-zero solution to MSIS^2 w.r.t. $pp'' = (n, k_1, k_2 + 1, q, \beta)$. □

4.4 Further Optimizations

In this section we present further optimizations that can be exploited when using the tree of commitments technique.

The first optimization addresses the question of how to select the number of masking vectors ℓ. Assume that the rejection sampling procedure accepts with probability $1/M$ when using only one masking vector, where $M \geq 1$ is the expected number of restarts. Then, when sampling ℓ masking vectors at once, a restart occurs with probability $(1 - 1/M)^\ell$. Therefore, we fix some desired aborting probability δ. This is the probability that the rejection sampling procedure does not accept using all chosen making vectors. Then, we choose ℓ such that the rejection sampling procedure accepts using at least one of the ℓ masking vectors, *i.e.*, we select the parameter ℓ such that

$$(1 - 1/M)^\ell \leq \delta.$$

Note that in order to completely eliminate restarts, ℓ has to be selected large enough such that the above probability is approximately zero, *e.g.*, $(1 - 1/M)^\ell \leq 2^{-80}$.

Let us consider an illustrative example. Let $\mathbf{v}$ be the secret (or secret-related) vector that has to be masked, and $\|\mathbf{v}\| \leq T$ for some $T > 0$. Suppose that we use masking vectors with entries sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}$, where $\sigma = \alpha T$ and $\alpha = 11$. Then, when sampling only one masking vector in a protocol run, Lemma 2.8 implies that the success probability of the rejection sampling procedure is given by $(1 - 2^{-100})/M \approx 0.33$, where $\exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right) \leq M$, and the expected number of restarts that is required to return a response $rp \neq \perp$ is given by $M \approx 3$. The total amount of communication includes three commitments and a response, *i.e.*, three vectors from $R_q^{k_1}$ and a Gaussian vector with standard deviation $\sigma = 11T$. However, by setting $\alpha = 22$ we need only $\ell = 8$ masking vectors in order to hide $\mathbf{v}$ with probability at least $1 - 2^{-10} \approx 0.999$. This means we need a

tree of commitments of height $h = 3$, which is a very small tree. Regarding communication complexity, both the commitment and response consist of only four hash values, a positive integer $k < 3$, and a Gaussian vector $\mathbf{z}$ with standard deviation $\sigma = 22T$, *i.e.*, $cm = root$ and $rp = (\mathbf{z}, auth = (k, \mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2))$. This is much better compared to the case where only one masking vector is used and $\alpha = 11$. Note that in order to hide $\mathbf{v}$ with probability $1 - 2^{-10}$ using a single masking vector, *i.e.*, without using trees of commitments, we need to set $\alpha = 246$, which induces a much larger response and requires larger modulus q to retain security, and hence increases the size of the commitments.

In addition to the selection of the parameter ℓ, we can improve the performance of protocols employing the tree of commitments technique as follows: We can save the masking vectors that were sampled, but not consumed in a protocol run, and use them later in a subsequent execution of the protocol. This reduces the number of new masking vectors to be sampled as well as the number of hash computations and multiplications modulo q, which are required to compute the commitments. For instance, consider the tree of commitments depicted in Figure 4.3, where the rejection sampling procedure accepts for the first time at index $k = 3$, *i.e.*, after three unsuccessful trials. For the next protocol run we can simply reuse the whole right subtree of height 2, whose root is the node $v_2[1]$. In this case, we only need to compute a new left subtree of height 2 and combine both subtrees to obtain a new tree of height $h = 3$. This saves seven hash computations in addition to sampling four new masking vectors and computing their commitments.

Finally, by following the standard of the hash-based signature scheme XMSS [HBG+18], we can generate trees of commitments using the so-called *randomized hashing* technique. This technique allows to reduce the security requirement of the hash function F, *i.e.*, it requires F to be only second preimage resistant rather than collision resistant. Consequently, we can set the output length of F to $\ell_{\mathsf{F}} \geq \lambda$ rather than $\ell_{\mathsf{F}} \geq 2\lambda$, assuming that λ bits of classical security is desired. This allows to save space and further reduce the communication complexity, since the root and the authentication path are reduced to one half of their original sizes.

5 Blind Signature Schemes

A blind signature scheme is an interactive protocol that allows a user, holding a message m, to generate a blind signature on m under the signer's secret key. The scheme is required to satisfy two security properties called *blindness* and *one-more unforgeability*. Informally, the first property indicates that the signer gets no information about m during the interaction with the user, while the second one ensures that the user cannot generate any valid blind signature without interacting with the signer. Blind signatures were first introduced by Chaum [Cha82] in the context of an anonymous e-cash system. They have since become a fundamental building block in privacy-preserving cryptography, and have been standardized as *ISO/IEC 18370*. One of the main applications of blind signatures is anonymous credentials [BL13a], which allow users to privately obtain and prove possession of credentials while revealing as little about themselves as possible. This complies, for example, with the European privacy standards [PotEU09] and the National Strategy for Trusted Identities in Cyberspace [Coo10]. An established real-life use case of blind signatures in the context of anonymous credentials is the U-Prove technology [Paq13] designed by Microsoft. U-Prove is one of the technologies, to which the Microsoft's Open Specification Promise [Cor07] applies. It is deployed, for example, in smart card devices produced by Gemalto [Com11] – a leading digital security company – in order to enhance privacy. Further applications of blind signatures include e-voting [KKS17] and blockchain protocols [HBG16].

Currently, the real-world applications employing blind signatures rely on classical blind signature schemes, where the security is based on the hardness of number-theoretic assumptions. For instance, the U-Prove protocol utilizes blind signature schemes that are secure as long as computing discrete logarithms in certain cyclic groups is hard [Paq13]. As it is meanwhile known, number-theoretic assumptions are not secure for the long-term, especially when taking into account the recent developments of quantum computers. Therefore, classical blind signature schemes have to be replaced with constructions that are secure, or at least conjectured to be secure, under quantum computer attacks. More concretely, we need post-quantum candidates of blind signature schemes in order to further preserve privacy standards and anonymity considerations. While such proposals do exist, *e.g.* [Rüc10, PSM17, HKLN20], they cannot be deployed in practical applications due to their poor performance as well as large sizes of keys and signatures.

Contributions

In this chapter we present three new blind signature schemes called BLAZE, BLAZE$^+$, and BlindOR. They provide statistical blindness and computational one-more unforgeability in the ROM assuming the hardness of both MLWE and MSIS. We show that our schemes are efficient and can be used in practical applications. In particular, we propose concrete parameters targeting 128 bits of security, and provide the corresponding sizes of keys and signatures as well as the communication cost required to generate valid blind signatures. Furthermore, we prove that almost all two-round lattice-based constructions of blind signatures found in the literature are insecure. More precisely, we show for the proposal given in [ZTZ+17] how the secret key can simply be recovered already after two executions of its signing protocol. Moreover, we show for the schemes proposed in [CCT+11, ZM14, ZH16, GHWX16, GHW+17] that any user is able to solve the underlying lattice problem in just one execution of the signing protocol.

Our first scheme BLAZE improves upon the first lattice-based scheme introduced by Rückert [Rüc10], which is based on the canonical identification scheme given in [Lyu08]. We observed that the main reason for its inefficiency is the first security check (rejection sampling procedure) that is carried out at the first user stage. Our scheme removes this check via a new technique that we call *partitioning and permutation*, which may be of independent interest. More precisely, the signing protocol of BLAZE consists of four moves between the signer and user, and it involves two security checks as opposed to the scheme given in [Rüc10], which involves three security checks. At the first user stage, BLAZE utilizes the partitioning and permutation technique so that there is no need to carry out any security check and any potential restart in order to blind the hash value that is output by the underlying cryptographic hash function. Moreover, this technique allows to use hash functions that output hash values from the set $\mathbb{T}_\kappa^n = \{c \in R_q = \mathbb{Z}_q[X]/\langle X^n + 1\rangle \mid (\|f\|_\infty = 1) \wedge (\|f\|_1 = \kappa)\}$ instead of polynomials in R_q with unspecified Hamming weight. At a high level, this technique works as follows: Let $c \in \mathbb{T}_\kappa^n$ be the hash value that has to be blinded. Rather than adding a large enough random masking term to c, we split c into partitions of the form $\pm x^i$, where $0 \leq i < n$. Then, we permute each partition by multiplying it with a random masking monomial of the same form. We prove that the resulting elements are independently distributed from c so that there is no need to apply any security check to ensure blindness. This significantly reduces the sizes of the parameters that have to be selected for the scheme, since blinding c does not require the selection of large masking elements anymore. This in turn reduces the size of keys and signatures and speeds up the signing process, but at the cost of increasing the communication complexity. This is because the signer must send a commitment message for each partition of c instead of only one commitment message for c.

Our second construction BLAZE$^+$ is built upon BLAZE. It reduces the keys/signature sizes and communication complexity of the signing protocol by utilizing the tree of commitments technique introduced in Chapter 4. More precisely, at the first user stage many masking terms are generated at once in order to blind the signature part that will be output from the second user stage. The number of these masking terms is chosen large enough such that a valid blind signature is always generated with probability approximately 1. Hence, and unlike BLAZE, there is no need to request a protocol restart from the signer anymore.

This allows to safely eliminate the last move of the signing protocol, which includes either an *ok* message or a proof of failure that allows the signer to verify the invalidity of the blind signature computed by the user. We obtain a three-move scheme similar to the basic structure of the underlying canonical identification scheme. As opposed to BLAZE, BLAZE$^+$ uses the Gaussian distribution for masking instead of the uniform distribution over small sets. The latter distribution is known to produce signatures of larger size.

While the one-more unforgeability property of both BLAZE and BLAZE$^+$ is currently not supported with a security reduction from lattice problems (see below), our third scheme BlindOR makes a significant progress towards a three-move lattice-based construction of blind signatures that is simultaneously practical and provably secure. BlindOR uses the OR-technique introduced by Cramer *et al.* [CDS94]. At a high level, an OR-proof is a Sigma protocol that proves the knowledge of a witness for one of two (or more) statements, without revealing which one. The design of BlindOR also involves the partitioning and permutation technique as well as the tree of commitments technique in order to reduce/remove the number of restarts induced by applying the rejection sampling procedure. For the first time, we are able to prove the one-more unforgeability property of lattice-based blind signatures assuming the hardness of both MLWE and MSIS rather than only MSIS. This results in a much more efficient construction. More importantly, using the OR-technique allows us to sidestep a subtle security argument, which is pointed out by Hauck *et al.* [HKLN20] and is missing in the proof of the one-more unforgeability property of all prior lattice-based schemes starting from the first one [Rüc10]. This missing argument guarantees that the security reduction returns a non-trivial solution to MSIS with high probability. We refer to Section 5.5 for more details.

We remark that it is possible to provide the parameters of BLAZE and BLAZE$^+$ with further security requirements in order to cover the missing security argument mentioned above. However, the resulting schemes would look similar to the one proposed by Hauck *et al.* [HKLN20], which is of theoretical interest only. This is due to the huge sizes of parameters that must be used so that their solution applies and yields a correct proof. In particular, the scheme given in [HKLN20] has a public and secret key of size 443.75 KB and 4275 KB, respectively, and generates blind signatures of size 7915.52 KB. While BLAZE and BLAZE$^+$ currently lack a security reduction from lattice problems, we believe that they are still one-more unforgeable as long as both MLWE and MSIS are hard to solve. This argument is based on the fact that there is no attack known yet that is able to forge blind signatures generated by BLAZE and BLAZE$^+$ other than solving MLWE and/or MSIS. Moreover, a similar argument has already been taken many times for cryptographic schemes lacking provable security. For instance, the blind signature scheme underlying the U-Prove protocol [Paq13] is not provably secure, and it was even shown by Baldimtsi and Lysyanskaya [BL13b] that no security proof can be established in the ROM using the currently known approaches under any hardness assumption. Despite this fact, the protocol is used in practice and considered to be secure for a suitable choice of parameters.

Organization

In Section 5.1 we give a formal definition of blind signature schemes in addition to their security properties. We describe our partitioning and permutation technique in Section 5.2.

In Sections 5.3 to 5.5 we present our schemes BLAZE, BLAZE$^+$, and BlindOR, respectively. We propose concrete parameters for our schemes and a practical comparison in Section 5.6. Finally, we describe in Section 5.7 two attacks on previous two-round lattice-based blind signature schemes.

Publications

This chapter is based on the publications [A2,A3], and [A4], which were presented at FC'20, ACISP'20, and CANS'21, respectively. This chapter extends these contributions as follows: We upgrade the structure of both blind signature schemes BLAZE and BLAZE$^+$ so that it is based on lattices over modules rather than lattices over rings, and improve their proofs of both correctness and statistical blindness accordingly. The generalized module structure allows for more flexibility when choosing concrete parameters. We also provide plausible security arguments about their one-more unforgeability property assuming the hardness of both MLWE and MSIS. Furthermore, we select new parameters for all our schemes and provide a practical comparison.

5.1 Blind Signature Schemes

In this section we define the syntax of blind signature schemes in addition to their security properties.

A blind signature scheme is an interactive protocol between two parties called the signer and user. It allows the user to generate a blind signature on a message of his choice. In the following definition we assume that the protocol is always initiated by the signer, which is the case in the schemes we deal with in this thesis.

Definition 5.1. A *blind signature scheme* is a tuple of polynomial-time algorithms

$$\mathsf{BS} = (\mathsf{BS.PGen}, \mathsf{BS.KGen}, \mathsf{BS.S}, \mathsf{BS.U}, \mathsf{BS.Verify}),$$

where:

$\mathsf{BS.PGen}(1^\lambda)$ is a PPT parameter generation algorithm that, on input the security parameter λ, returns a set of public parameters pp, which implicitly contains λ in unary, *i.e.*, $pp \leftarrow\!\!\$\ \mathsf{BS.PGen}(1^\lambda)$. We assume that pp identifies the message space $\mathcal{M}$ of BS.

$\mathsf{BS.KGen}(pp)$ is a PPT key generation algorithm that, on input a set of public parameters pp, returns a key pair (pk, sk), where pk is a public key and sk is a secret key, *i.e.*, $(pk, sk) \leftarrow\!\!\$\ \mathsf{BS.KGen}(pp)$.

$\mathsf{BS.S}(pp, pk, sk)$ is an interactive algorithm that is called *signer* and consists of algorithms $\mathsf{BS.S} = (\mathsf{BS.S}_1, \ldots, \mathsf{BS.S}_k)$, where:

- $\mathsf{BS.S}_1$ is a PPT algorithm that, on input a set of public parameters pp, a public key pk, and a secret key sk, returns a signer message s_1 and a state st_S, *i.e.*, $(s_1, st_\mathsf{S}) \leftarrow\!\!\$\ \mathsf{BS.S}_1(pp, pk, sk)$.

- For all $i \in \{2, \ldots, k\}$, $\mathsf{BS.S}_i$ is a DPT algorithm that, on input a set of public parameters pp, a public key pk, a secret key sk, a state st_{S}, and the $(i-1)^{\text{th}}$ user message u_{i-1}, returns the i^{th} signer message s_i and an updated state st_{S}, *i.e.*, $(s_i, st_{\mathsf{S}}) \leftarrow \mathsf{BS.S}_i(pp, pk, sk, st_{\mathsf{S}}, u_{i-1})$. The state output by the algorithm $\mathsf{BS.S}_k$ is set to $st_{\mathsf{S}} = view$, where $view$ is possibly an empty string $null$, or a status message, *e.g.*, ok.

$\mathsf{BS.U}(pp, pk, m)$ is an interactive algorithm that is called *user* and consists of algorithms $\mathsf{BS.U} = (\mathsf{BS.U}_1, \ldots, \mathsf{BS.U}_t)$, where:

- $\mathsf{BS.U}_1$ is a PPT algorithm that, on input a set of public parameters pp, a public key pk, a message m from the message space $\mathcal{M}$, and the first signer message s_1, returns a user message u_1 and a state st_{U}, *i.e.*, $(u_1, st_{\mathsf{U}}) \leftarrow\!\!\$\ \mathsf{BS.U}_1(pp, pk, m, s_1)$.
- For all $j \in \{2, \ldots, t\}$, $\mathsf{BS.U}_j$ is a DPT algorithm that, on input a set of public parameters pp, a public key pk, a message m, a state st_{U}, and the j^{th} signer message s_j, returns the j^{th} user message u_j and an updated state st_{U}, *i.e.*, $(u_j, st_{\mathsf{U}}) \leftarrow \mathsf{BS.U}_j(pp, pk, m, st_{\mathsf{U}}, s_j)$. The state output by the algorithm $\mathsf{BS.U}_t$ is set to $st_{\mathsf{U}} = null$, and the last user message u_t includes a blind signature sig on the message m. We let $sig = \perp$ denote failure.

$\mathsf{BS.Verify}(pp, pk, m, sig)$ is a DPT verification algorithm that, on input a set of public parameters pp, a public key pk, a message m, and a signature sig, returns 1 if sig is valid and 0 otherwise, *i.e.*, $b \leftarrow \mathsf{BS.Verify}(pp, pk, m, sig)$, where $b \in \{0, 1\}$.

Let $\mathsf{BS} = (\mathsf{BS.PGen}, \mathsf{BS.KGen}, \mathsf{BS.S}, \mathsf{BS.U}, \mathsf{BS.Verify})$ be a blind signature scheme. We write $(view, sig) \leftarrow\!\!\$\ \langle \mathsf{BS.S}(pp, pk, sk), \mathsf{BS.U}(pp, pk, m) \rangle$ to denote the joint execution of $\mathsf{BS.S}$ and $\mathsf{BS.U}$ in the signing protocol of BS with private inputs (pp, pk, sk) for $\mathsf{BS.S}$ and (pp, pk, m) for $\mathsf{BS.U}$, as well as private outputs $view$ for $\mathsf{BS.S}$ and sig for $\mathsf{BS.U}$. Accordingly, we write $\mathsf{BS.S}^{\langle \cdot, \mathsf{BS.U}(pp,pk,m) \rangle^k}(pp, pk, sk)$ if $\mathsf{BS.S}$ can invoke up to k executions of the signing protocol with $\mathsf{BS.U}$. Similarly, we write $\mathsf{BS.U}^{\langle \mathsf{BS.S}(pp,pk,sk), \cdot \rangle^k}(pp, pk, m)$ if $\mathsf{BS.U}$ can invoke up to k executions of the signing protocol with $\mathsf{BS.S}$.

Blind signature schemes are required to satisfy the correctness property given in the following definition:

Definition 5.2. Let BS be a blind signature scheme, and $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that BS is $\mathrm{corr}_{\mathsf{BS}}$-*correct* w.r.t. pp if, for every key pair $(pk, sk) \in \mathsf{BS.KGen}(pp)$ and every message $m \in \mathcal{M}$, we have

$$\Pr_{(view,sig) \leftarrow\!\!\$ \langle \mathsf{BS.S}(pp,pk,sk), \mathsf{BS.U}(pp,pk,m) \rangle} [\mathsf{BS.Verify}(pp, pk, m, sig) \neq 1] \leq \mathrm{corr}_{\mathsf{BS}}.$$

The security of blind signature schemes is defined by two security notions called blindness and one-more unforgeability [JLO97, PS00]. We start with the blindness property, which prevents the (malicious) signer to learn information about signed messages.

Definition 5.3. Let BS be a blind signature scheme, and $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that BS is (t, ε)-*blind* w.r.t. pp if, for every adversarial signer S^* running in time at most t and

$\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp)$:

1: $b \leftarrow\!\!\$\ \{0,1\}$
2: $(pk, sk) \leftarrow\!\!\$\ \mathsf{BS.KGen}(pp)$
3: $(m_0, m_1, st_{\mathsf{find}}) \leftarrow\!\!\$\ \mathsf{S}^*(\mathsf{find}, pp, pk, sk)$
4: $st_{\mathsf{issue}} \leftarrow\!\!\$\ \mathsf{S}^{*\langle\cdot,\mathsf{BS.U}(pp,pk,m_b)\rangle^1,\langle\cdot,\mathsf{BS.U}(pp,pk,m_{1-b})\rangle^1}(\mathsf{issue}, st_{\mathsf{find}})$
5: Let sig_b, sig_{1-b} denote the local output of $\mathsf{BS.U}(pp, pk, m_b), \mathsf{BS.U}(pp, pk, m_{1-b})$.
6: **if** $(sig_0 = \bot) \vee (sig_1 = \bot)$ **then**
7: $\quad (sig_0, sig_1) \leftarrow (\bot, \bot)$
8: $b^* \leftarrow\!\!\$\ \mathsf{S}^*(\mathsf{guess}, sig_0, sig_1, st_{\mathsf{issue}})$
9: **if** $b = b^*$ **then**
10: $\quad$ **return** 1
11: **return** 0

$\mathbf{Exp}^{\text{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp)$:

21: $(pk, sk) \leftarrow\!\!\$\ \mathsf{BS.KGen}(pp)$
22: $((m_1, sig_1), \ldots, (m_\ell, sig_\ell)) \leftarrow\!\!\$\ \mathsf{U}^{*\langle\mathsf{BS.S}(pp,pk,sk),\cdot\rangle^\infty}(pp, pk, \cdot)$
23: Let k denote the number of successful interactions between $\mathsf{BS.S}$ and U^*.
24: **if** $(m_i \neq m_j) \wedge (\mathsf{BS.Verify}(pp, pk, m_i, sig_i) = 1) \wedge (k + 1 = \ell)$ **then** // $\forall i, j \in [\ell], i \neq j$
25: $\quad$ **return** 1
26: **return** 0

Figure 5.1: Definition of the experiments $\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ and $\mathbf{Exp}^{\text{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$.

working in modes find, issue, and guess, we have

$$\mathbf{Adv}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp) = 2 \cdot \left| \Pr\left[\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp) = 1\right] - \frac{1}{2} \right| \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ is depicted in Figure 5.1. We say that BS is ε-*statistically blind* if it is (t, ε)-blind for every t.

In the experiment $\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}$, the adversarial signer S^* chooses two messages m_0 and m_1 in mode find. Then, S^* executes two interactions with the honest user $\mathsf{BS.U}$ in mode issue. Depending on the random bit b, $\mathsf{BS.U}$ computes blind signatures sig_b and sig_{1-b} in the first and second interaction with S^*, respectively. In mode guess, S^* obtains sig_0 and sig_1 in the original order, and has to decide which of the two messages has been signed first. If $\mathsf{BS.U}$ returns $\bot$ in at least one of the two executions, then S^* is informed about the failure and does not get any signature.

Next, we define the one-more unforgeability property of blind signature schemes. It ensures that each successful execution of the signing protocol yields at most one valid blind signature.

Definition 5.4. Let BS be a blind signature scheme, and $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that BS is $(t, q_{\mathsf{Sign}}, \varepsilon)$-*one-more unforgeable* w.r.t. pp if, for every adversarial user U^* running in time at most t and making at most q_{Sign} interactions with $\mathsf{BS.S}$, we have

$$\mathbf{Adv}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp) = \Pr\left[\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$ is depicted in Figure 5.1.

In the experiment $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$, the goal of the adversarial user U^* is to return $k+1$ valid pairs $(m_1, sig_1), \ldots, (m_{k+1}, sig_{k+1})$ after k successful interactions with the honest signer $\mathsf{BS.S}$, where $k \leq q_{\mathsf{Sign}}$.

5.1.1 Further Security Properties

In this section we consider further attacks and extensions to the standard security notions of blindness and one-more unforgeability of blind signature schemes.

Schröder and Unruh [SU17] introduced a stronger security notion called *honest-user unforgeability*. They showed that this notion is more convenient for the security of blind signature schemes as it removes certain types of attacks not captured in the traditional security model of one-more unforgeability due to Pointcheval and Stern [PS00]. Assuming that the one-more unforgeability property holds, we can establish honest-user unforgeability by signing a commitment to the message instead of the message itself via any commitment scheme [Blu81] that is statistically hiding but computationally binding (see, *e.g.* [DPP94]).

Camenisch *et al.* [CNs07] introduced the strengthened security notion of *selective failure blindness*. It prevents a malicious signer in the blindness experiment to choose a message from a distribution that makes the signing protocol fail. Fischlin and Schröder [FS09] showed that the blindness property of any blind signature scheme can easily be extended to cover such kind of attacks. This can also be established by signing a commitment to the message instead of the message itself, and the commitment scheme has to be statistically hiding but computationally binding.

Fischlin [Fis06] considered a blindness experiment under maliciously generated keys. More concretely, instead of letting the experiment generates a key pair, the signer further outputs a public key pk, *i.e.*, $(pk, m_0, m_1, st_{\mathsf{find}}) \leftarrow\!\!\$\ \mathsf{S}^*(\mathsf{find}, pp)$ (*cf.* Figure 5.1). Blindness under this setting is harder to achieve, and it is still unclear how to prove the blindness property of our schemes under maliciously generated keys. The same is true for previous schemes, *e.g.*, for the scheme by Hauck *et al.* [HKLN20].

Fischlin and Schröder [FS10] showed that it is infeasible to find a black-box reduction from some non-interactive cryptographic assumption to the one-more unforgeability property in the *standard model* for blind signature schemes that satisfy some specific properties. This impossibility result does not apply to our blind signature schemes, since we consider security in the ROM.

In the context of blind signatures, Schnorr [Sch01] defined a computational problem called *Random inhomogenities in an Overdetermined, Solvable system of linear equations* (ROS). He showed that solving the ROS problem implies the forgeability of number-theoretic blind signature schemes such as [PS00, AO00]. This attack was significantly improved, *e.g.*, by

Wagner [Wag02] and most recently by Benhamouda *et al.* [BLL+21]. However, Hauck *et al.* [HKLN20] showed that the ROS attack cannot be applied to lattice-based schemes due to their algebraic structure. They defined a lattice variant of this problem called the *Generalized* ROS problem, and leave solving this variant in sub-exponential time as an open problem.

5.2 The Partitioning and Permutation Technique

In this section we introduce the partitioning and permutation technique. It allows a user in a blind signature scheme to mask the hash value that is output by some cryptographic hash function, so that blindness is satisfied at the first user stage without the need to apply any security check (rejection sampling procedure) or restart the algorithm $\mathsf{BS.U}_1$ (*cf.* Definition 5.1). As a result, smaller parameters can be selected for the scheme, which reduces the size of signatures and speeds up the signing process. It also allows to use cryptographic hash functions that output polynomials from $\mathbb{T}_\kappa^n$ instead of polynomials with unspecified Hamming weight.

Definition 5.5. We define by $\mathbb{T} := \left\{ (-1)^b \cdot X^i \;\middle|\; b \in \{0,1\}, i \in \mathbb{Z} \right\}$ the set of signed permutation polynomials that represent a rotation multiplied by a sign.

Lemma 5.6. *The set $\mathbb{T}$ defines a group with respect to multiplication in R. The inverse of any $p = (-1)^b \cdot X^i \in \mathbb{T}$ is given by $p^{-1} = (-1)^{1-b} \cdot X^{n-i} \in \mathbb{T}$.*

Proof. Let $p_1 = (-1)^{b_1} \cdot X^{i_1}$, $p_2 = (-1)^{b_2} \cdot X^{i_2} \in \mathbb{T}$. Then, $p_1 \cdot p_2 = (-1)^{b_1+b_2} \cdot X^{i_1+i_2} \in \mathbb{T}$, and $p \cdot p^{-1} = (-1)^b \cdot X^i \cdot (-1)^{1-b} \cdot X^{n-i} = -X^n \equiv 1 \bmod \langle X^n + 1 \rangle$. Thus, every $p \in \mathbb{T}$ has an inverse $p^{-1} \in \mathbb{T}$ and the neutral element is given by the polynomial 1. □

The partitioning and permutation technique works as follows: Let c be a polynomial from $\mathbb{T}_\kappa^n$. The goal is to mask c to obtain a blinded element, and then send this element to the signer in order to compute a signature. To this end, the polynomial c is split into partitions $c_1, \ldots, c_\kappa \in \mathbb{T}$ such that $c = \sum_{j=1}^{\kappa} c_j$ and each partition c_j is the j^{th} non-zero entry of c at exactly the same position. Then, each partition c_j is masked by computing $c_j^* = p_j^{-1} \cdot c_j$ for all $j \in [\kappa]$, where p_j are chosen from the uniform distribution over $\mathbb{T}$. The masked element that is sent to the signer is given by the vector $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*) \in \mathbb{T}^\kappa$. The following lemma shows that the vector $\mathbf{c}^*$ is independently distributed from the polynomial c. It is a crucial result that will be used to prove the statistical blindness of our blind signature schemes.

Lemma 5.7. *Let $c \in \mathbb{T}_\kappa^n$, and $c_1, \ldots, c_\kappa$ be a partition of c such that $c = \sum_{j=1}^{\kappa} c_j$ and each c_j consists of the j^{th} non-zero entry of c at the j^{th} position. Furthermore, let $c_j^* = p_j^{-1} c_j$ for random signed rotations $p_1, \ldots, p_\kappa \in \mathbb{T}$. Then, $c_j^*, c_j \in \mathbb{T}$ and we have:*

$$\Pr_{p_1,\ldots,p_\kappa \leftarrow\$ \mathbb{T}} \left[(c_1^*, \ldots, c_\kappa^*) = \left(p_1^{-1} c_1, \ldots, p_\kappa^{-1} c_\kappa\right) \;\middle|\; c \right] =$$
$$\Pr_{\substack{p_1,\ldots,p_\kappa \leftarrow\$ \mathbb{T} \\ c \leftarrow\$ \mathbb{T}_\kappa^n}} \left[(c_1^*, \ldots, c_\kappa^*) = \left(p_1^{-1} c_1, \ldots, p_\kappa^{-1} c_\kappa\right) \right] = (2n)^{-\kappa}.$$

Proof. For any partitioning we have $c_j \in \mathbb{T}$, since it contains only one ± 1 at exactly the same position as c. Furthermore, each $c_j \in \mathbb{T}$ can be transformed into any element of $\mathbb{T}$ via a signed rotation $p \in \mathbb{T}$, hence $c_j^* \in \mathbb{T}$.

Let $c \in \mathbb{T}_\kappa^n$, and $c_1, \ldots, c_\kappa$ be a partition of c. Then, for any fixed $c_j^* \in \mathbb{T}$ there exists exactly one set of elements $p_1^{-1}, \ldots, p_\kappa^{-1} \in \mathbb{T}$ such that $c_1^* = p_1^{-1}c_1, \ldots, c_\kappa^* = p_\kappa^{-1}c_\kappa$. Thus, we have

$$\Pr_{p_1,\ldots,p_\kappa \leftarrow\$ \mathbb{T}}\left[(c_1^*, \ldots, c_\kappa^*) = \left(p_1^{-1}c_1, \ldots, p_\kappa^{-1}c_\kappa\right) \mid c\right] = (2n)^{-\kappa}.$$

Next, we recall that for any fixed $c_j^*, c_j \in \mathbb{T}$, there exists exactly one $p_j \in \mathbb{T}$ for each $j \in [\kappa]$ such that $c_j^* = p_j^{-1}c_j$. Thus, we have

$$\Pr_{\substack{p_1,\ldots,p_\kappa \leftarrow\$ \mathbb{T} \\ c \leftarrow\$ \mathbb{T}_\kappa^n}}\left[(c_1^*, \ldots, c_\kappa^*) = \left(p_1^{-1}c_1, \ldots, p_\kappa^{-1}c_\kappa\right)\right] =$$

$$\sum_{c \in \mathbb{T}_\kappa^n} \Pr_{p_1,\ldots,p_\kappa \leftarrow\$ \mathbb{T}}\left[(c_1^*, \ldots, c_\kappa^*) = \left(p_1^{-1}c_1, \ldots, p_\kappa^{-1}c_\kappa\right) \mid c\right] \cdot \Pr[c] = (2n)^{-\kappa}.$$

□

5.3 The Blind Signature Scheme BLAZE

In this section we introduce our first blind signature scheme BLAZE. Its signing protocol consists of four moves between the signer and user. As opposed to the first lattice-based construction of blind signatures [Rüc10], any blind signature generated by BLAZE has to pass two security checks (rejection sampling procedures) rather than three. The first one is carried out by the signer in order to conceal the secret key, and the second one is performed by the user to ensure blindness. If the first check fails, the signer simply restarts the signing protocol. However, the user requests a protocol restart from the signer in case the computed blind signature does not pass the security check. This is why the fourth move is required. That is, this move allows the signer to check if the user was not able to compute a blind signature, and to restart the interaction. In Section 5.3.1, we give a detailed description of BLAZE and prove its correctness. Then, we analyze its statistical blindness and computational one-more unforgeability in Section 5.3.2.

5.3.1 Description of the Scheme

We let $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ be a cryptographic hash function. In order to ensure security over restarts, BLAZE uses a commitment scheme [Blu81] that is statistically hiding but computationally binding (see, *e.g.* [DPP94]). This kind of schemes allows to commit to a message while keeping it secret and unmodified. It defines a commitment function

$$\mathsf{Com}\colon \{0,1\}^* \times \{0,1\}^{\ell_{str}} \to \{0,1\}^{\ell_\tau},\ (m; str) \mapsto \tau,$$

where $\ell_{str}, \ell_\tau \geq \lambda$. The input of Com is a message m of arbitrary length and a randomness str of length ℓ_{str}, and the output is a commitment τ of length ℓ_τ. This function provides two basic security properties called the statistical hiding and computational binding. The

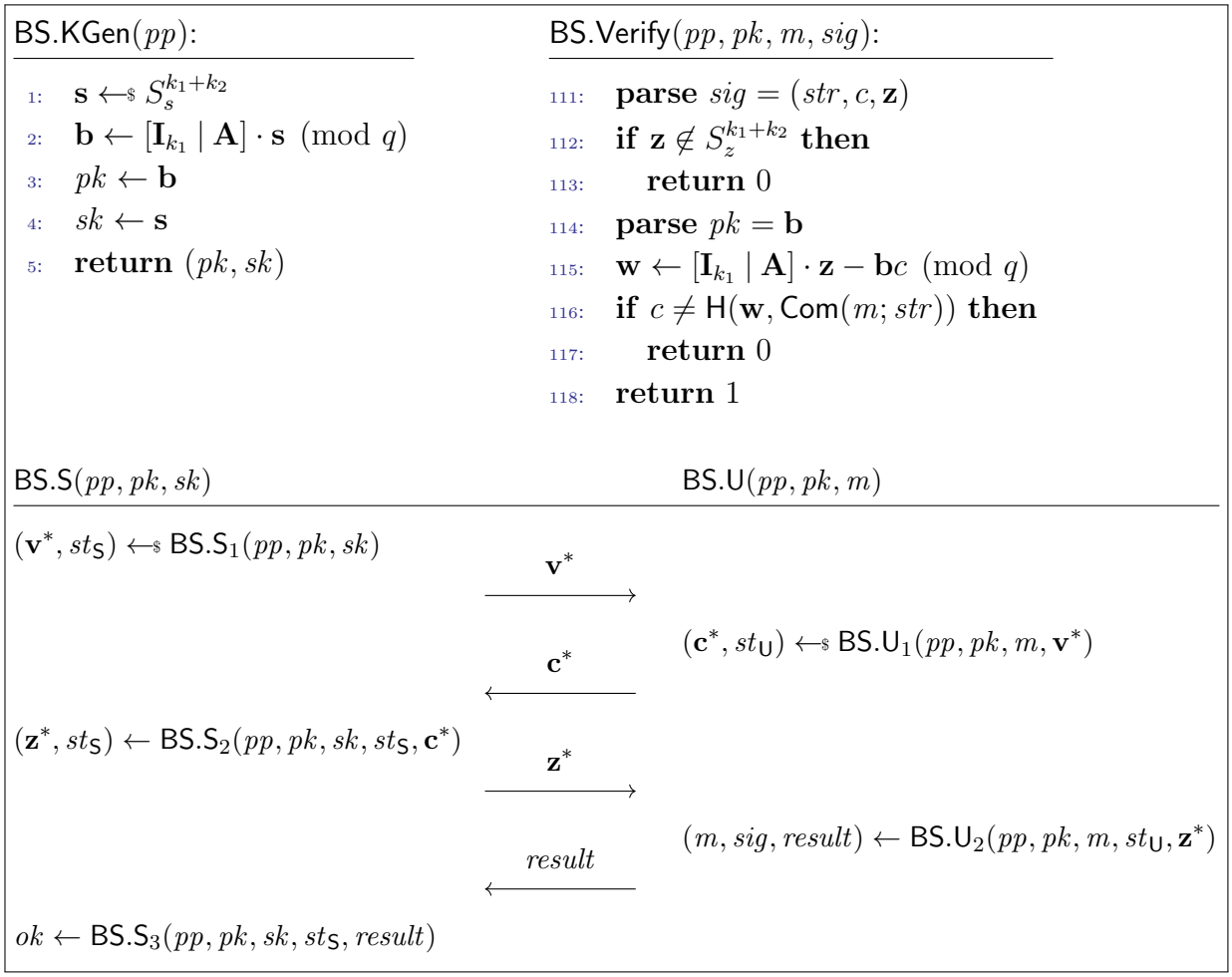

Figure 5.2: Overview of the blind signature scheme BLAZE. The algorithm BS.PGen is explained in the text, and the algorithms BS.S, BS.U are described in Figure 5.3.

first one indicates that the commitment τ does not reveal any statistical information about the message m so that the distributions of τ and any other commitment τ' to a message m' are ε-statistically close, where $\varepsilon \approx 0$. The latter property ensures that it is computationally infeasible to find a message $m' \neq m$ and a randomness str' satisfying

$$\mathsf{Com}(m'; str') = \tau = \mathsf{Com}(m; str).$$

Furthermore, any commitment scheme requires the correctness property, which ensures that any honestly created commitment can be correctly verified. For further details about commitment schemes and formal definitions we refer, for example, to [Alk15].

Next, we describe our blind signature scheme BLAZE. Its respective algorithms are formalized in Figures 5.2 to 5.4.

Parameter generation. Given 1^λ, BS.PGen generates a set of public parameters subject to the constraints given in Table 5.1. This set is given by

$$pp = \left(1^\lambda, n, k_1, k_2, q, s, \kappa, y, z^*, e, z, \ell_{seed}, \ell_{str}, \ell_\tau, \mathbf{A}\right),$$

$\mathsf{BS.S}_1(pp, pk, sk)$:

11: **for** $j = 1$ **to** κ **do**
12: $\quad \mathbf{y}_j \leftarrow\!\!\$\ S_y^{k_1+k_2}$
13: $\quad \mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
14: $\mathbf{y} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
15: $\mathbf{v}^* \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
16: $st_\mathsf{S} \leftarrow (\mathbf{y}, \mathbf{v}^*)$
17: **return** $(\mathbf{v}^*, st_\mathsf{S})$

$\mathsf{BS.U}_1(pp, pk, m, \mathbf{v}^*)$:

21: **parse** $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
22: $str \leftarrow\!\!\$\ \{0,1\}^{\ell_{str}}$
23: $\tau \leftarrow \mathsf{Com}(m; str)$
24: $\rho \leftarrow\!\!\$\ \{0,1\}^{\ell_{seed}}$
25: $\mathbf{e} \in S_e^{k_1+k_2} \leftarrow \mathsf{Expand}(\rho)$
26: $\mathbf{p} = (p_1, \ldots, p_\kappa) \leftarrow\!\!\$\ \mathbb{T}^\kappa$
27: $\mathbf{v} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$
28: $c = \sum_{j=1}^{\kappa} c_j \leftarrow \mathsf{H}(\mathbf{v}, \tau) \quad /\!/\ c_j \in \mathbb{T}$
29: $\mathbf{c}^* \leftarrow (c_1 p_1^{-1}, \ldots, c_\kappa p_\kappa^{-1})$
30: $st_\mathsf{U} \leftarrow (\mathbf{v}^*, str, \tau, \rho, \mathbf{p}, \mathbf{e}, c, \mathbf{c}^*)$
31: **return** $(\mathbf{c}^*, st_\mathsf{U})$

$\mathsf{BS.S}_3(pp, pk, sk, st_\mathsf{S}, result)$:

81: **if** $result \neq ok$ **then**
82: $\quad$ **if** $\mathsf{Proof}(pp, pk, st_\mathsf{S}, result) = 1$ **then**
83: $\quad\quad$ **restart** BS
84: **return** ok

$\mathsf{BS.S}_2(pp, pk, sk, st_\mathsf{S}, \mathbf{c}^*)$:

41: **parse** $sk = \mathbf{s}$
42: **parse** $st_\mathsf{S} = (\mathbf{y}, \mathbf{v}^*)$
43: **parse** $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
44: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*) \quad /\!/\ c_j^* \in \mathbb{T}$
45: **for** $j = 1$ **to** κ **do**
46: $\quad \mathbf{z}_j \leftarrow \mathbf{y}_j + \mathbf{s}c_j^*$
47: $\mathbf{z}^* \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
48: **if** $\mathbf{z}^* \notin S_{z^*}^{(k_1+k_2)\kappa}$ **then**
49: $\quad$ **return** $\perp$
50: $st_\mathsf{S} \leftarrow (\mathbf{y}, \mathbf{v}^*, \mathbf{c}^*, \mathbf{z}^*)$
51: **return** $(\mathbf{z}^*, st_\mathsf{S})$

$\mathsf{BS.U}_2(pp, pk, m, st_\mathsf{U}, \mathbf{z}^*)$:

61: **if** $\mathbf{z}^* \notin S_{z^*}^{(k_1+k_2)\kappa}$ **then**
62: $\quad$ **return** $(\perp, \perp, \perp)$
63: **parse** $st_\mathsf{U} = (\mathbf{v}^*, str, \tau, \rho, \mathbf{p}, \mathbf{e}, c, \mathbf{c}^*)$
64: **parse** $pk = \mathbf{b}$
65: **parse** $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
66: **parse** $\mathbf{p} = (p_1, \ldots, p_\kappa)$
67: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$
68: **parse** $\mathbf{z}^* = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
69: **for** $j = 1$ **to** κ **do**
70: $\quad \mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j^* \pmod q$
71: $\quad$ **if** $\mathbf{v}_j \neq \mathbf{w}_j$ **then**
72: $\quad\quad$ **return** $(\perp, \perp, \perp)$
73: $\mathbf{z} \leftarrow \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$
74: **if** $\mathbf{z} \notin S_z^{k_1+k_2}$ **then**
75: $\quad result \leftarrow (\tau, \rho, \mathbf{p}, c)$
76: $\quad$ **return** $(\perp, \perp, result)$
77: $result \leftarrow ok$
78: $sig \leftarrow (str, c, \mathbf{z})$
79: **return** $(m, sig, result)$

Figure 5.3: A formal description of the algorithms BS.S and BS.U of BLAZE (*cf.* Figure 5.2). The signer restarts the protocol if $\mathsf{BS.S}_2$ returns $\perp$. The algorithm Proof is given in Figure 5.4.

$\mathsf{Proof}(pp, pk, st_{\mathsf{S}}, result)$:

91: **parse** $pk = \mathbf{b}$
92: **parse** $st_{\mathsf{S}} = (\mathbf{y}, \mathbf{v}^*, \mathbf{c}^*, \mathbf{z}^*)$
93: **parse** $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
94: **parse** $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
95: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$
96: **parse** $\mathbf{z}^* = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
97: **parse** $result = (\tau, \rho, \mathbf{p}, c)$
98: $\mathbf{e} \in S_e^{k_1+k_2} \leftarrow \mathsf{Expand}(\rho)$
99: $\mathbf{z} \leftarrow \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$
100: $\mathbf{v} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$
101: $\mathbf{w} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$
102: **if** $\left(\sum_{j=1}^{\kappa} c_j^* p_j = c = \mathsf{H}(\mathbf{v}, \tau)\right) \wedge (c = \mathsf{H}(\mathbf{w}, \tau)) \wedge \left(\mathbf{z} \notin S_z^{k_1+k_2}\right)$ **then**
103: **return** 1
104: **return** 0

Figure 5.4: The algorithm carried out by the signer in order to verify the proof of failure that is sent by the user (see Figure 5.3).

where $\mathbf{A}$ is a matrix chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. The remaining parameters are described below.

Key generation. Given a set of public parameters pp, $\mathsf{BS.KGen}$ samples a vector $\mathbf{s}$ from the uniform distribution over $S_s^{k_1+k_2}$, and computes $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. The public key is given by $pk = \mathbf{b}$, while the secret key is set to $sk = \mathbf{s}$.

Signing. The signing protocol consists of four moves between the signer $\mathsf{BS.S}$ and the user $\mathsf{BS.U}$, and it is initiated by the signer.

Given (pp, pk, sk), $\mathsf{BS.S}_1$ selects masking vectors $\mathbf{y}_1, \ldots, \mathbf{y}_\kappa$ from the uniform distribution over $S_y^{k_1+k_2}$. Then, it sets $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$, and sends the vector $\mathbf{v}^*$ to $\mathsf{BS.U}$, where $\mathbf{v}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$ for all $j \in [\kappa]$.

Let m be the message being blindly signed by $\mathsf{BS.S}$. On input $(pp, pk, m, \mathbf{v}^*)$, $\mathsf{BS.U}_1$ computes the commitment $\tau = \mathsf{Com}(m; str)$, where str is chosen randomly from the uniform distribution over $\{0,1\}^{\ell_{str}}$. Afterwards, it samples a random string ρ from the uniform distribution over $\{0,1\}^{\ell_{seed}}$, and expands it via the function Expand to obtain the vector $\mathbf{e} \in S_e^{k_1+k_2}$. Note that $\mathbf{e}$ is not directly sampled from the uniform distribution over $S_e^{k_1+k_2}$. This reduces the communication complexity of the signing

protocol because a proof of failure sent by $\mathsf{BS.U}_2$ (see below) will include only ρ rather than the whole vector $\mathbf{e}$. After that, $\mathsf{BS.U}_1$ selects a vector $\mathbf{p} = (p_1, \ldots, p_\kappa)$ from the uniform distribution over $\mathbb{T}^\kappa$. Then, it computes $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$, and generates the hash value $c = \mathsf{H}(\mathbf{v}, \tau)$, where c is split into the partitions $c_j \in \mathbb{T}$ for all $j \in [\kappa]$ such that $c = \sum_{j=1}^{\kappa} c_j$ and the j^{th} partition c_j consists of the j^{th} non-zero entry of c at the j^{th} position. Subsequently, $\mathsf{BS.U}_1$ masks the partitions c_j by computing $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$, where $c_j^* = c_j p_j^{-1}$ for all $j \in [\kappa]$. Thanks to the partitioning and permutation technique introduced in Section 5.2, the vector $\mathbf{c}^*$ already blinds the hash value c without the need to carry out any security check (rejection sampling procedure) and any potential restart. Then, $\mathsf{BS.U}_1$ sends the vector $\mathbf{c}^*$ to $\mathsf{BS.S}_2$.

Using the partitions $c_1^*, \ldots, c_\kappa^*$, $\mathsf{BS.S}_2$ computes $\mathbf{z}^* = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$, where $\mathbf{z}_j = \mathbf{y}_j + \mathbf{s} c_j^*$ for all $j \in [\kappa]$. Then, it checks if $\mathbf{z}^* \in S_{z^*}^{(k_1+k_2)\kappa}$. This check ensures that the vector $\mathbf{z}^*$ follows the uniform distribution over $S_{z^*}^{(k_1+k_2)\kappa}$, and that it does not leak any information about the secret key. If this is the case, then $\mathsf{BS.S}_2$ sends $\mathbf{z}^*$ to $\mathsf{BS.U}_2$. Otherwise, $\mathsf{BS.S}_2$ restarts the protocol.

Upon receiving the vector $\mathbf{z}^*$, $\mathsf{BS.U}_2$ verifies that $\mathbf{z}^* \in S_{z^*}^{(k_1+k_2)\kappa}$, and that the commitments $\mathbf{v}_1, \ldots, \mathbf{v}_\kappa$ received from $\mathsf{BS.S}$ in the first move correspond to both vectors $\mathbf{c}^*$ and $\mathbf{z}^*$, *i.e.*, $\mathbf{v}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b} c_j^* \pmod q$ for all $j \in [\kappa]$. These checks detect malicious signers and ensure that the generated signatures are valid and blind. However, they can be skipped in applications with trustworthy signers. Afterwards, $\mathsf{BS.U}_2$ computes $\mathbf{z} = \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$. This brings $\mathbf{z}$ into the form $\mathbf{z} = \mathbf{y}' + \mathbf{s} c$ for some $\mathbf{y}'$, and hence allows the verification algorithm to succeed. In fact, we must have $[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}' = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$. Therefore, $\mathsf{BS.U}_2$ computes $\mathbf{z}$ that way, and then checks if $\mathbf{z} \in S_z^{k_1+k_2}$ in order ensure that $\mathbf{z}$ is uniformly distributed over $S_z^{k_1+k_2}$, and that all $\mathbf{z}_j$ are concealed within $\mathbf{z}$. Finally, $\mathsf{BS.U}_2$ sends $\mathit{result} = \mathit{ok}$ to $\mathsf{BS.S}_3$ indicating that it has obtained a valid blind signature. The blind signature is given by $\mathit{sig} = (\mathit{str}, c, \mathbf{z})$. If $\mathbf{z} \notin S_z^{k_1+k_2}$, then $\mathsf{BS.U}_2$ sends $\mathsf{BS.S}_3$ a proof of failure by setting $\mathit{result} = (\tau, \mathbf{p}, \rho, c)$. This tuple allows $\mathsf{BS.S}_3$ to verify if $\mathsf{BS.U}$ was indeed not able to obtain a valid blind signature (see Line 102 in Figure 5.4). If this proof of failure is valid, then $\mathsf{BS.S}_3$ restarts the protocol. Otherwise, it assumes that $\mathsf{BS.U}$ was already successful and terminates the protocol.

Verification. On input (pp, pk, m, sig), $\mathsf{BS.Verify}$ first makes sure that $\mathbf{z} \in S_z^{k_1+k_2}$. If this is not the case, then the signature sig is rejected. Then, it computes the vector $\mathbf{w} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b} c \pmod q$, and accepts if and only if the output of the hash function H on input $(\mathbf{w}, \mathsf{Com}(m; \mathit{str}))$ is equal to c, *i.e.*, $c = \mathsf{H}(\mathbf{w}, \mathsf{Com}(m; \mathit{str}))$.

The following theorem states the correctness of BLAZE.

Theorem 5.8. *Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be a cryptographic hash function, and* Com *be the commitment function of a commitment scheme that is statistically hiding but computationally binding. The blind signature scheme* BLAZE *is* $\mathrm{corr}_{\mathsf{BS}}$*-correct w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$*, where*

$$\mathrm{corr}_{\mathsf{BS}} = 2 - \frac{1}{M_{\mathsf{S}}} - \frac{1}{M_{\mathsf{U}}}.$$

Table 5.1: A review of the parameters of BLAZE.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 1 \pmod{2n}$
s	Bound of $\|\mathbf{s}\|_\infty$	$s \in \mathbb{Z}_{\geq 1}$
κ	Specifies the set $\mathbb{T}_\kappa^n$	$2^\kappa \binom{n}{\kappa} \geq 2^\lambda$
M_S	No. restarts induced by BS.S	$M_\mathsf{S} \geq 1$
M_U	No. restarts induced by BS.U	$M_\mathsf{U} \geq 1$
M	Total No. restarts of BS	$M = M_\mathsf{S} M_\mathsf{U}$
y	Bound of $\|\mathbf{y}\|_\infty$	$\exp\left(\frac{-(k_1+k_2)\kappa n s}{y}\right) \geq 1/M_\mathsf{S}$
z^*	Bound of $\|\mathbf{z}^*\|_\infty$	$z^* = y - s$
e	Bound of $\|\mathbf{e}\|_\infty$	$\exp\left(\frac{-(k_1+k_2)n\kappa z^*}{e}\right) \geq 1/M_\mathsf{U}$
z	Bound of $\|\mathbf{z}\|_\infty$	$z = e - \kappa z^*$
$\ell_{str}, \ell_\tau, \ell_{seed}$	In-/output length of functions Com and Expand	$\ell_{str}, \ell_\tau, \ell_{seed} \geq \lambda$

Proof. Let $sig = (str, c, \mathbf{z})$ be a valid blind signature that is generated by the signing protocol of BLAZE on a message m. Then, we have $\mathbf{z} \in S_z^{k_1+k_2}$. Moreover, the condition

$$c = \mathsf{H}(\mathbf{w}, \mathsf{Com}(m; str))$$

in the verification algorithm (*cf.* Figure 5.2) is satisfied due to the correctness property of the commitment scheme[3], *i.e.*, $\mathsf{Com}(m; str) = \tau$, and due to the following:

$$\begin{aligned}
\mathbf{w} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \left(\mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{z}_j p_j\right) - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \left(\mathbf{e} + \sum_{j=1}^{\kappa} (\mathbf{y}_j + \mathbf{s}c_j^*) p_j\right) - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} ([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j) p_j + \sum_{j=1}^{\kappa} ([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}) c_j^* p_j - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j + \sum_{j=1}^{\kappa} \mathbf{b}c_j - \mathbf{b}c
\end{aligned}$$

[3]We remark that Theorem 5.8 requires that the commitment scheme is perfectly correct. This means that its related commitment function Com always generates commitments that can be verified correctly, *i.e.*, with probability 1.

$$= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j$$
$$= \mathbf{v} \pmod{q}.$$

Next, we observe that a protocol restart is first triggered by the algorithm $\mathsf{BS.S}_2$. This occurs if $\mathbf{z}^* \notin S_{z^*}^{(k_1+k_2)\kappa}$ (see Line 48 in Figure 5.3). The probability that $\mathbf{z}^* \in S_{z^*}^{(k_1+k_2)\kappa}$ is given by

$$\left(\frac{2z^*+1}{2y+1}\right)^{(k_1+k_2)\kappa n} = \left(\frac{2y-2s+1}{2y+1}\right)^{(k_1+k_2)\kappa n} = \left(1-\frac{s}{y+\frac{1}{2}}\right)^{(k_1+k_2)\kappa n}$$
$$\approx \left(1-\frac{s}{y}\right)^{(k_1+k_2)\kappa n} \approx \exp\left(\frac{-(k_1+k_2)\kappa n s}{y}\right),$$

where we used the fact that the parameter selection always require a value of y that is very large compared to $1/2$. Therefore, we select y such that the above probability is at least $1/M_{\mathsf{S}}$, where $M_{\mathsf{S}} \geq 1$ is a fixed number of the restarts induced by $\mathsf{BS.S}$. Thus, the signer returns $\perp$ with probability at most $1 - \frac{1}{M_{\mathsf{S}}}$. A protocol restart is also induced by $\mathsf{BS.U}_2$ if $\mathbf{z} \notin S_z^{k_1+k_2}$ (see Line 74 in Figure 5.3). The probability that $\mathbf{z} \in S_z^{k_1+k_2}$ is given by

$$\left(\frac{2z+1}{2e+1}\right)^{(k_1+k_2)n} \approx \left(1-\frac{\kappa z^*}{e}\right)^{(k_1+k_2)n} \approx \exp\left(\frac{-(k_1+k_2)n\kappa z^*}{e}\right),$$

where we also used the fact that e is very large compared to $1/2$. Similarly, e is chosen such that the latter probability is at least $1/M_{\mathsf{U}}$, where $M_{\mathsf{U}} \geq 1$ is a fixed number of the restarts induced by $\mathsf{BS.U}$. Thus, the user returns $(\perp, \perp, \mathit{result})$ with probability at most $1 - \frac{1}{M_{\mathsf{U}}}$. Hence, the correctness error of BLAZE is at most $\mathrm{corr}_{\mathsf{BS}} = 2 - \frac{1}{M_{\mathsf{S}}} - \frac{1}{M_{\mathsf{U}}}$, and the total number of protocol restarts that is required to obtain a valid blind signature is given by $M_{\mathsf{S}} M_{\mathsf{U}} = M$. □

5.3.2 Security Analysis

In this section we prove the statistical blindness of BLAZE and provide our security argument about its computational one-more unforgeability.

Theorem 5.9. *Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be a cryptographic hash function modeled as a random oracle, and* Com *be the commitment function of a commitment scheme that is statistically hiding but computationally binding. Furthermore, let* $\varepsilon' > 0$ *be the statistical distance between the distributions of any two commitments generated by the function* Com*. The blind signature scheme* BLAZE *is* ε*-statistically blind w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM, where* $\varepsilon = \max\{\varepsilon', (2n)^{-\kappa}\}$.

Proof. In the blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ (*cf.* Definition 5.3) the adversarial signer S^* selects two messages m_0, m_1, and interacts with the honest user $\mathsf{BS.U}$ twice. We show that after both interactions, the messages output by the user (interpreted as random variables) are independently distributed and do not leak any information about the signed messages and the respective interactions.

The vector $\mathbf{c}^*$ and the blind signature $sig = (str, c, \mathbf{z})$ are both uniformly distributed over $\mathbb{T}^\kappa$ and $\{0,1\}^{\ell_{str}} \times \mathbb{T}_\kappa^n \times S_z^{k_1+k_2}$, respectively, and hence they do not leak any information. Moreover, Lemma 5.7 ensures that the vector $\mathbf{c}^*$ is independently distributed from the polynomial c, and S^* can link both elements only with probability $(2n)^{-\kappa}$ over guessing.

If the signing protocol needs to be restarted, then $\mathsf{BS.U}$ generates fresh random elements $str, \rho, \mathbf{p}$. Therefore, the protocol executions are independent of each other, and hence S^* does not get any information about the message being signed. Furthermore, the string τ included in the proof of failure also maintains blindness due to the statistical hiding property of Com. □

Conjecture 5.10. *Let* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be a cryptographic hash function modeled as a random oracle, and* Com *be the commitment function of a commitment scheme that is statistically hiding but computationally binding. The blind signature scheme* BLAZE *is one-more unforgeable w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM as long as* MLWE^∞ *is hard w.r.t.* $pp' = (n, k_1, k_2, q, s, \mathbf{A})$ *and* MSIS^∞ *is hard w.r.t.* $pp'' = (n, k_1, k_2 + 1, q, 2e)$.

In order to forge blind signatures, an attacker can attempt to recover the secret key from the public parameters and public key. This is based on the hardness of MLWE^∞ w.r.t. $pp' = (n, k_1, k_2, q, s, \mathbf{A})$.

An attacker can also succeed in forging blind signatures if he can break the binding property of the commitment scheme underlying BLAZE. In this case, he can simply interact with the signer to obtain a blind signature $sig = (str, c, \mathbf{z})$ on a message m, and then find another message $m' \neq m$ and a randomness str' satisfying $\mathsf{Com}(m'; str') = \mathsf{Com}(m; str)$. This allows to obtain the forgery $(str', c, \mathbf{z})$ on the message m'.

Assuming that the attacker does not know the secret key and that the binding property of the underlying commitment scheme holds, we argue in the following that forging a signature implies solving MSIS^∞ w.r.t. $pp'' = (n, k_1, k_2 + 1, q, 2e)$. Let $sig = (str, c, \mathbf{z})$ be a valid forgery on a message m. Then, we have $\mathbf{z} \in S_z^{k_1+k_2}$ and

$$c = \mathsf{H}([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q, \mathsf{Com}(m; str)).$$

By setting

$$\tau = \mathsf{Com}(m; str), \ \mathbf{A}' = [\mathbf{A} \mid -\mathbf{b}] \in R_q^{k_1 \times (k_2+1)}, \text{ and } \mathbf{x} = [\mathbf{z} \mid c]^\top \in R^{k_1+k_2+1},$$

we can write

$$c = \mathsf{H}([\mathbf{I}_{k_1} \mid \mathbf{A}'] \cdot \mathbf{x} \pmod q, \tau),$$

where $\|\mathbf{x}\|_\infty \leq z < e$. Therefore, the pair $(\mathbf{x}, \tau) \in R^{k_1+k_2+1} \times \{0,1\}^{\ell_\tau}$ constitutes a solution to $\mathsf{SelfTargetMSIS}^\infty$ with respect to the public parameters given by the set $(n, k_1, k_2 + 1, q, \kappa, e, \mathsf{H})$ and the matrix $\mathbf{A}'$. By Lemma 2.12, this implies solving MSIS^∞ w.r.t. $pp'' = (n, k_1, k_2 + 1, q, 2e)$ and the matrix $\mathbf{A}'$.

Finally, we analyze the case that a user is able to generate a valid blind signature after an aborted interaction with the signer. From the aborted interaction we obtain a proof of failure $result = (\tau, \rho, \mathbf{p}, c)$. It satisfies the following conditions (see Line 102 in Figure 5.4):

$$\sum_{j=1}^{\kappa} c_j^* p_j = c = \mathsf{H}(\mathbf{v}, \tau) \tag{5.1}$$

$$c = \mathsf{H}(\mathbf{w}, \tau) \tag{5.2}$$
$$\mathbf{z} \notin S_z^{k_1+k_2} \tag{5.3}$$

Now, assume that the user obtains a valid blind signature $sig' = (str', c', \mathbf{z}')$ after an aborted interaction. If $c' = c$, then by Equation (5.2) together with the verification algorithm and the binding property of Com we obtain

$$[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{z} - \mathbf{z}') = [\mathbf{I}_{k_1} \mid \mathbf{A}'] \cdot \mathbf{x} = \mathbf{0} \pmod{q},$$

where $\mathbf{x} = [\mathbf{z} - \mathbf{z}' \mid \mathbf{0}]^\top$. In addition, we have $\mathbf{z} \neq \mathbf{z}'$, because the case $\mathbf{z} = \mathbf{z}'$ contradicts Equation (5.3). Note that $\|\mathbf{z}\|_\infty \leq e + \kappa z^*$, and hence $\|\mathbf{z} - \mathbf{z}'\|_\infty \leq z + e + \kappa z^* = 2e$. Therefore, the vector $\mathbf{x}$ constitutes a solution to MSIS$^\infty$ w.r.t. $pp'' = (n, k_1, k_2 + 1, q, 2e)$ and the matrix $\mathbf{A}'$. If $c' \neq c$, then by Equation (5.1) we must have $c_j^* = p_j^{-1} c_j = {p'}_j^{-1} c'_j$, where ${p'}_j \neq p_j$ for all $j \in [\kappa]$. Otherwise, sig' was not obtained from the aborted interaction. Thus, we have $p_j^{-1} = {p'}_j^{-1} c'_j c_j^{-1}$. Hence, the user must have predicted the output of H in order to determine p_j^{-1}, which is assumed to be computationally infeasible.

5.4 The Blind Signature Scheme BLAZE$^+$

In this section we introduce our second blind signature scheme BLAZE$^+$. We recall that the signing protocol of BLAZE consists of four moves between the signer BS.S and the user BS.U. Protocol restarts are triggered due to two security checks. The first check is carried out by BS.S in order to hide the secret key, and the second check by BS.U to maintain blindness. In case the latter check fails, BS.U requests a protocol restart by sending BS.S a proof of failure. This is why the last move is needed in the protocol, as opposed to the standard three-move structure of the canonical identification scheme underlying BLAZE. Furthermore, BS.U must also use a statistically hiding but computationally binding commitment scheme in order to keep the message hidden from BS.S when a protocol restart is required.

BLAZE$^+$ is basically developed by applying our tree of commitments technique introduced in Chapter 4 to BLAZE. In particular, protocol restarts induced by BS.U are completely removed. This is accomplished by generating a large enough number of masking vectors all at once such that blinding the signature created by BS.S can be achieved with a probability very close to 1, *e.g.*, $1 - 2^{-80}$. This allows to safely eliminate the last move of the signing protocol including the proof of failure, and obtain a three-move protocol. Consequently, the statistically hiding but computationally binding commitment scheme that is used to keep the message hidden over restarts is not required anymore.

Furthermore and unlike BLAZE, the secret key of BLAZE$^+$ is Gaussian distributed, and both BS.S and BS.U sample the masking vectors from the Gaussian distribution rather than the uniform distribution over different subsets of R_q. Using the Gaussian distribution for masking allows to generate blind signatures with smaller size. We remark that exploiting this fact in BLAZE may lead the verification algorithm to accept even if the security check carried out by BS.U fails. This allows a (malicious) user to request a protocol restart in order to obtain another signature.

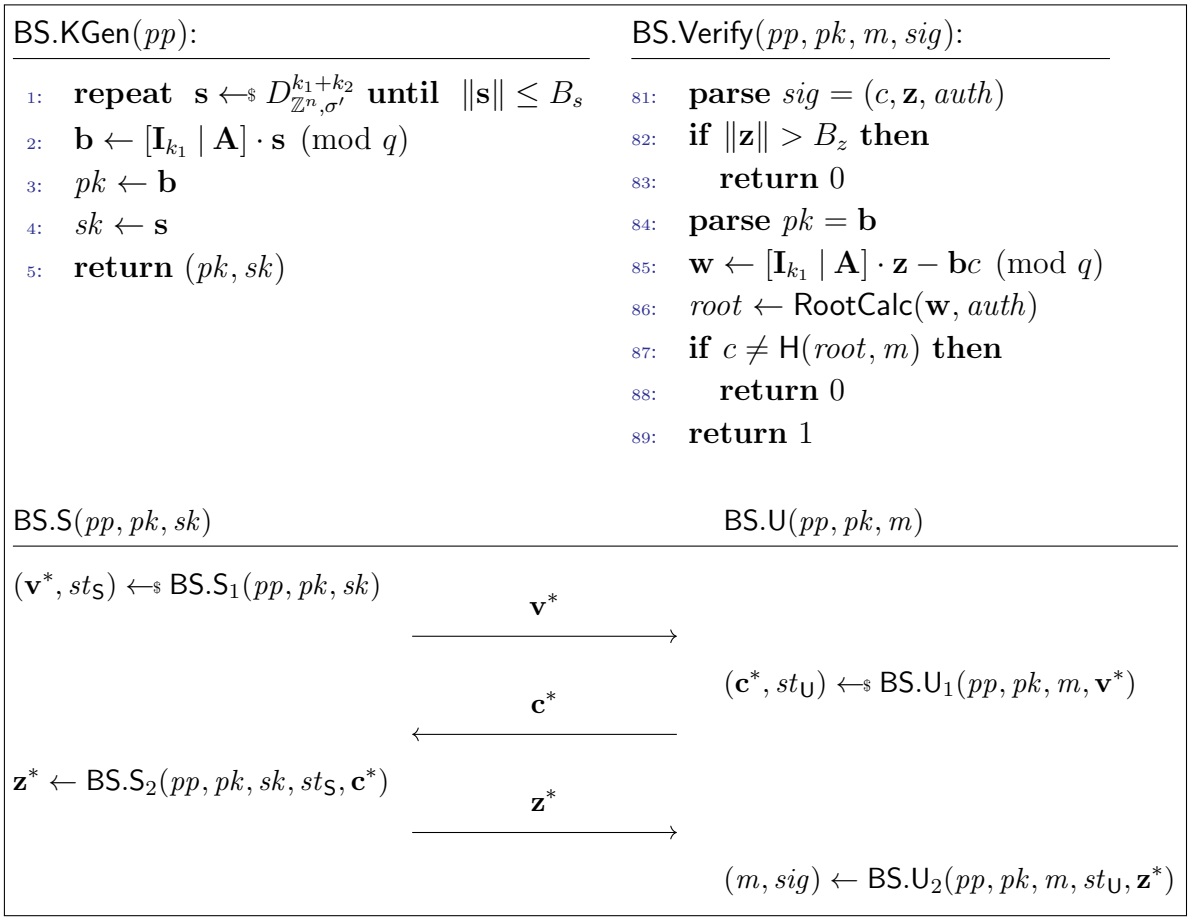

Figure 5.5: Overview of the blind signature scheme BLAZE^+. The parameter generation algorithm $\mathsf{BS.PGen}$ is explained in the text. The algorithms $\mathsf{BS.S}, \mathsf{BS.U}$ are formalized in Figure 5.6.

We give a detailed description of BLAZE^+ in Section 5.4.1 including a proof of its correctness. Then, we analyze its statistical blindness and computational one-more unforgeability in Section 5.4.2.

5.4.1 Description of the Scheme

We let $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ and $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ be two cryptographic hash functions. The first one is used to construct trees of commitments (see Section 4.2), and the second one computes the hash values that are part of blind signatures. In the following we give a detailed description of BLAZE^+. The respective algorithms of the scheme are formally given in Figures 5.5 and 5.6.

Parameter generation. Given 1^λ, $\mathsf{BS.PGen}$ generates a set of public parameters given by

$$pp = \big(1^\lambda, n, k_1, k_2, q, \sigma', B_s, \kappa, \sigma^*, \ell, \sigma, M, B_{z^*}, U, h, B_z, \ell_\mathsf{F}, \mathbf{A}\big),$$

where the matrix $\mathbf{A}$ is chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. The remaining parameters are generated as specified in Table 5.2 and in the following.

Key generation. Given a set of public parameters pp, BS.KGen keeps sampling a vector $\mathbf{s}$ from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma'}^{k_1+k_2}$ until it satisfies $\|\mathbf{s}\| \leq B_s$, where the bound B_s is set to $B_s = \eta'\sigma'\sqrt{(k_1+k_2)n}$ for $\eta' > 0$ (see Lemma 2.6). Then, it computes the vector $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. The public and secret keys are given by the pair $(pk = \mathbf{b}, sk = \mathbf{s})$. The condition $\|\mathbf{s}\| \leq B_s$ represents a trade-off between the speed of generating keys and the size of signatures. A closer look at Table 5.2 reveals that a smaller value of η' decreases σ, where σ is the standard deviation of the vector $\mathbf{z}$ included in a blind signature. However, a small value of η' reduces the success probability of passing the condition $\|\mathbf{s}\| \leq B_s$, which may require to repeat sampling the vector $\mathbf{s}$. Note that the secret key can also be sampled from the uniform distribution over the set $S_s^{k_1+k_2}$ as specified in BLAZE.

Signing. The signing protocol of BLAZE$^+$ is initiated by the signer BS.S, and it consists of three moves.

Given (pp, pk, sk), $\mathsf{BS.S}_1$ computes $\mathbf{v}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$ for all $j \in [\kappa]$, and sends the vector $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$ to $\mathsf{BS.U}_1$, where $\mathbf{y}_j$ are masking vectors sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$.

Let m be the message being blindly signed by BS.S. On input $(pp, pk, m, \mathbf{v}^*)$, $\mathsf{BS.U}_1$ selects a random vector $\mathbf{p} = (p_1, \ldots, p_\kappa)$ from the uniform distribution over $\mathbb{T}^\kappa$. Then, it computes the vector $\mathbf{v}' = \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$, and subsequently the commitments $\mathbf{v}^{(k)} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} + \mathbf{v}' \pmod q$ for all $k \in \{0, \ldots, \ell - 1\}$, where $\mathbf{e}^{(k)}$ are masking vectors sampled from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$. These commitments are then used to generate a tree of commitments of height $h = \lceil \log(\ell) \rceil$ via the algorithm HashTree. This algorithm outputs $(root, tree)$, where $root$ is the root of the tree and $tree$ is the sequence of all other nodes of the tree. This root together with the message m are used as input to compute the hash value c via the function H. Similar to BLAZE, the partitioning and permutation technique (*cf.* Section 5.2) is then employed in order to blind c without the need to carry out any security check or trigger any restart. That is, $\mathsf{BS.U}_1$ splits c into the partitions $c_j \in \mathbb{T}$ such that $c = \sum_{j=1}^{\kappa} c_j$ and the j^{th} partition c_j consists of the j^{th} non-zero entry of c at exactly the same position. After that, $\mathsf{BS.U}_1$ masks the partitions c_j by computing $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$, where $c_j^* = c_j p_j^{-1}$ for all $j \in [\kappa]$. Then, it sends the vector $\mathbf{c}^*$ to $\mathsf{BS.S}_2$. Note that the generation of all $\mathbf{e}^{(k)}$ as well as performing the multiplications $[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} \pmod q$ can be precomputed by the user before starting the protocol with the signer. We further note that all $\mathbf{e}^{(k)}$ can be reused in case a protocol restart is triggered by $\mathsf{BS.S}_2$ (see below), because only the blind vector $\mathbf{c}^*$ is revealed, which will be different in each restart. This saves sampling a large number of Gaussian vectors over restarts.

Upon receiving $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$, $\mathsf{BS.S}_2$ uses the partitions c_j^* to compute the vector $\mathbf{z}^* = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$, where $\mathbf{z}_j = \mathbf{y}_j + \mathbf{s}c_j^*$ for all $j \in [\kappa]$. Then, it checks if $\mathbf{z}^*$ does not leak any information about sk. This is established by applying the rejection sampling procedure Rej on input $(pp, \mathbf{z}^*, \mathbf{s}^*)$, where $\mathbf{s}^* = (\mathbf{s}c_1^*, \ldots, \mathbf{s}c_\kappa^*)$, and verifying that $\mathbf{z}^*$ follows the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{(k_1+k_2)\kappa}$. If rejection sampling does not accept, then $\mathsf{BS.S}_2$ restarts the signing protocol. Otherwise, $\mathsf{BS.S}_2$ sends $\mathbf{z}^*$ to $\mathsf{BS.U}_2$.

$\mathsf{BS.S}_1(pp, pk, sk)$:

11: **for** $j = 1$ **to** κ **do**
12: $\quad \mathbf{y}_j \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n,\sigma^*}$
13: $\quad \mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
14: $\mathbf{y} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
15: $\mathbf{v}^* \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
16: $st_\mathsf{S} \leftarrow \mathbf{y}$
17: **return** $(\mathbf{v}^*, st_\mathsf{S})$

$\mathsf{BS.U}_1(pp, pk, m, \mathbf{v}^*)$:

21: **parse** $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
22: $\mathbf{p} = (p_1, \ldots, p_\kappa) \leftarrow\!\!\$\ \mathbb{T}^\kappa$
23: $\mathbf{v}' \leftarrow \sum_{j=1}^{\kappa} \mathbf{v}_j p_j \pmod q$
24: **for** $k = 0$ **to** $\ell - 1$ **do**
25: $\quad \mathbf{e}^{(k)} \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n,\sigma}$
26: $\quad \mathbf{v}^{(k)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} + \mathbf{v}' \pmod q$
27: $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\ell-1)})$
28: $c = \sum_{j=1}^{\kappa} c_j \leftarrow \mathsf{H}(root, m)$ // $c_j \in \mathbb{T}$
29: $\mathbf{c}^* \leftarrow (c_1 p_1^{-1}, \ldots, c_\kappa p_\kappa^{-1})$
30: $\mathbf{e} \leftarrow (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
31: $st_\mathsf{U} \leftarrow (\mathbf{v}^*, \mathbf{p}, \mathbf{e}, tree, c, \mathbf{c}^*)$
32: **return** $(\mathbf{c}^*, st_\mathsf{U})$

$\mathsf{IterateRej}(pp, \mathbf{e}, \mathbf{z}')$:

71: **parse** $\mathbf{e} = (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
72: **for** $k = 0$ **to** $\ell - 1$ **do**
73: $\quad \mathbf{z} \leftarrow \mathbf{e}^{(k)} + \mathbf{z}'$
74: $\quad$ **if** $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{z}') = 1$ **then**
75: $\quad\quad$ **return** $(\mathbf{z}, k)$
76: **return** $(\bot, \bot)$

$\mathsf{BS.S}_2(pp, pk, sk, st_\mathsf{S}, \mathbf{c}^*)$:

41: **parse** $sk = \mathbf{s}$
42: **parse** $st_\mathsf{S} = \mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
43: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$ // $c_j^* \in \mathbb{T}$
44: **for** $j = 1$ **to** κ **do**
45: $\quad \mathbf{z}_j \leftarrow \mathbf{y}_j + \mathbf{s}c_j^*$
46: $\mathbf{z}^* \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
47: $\mathbf{s}^* \leftarrow (\mathbf{s}c_1^*, \ldots, \mathbf{s}c_\kappa^*)$
48: **if** $\mathsf{Rej}(pp, \mathbf{z}^*, \mathbf{s}^*) = 0$ **then**
49: $\quad$ **return** $\bot$
50: **return** $\mathbf{z}^*$

$\mathsf{BS.U}_2(pp, pk, m, st_\mathsf{U}, \mathbf{z}^*)$:

51: **if** $\|\mathbf{z}^*\| > B_{z^*}$ **then**
52: $\quad$ **return** $(\bot, \bot)$
53: **parse** $st_\mathsf{U} = (\mathbf{v}^*, \mathbf{p}, \mathbf{e}, tree, c, \mathbf{c}^*)$
54: **parse** $pk = \mathbf{b}$
55: **parse** $\mathbf{v}^* = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
56: **parse** $\mathbf{p} = (p_1, \ldots, p_\kappa)$
57: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$
58: **parse** $\mathbf{z}^* = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
59: **for** $j = 1$ **to** κ **do**
60: $\quad \mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j^* \pmod q$
61: $\quad$ **if** $\mathbf{v}_j \neq \mathbf{w}_j$ **then**
62: $\quad\quad$ **return** $(\bot, \bot)$
63: $\mathbf{z}' \leftarrow \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$
64: $(\mathbf{z}, k) \leftarrow \mathsf{IterateRej}(pp, \mathbf{e}, \mathbf{z}')$
65: **if** $(\mathbf{z}, k) = (\bot, \bot)$ **then**
66: $\quad$ **return** $(\bot, \bot)$
67: $auth \leftarrow \mathsf{BuildAuth}(k, tree, h)$
68: $sig \leftarrow (c, \mathbf{z}, auth)$
69: **return** (m, sig)

Figure 5.6: A formal description of the algorithms BS.S and BS.U of BLAZE$^+$ (*cf.* Figure 5.5). The signer restarts the protocol if $\mathsf{BS.S}_2$ returns $\bot$.

Table 5.2: A review of the parameters of BLAZE^+.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 1 \pmod{2n}$
σ'	Standard deviation of $\mathbf{s}$	$\sigma' > 0$
κ	Specifies the set $\mathbb{T}_\kappa^n$	$2^\kappa \binom{n}{\kappa} \geq 2^\lambda$
σ^*	Standard deviation of $\mathbf{z}^*$	$\sigma^* = \alpha^* \sqrt{\kappa} B_s$, $\alpha^* > 0$
σ	Standard deviation of $\mathbf{z}$	$\sigma = \alpha\eta \frac{B_{z^*}}{\eta^*}$, $U = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$, $\alpha > 0$, $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell \leq \delta$, $\delta > 0$
ℓ	No. masking vectors $\mathbf{e}^{(k)}$	$\ell \in \mathbb{N}_{>1}$
h	Tree height	$h = \lceil \log(\ell) \rceil$
M	No. restarts of BS	$M = \exp\left(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\right)$
B_s	Bound of $\|\mathbf{s}\|$	$B_s = \eta' \sigma' \sqrt{(k_1 + k_2)n}$, $\eta' > 0$
B_{z^*}	Bound of $\|\mathbf{z}^*\|$	$B_{z^*} = \eta^* \sigma^* \sqrt{(k_1 + k_2)\kappa n}$, $\eta^* > 0$
B_z	Bound of $\|\mathbf{z}\|$	$B_z = \eta\sigma \sqrt{(k_1 + k_2)n}$, $\eta > 0$
ℓ_F	Output length of F	$\ell_\mathsf{F} \geq 2\lambda$

Given $\mathbf{z}^*$, $\mathsf{BS.U}_2$ verifies that $\|\mathbf{z}^*\| \leq B_{z^*}$, and that $\mathbf{v}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j^* \pmod{q}$ for all $j \in [\kappa]$. Then, it computes the vector $\mathbf{z}' = \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$ and runs the algorithm $\mathsf{IterateRej}$ on input $(pp, \mathbf{e}, \mathbf{z}')$, where $\mathbf{e} = (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$. This algorithm repeatedly keeps applying rejection sampling on input $\left(pp, \mathbf{e}^{(k)} + \mathbf{z}', \mathbf{z}'\right)$ until it accepts for some masking vector $\mathbf{e}^{(k)}$, where $k \in \{0, \ldots, \ell - 1\}$. The algorithm $\mathsf{IterateRej}$ outputs $(\mathbf{z}, k)$, where $\mathbf{z} = \mathbf{e}^{(k)} + \mathbf{z}'$ and k corresponds to the index of the masking vector $\mathbf{e}^{(k)}$, for which $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{z}') = 1$. After that, $\mathsf{BS.U}_2$ computes the authentication path $auth$ associated to the index k by using the algorithm $\mathsf{BuildAuth}$ on input $(k, tree, h)$. This allows to compute the root of the corresponding tree of commitments, and hence verifying the blind signature. Finally, $\mathsf{BS.U}_2$ returns the message m and its blind signature $sig = (c, \mathbf{z}, auth)$. Note that the number ℓ of masking vectors is chosen large enough such that the algorithm $\mathsf{IterateRej}$ returns $(\perp, \perp)$ with a probability very close to zero, *e.g.*, 2^{-80}.

Verification. On input (pp, pk, m, sig), $\mathsf{BS.Verify}$ first checks that $\|\mathbf{z}\| \leq B_z$ and rejects if this is not the case. Otherwise, it computes $\mathbf{w} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod{q}$, and runs the algorithm $\mathsf{RootCalc}$ on input $(\mathbf{w}, auth)$ to compute the root of the tree of commitments that includes the leaf $\mathsf{F}(\mathbf{w})$ and its authentication path $auth$. The algorithm accepts if and only if $c = \mathsf{H}(root, m)$.

The following theorem states the correctness of BLAZE^+.

Theorem 5.11. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be two cryptographic hash functions. The blind signature scheme* BLAZE^+ *is* $\mathrm{corr}_{\mathsf{BS}}$*-correct w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$*, where*

$$\mathrm{corr}_{\mathsf{BS}} = 1 - \frac{1}{M} + \varepsilon^* + \delta + \varepsilon.$$

The term $1 - 1/M$ *defines the probability that* $\mathsf{BS.S}_2$ *returns* $\perp$*,* ε^* *is the probability that* $\|\mathbf{z}^*\| > B_{z^*}$*,* δ *is the probability that* IterateRej *returns* $(\perp, \perp)$*, and* ε *is the probability that* $\|\mathbf{z}\| > B_z$*,* i.e.*, the probability that* BS.Verify *rejects a valid blind signature.*

Proof. Let $sig = (c, \mathbf{z}, auth)$ be a valid blind signature that is generated by the signing protocol of BLAZE^+ on a message m. Then, the vector $\mathbf{z}$ is distributed according to the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$. By Lemma 2.6, $\|\mathbf{z}\| > B_z = \eta\sigma\sqrt{(k_1+k_2)n}$ with probability

$$\varepsilon = \eta^{(k_1+k_2)n} \exp\left(\frac{(k_1+k_2)n}{2}\left(1-\eta^2\right)\right),$$

where $\eta > 0$. By a suitable choice of η we obtain $\|\mathbf{z}\| \leq B_z$ with probability at least $1-\varepsilon$, *e.g.*, $1 - 2^{-80}$.

Similarly, after receiving the vector $\mathbf{z}^*$, the algorithm $\mathsf{BS.U}_2$ verifies that $\|\mathbf{z}^*\| \leq B_{z^*}$. By Lemma 2.6, the probability that this condition does not hold is given by

$$\varepsilon^* = \eta^{*(k_1+k_2)\kappa n} \exp\left(\frac{(k_1+k_2)\kappa n}{2}\left(1-\eta^{*2}\right)\right),$$

where $\eta^* > 0$ can be chosen such that $\|\mathbf{z}^*\| > B_{z^*}$ with probability at most ε^*.

Next, we show that the condition $c = \mathsf{H}(root, m)$ in the verification algorithm is also satisfied. To this end, we first show that the vector $\mathbf{w}$ computed in BS.Verify matches the commitment $\mathbf{v}$ that is used in $\mathsf{BS.U}_1$ to build the related tree of commitments with authentication path *auth*, *i.e.*,

$$\begin{aligned}
\mathbf{w} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \left(\mathbf{e}^{(k)} + \sum_{j=1}^{\kappa} \mathbf{z}_j p_j\right) - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \left(\mathbf{e}^{(k)} + \sum_{j=1}^{\kappa} \left(\mathbf{y}_j + \mathbf{s}c_j^*\right) p_j\right) - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} + \sum_{j=1}^{\kappa} ([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j) p_j + \sum_{j=1}^{\kappa} ([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}) c_j^* p_j - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} + \sum_{j=1}^{\kappa} \mathbf{v}_j p_j + \sum_{j=1}^{\kappa} \mathbf{b}c_j - \mathbf{b}c \\
&= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}^{(k)} + \mathbf{v}' \\
&= \mathbf{v}^{(k)} \pmod q.
\end{aligned}$$

Therefore, given the commitment $\mathbf{w}$ and its correct authentication path *auth*, the algorithm RootCalc returns the correct root of the related tree of commitments.

Furthermore, observe that $\mathsf{BS.S}_2$ returns $\perp$, *i.e.*, it restarts the signing protocol, if $\mathsf{Rej}(pp, \mathbf{z}^*, \mathbf{s}^*) = 0$ (see Line 48 in Figure 5.6). By Lemma 2.8, this occurs with probability $1 - 1/M$, where $M = \exp\left(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\right)$ is the expected number of protocol restarts and $\sigma^* = \alpha^* \|\mathbf{s}^*\|$. Since $\|\mathbf{s}^*\| \leq \sqrt{\kappa} B_s$, we set $\sigma^* = \alpha^* \sqrt{\kappa} B_s$.

Finally, $\mathsf{BS.U}_2$ returns $(\perp, \perp)$ if $\mathsf{IterateRej}$ returns $(\perp, \perp)$. This event occurs with probability at most δ. More concretely, Lemma 2.8 states that when using only one masking vector, say $\mathbf{e}^{(0)}$, we have $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{z}') = 1$ with probability $(1 - 2^{-100})/U$ after an average number of $U = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$ restarts, where $\mathbf{z} = \mathbf{e}^{(0)} + \mathbf{z}'$ and $\sigma = \alpha \|\mathbf{z}'\|$. By Lemma 2.7, the vector $\mathbf{z}' = \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$ follows the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n, \sqrt{\kappa}\sigma^*}$, since the vectors $\mathbf{z}_j$ are distributed according to $D^{k_1+k_2}_{\mathbb{Z}^n, \sigma^*}$. This means we have $\|\mathbf{z}'\| \leq \eta \frac{B_{z^*}}{\eta^*}$, and hence we set $\sigma = \alpha\eta \frac{B_{z^*}}{\eta^*}$. When using ℓ masking vectors, the algorithm $\mathsf{IterateRej}$ returns $(\mathbf{z}, k) \neq (\perp, \perp)$ with probability $1 - \left(1 - \frac{1-2^{-100}}{U}\right)^{\ell}$ after at most $1/(1-\delta) \approx 1$ restarts, where ℓ and δ are chosen such that $\left(1 - \frac{1-2^{-100}}{U}\right)^{\ell} \leq \delta$ and $\delta \approx 0$, *e.g.*, 2^{-80}. Hence, the correctness error of BLAZE$^+$ is at most $\mathrm{corr}_{\mathsf{BS}} = 1 - \frac{1}{M} + \varepsilon^* + \delta + \varepsilon$, and the total number of protocol restarts that is required to obtain a valid blind signature is given by M. □

5.4.2 Security Analysis

In this section we prove the statistical blindness of BLAZE$^+$ and provide our security argument about its computational one-more unforgeability.

Theorem 5.12. *Let* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}^n_\kappa$ *be two cryptographic hash functions modeled as random oracles. The blind signature scheme* BLAZE$^+$ *is* ε*-statistically blind w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM, where* $\varepsilon = \max\{2^{-100}/U, (2n)^{-\kappa}\}$.

Proof. In the blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ the adversarial signer S^* selects two messages m_0, m_1 and interacts with the honest user $\mathsf{BS.U}$ twice. We show that after both interactions, the messages output by the user, *i.e.*, two blind vectors of the form $\mathbf{c}^* \in \mathbb{T}^\kappa$ and two signatures of the form $sig = (c, \mathbf{z}, auth)$, are independently distributed and do not leak any information about the signed messages and the respective interaction.

The vector $\mathbf{c}^*$ and the polynomial c are uniformly distributed over $\mathbb{T}^\kappa$ and $\mathbb{T}^n_\kappa$, respectively. Hence, they do not leak any information. Moreover, Lemma 5.7 ensures that the vector $\mathbf{c}^*$ is independently distributed from the hash value c, and S^* can link both elements only with probability $(2n)^{-\kappa}$ over guessing. Furthermore, the authentication path *auth* includes hash values that are uniformly distributed over $\{0,1\}^{\ell_\mathsf{F}}$. By Lemma 2.8, the vector $\mathbf{z}$ is independently distributed from $\mathbf{z}' = \sum_{j=1}^{\kappa} \mathbf{z}_j p_j$ and completely masks $\mathbf{z}'$, and hence $\mathbf{z}^*$. By the same lemma, $\mathbf{z}$ is also within statistical distance of $2^{-100}/U$ from the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n, \sigma}$.

Finally, if a protocol restart is triggered by S^*, then $\mathsf{BS.U}$ generates fresh random elements. Therefore, the protocol executions are independent of each other, and hence S^* does not get any information about the message being signed. □

$\mathbf{Exp}^{\mathsf{Tree\text{-}SelfTargetMSIS}}_{\mathsf{A}^*}(pp)$:

1: **parse** $pp = (n, k_1, k_2, q, \kappa, \ell, \beta, \mathsf{F}, \mathsf{H})$
2: $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$
3: $(\mathbf{x} = [\mathbf{z} \mid c]^\top, auth, m) \leftarrow\!\!\$\ \mathsf{A}^{*\mathsf{F},\mathsf{H}}(pp, \mathbf{A})$ // $(\mathbf{x}, auth, m) \in R^{k_1+k_2+1} \times (\{0,1\}^{\ell_\mathsf{F}})^{\lceil \log(\ell) \rceil} \times \{0,1\}^*$
4: **if** $(\|\mathbf{x}\|_p \leq \beta) \wedge (c = \mathsf{H}(\mathsf{RootCalc}([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} \pmod q), auth), m))$ **then**
5: **return** 1
6: **return** 0

Figure 5.7: Definition of the experiment $\mathbf{Exp}^{\mathsf{Tree\text{-}SelfTargetMSIS}}_{\mathsf{A}^*}$. The algorithm RootCalc is given in Definition 4.2.

Conjecture 5.13. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be two cryptographic hash functions modeled as random oracles. The blind signature scheme* BLAZE^+ *is one-more unforgeable w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM as long as* MLWE^2 *is hard w.r.t.* $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$ *and* MSIS^2 *is hard w.r.t.* $pp'' = \left(n, k_1, k_2 + 1, q, 2\sqrt{B_z^2 + \kappa}\right)$.

Similar to our first blind signature scheme BLAZE, an attacker can attempt to recover the secret key sk, given the set of public parameters pp and public key pk, in order to forge blind signatures under BLAZE^+. This is based on the hardness of MLWE^2 w.r.t. $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$.

In the following we show that forging a signature without knowing the secret key implies solving MSIS^2 w.r.t. $pp'' = \left(n, k_1, k_2 + 1, q, 2\sqrt{B_z^2 + \kappa}\right)$. To this end, we define an extended version of the problem SelfTargetMSIS (see Definition 2.11). We denote this extended version by Tree-SelfTargetMSIS.

Definition 5.14. Let $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ and $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa^n$ be two cryptographic hash functions. Furthermore, let $pp = (n, k_1, k_2, q, \kappa, \ell, \beta, \mathsf{F}, \mathsf{H})$, where $n, k_1, k_2, q, \kappa, \ell \in \mathbb{N}_{>0}$ and $\beta \in \mathbb{R}_{>0}$. We say that the Tree-SelfTargetMSIS problem is (t, ε)-*hard w.r.t.* pp if, for every algorithm A^* running in time at most t, we have

$$\mathbf{Adv}^{\mathsf{Tree\text{-}SelfTargetMSIS}}_{\mathsf{A}^*}(pp) := \Pr\left[\mathbf{Exp}^{\mathsf{Tree\text{-}SelfTargetMSIS}}_{\mathsf{A}^*}(pp) = 1\right] \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathsf{Tree\text{-}SelfTargetMSIS}}_{\mathsf{A}^*}$ is depicted in Figure 5.7.

Unlike SelfTargetMSIS, the problem Tree-SelfTargetMSIS does not define the "commitment" vector $[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} \pmod q$ as the left part of the input to the hash function H. Rather, its hash value under the hash function F represents a leaf of a tree of commitments, and the root of this tree is the left part of the input to H. Therefore, a solution to Tree-SelfTargetMSIS w.r.t. $pp = (n, k_1, k_2, q, \kappa, \ell, \beta, \mathsf{F}, \mathsf{H})$ further includes an authentication path $auth$ of this leaf, which together with the commitment allows to compute the root of the tree via the algorithm RootCalc. In the following lemma we show that Tree-SelfTargetMSIS is at least as hard as MSIS, when modeling the functions F and H as random oracles.

Lemma 5.15. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ *be two cryptographic hash functions modeled as random oracles. Furthermore, let* $n, k_1, k_2, q, \kappa, \ell \in \mathbb{N}_{>0}$, $\beta \in \mathbb{R}_{>0}$, $\ell_\mathsf{F} \geq 2\lambda$, *and* $2^\kappa \binom{n}{\kappa} \geq 2^\lambda$. *Define by* q_F *and* q_H *the maximum number of hash queries that can be made to the oracles* O_F *and* O_H, *respectively. The problem* Tree-SelfTargetMSIS *is* (t,ε)*-hard w.r.t.* $pp = (n, k_1, k_2, q, \kappa, \ell, \beta, \mathsf{F}, \mathsf{H})$ *in the ROM if the problem* MSIS *is* (t',ε')*-hard w.r.t.* $pp' = (n, k_1, k_2, q, 2\beta)$, *where*

$$t' \approx 2t \;\wedge\; \varepsilon' \approx \left(\varepsilon - \frac{q_\mathsf{F}^2 + q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{1}{|\mathbb{T}_\kappa^n|}\right) \cdot \left(\frac{\varepsilon - \frac{q_\mathsf{F}^2+q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{1}{|\mathbb{T}_\kappa^n|}}{q_\mathsf{H}\ell} - \frac{1}{|\mathbb{T}_\kappa^n|}\right).$$

Proof. Let A^* be an algorithm that runs in time t and solves Tree-SelfTargetMSIS w.r.t. $pp = (n, k_1, k_2, q, \kappa, \ell, \beta, \mathsf{F}, \mathsf{H})$ with probability ε. We construct a reduction algorithm R that runs A^* as a subroutine, and answers the hash queries made by A^* to the oracles O_F and O_H in order to solve MSIS w.r.t. $pp' = (n, k_1, k_2, q, 2\beta)$.

Given a uniformly random matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$, R runs A^* on input $\mathbf{A}$ as well. Queries to the oracles O_F and O_H are answered similar to the way explained in the proof of Theorem 4.3. Whenever A^* queries the oracle O_H on some input $(root, m)$, R answers with a uniformly random $c \in \mathbb{T}_\kappa^n$. However, queries to the oracle O_F are answered in a way that excludes collisions and chains (see Figure 4.5). This is within statistical distance of at most $\frac{q_\mathsf{F}^2+q_\mathsf{F}}{2^{\ell_\mathsf{F}}}$ from an oracle that allows the existence of collisions and chains.

Note that the probability that A^* succeeds in finding a solution $(\mathbf{x} = [\mathbf{z} \mid c]^\top, auth, m)$, where c corresponds to one of the random oracle queries made by A^* is at least $\varepsilon - 1/|\mathbb{T}_\kappa^n|$. When A^* returns a solution $(\mathbf{x} = [\mathbf{z} \mid c]^\top, auth, m)$ to Tree-SelfTargetMSIS w.r.t. pp, R invokes A^* again with the same random tape, but reprograms the query $\mathsf{O}_\mathsf{H}(root, m)$ to a different random answer $c' \neq c$, where $(root, m)$ corresponds to the solution $(\mathbf{x}, auth, m)$. By the forking lemma (Lemma 2.18), A^* returns a solution $(\mathbf{x}' = [\mathbf{z}' \mid c']^\top, auth', m')$ using the hash query $(root', m')$ and with the probability ε' that is given in the statement of this lemma, such that

$$(c \neq c') \wedge (root = root') \wedge (m = m') \wedge (k = k'),$$

where $k, k' \in \{0, \ldots, \ell - 1\}$ are the indices included in $auth, auth'$, respectively. Furthermore, answering queries to the oracle O_F as described in Figure 4.5 ensures that both $auth, auth'$ include the same sequence of hash values. Therefore, we have $auth = auth'$ and

$$[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x}' \pmod q \quad \Rightarrow \quad [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{x} - \mathbf{x}') = \mathbf{0} \pmod q.$$

Since $c \neq c'$, we have $\mathbf{x} \neq \mathbf{x}'$ and $0 < \|\mathbf{x} - \mathbf{x}'\|_p \leq 2\beta$. Therefore, R returns the non-zero solution $\mathbf{x} - \mathbf{x}'$ to MSIS w.r.t pp'. □

Finally, we complete our security argument for Conjecture 5.13, *i.e.*, we show how to solve MSIS^2 w.r.t. $pp'' = \left(n, k_1, k_2 + 1, q, 2\sqrt{B_z^2 + \kappa}\right)$, given a valid forgery $sig = (c, \mathbf{z}, auth)$ on a message m. From the verification algorithm we have

$$\|\mathbf{z}\| \leq B_z \text{ and } c = \mathsf{H}(\mathsf{RootCalc}([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q, auth), m).$$

By setting $\mathbf{A}' = [\mathbf{A} \mid -\mathbf{b}] \in R_q^{k_1 \times (k_2+1)}$ and $\mathbf{x} = [\mathbf{z} \mid c]^\top \in R^{k_1+k_2+1}$, where $\|\mathbf{x}\| \leq \sqrt{B_z^2 + \kappa}$, we can write

$$c = \mathsf{H}(\mathsf{RootCalc}([\mathbf{I}_{k_1} \mid \mathbf{A}'] \cdot \mathbf{x} \pmod q, \mathit{auth}), m).$$

Therefore, the tuple $(\mathbf{x}, \mathit{auth}, m)$ constitutes a solution to $\mathsf{Tree\text{-}SelfTargetMSIS}^2$ with respect to the set of parameters given by $\left(n, k_1, k_2+1, q, \kappa, \ell, \sqrt{B_z^2+\kappa}, \mathsf{F}, \mathsf{H}\right)$. By Lemma 5.15, this implies solving MSIS^2 w.r.t. $pp'' = \left(n, k_1, k_2+1, q, 2\sqrt{B_z^2+\kappa}\right)$.

5.5 The Blind Signature Scheme BlindOR

In this section we present our third blind signature scheme BlindOR. Recently, Hauck *et al.* [HKLN20] pointed out that the proof of the one-more unforgeability property that originally turns back to Pointcheval and Stern [PS00] and later adapted by Rückert to the lattice setting [Rüc10], has not been transformed correctly to a valid security reduction that replaces the discrete logarithm assumption with lattice assumptions. Indeed, the main idea of the reduction given in [PS00] is to select a secret key sk and then run the forger with the related public key pk, which represents an instance of a computationally hard problem that admits more than one solution. In other words, pk is related to more than one sk, and the forger cannot distinguish which sk is used by the reduction. Note that it is crucial for the reduction to know a secret key because, unlike standard Fiat-Shamir signature schemes, the signer cannot be simulated without knowledge of a secret key (otherwise the scheme would be universally forgeable [PS00]). After running the forger and obtaining an element x, the reduction rewinds the forger with the same random tape and partially different random oracle answers to obtain an element x'. Then, the proof in [PS00] uses a subtle argument to ensure that $x \neq x'$ with noticeable probability. This allows the reduction to compute a solution to the underlying instance of the discrete logarithm problem.

In lattice-based schemes so far, the security proof is a reduction from the SIS problem (or its ring variant RSIS) to the one-more unforgeability property. In this context, after obtaining x and x' from the forger, the reduction returns $x - x'$ as a non-zero solution to SIS (or RSIS). The problem, as discussed in [HKLN20], is that Rückert's argument is not sufficient to ensure that $x \neq x'$ with high probability, and further assumptions are required to guarantee that a transcript of the scheme with a given secret key sk can be preserved with high probability when switching to a different valid secret key. Based on this observation, Hauck *et al.* [HKLN20] extended the modular framework for blind signatures from linear functions given in [HKL19] to the lattice setting, and provide a proof that covers the missing argument.

Unfortunately, and as stated by the authors themselves, their framework is mostly of theoretical interest. Indeed, the solution presented in [HKLN20] to cover the missing argument in the security proof entails parameters of huge size and renders the scheme impractical. In particular, the RSIS-based instantiation given in [HKLN20] has public and secret keys of size 443.75 KB and 4275 KB, respectively, and generates blind signatures of size 7915.52 KB. This classifies all known (three-move and four-move) lattice-based constructions of blind signatures so far to schemes that either are not backed by a correct security proof, or need impractically huge parameters to obtain a provably secure instantiation.

As stated at the beginning of this chapter, our scheme BlindOR, which is simultaneously practical and provably secure, makes use of OR-proofs introduced by Cramer *et al.* [CDS94]. An OR-proof is a Sigma protocol that allows to prove the knowledge of a witness for one of two (or more) statements, without revealing which one. BlindOR shows how to construct provably secure blind signatures from lattice-based OR-proofs in a way that allows to sidestep the missing security argument pointed out in [HKLN20]. The public key of BlindOR consists of two statements (two instances of a hard lattice problem), and the secret key includes a witness for one of them. Consequently, the hardness assumption underlying the public key does not have to "natively" admit multiple solutions because the OR-proof already takes care of this. In particular, the public key of BlindOR consists of two instances of MLWE, which results in a much more efficient construction. Signing is carried out by proving the possession of the witness for one of the two instances of MLWE. This is the witness included in the secret key. A user interacting with the signer blinds the two transcripts generated by the signer without being able to distinguish for which instance the signer holds a witness. We capture these blinding steps in a set of algorithms and show that BlindOR is statistically blind. The one-more unforgeability property of our scheme is proven in the ROM assuming the hardness of both MLWE and MSIS. The security reduction creates one instance of the hard problem with a witness in order to simulate the signing oracle, and tries to solve the other instance, which is given to the reduction as input. That is, the reduction does not know a witness for its input. This is analogous to the security proof of ordinary signature schemes based on both MLWE and MSIS, and hence no further conditions on the parameters are required to ensure the correctness and success of the reduction with high probability. This is in contrast to all previous three-move and four-move lattice-based constructions of blind signatures, as observed in [HKLN20].

Similar to BLAZE and BLAZE$^+$, BlindOR also uses the partitioning and permutation technique, which allows to remove a rejection sampling procedure in the first user stage. Another advantage of using this technique is that it can be used to construct OR-proofs based on lattice assumptions, because it allows to use the set $\mathbb{T}^\kappa$ as a challenge space. In contrast to the typical challenge space $\mathbb{T}_\kappa^n$ that is used in current lattice-based schemes, the set $\mathbb{T}^\kappa$ defines an abelian group, which is a crucial requirement for OR-proofs. Furthermore and similar to BLAZE$^+$, the tree of commitments technique is used in BlindOR in order to remove the restarts induced by the user when blinding the signature generated by the signer. We extend the employment of this technique in BlindOR to further reduce (remove) the potential restarts induced by the signer when computing signatures, which must be distributed independently from the secret key.

In summary, although BlindOR requires twice as many public key and signature parts, which is inherent in OR-proofs, it yields much smaller sizes compared to the provably secure construction due to [HKLN20], resulting in a practical scheme overall.

Finally, we remark that the design of BlindOR is inspired by Abe and Okamoto [AO00], who used OR-proofs to build partially blind signatures with security based on the hardness of the discrete logarithm problem. Informally, a partially blind signature scheme, first introduced by Abe and Fujisaki [AF96], allows to include common information *info*, *e.g.*, the date of issue, to the blind signature under some agreement between the signer and user. Unlike our construction, the public key of their scheme consists of only one instance of the hard problem. The second instance is created within the signing protocol by embedding

info into a cryptographic hash function that transforms *info* into a random public key whose secret key in unknown to anybody. Therefore, the second problem instance depends on *info*, and hence the user knows for which instance the signer holds a witness. Observe that we cannot simply convert their scheme to the lattice setting because this would force us to use MSIS (instead of MLWE) and result in an inefficient scheme, as we must consider the additional requirements on the parameters that are given in [HKLN20] to obtain a gap-free security reduction. The change to MLWE is possible because there is no common information to consider in our case.

In Section 5.5.1 we recall the syntax of Sigma protocols and OR-proofs. In Section 5.5.2 we present the Sigma protocol underlying our scheme BlindOR. The description of BlindOR is given Section 5.5.3, and its security analysis in Section 5.5.4.

5.5.1 Sigma Protocols and OR-Proofs

In this section we give a formal definition of Sigma protocols and OR-proofs. We first define the synatx of the relations, for which Sigma protocols and OR-proofs are defined.

Definition 5.16. A *relation* is a tuple

$$\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen}),$$

where:

$\mathcal{R}.\mathsf{PGen}(1^\lambda)$ is a PPT parameter generation algorithm that, on input the security parameter λ, returns a set of public parameters pp, which implicitly contains λ in unary.

$\mathcal{R}.\mathsf{RSet}(pp)$ is a set called the *relation set*. It is indexed by a set of public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, and defines a collection of sets.

$\mathcal{R}.\mathsf{IGen}(pp, b)$ is a PPT instance generator algorithm that, on input a set of public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$ and $b \in \{0,1\}$, returns a pair $(x, w) \in \mathcal{R}.\mathsf{RSet}(pp)$ if $b = 1$, and x if $b = 0$. If $b = 1$, then x is called a *yes-instance* for $\mathcal{R}$ w.r.t. pp, and w a corresponding *witness*. Otherwise, x is called a *no-instance* for $\mathcal{R}$ w.r.t. pp.

Next, we define the OR-relation $\mathcal{R}_{\mathsf{OR}}$ on a relation $\mathcal{R}$. For a set of public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, a yes-instance for $\mathcal{R}_{\mathsf{OR}}$ w.r.t. pp is a pair (x_0, x_1), where both x_0 and x_1 are yes-instances for $\mathcal{R}$ w.r.t. pp. A witness for such an instance is a witness for one of the two coordinates, *i.e.*, a pair (d, w), where $d \in \{0,1\}$ and w is a witness for x_d. On the other hand, a no-instance for $\mathcal{R}_{\mathsf{OR}}$ w.r.t. pp consists of a pair (x_0, x_1), where at least one coordinate is a no-instance for $\mathcal{R}$ w.r.t. pp.

Definition 5.17. Let $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ be a relation. The *OR-relation on* $\mathcal{R}$ is a tuple

$$\mathcal{R}_{\mathsf{OR}} = (\mathcal{R}_{\mathsf{OR}}.\mathsf{PGen}, \mathcal{R}_{\mathsf{OR}}.\mathsf{RSet}, \mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}),$$

where its parameter generation algorithm is defined by $\mathcal{R}_{\mathsf{OR}}.\mathsf{PGen} := \mathcal{R}.\mathsf{PGen}$, its relation set is given by

$$\mathcal{R}_{\mathsf{OR}}.\mathsf{RSet}(pp) := \{((x_0, x_1), (d, w)) \mid (x_d, w), (x_{1-d}, \cdot) \in \mathcal{R}.\mathsf{IGen}(pp, 1)\},$$

where $pp \in \mathcal{R}_{\mathsf{OR}}.\mathsf{PGen}(1^\lambda)$, and its instance generator algorithm is depicted in Figure 5.8.

$\mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}(pp, b)$:

1: **if** $b = 0$ **then**
2: $d, d' \leftarrow\!\!\$\ \{0,1\}$
3: $x_d \leftarrow\!\!\$\ \mathcal{R}.\mathsf{IGen}(pp, 0)$
4: $x_{1-d} \leftarrow\!\!\$\ \mathcal{R}.\mathsf{IGen}(pp, d')$
5: **return** (x_0, x_1)
6: **else**
7: $d \leftarrow\!\!\$\ \{0,1\}$
8: $(x_0, w_0) \leftarrow\!\!\$\ \mathcal{R}.\mathsf{IGen}(pp, 1)$
9: $(x_1, w_1) \leftarrow\!\!\$\ \mathcal{R}.\mathsf{IGen}(pp, 1)$
10: $w \leftarrow w_d$
11: **return** $((x_0, x_1), (d, w))$

Figure 5.8: Definition of the instance generator algorithm $\mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}$ of the relation $\mathcal{R}_{\mathsf{OR}}$. Note that in Line 4 we slightly abuse notation. That is, if $d' = 1$ (*i.e.*, the instance generator creates a yes-instance), we only consider the first component of the output and ignore the witness in the second coordinate.

We now proceed to the definition of a Sigma protocol for a relation $\mathcal{R}$, first introduced by Cramer [Cra96].

Definition 5.18. Let $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ be a relation and $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$. A *Sigma protocol for* $\mathcal{R}$ is a tuple of polynomial-time algorithms

$$\Sigma = (\Sigma.\mathsf{P}, \Sigma.\mathsf{V}, \Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}, \Sigma.\mathsf{Rec}),$$

where:

$\Sigma.\mathsf{P}(pp, x, w)$ is an interactive algorithm that is called *prover* and consists of two algorithms $\Sigma.\mathsf{P} = (\Sigma.\mathsf{P}_1, \Sigma.\mathsf{P}_2)$, where:

- $\Sigma.\mathsf{P}_1$ is a PPT algorithm that, on input the set of public parameters pp, a yes-instance x for $\mathcal{R}$ w.r.t. pp, and a corresponding witness w, returns a message cm, called the *commitment*, and a state $st_{\Sigma.\mathsf{P}}$, *i.e.*, $(cm, st_{\Sigma.\mathsf{P}}) \leftarrow\!\!\$\ \Sigma.\mathsf{P}_1(pp, x, w)$.
- $\Sigma.\mathsf{P}_2$ is a DPT algorithm that, on input the set of public parameters pp, an instance-witness pair (x, w) for $\mathcal{R}$ w.r.t. pp, a state $st_{\Sigma.\mathsf{P}}$, and a verifier message ch, returns a message rp, called the *response*, *i.e.*, $rp \leftarrow \Sigma.\mathsf{P}_2(pp, x, w, st_{\Sigma.\mathsf{P}}, ch)$. We write $rp = \perp$ to denote failure.

$\Sigma.\mathsf{V}(pp, x)$ is an interactive algorithm that is called *verifier* and consists of two algorithms $\Sigma.\mathsf{V} = (\Sigma.\mathsf{V}_1, \Sigma.\mathsf{V}_2)$, where:

- $\Sigma.\mathsf{V}_1$ is a PPT algorithm that, on input the set of public parameters pp, an instance x for $\mathcal{R}$ w.r.t. pp, and a commitment cm, returns a message ch, called the *challenge*, and a state $st_{\Sigma.\mathsf{V}} = (cm, ch)$, *i.e.*, $(ch, st_{\Sigma.\mathsf{V}}) \leftarrow\!\!\$\ \Sigma.\mathsf{V}_1(pp, x, cm)$,

where $st_{\Sigma.\mathsf{V}} = (cm, ch)$. The challenge ch is sampled from the uniform distribution over a finite abelian group $\mathcal{C}$, called the *challenge space*. We assume that pp implicitly defines $\mathcal{C}$.

- $\Sigma.\mathsf{V}_2$ is a DPT algorithm that, on input the set of public parameters pp, an instance x for $\mathcal{R}$ w.r.t. pp, a state $st_{\Sigma.\mathsf{V}} = (cm, ch)$, and a response rp, returns a pair (b, int), where $b \in \{0, 1\}$ and $int \in \mathbb{Z}$, *i.e.*, $(b, int) \leftarrow \Sigma.\mathsf{V}_2(pp, x, st_{\Sigma.\mathsf{V}}, rp)$. We say that the verifier accepts the transcript (cm, ch, rp) if $b = 1$, and that it rejects if $b = 0$.

$\Sigma.\mathsf{Sim}(pp, x, ch)$ is a PPT algorithm called *simulator*. On input the set of public parameters pp, an instance x for $\mathcal{R}$ w.r.t. pp, and a challenge $ch \in \mathcal{C}$, returns a commitment cm and a response rp, *i.e.*, $(cm, rp) \leftarrow\!\!\$\ \Sigma.\mathsf{Sim}(pp, x, ch)$. We write $(cm, rp) = (\bot, \bot)$ to denote failure.

$\Sigma.\mathsf{Ext}(pp, x, (cm, ch, rp), (cm, ch', rp'))$ is a DPT algorithm called *extractor*. On input the set of public parameters pp, an instance x for $\mathcal{R}$ w.r.t. pp, and two transcripts (cm, ch, rp) and (cm, ch', rp') such that $ch \neq ch'$ and $\Sigma.\mathsf{V}_2$ returns the same output $(1, int)$ for both transcripts, $\Sigma.\mathsf{Ext}$ returns a witness w such that $(x, w) \in \mathcal{R}.\mathsf{RSet}(pp)$.

$\Sigma.\mathsf{Rec}(pp, x, ch, rp)$ is a DPT algorithm called the *commitment recovering* algorithm. On input the set of public parameters pp, an instance x for $\mathcal{R}$ w.r.t. pp, a challenge ch from the challenge space $\mathcal{C}$, and a response rp, the algorithm returns a commitment cm, *i.e.*, $cm \leftarrow \Sigma.\mathsf{Rec}(pp, x, ch, rp)$. We write $cm = \bot$ to denote failure.

Let $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ be a relation and $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$ be a set of public parameters. Let $\Sigma = (\Sigma.\mathsf{P}, \Sigma.\mathsf{V}, \Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}, \Sigma.\mathsf{Rec})$ be a Sigma protocol for $\mathcal{R}$. We write $(view, (b, int)) \leftarrow\!\!\$\ \langle \Sigma.\mathsf{P}(pp, x, w), \Sigma.\mathsf{V}(pp, x) \rangle$ to denote the joint execution of $\Sigma.\mathsf{P}$ and $\Sigma.\mathsf{V}$ in the Sigma protocol Σ with private inputs (pp, x, w) for $\Sigma.\mathsf{P}$ and (pp, x) for $\Sigma.\mathsf{V}$, as well as private outputs $view$ for $\Sigma.\mathsf{P}$ and (b, int) for $\Sigma.\mathsf{V}$, where $view$ is possibly an empty string $null$, or a status message, *e.g.*, ok.

The Sigma protocols we consider must satisfy a few properties, which we collect in the following definition:

Definition 5.19. Let $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ be a relation, $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, and $\Sigma = (\Sigma.\mathsf{P}, \Sigma.\mathsf{V}, \Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}, \Sigma.\mathsf{Rec})$ be a Sigma protocol for $\mathcal{R}$.

1. We say that Σ is corr_Σ-*correct* w.r.t. pp if, for every $(x, w) \in \mathcal{R}.\mathsf{IGen}(pp, 1)$, we have

$$\Pr[(view, (0, int)) \leftarrow\!\!\$\ \langle \Sigma.\mathsf{P}(pp, x, w), \Sigma.\mathsf{V}(pp, x) \rangle] \leq \mathrm{corr}_\Sigma.$$

2. We say that $\Sigma.\mathsf{Rec}$ is $\mathrm{corr}_{\Sigma.\mathsf{Rec}}$-*correct* w.r.t. pp if, for every $(x, w) \in \mathcal{R}.\mathsf{IGen}(pp, 1)$, we have

$$\Pr\left[\mathbf{Exp}^{\mathrm{Rec}}_\Sigma(pp, x, w) = 0\right] \leq \mathrm{corr}_{\Sigma.\mathsf{Rec}},$$

where the experiment $\mathbf{Exp}^{\mathrm{Rec}}_\Sigma$ is depicted in Figure 5.9.

$\mathbf{Exp}^{\mathrm{Rec}}_{\Sigma}(pp, x, w)$:

1: $(cm, st_{\Sigma.\mathsf{P}}) \leftarrow\!\!\$\ \Sigma.\mathsf{P}_1(pp, x, w)$
2: $(ch, st_{\Sigma.\mathsf{V}}) \leftarrow\!\!\$\ \Sigma.\mathsf{V}_1(pp, x, cm)$
3: $rp \leftarrow \Sigma.\mathsf{P}_2(pp, x, w, st_{\Sigma.\mathsf{P}}, ch)$
4: $cm' \leftarrow \Sigma.\mathsf{Rec}(pp, x, ch, rp)$
5: $st_{\Sigma.\mathsf{V}} \leftarrow (cm', ch)$
6: $(b, int) \leftarrow \Sigma.\mathsf{V}_2(pp, x, st_{\Sigma.\mathsf{V}}, rp)$
7: **return** b

$\mathbf{Exp}^{\mathrm{WI}}_{\Sigma,\mathsf{D}^*}(pp, x, w_0, w_1)$:

11: $b \leftarrow\!\!\$\ \{0,1\}$
12: $b^* \leftarrow\!\!\$\ \mathsf{D}^{*\langle \Sigma.\mathsf{P}(pp,x,w_b),\cdot\rangle^{\infty}}(pp, x, w_0, w_1)$
13: **if** $b = b^*$ **then**
14: **return** 1
15: **return** 0

$\mathbf{Exp}^{\mathrm{SHVZK}}_{\Sigma,\mathsf{D}^*}(pp, x, w, ch)$:

21: $b \leftarrow\!\!\$\ \{0,1\}$
22: **if** $b = 0$ **then**
23: $(cm, st_{\Sigma.\mathsf{P}}) \leftarrow\!\!\$\ \Sigma.\mathsf{P}_1(pp, x, w)$
24: $rp \leftarrow \Sigma.\mathsf{P}_2(pp, x, w, st_{\Sigma.\mathsf{P}}, ch)$
25: **if** $rp = \perp$ **then**
26: $cm \leftarrow \perp$
27: **else**
28: $(cm, rp) \leftarrow\!\!\$\ \Sigma.\mathsf{Sim}(pp, x, ch)$
29: $b^* \leftarrow\!\!\$\ \mathsf{D}^*(pp, x, cm, ch, rp)$
30: **if** $b = b^*$ **then**
31: **return** 1
32: **return** 0

Figure 5.9: Definition of the experiments $\mathbf{Exp}^{\mathrm{Rec}}_{\Sigma}$, $\mathbf{Exp}^{\mathrm{WI}}_{\Sigma,\mathsf{D}^*}$, and $\mathbf{Exp}^{\mathrm{SHVZK}}_{\Sigma,\mathsf{D}^*}$.

3. We say that Σ is $(t, q_{\mathsf{WI}}, \varepsilon)$-*witness indistinguishable* w.r.t. pp if, for every yes-instance x for $\mathcal{R}$ w.r.t. pp with two corresponding witnesses w_0, w_1 and every distinguisher D^* running in time at most t and making at most q_{WI} interactions with the honest prover $\Sigma.\mathsf{P}$, we have

$$\mathbf{Adv}^{\mathrm{WI}}_{\Sigma,\mathsf{D}^*}(pp, x, w_0, w_1) = 2 \cdot \left| \Pr\left[\mathbf{Exp}^{\mathrm{WI}}_{\Sigma,\mathsf{D}^*}(pp, x, w_0, w_1) = 1\right] - \frac{1}{2} \right| \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathrm{WI}}_{\Sigma,\mathsf{D}^*}$ is depicted in Figure 5.9. We remark that witness indistinguishability is already satisfied if x has only one witness [FS90].

4. We say that Σ is (t, ε)-*special honest-verifier zero-knowledge* w.r.t. pp if, for every $(x, w) \in \mathcal{R}.\mathsf{IGen}(pp, 1)$, every challenge $ch \in \mathcal{C}$, and every distinguisher D^* running in time at most t, we have

$$\mathbf{Adv}^{\mathrm{SHVZK}}_{\Sigma,\mathsf{D}^*}(pp, x, w, ch) = 2 \cdot \left| \Pr\left[\mathbf{Exp}^{\mathrm{SHVZK}}_{\Sigma,\mathsf{D}^*}(pp, x, w, ch) = 1\right] - \frac{1}{2} \right| \leq \varepsilon,$$

where the experiment $\mathbf{Exp}^{\mathrm{SHVZK}}_{\Sigma,\mathsf{D}^*}$ is depicted in Figure 5.9.

Next, we give a formal description of OR-proofs. An *OR-proof* is a Sigma protocol that is obtained by combining several Sigma protocols. It allows to prove the possession of a witness for one of several statements, or to prove that one out of many statements is true. This problem was first considered by Cramer *et al.* [CDS94] under the assumption that the underlying Sigma protocols are special honest-verifier zero-knowledge. An important

$\Sigma_{\mathsf{OR}}.\mathsf{P}_1(pp, (x_0, x_1), (d, w))$:

1: $(cm_d, st_{\Sigma_d.\mathsf{P}}) \leftarrow\!\!\$\ \Sigma_d.\mathsf{P}_1(pp, x_d, w)$
2: $ch_{1-d} \leftarrow\!\!\$\ \mathcal{C}$
3: $(cm_{1-d}, rp_{1-d}) \leftarrow\!\!\$\ \Sigma_{1-d}.\mathsf{Sim}(pp, x_{1-d}, ch_{1-d})$
4: **if** $(cm_{1-d}, rp_{1-d}) = (\bot, \bot)$ **then**
5: **restart** $\Sigma_{1-d}.\mathsf{Sim}$
6: $cm \leftarrow (cm_0, cm_1)$
7: $st_{\Sigma_{\mathsf{OR}}.\mathsf{P}} \leftarrow (st_{\Sigma_d.\mathsf{P}}, ch_{1-d}, rp_{1-d})$
8: **return** $(cm, st_{\Sigma_{\mathsf{OR}}.\mathsf{P}})$

$\Sigma_{\mathsf{OR}}.\mathsf{V}_1(pp, (x_0, x_1), cm)$:

11: $ch \leftarrow\!\!\$\ \mathcal{C}$
12: $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} \leftarrow (cm, ch)$
13: **return** $(ch, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$

$\Sigma_{\mathsf{OR}}.\mathsf{P}_2(pp, (x_0, x_1), (d, w), st_{\Sigma_{\mathsf{OR}}.\mathsf{P}}, ch)$:

21: **parse** $st_{\Sigma_{\mathsf{OR}}.\mathsf{P}} = (st_{\Sigma_d.\mathsf{P}}, ch_{1-d}, rp_{1-d})$
22: $ch_d \leftarrow ch - ch_{1-d}$
23: $rp_d \leftarrow \Sigma_d.\mathsf{P}_2(pp, x_d, w, st_{\Sigma_d.\mathsf{P}}, ch_d)$
24: **if** $rp_d = \bot$ **then**
25: **return** $\bot$
26: $rp \leftarrow (ch_0, ch_1, rp_0, rp_1)$
27: **return** rp

$\Sigma_{\mathsf{OR}}.\mathsf{V}_2(pp, (x_0, x_1), st_{\Sigma_{\mathsf{OR}}.\mathsf{V}}, rp)$:

31: **parse** $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} = (cm_0, cm_1, ch)$
32: **parse** $rp = (ch_0, ch_1, rp_0, rp_1)$
33: **if** $ch \neq ch_0 + ch_1$ **then**
34: **return** $(0, (-1, -1))$
35: $st_{\Sigma_0.\mathsf{V}} \leftarrow (cm_0, ch_0)$
36: $st_{\Sigma_1.\mathsf{V}} \leftarrow (cm_1, ch_1)$
37: $(b_0, int_0) \leftarrow \Sigma_0.\mathsf{V}_2(pp, x_0, st_{\Sigma_0.\mathsf{V}}, rp_0)$
38: $(b_1, int_1) \leftarrow \Sigma_1.\mathsf{V}_2(pp, x_1, st_{\Sigma_1.\mathsf{V}}, rp_1)$
39: **if** $(b_0 = 0) \vee (b_1 = 0)$ **then**
40: **return** $(0, (-1, -1))$
41: **return** $(1, (int_0, int_1))$

$\Sigma_{\mathsf{OR}}.\mathsf{Sim}(pp, (x_0, x_1), ch)$:

51: $ch_0 \leftarrow\!\!\$\ \mathcal{C}$
52: $ch_1 \leftarrow ch - ch_0$
53: $(cm_0, rp_0) \leftarrow\!\!\$\ \Sigma_0.\mathsf{Sim}(pp, x_0, ch_0)$
54: $(cm_1, rp_1) \leftarrow\!\!\$\ \Sigma_1.\mathsf{Sim}(pp, x_1, ch_1)$
55: **if** $((cm_0, rp_0) = (\bot, \bot)) \vee ((cm_1, rp_1) = (\bot, \bot))$ **then**
56: **return** $(\bot, \bot)$
57: $cm \leftarrow (cm_0, cm_1)$
58: $rp \leftarrow (ch_0, ch_1, rp_0, rp_1)$
59: **return** (cm, rp)

$\Sigma_{\mathsf{OR}}.\mathsf{Ext}(pp, (x_0, x_1), T, T')$:

61: **parse** $T = (cm, ch, rp)$
62: **parse** $T' = (cm, ch', rp')$
63: **parse** $cm = (cm_0, cm_1)$
64: **parse** $rp = (ch_0, ch_1, rp_0, rp_1)$
65: **parse** $rp' = (ch'_0, ch'_1, rp'_0, rp'_1)$
66: **if** $ch_0 \neq ch'_0$ **then**
67: $T_0 \leftarrow (cm_0, ch_0, rp_0)$
68: $T'_0 \leftarrow (cm_0, ch'_0, rp'_0)$
69: $w \leftarrow \Sigma_0.\mathsf{Ext}(pp, x_0, T_0, T'_0)$
70: **return** $(0, w)$
71: **else**
72: $T_1 \leftarrow (cm_1, ch_1, rp_1)$
73: $T'_1 \leftarrow (cm_1, ch'_1, rp'_1)$
74: $w \leftarrow \Sigma_1.\mathsf{Ext}(pp, x_1, T_1, T'_1)$
75: **return** $(1, w)$

$\Sigma_{\mathsf{OR}}.\mathsf{Rec}(pp, (x_0, x_1), ch, rp)$:

81: **parse** $rp = (ch_0, ch_1, rp_0, rp_1)$
82: **if** $ch \neq ch_0 + ch_1$ **then**
83: **return** $\bot$
84: $cm_0 \leftarrow \Sigma_0.\mathsf{Rec}(pp, x_0, ch_0, rp_0)$
85: $cm_1 \leftarrow \Sigma_1.\mathsf{Rec}(pp, x_1, ch_1, rp_1)$
86: **if** $(cm_0 = \bot) \vee (cm_1 = \bot)$ **then**
87: **return** $\bot$
88: $cm \leftarrow (cm_0, cm_1)$
89: **return** cm

Figure 5.10: A formal description of the Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma_0, \Sigma_1]$ given in [CDS94]. We assume that $+$ is the group operation defined on the challenge space $\mathcal{C}$, and $-$ is its inverse.

property of their solution is that the resulting Sigma protocol is witness indistinguishable, *i.e.*, the verifier does not learn which particular witness was used to generate the proof.

In the following we recall the construction introduced in [CDS94]. Our focus is on provers holding only two statements (x_0, x_1) and a witness w for x_d, where $d \in \{0, 1\}$. The goal of such a prover is to convince a verifier that it holds a witness for one of the two statements, without revealing which one. More formally, let Σ_0 and Σ_1 be two Sigma protocols for a relation $\mathcal{R}$. Given a set of public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$ and an instance-witness pair $((x_0, x_1), (d, w)) \in \mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}(pp, 1)$, the construction given in [CDS94] allows to combine Σ_0 and Σ_1 into a new Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma_0, \Sigma_1]$ for the relation $\mathcal{R}_{\mathsf{OR}}$. The key idea of the construction is that the prover $\Sigma_{\mathsf{OR}}.\mathsf{P}$ splits the challenge ch received from $\Sigma_{\mathsf{OR}}.\mathsf{V}$ into two random shares $ch = ch_0 + ch_1 \in \mathcal{C}$, where $+$ is the group operation defined on the challenge space $\mathcal{C}$. Using the challenge parts ch_0 and ch_1, $\Sigma_{\mathsf{OR}}.\mathsf{P}$ provides two accepting transcripts (cm_0, ch_0, rp_0) and (cm_1, ch_1, rp_1) for both statements x_0 and x_1, respectively. If $\Sigma_{\mathsf{OR}}.\mathsf{P}$ knows the witness w for statement x_d, then the transcript (cm_d, ch_d, rp_d) is generated using w, while the transcript $\left(cm_{1-d}, ch_{1-d}, rp_{1-d}\right)$ is created using the simulator of Σ_{1-d}. The protocol is formally described in Figure 5.10.

For the remainder of this chapter, we are interested in the situation where a Sigma protocol is combined with itself. That is, given a Sigma protocol Σ for a relation $\mathcal{R}$, we obtain a Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ for the relation $\mathcal{R}_{\mathsf{OR}}$. In addition to the witness indistinguishability property, the protocol Σ_{OR} inherits many properties of Σ, *e.g.*, correctness and special honest-verifier zero-knowledge.

5.5.2 The Underlying Sigma Protocol

In this section we present the Sigma protocol that constitutes the main building block of BlindOR.

In lattice-based cryptography, it is common to prove in zero-knowledge the possession of a witness $\mathbf{s}$ with small entries such that $\mathbf{b} = \mathbf{A}\mathbf{s}$, given a matrix $\mathbf{A}$ and a vector $\mathbf{b}$ over some ring, typically $\mathbb{Z}_q$ or R_q. One approach to perform this kind of proofs is the so-called *Fiat-Shamir with Aborts* technique [Lyu09]. However, rather than proving knowledge of the vector $\mathbf{s}$ itself, this method allows to prove knowledge of a pair $(\bar{\mathbf{s}}, \bar{c})$ satisfying $\mathbf{b}\bar{c} = \mathbf{A}\bar{\mathbf{s}}$, where the entries of $\bar{\mathbf{s}}$ are still small but slightly larger than those of $\mathbf{s}$, and $\bar{c}$ is small as well. More precisely, the Fiat-Shamir with Aborts approach allows to prove the possession of a witness of the form $(\bar{\mathbf{s}}, \bar{c}) \in B_1 \times B_2$, where B_1 and B_2 are some predefined sets, even though the prover actually holds a witness of the form $(\mathbf{s}, 1) \in B_1' \times B_2$, where $B_1' \subseteq B_1$. It was shown that this relaxation is sufficient for many cryptographic applications such as digital signatures [Lyu12], commitment schemes [BCK+14], and verifiable encryption [LN17]. We extend this line of applications to blind signatures, *i.e.*, we show how this kind of proofs can be used to build blind signatures.

Our blind signature scheme BlindOR is built on a variant of the Sigma protocol introduced in [Lyu09]. Therefore, we briefly recall this construction before presenting our modified Sigma protocol. Given a public matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$ and a yes-instance of the form $\mathbf{b} \in R_q^{k_1}$, the prover holds a corresponding witness of the form $(\mathbf{s}, 1) \in B_1' \times B_2 \subseteq R^{k_1+k_2} \times R_q$ such that $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. A successful execution of the Sigma protocol allows to prove knowledge of a witness of the form $(\bar{\mathbf{s}}, \bar{c}) \in B_1 \times B_2$ satisfying $\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod q$,

$\mathcal{R}.\mathsf{IGen}(pp, b)$:

1: **if** $b = 0$ **then**
2: $\quad \mathbf{b} \leftarrow\!\!\$\ R_q^{k_1}$
3: $\quad$ **return** $\mathbf{b}$
4: **else**
5: $\quad$ **repeat** $\mathbf{s} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma'}^{k_1+k_2}$ **until** $\|\mathbf{s}\| \leq B_s$
6: $\quad \mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
7: $\quad$ **return** $(\mathbf{b}, \mathbf{s})$

Figure 5.11: The instance generator algorithm of the Sigma protocol underlying BlindOR.

where $B_1' \subseteq B_1 \subseteq R^{k_1+k_2}$. The commitment message is given by $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y} \pmod q$, where $\mathbf{y}$ is a masking vector with small entries. Upon receiving a random challenge c chosen uniformly from the challenge space $\mathbb{T}_\kappa^n$, the response is computed as $\mathbf{z} = \mathbf{y} + \mathbf{s}c$, and is sent to the verifier only if it follows a specified distribution, typically the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$, for some $\sigma > 0$, or the uniform distribution over a small subset of $R^{k_1+k_2}$. This ensures that $\mathbf{y}$ masks the secret-related term $\mathbf{s}c$, and that $\mathbf{z}$ is independently distributed from $\mathbf{s}$. If $\mathbf{z}$ does not follow the target distribution, the prover restarts the protocol with a fresh $\mathbf{y}$. The verifier accepts if $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$, and if $\|\mathbf{z}\|_p$ is bounded by some predefined value, where $p \in \{2, \infty\}$, depending on the distribution of $\mathbf{z}$.

Next, we describe our modified Sigma protocol Σ, which is built on top of the Sigma protocol briefly described in the previous paragraph. We start by introducing the relation $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ for which our Sigma protocol Σ is defined. On input 1^λ, the parameter generation algorithm $\mathcal{R}.\mathsf{PGen}$ generates a set of public parameters of the following form:

$$pp = (1^\lambda, n, k_1, k_2, q, \sigma', B_s, \omega, \kappa, \sigma^*, S, B_{z^*}, \delta^*, \mathbf{A}), \tag{5.4}$$

where the matrix $\mathbf{A}$ is chosen randomly from the uniform distribution over the set $R_q^{k_1 \times k_2}$. The remaining parameters are generated according to the constraints given in Table 5.3. The relation set associated to $\mathcal{R}$ is given by

$$\mathcal{R}.\mathsf{RSet}(pp) := \left\{ \left(\mathbf{b}, (\bar{\mathbf{s}}, \bar{c})\right) \in R_q^{k_1} \times (R^{k_1+k_2} \times \bar{\mathcal{C}}) \;\middle|\; (\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod q) \wedge \right.$$
$$\left. (\|\bar{\mathbf{s}}\| \leq 2B_{z^*}) \ \wedge \ \left(\bar{\mathcal{C}} = \left\{c - c' \mid c, c' \in \mathbb{T},\ c \neq c'\right\}\right) \right\}. \tag{5.5}$$

The instance generator algorithm $\mathcal{R}.\mathsf{IGen}$ is provided in Figure 5.11.

The actual witness the prover possesses is of the form $(\mathbf{s}, 1)$, where $\|\mathbf{s}\| \leq B_s < B_{z^*}$ and $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. In order to construct an OR-proof from the Sigma protocol Σ as shown in Section 5.5.1, we will directly use the abelian group $\mathbb{T}^\kappa$ as a challenge space rather than the set $\mathbb{T}_\kappa^n$, since $\mathbb{T}_\kappa^n$ does not define a group structure. The respective algorithms of Σ are formalized in Figures 5.12 and 5.13.

$\Sigma.\mathsf{P}_1(pp, \mathbf{b}, \mathbf{s})$:

1: **for** $i = 0$ **to** $\omega - 1$ **do**
2: **for** $j = 1$ **to** κ **do**
3: $\mathbf{y}_j \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n,\sigma^*}$
4: $\mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
5: $\mathbf{y}^{(i)} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
6: $\mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
7: $\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
8: $st_{\Sigma.\mathsf{P}} \leftarrow (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\omega-1)})$
9: **return** $(\mathbf{v}, st_{\Sigma.\mathsf{P}})$

$\Sigma.\mathsf{P}_2(pp, \mathbf{b}, \mathbf{s}, st_{\Sigma.\mathsf{P}}, \mathbf{c})$:

21: **parse** $st_{\Sigma.\mathsf{P}} = (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\omega-1)})$
22: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
23: $T \leftarrow \{0, \ldots, \omega - 1\}$
24: **while** $T \neq \emptyset$ **do**
25: $i \leftarrow\!\!\$\ T$
26: $T \leftarrow T \setminus \{i\}$
27: **parse** $\mathbf{y}^{(i)} = (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
28: **for** $j = 1$ **to** κ **do**
29: $\mathbf{z}_j \leftarrow \mathbf{y}_j + \mathbf{s}c_j$
30: $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
31: $\mathbf{s}^* \leftarrow (\mathbf{s}c_1, \ldots, \mathbf{s}c_\kappa)$
32: **if** $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{s}^*) = 1$ **then**
33: **return** $\mathbf{z}$
34: **return** $\perp$

$\Sigma.\mathsf{V}_1(pp, \mathbf{b}, \mathbf{v})$:

11: $\mathbf{c} = (c_1, \ldots, c_\kappa) \leftarrow\!\!\$\ \mathbb{T}^\kappa$
12: $st_{\Sigma.\mathsf{V}} \leftarrow (\mathbf{v}, \mathbf{c})$
13: **return** $(\mathbf{c}, st_{\Sigma.\mathsf{V}})$

$\Sigma.\mathsf{V}_2(pp, \mathbf{b}, st_{\Sigma.\mathsf{V}}, \mathbf{z})$:

41: **if** $\|\mathbf{z}\| > B_{z^*}$ **then**
42: **return** $(0, -1)$
43: **parse** $st_{\Sigma.\mathsf{V}} = (\mathbf{v}, \mathbf{c})$
44: **parse** $\mathbf{v} = (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
45: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
46: **parse** $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
47: **for** $j = 1$ **to** κ **do**
48: $\mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
49: **for** $i = 0$ **to** $\omega - 1$ **do**
50: $int \leftarrow 0$
51: **parse** $\mathbf{v}^{(i)} = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
52: **for** $j = 1$ **to** κ **do**
53: **if** $\mathbf{w}_j = \mathbf{v}_j$ **then**
54: $int = int + 1$
55: **if** $int = \kappa$ **then**
56: **return** $(1, i)$
57: **return** $(0, -1)$

Figure 5.12: The Sigma protocol underlying BlindOR. The remaining algorithms $\Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}$, and $\Sigma.\mathsf{Rec}$ are provided in Figure 5.13. Note that $\Sigma.\mathsf{P}$ restarts the protocol if $\Sigma.\mathsf{P}_2$ returns $\perp$.

At a high level, the protocol can be seen as a generalized version of the one given in [Lyu09] and explained above. In particular, it is optimized to work for BlindOR. Rather than computing only one commitment to a masking vector in $\Sigma.\mathsf{P}_1$, the prover computes commitments to $\omega \geq 1$ such vectors and sends them to the verifier all at once. The choice of $\omega > 1$ allows to reduce the number of protocol restarts induced by applying the rejection sampling procedure Rej in $\Sigma.\mathsf{P}_2$. That is, the probability of masking the secret-related vector $\mathbf{s}^* = (\mathbf{s}c_1, \ldots, \mathbf{s}c_\kappa)$ without restarting the protocol is increased when $\omega > 1$. The masking vectors are chosen according to the Gaussian distribution $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma^*}$. Upon receiving the challenge $\mathbf{c} \in \mathbb{T}^\kappa$, the prover sends the first response $\mathbf{z}$ for which rejection sampling accepts, *i.e.*, for the masking vector $\mathbf{y}^{(i)}$ such that $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{s}^*) = 1$,

$\Sigma.\mathsf{Sim}(pp, \mathbf{b}, \mathbf{c})$:

61: **return** $(\perp, \perp)$ with probability δ^*
62: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
63: $i \leftarrow\!\!\$\ \{0, \ldots, \omega - 1\}$
64: **for** $j = 1$ **to** κ **do**
65: $\quad \mathbf{z}_j \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n, \sigma^*}$
66: $\quad \mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
67: $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
68: $\mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
69: **for** $k = 0$ **to** $\omega - 1$ **do**
70: $\quad$ **if** $k = i$ **then**
71: $\quad\quad$ **continue**
72: $\quad$ **for** $j = 1$ **to** κ **do**
73: $\quad\quad \mathbf{y}_j \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n, \sigma^*}$
74: $\quad\quad \mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
75: $\quad \mathbf{v}^{(k)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
76: $\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
77: **return** $(\mathbf{v}, \mathbf{z})$ with probability $1 - \delta^*$

$\Sigma.\mathsf{Ext}(pp, \mathbf{b}, (\mathbf{v}, \mathbf{c}, \mathbf{z}), (\mathbf{v}, \mathbf{c}', \mathbf{z}'))$:

81: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
82: **parse** $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
83: **parse** $\mathbf{c}' = (c'_1, \ldots, c'_\kappa)$
84: **parse** $\mathbf{z}' = (\mathbf{z}'_1, \ldots, \mathbf{z}'_\kappa)$
85: Pick $j \in [\kappa]$ such that $c_j \neq c'_j$
86: $\bar{\mathbf{s}} \leftarrow \mathbf{z}_j - \mathbf{z}'_j$
87: $\bar{c} \leftarrow c_j - c'_j$
88: **return** $(\bar{\mathbf{s}}, \bar{c})$

$\Sigma.\mathsf{Rec}(pp, \mathbf{b}, \mathbf{c}, \mathbf{z})$:

91: **if** $\|\mathbf{z}\| > B_{z^*}$ **then**
92: $\quad$ **return** $\perp$
93: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
94: **parse** $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
95: **for** $j = 1$ **to** κ **do**
96: $\quad \mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
97: $\mathbf{v}^{(0)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
98: **for** $i = 1$ **to** $\omega - 1$ **do**
99: $\quad \mathbf{v}^{(i)} \leftarrow (\mathbf{0}, \ldots, \mathbf{0}) \in \left(R_q^{k_1}\right)^\kappa$
100: $\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
101: **return** $\mathbf{v}$

Figure 5.13: The Sigma protocol underlying BlindOR. The algorithms $\Sigma.\mathsf{P}$ and $\Sigma.\mathsf{V}$ are given in Figure 5.12.

and i is chosen from the uniform distribution over the set $T \subseteq \{0, \ldots, \omega - 1\}$ (see $\Sigma.\mathsf{P}_2$ in Figure 5.12). The random choice of the index i ensures that the simulator $\Sigma.\mathsf{Sim}$ returns $(\mathbf{v}, \mathbf{z}) \neq (\perp, \perp)$ with the same probability as the prover (*cf.* Figure 5.13). Note that each of the ω commitments consists of κ components, where κ defines the challenge space $\mathbb{T}^\kappa$. This allows to use the partitioning and permutation technique in BlindOR. In order to verify a transcript $(\mathbf{v}, \mathbf{c}, \mathbf{z})$, where $\mathbf{v} = (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$, the verifier first finds out which of the commitments $\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)}$ is related to the response $\mathbf{z}$. The index $i \in \{0, \ldots, \omega - 1\}$ of the corresponding commitment $\mathbf{v}^{(i)}$ is part of the verifier's output, *i.e.*, $\Sigma.\mathsf{V}_2$ returns $(1, i)$ if the transcript is accepted. Otherwise, $\Sigma.\mathsf{V}_2$ returns $(0, -1)$.

Theorem 5.20. *Let* $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$ *be the relation described above, and* $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, i.e., *pp has the form given in Equation (5.4), while* $\mathcal{R}.\mathsf{RSet}$ *and* $\mathcal{R}.\mathsf{IGen}$ *are the relation set and instance generator that are given in Equation (5.5) and Figure 5.11, respectively. The protocol depicted in Figures 5.12 and 5.13 is a Sigma protocol for* $\mathcal{R}$.

Proof. First, we note that the prover computes ω commitments in one execution of the

protocol and sends them to the verifier all at once. By Lemma 2.8, $\Sigma.\mathsf{P}_2$ returns $\mathbf{z} \neq \perp$ with probability $(1 - 2^{-100})/S$ after an average number of S restarts, where $S = \exp\left(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\right)$ and $\omega = 1$. Therefore, when $\omega \geq 1$, the prover sends a response $\mathbf{z} \neq \perp$ with probability $1 - \left(1 - \frac{1-2^{-100}}{S}\right)^{\omega}$ after at most $M = 1/(1 - \delta^*)$ restarts, where ω is chosen such that

$$\left(1 - \frac{1 - 2^{-100}}{S}\right)^{\omega} \leq \delta^*.$$

This means that the prover returns a response $\mathbf{z} = \perp$ with probability at most δ^*, and after $M = 1/(1 - \delta^*)$ restarts, the prover returns a response $\mathbf{z} \neq \perp$.

Let $(\mathbf{v}, \mathbf{c}, \mathbf{z})$ be an honestly created transcript. Then, its response $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$ follows the Gaussian distribution $D_{\mathbb{Z}^n, \sigma^*}^{(k_1+k_2)\kappa}$. By Lemma 2.6, $\|\mathbf{z}\| > B_{z^*} = \eta^* \sigma^* \sqrt{(k_1 + k_2)\kappa n}$ with probability at most

$$\varepsilon^* = \eta^{*(k_1+k_2)\kappa n} \exp\left(\frac{(k_1 + k_2)\kappa n}{2}(1 - \eta^{*2})\right),$$

where $\eta^* > 0$ can be chosen such that $\|\mathbf{z}\| \leq B_{z^*}$ with probability almost 1, *e.g.*, $1 - 2^{-80}$, where $\varepsilon^* = 2^{-80}$. Moreover, given the correct index $i \in \{0, \ldots, \omega - 1\}$ we have

$$\mathbf{w}_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b} c_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{y}_j + \mathbf{s} c_j) - \mathbf{b} c_j = \mathbf{v}_j \pmod{q}, \text{ for all } j \in [\kappa].$$

Thus, $\Sigma.\mathsf{V}_2$ returns $(0, -1)$ with probability ε^*, and hence the protocol is corr_Σ-correct, where $\mathrm{corr}_\Sigma = \delta^* + \varepsilon^*$. The same argument shows that $\Sigma.\mathsf{Rec}$ is $\mathrm{corr}_{\Sigma.\mathsf{Rec}}$-correct, where $\mathrm{corr}_{\Sigma.\mathsf{Rec}} = \mathrm{corr}_\Sigma$.

Next, we remark that the witness indistinguishability property directly follows from the base Sigma protocol given in [Lyu09]. We further observe that the special honest-verifier zero-knowledge property is satisfied as well. By Lemma 2.8, the distribution of the response $\mathbf{z}$ does not depend on the witness. Furthermore, real and simulated transcripts are statistically indistinguishable and within statistical distance of at most $2^{-100}/M$.

Finally, we show that the extractor $\Sigma.\mathsf{Ext}$ returns a witness in $\mathcal{R}$. Assume we have two correctly verified transcripts $(\mathbf{v}, \mathbf{c}, \mathbf{z})$ and $(\mathbf{v}, \mathbf{c}', \mathbf{z}')$, where $\mathbf{c} \neq \mathbf{c}'$ and $\Sigma.\mathsf{V}_2$ returns the same output $(1, i)$ when given both inputs $(pp, \mathbf{b}, st_{\Sigma.\mathsf{V}} = (\mathbf{v}, \mathbf{c}), \mathbf{z})$ and $(pp, \mathbf{b}, st_{\Sigma.\mathsf{V}} = (\mathbf{v}, \mathbf{c}'), \mathbf{z}')$, where $i \in \{0, \ldots, \omega - 1\}$. This means that

$$(\|\mathbf{z}\| \leq B_{z^*}) \wedge (\|\mathbf{z}'\| \leq B_{z^*}),$$

and for all $j \in [\kappa]$, we have

$$[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b} c_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_j - \mathbf{b} c'_j \pmod{q}.$$

For all $j \in [\kappa]$, this implies

$$\mathbf{b}(c_j - c'_j) = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{z}_j - \mathbf{z}'_j) \pmod{q}.$$

Since $\mathbf{c} \neq \mathbf{c}'$, where $\mathbf{c} = (c_1, \ldots, c_\kappa)$ and $\mathbf{c}' = (c'_1, \ldots, c'_\kappa)$, then $c_j \neq c'_j$ for at least one index j, where $j \in [\kappa]$. We set $\bar{\mathbf{s}} = \mathbf{z}_j - \mathbf{z}'_j$ and $\bar{c} = c_j - c'_j$ to obtain $\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod{q}$. Note that $\|\bar{\mathbf{s}}\| \leq 2B_{z^*}$ and $\bar{c} \in \bar{\mathcal{C}}$. Moreover, by Lemma 2.1, the polynomial $\bar{c}$ is invertible in R_q if q satisfies $\|\bar{c}\| = 2 < q^{1/p}$, where $q = 2p + 1 \pmod{4p}$ and p is a power of two such that $n \geq p > 1$. This implies that $\mathbf{b}\bar{c} \neq \mathbf{0} \pmod{q}$, and we can extract a witness $(\bar{\mathbf{s}}, \bar{c})$ in $\mathcal{R}$ as given in Figure 5.13. □

Observe that being able to extract a witness $(\bar{\mathbf{s}}, \bar{c})$ as shown in the proof of Theorem 5.20 implies a solution to MSIS^2 with respect to the parameters $(n, k_1, k_2 + 1, q, 2\sqrt{B_{z^*}^2 + 1})$. We further remark that when constructing the Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ as depicted in Figure 5.10, where Σ is the protocol introduced above, we have to consider the group operation defined on the challenge space $\mathbb{T}^\kappa$. More precisely, $\Sigma_{\mathsf{OR}}.\mathsf{P}_1$ samples a challenge share $\mathbf{c}_{1-d} = (c_{1,1-d}, \ldots, c_{\kappa,1-d})$ uniformly from $\mathbb{T}^\kappa$, and then runs the simulator $\Sigma_{1-d}.\mathsf{Sim}$ on input $(pp, \mathbf{b}_{1-d}, \mathbf{c}_{1-d})$. Upon receiving a challenge $\mathbf{c} = (c_1, \ldots, c_\kappa) \in \mathbb{T}^\kappa$ from $\Sigma_{\mathsf{OR}}.\mathsf{V}_1$, $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ computes the challenge share $\mathbf{c}_d = (c_1 c_{1,1-d}^{-1}, \ldots, c_\kappa c_{\kappa,1-d}^{-1})$, and then runs $\Sigma_d.\mathsf{P}_2$ on input $(pp, \mathbf{b}_d, \mathbf{s}, st_{\Sigma_d.\mathsf{P}}, \mathbf{c}_d)$. Therefore, we have

$$\mathbf{c} = \mathbf{c}_d \cdot \mathbf{c}_{1-d} = (c_{1,d}c_{1,1-d}, \ldots, c_{\kappa,d}c_{\kappa,1-d}).$$

5.5.3 Description of the Scheme

Let BS be a three-move blind signature scheme following the Fiat-Shamir approach, *e.g.*, BLAZE^+. We recall how signing and verification of such a scheme works. The signer computes and sends a commitment cm^* to the user. The user blinds cm^* to obtain a blind commitment cm, and then computes a blind challenge ch, which is usually generated by evaluating a cryptographic hash function H on input (cm, m), *i.e.*, $ch = \mathsf{H}(cm, m)$, where m is the message being signed. After that, the user unblinds ch to obtain a challenge ch^* and sends it to the signer. Using ch^*, the signer computes a response rp^* and sends it back to the user. Finally, the user blinds rp^* to obtain a blind response rp, and outputs the blind signature $sig = (ch, rp)$. Note that we call ch a blind challenge, since it is a part of the blind signature sig. Verifying the validity of sig is established by computing a commitment cm corresponding to ch and rp, and then checking if ch matches $\mathsf{H}(cm, m)$.

Observe that while the steps carried out by the signer are actually what a prover in a Sigma protocol does when proving the possession of a witness for a statement, the steps performed by the user consist of blinding the transcript (cm^*, ch^*, rp^*) while interacting with the prover. In $\mathsf{BlindOR}$, we capture these blinding steps by algorithms $\mathsf{Com}, \mathsf{Cha}, \mathsf{Rsp}$, and Rec, which we describe next.

For the remainder of this section we let $\Sigma = (\Sigma.\mathsf{P}, \Sigma.\mathsf{V}, \Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}, \Sigma.\mathsf{Rec})$ be the Sigma protocol depicted in Figures 5.12 and 5.13 for the relation $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{IGen})$, whose relation set and instance generator are described in Equation (5.5) and Figure 5.11, respectively. Let $\mathbf{b} \in R_q^{k_1}$ be an instance for $\mathcal{R}$ w.r.t. pp', where $pp' \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$ is a set of public parameters having the form given in Equation (5.4). For $\omega \in \mathbb{N}_{>0}$ and $\ell \in \mathbb{N}_{>1}$, we define the following bijective mapping:

$$\mathsf{IntIndex}\colon \{0, \ldots, \omega - 1\} \times \{0, \ldots, \ell - 1\} \to \{0, \ldots, \omega\ell - 1\},\ (i, k) \mapsto k + i\ell.$$

The function $\mathsf{IntIndex}$ converts the pair (i, k) into a unique positive integer. It will be used to build authentication paths via the algorithm $\mathsf{BuildAuth}$. These authentication paths are associated to trees of commitments with height $h = \lceil \log(\omega\ell) \rceil$ (see Chapter 4). Let $pp \in \mathsf{BS}.\mathsf{PGen}(1^\lambda)$ be a set of public parameters, where $pp' \subset pp$ and $\mathsf{BS}.\mathsf{PGen}$ is the parameter generation algorithm of $\mathsf{BlindOR}$ (see below). We define the following algorithms, which are formally described in Figure 5.14:

$\mathsf{Com}(pp, \mathbf{v}^*)$:

1: **parse** $\mathbf{v}^* = (\mathbf{v}^{*(0)}, \ldots, \mathbf{v}^{*(\omega-1)})$
2: $\mathbf{p} = (p_1, \ldots, p_\kappa) \leftarrow\!\!\$\ \mathbb{T}^\kappa$
3: **for** $i = 0$ **to** $\omega - 1$ **do**
4: **parse** $\mathbf{v}^{*(i)} = (\mathbf{v}_1^*, \ldots, \mathbf{v}_\kappa^*)$
5: **for** $k = 0$ **to** $\ell - 1$ **do**
6: **for** $j = 1$ **to** κ **do**
7: $\mathbf{e}_j^{(k)} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$
8: $\mathbf{v}_j^{(i,k)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}_j^{(k)} + \mathbf{v}_j^* p_j \pmod q$
9: $\mathbf{e}^{(k)} \leftarrow (\mathbf{e}_1^{(k)}, \ldots, \mathbf{e}_\kappa^{(k)})$
10: $\mathbf{v}^{(i,k)} \leftarrow (\mathbf{v}_1^{(i,k)}, \ldots, \mathbf{v}_\kappa^{(i,k)})$
11: $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{v}^{(0,0)}, \ldots, \mathbf{v}^{(\omega-1,\ell-1)})$
12: $\mathbf{e} \leftarrow (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
13: $st_{\mathsf{Com}} \leftarrow (\mathbf{p}, \mathbf{e}, tree)$
14: **return** $(root, st_{\mathsf{Com}})$

$\mathsf{IterateRej}(pp, \mathbf{e}, \mathbf{z}')$:

41: **parse** $\mathbf{e} = (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
42: **parse** $\mathbf{z}' = (\mathbf{z}'_1, \ldots, \mathbf{z}'_\kappa)$
43: **for** $k = 0$ **to** $\ell - 1$ **do**
44: **parse** $\mathbf{e}^{(k)} = (\mathbf{e}_1^{(k)}, \ldots, \mathbf{e}_\kappa^{(k)})$
45: **for** $j = 1$ **to** κ **do**
46: $\mathbf{z}_j \leftarrow \mathbf{e}_j^{(k)} + \mathbf{z}'_j$
47: $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
48: **if** $\mathsf{Rej}(pp, \mathbf{z}, \mathbf{z}') = 1$ **then**
49: **return** $(\mathbf{z}, k)$
50: **return** $(\perp, \perp)$

$\mathsf{Cha}(pp, \mathbf{p}, \mathbf{c}^*, b)$:

21: **parse** $\mathbf{p} = (p_1, \ldots, p_\kappa)$
22: **parse** $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$
23: **if** $b = 0$ **then**
24: $\mathbf{c} \leftarrow (c_1^* p_1^{-1}, \ldots, c_\kappa^* p_\kappa^{-1})$
25: **else**
26: $\mathbf{c} \leftarrow (c_1^* p_1, \ldots, c_\kappa^* p_\kappa)$
27: **return** $\mathbf{c}$

$\mathsf{Rsp}(pp, st_{\mathsf{Com}}, \mathbf{z}^*, i)$:

31: **parse** $st_{\mathsf{Com}} = (\mathbf{p}, \mathbf{e}, tree)$
32: **parse** $\mathbf{p} = (p_1, \ldots, p_\kappa)$
33: **parse** $\mathbf{z}^* = (\mathbf{z}_1^*, \ldots, \mathbf{z}_\kappa^*)$
34: $\mathbf{z}' \leftarrow (\mathbf{z}_1^* p_1, \ldots, \mathbf{z}_\kappa^* p_\kappa)$
35: $(\mathbf{z}, k) \leftarrow \mathsf{IterateRej}(pp, \mathbf{e}, \mathbf{z}')$
36: **if** $(\mathbf{z}, k) = (\perp, \perp)$ **then**
37: **return** $(\perp, \perp)$
38: $int \leftarrow \mathsf{IntIndex}(i, k)$
39: $auth \leftarrow \mathsf{BuildAuth}(int, tree, h)$
40: **return** $(\mathbf{z}, auth)$

$\mathsf{Rec}(pp, \mathbf{b}, \mathbf{c}, (\mathbf{z}, auth))$:

51: **if** $\|\mathbf{z}\| > B_z$ **then**
52: **return** $\perp$
53: **parse** $\mathbf{c} = (c_1, \ldots, c_\kappa)$
54: **parse** $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
55: **for** $j = 1$ **to** κ **do**
56: $\mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b} c_j \pmod q$
57: $\mathbf{w} \leftarrow (\mathbf{w}_1, \ldots, \mathbf{w}_\kappa)$
58: $root \leftarrow \mathsf{RootCalc}(\mathbf{w}, auth)$
59: **return** $root$

Figure 5.14: A formal description of the blinding algorithms Com, Cha, Rsp, and Rec.

$\mathsf{Com}(pp, \mathbf{v}^*)$ is a PPT algorithm that, on input the set of public parameters pp and a commitment $cm^* = \mathbf{v}^*$, which is generated by $\Sigma.\mathsf{P}_1$, returns a blind commitment $cm = root$ and a state $st_{\mathsf{Com}} = (\mathbf{p}, \mathbf{e}, tree)$, *i.e.*, $(root, st_{\mathsf{Com}}) \leftarrow\!\!\$\ \mathsf{Com}(pp, \mathbf{v}^*)$, where $st_{\mathsf{Com}} = (\mathbf{p}, \mathbf{e}, tree)$.

$\mathsf{Cha}(pp, \mathbf{p}, \mathbf{c}^*, b)$ is a DPT algorithm that, on input the set of public parameters pp, a

randomness $\mathbf{p} = (p_1, \ldots, p_\kappa) \in \mathbb{T}^\kappa$, a challenge $ch^* = \mathbf{c}^* \in \mathbb{T}^\kappa$, and an auxiliary bit $b \in \{0,1\}$, returns a challenge $ch = \mathbf{c} \in \mathbb{T}^\kappa$, *i.e.*, $\mathbf{c} \leftarrow \mathsf{Cha}(pp, \mathbf{p}, \mathbf{c}^*, b)$. Observe that the bit b determines if the challenge $\mathbf{c}^*$ will be blinded using the randomness $\mathbf{p}$ or its inverse $\mathbf{p}^{-1} = \left(p_1^{-1}, \ldots, p_\kappa^{-1}\right)$ with respect to the group operation defined on the challenge space $\mathbb{T}^\kappa$.

$\mathsf{Rsp}(pp, st_{\mathsf{Com}}, \mathbf{z}^*, i)$ is a DPT algorithm that, on input the set of public parameters pp, a state $st_{\mathsf{Com}} = (\mathbf{p}, \mathbf{e}, tree)$, a response $rp^* = \mathbf{z}^*$, which is generated by the algorithm $\Sigma.\mathsf{P}_2$, and an integer $i \in \{0, \ldots, \omega - 1\}$, returns a blind response $rp = (\mathbf{z}, auth)$, *i.e.*, $(\mathbf{z}, auth) \leftarrow \mathsf{Rsp}(pp, st_{\mathsf{Com}}, \mathbf{z}^*, i)$, where $st_{\mathsf{Com}} = (\mathbf{p}, \mathbf{e}, tree)$. We write $rp = (\perp, \perp)$ to denote failure.

$\mathsf{Rec}(pp, \mathbf{b}, \mathbf{c}, (\mathbf{z}, auth))$ is a DPT algorithm that, on input the set of public parameters pp, the instance $\mathbf{b}$, a challenge $ch = \mathbf{c}$, and a response $rp = (\mathbf{z}, auth)$, returns a commitment $cm = root$, *i.e.*, $root \leftarrow \mathsf{Rec}(pp, \mathbf{b}, \mathbf{c}, (\mathbf{z}, auth))$. We write $cm = \perp$ to denote failure.

The blinding algorithms depicted in Figure 5.14 capture the steps that are implicitly carried out by the user and verifier of our blind signature scheme BLAZE^+ (see Section 5.4). Additionally, the algorithm Com blinds commitments $\mathbf{v}^{*(0)}, \ldots, \mathbf{v}^{*(\omega-1)}$ rather than only one commitment generated by the algorithm $\Sigma.\mathsf{P}_1$. That is, the tree of commitments technique used in BLAZE^+ is extended to further include ω commitments created by the prover. These ω commitments are then combined with ℓ commitments generated within the algorithm Com to compute the root related to a tree of $\omega\ell$ commitments with height $h = \lceil \log(\omega\ell) \rceil$. Similar to BLAZE^+, we require ℓ to be chosen large enough such that the algorithm Rsp returns a blind response $(\mathbf{z}, auth) \neq (\perp, \perp)$ with probability very close to 1, *e.g.*, $1 - 2^{-80}$. This is crucial for $\mathsf{BlindOR}$ (and BLAZE^+) since otherwise, we would need an extra move between the signer and user as in our blind signature scheme BLAZE (*cf.* Section 5.3). This additional move would allow the user to request a restart of the signing protocol in case the algorithm $\mathsf{IterateRej}$ returns $(\perp, \perp)$. This would increase the communication complexity and force the signer to carry out almost all computations performed by the user before triggering a protocol restart. Moreover, this extra move would not allow the use of Gaussian distributed masking vectors $\mathbf{e}$. This is because a blind signature could be correctly verified even if rejection sampling does not accept. This would enable the user to request a protocol restart and obtain two different signatures on the same message. The advantage of using the Gaussian distribution for masking is that it allows to generate blind signatures with a size smaller than signatures generated using masking vectors that are uniformly distributed over a small subset of R_q.

We are now ready to give a detailed description of $\mathsf{BlindOR}$. Its respective algorithms are formalized in Figure 5.15. Let $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ and $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$, $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ be two cryptographic hash functions, where $\ell_\mathsf{F} \geq 2\lambda$ and $\mathbb{T}^\kappa$ is the challenge space of Σ. The first function is used to build trees of commitments, while the second one hashes the blind commitments together with the message being signed to obtain a blind challenge.

Parameter generation. Given 1^λ, $\mathsf{BS.PGen}$ generates and returns a set of public parame-

$\mathsf{BS.KGen}(pp)$:

1: $((\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s})) \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}(pp, 1)$
2: $pk \leftarrow (\mathbf{b}_0, \mathbf{b}_1)$
3: $sk \leftarrow (d, \mathbf{s})$
4: **return** (pk, sk)

$\mathsf{BS.S}_1(pp, pk, sk)$:

11: **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
12: **parse** $sk = (d, \mathbf{s})$
13: $(cm^*, st_{\mathsf{S}}) \leftarrow\!\!\$\ \Sigma_{\mathsf{OR}}.\mathsf{P}_1(pp, (\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s}))$
14: **return** (cm^*, st_{S})

$\mathsf{BS.U}_1(pp, pk, m, cm^*)$:

21: **parse** $cm^* = (\mathbf{v}_0^*, \mathbf{v}_1^*)$
22: $(root_0, st^0_{\mathsf{Com}}) \leftarrow\!\!\$\ \mathsf{Com}(pp, \mathbf{v}_0^*)$
23: $(root_1, st^1_{\mathsf{Com}}) \leftarrow\!\!\$\ \mathsf{Com}(pp, \mathbf{v}_1^*)$
24: $\mathbf{c} \leftarrow \mathsf{H}(root_0, root_1, m)$
25: **parse** $st^0_{\mathsf{Com}} = (\mathbf{p}_0, \mathbf{e}_0, tree_0)$
26: **parse** $st^1_{\mathsf{Com}} = (\mathbf{p}_1, \mathbf{e}_1, tree_1)$
27: $ch^* \leftarrow \mathsf{Cha}(pp, \mathbf{p}_0 \cdot \mathbf{p}_1, \mathbf{c}, 0)$
28: $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} \leftarrow (cm^*, ch^*)$
29: $st_{\mathsf{U}} \leftarrow (st^0_{\mathsf{Com}}, st^1_{\mathsf{Com}}, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$
30: **return** (ch^*, st_{U})

$\mathsf{BS.S}_2(pp, pk, sk, st_{\mathsf{S}}, ch^*)$:

41: **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
42: **parse** $sk = (d, \mathbf{s})$
43: $rp^* \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{P}_2(pp, (\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s}), st_{\mathsf{S}}, ch^*)$
44: **if** $rp^* = \bot$ **then**
45: **return** $\bot$
46: **return** rp^*

$\mathsf{BS.U}_2(pp, pk, m, st_{\mathsf{U}}, rp^*)$:

51: **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
52: **parse** $st_{\mathsf{U}} = (st^0_{\mathsf{Com}}, st^1_{\mathsf{Com}}, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$
53: $(b, (i_0, i_1)) \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{V}_2(pp, (\mathbf{b}_0, \mathbf{b}_1), st_{\Sigma_{\mathsf{OR}}.\mathsf{V}}, rp^*)$
54: **if** $b = 0$ **then**
55: **return** $(\bot, \bot)$
56: **parse** $st^0_{\mathsf{Com}} = (\mathbf{p}_0, \mathbf{e}_0, tree_0)$
57: **parse** $st^1_{\mathsf{Com}} = (\mathbf{p}_1, \mathbf{e}_1, tree_1)$
58: **parse** $rp^* = (\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$
59: $\mathbf{c}_0 \leftarrow \mathsf{Cha}(pp, \mathbf{p}_0, \mathbf{c}_0^*, 1)$
60: $\mathbf{c}_1 \leftarrow \mathsf{Cha}(pp, \mathbf{p}_1, \mathbf{c}_1^*, 1)$
61: $(\mathbf{z}_0, auth_0) \leftarrow \mathsf{Rsp}(pp, st^0_{\mathsf{Com}}, \mathbf{z}_0^*, i_0)$
62: $(\mathbf{z}_1, auth_1) \leftarrow \mathsf{Rsp}(pp, st^1_{\mathsf{Com}}, \mathbf{z}_1^*, i_1)$
63: **if** $((\mathbf{z}_0, auth_0) = (\bot, \bot)) \vee ((\mathbf{z}_1, auth_1) = (\bot, \bot))$ **then**
64: **return** $(\bot, \bot)$
65: $sig \leftarrow (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$
66: **return** (m, sig)

$\mathsf{BS.Verify}(pp, pk, m, sig)$:

71: **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
72: **parse** $sig{=}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$
73: $root_0 \leftarrow \mathsf{Rec}(pp, \mathbf{b}_0, \mathbf{c}_0, (\mathbf{z}_0, auth_0))$
74: $root_1 \leftarrow \mathsf{Rec}(pp, \mathbf{b}_1, \mathbf{c}_1, (\mathbf{z}_1, auth_1))$
75: **if** $(root_0 = \bot) \vee (root_1 = \bot)$ **then**
76: **return** 0
77: $\mathbf{c} \leftarrow \mathsf{H}(root_0, root_1, m)$
78: **if** $\mathbf{c} \neq \mathbf{c}_0 \cdot \mathbf{c}_1$ **then**
79: **return** 0
80: **return** 1

Figure 5.15: A formal description of the blind signature scheme BlindOR. The description of BS.PGen is given in the text. The signer restarts the signing protocol if $rp^* = \bot$.

ters of the following form:

$$pp = (1^\lambda, n, k_1, k_2, q, \sigma', B_s, \omega, \kappa, \sigma^*, \ell, \sigma, S, B_{z^*}, U, h, B_z, \ell_{\mathsf{F}}, \mathbf{A}). \tag{5.6}$$

The matrix $\mathbf{A}$ is chosen randomly from the uniform distribution over $R_q^{k_1 \times k_2}$. The

description of the remaining parameters is summarized in Table 5.3. We remark that pp includes the set of public parameters pp' associated to the relation $\mathcal{R}$ for which the Sigma protocol Σ is defined, *i.e.*, $pp' \subset pp$. In other words, $\mathsf{BS.PGen}$ may first run the algorithm $\mathcal{R}.\mathsf{PGen}(1^\lambda)$ to obtain pp', and then proceed by generating the remaining parameters of $\mathsf{BlindOR}$. For simplicity, we include the larger set pp in the input of the algorithms related to Σ.

Key generation. Given a set of public parameters pp, $\mathsf{BS.KGen}$ runs the instance generator algorithm $\mathcal{R}_{\mathsf{OR}}.\mathsf{IGen}$ on input $(pp, 1)$ to obtain a yes-instance $(\mathbf{b}_0, \mathbf{b}_1) \in R_q^{k_1} \times R_q^{k_1}$ for the relation $\mathcal{R}_{\mathsf{OR}}$ on $\mathcal{R}$ together with a witness $(d, \mathbf{s})$, where $d \in \{0,1\}$ and $\mathbf{s} \in D_{\mathbb{Z}^n,\sigma'}^{k_1+k_2}$ is a witness for $\mathbf{b}_d$. The algorithm returns the public key $pk = (\mathbf{b}_0, \mathbf{b}_1)$ and the secret key $sk = (d, \mathbf{s})$.

Signing. Let m be the message being blindly signed by $\mathsf{BS.S}$. The signing protocol of $\mathsf{BlindOR}$ is initiated by the signer, which plays the role of the prover of the Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$. The user acts as the verifier of Σ_{OR}, and further blinds both transcripts created by the signer via the blinding algorithms Com, Cha, and Rsp. In the following we describe the interaction between the signer and user.

Given (pp, pk, sk), $\mathsf{BS.S}_1$ runs the first prover algorithm $\Sigma_{\mathsf{OR}}.\mathsf{P}_1$ to obtain the commitment $cm^* = (\mathbf{v}_0^*, \mathbf{v}_1^*)$, which is sent to the user.

On input (pp, pk, m, cm^*), $\mathsf{BS.U}_1$ computes two blind commitments $root_0$ and $root_1$ by running the algorithm Com twice. Afterwards, the hash function H is evaluated on input $(root_0, root_1, m)$ to obtain the blind challenge $\mathbf{c}$. The challenge $\mathbf{c}$ is then "unblinded" to a challenge $\mathbf{c}^*$ via the algorithm Cha, which uses the randomness $\mathbf{p}_0 \cdot \mathbf{p}_1 = (p_{1,0}p_{1,1}, \ldots, p_{\kappa,0}p_{\kappa,1})$. The unblinded challenge $\mathbf{c}^*$ is then sent to the signer.

Upon receiving $\mathbf{c}^*$, $\mathsf{BS.S}_2$ runs the second prover algorithm $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ to obtain the response $rp^* = (\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$. The signing protocol is restarted if $rp^* = \perp$. Otherwise, rp^* is sent to the user.

Given rp^*, $\mathsf{BS.U}_2$ verifies the validity of the both transcripts $(\mathbf{v}_0^*, \mathbf{c}_0^*, \mathbf{z}_0^*)$ and $(\mathbf{v}_1^*, \mathbf{c}_1^*, \mathbf{z}_1^*)$ by running $\Sigma_{\mathsf{OR}}.\mathsf{V}_2$. Then, it blinds both challenge shares $\mathbf{c}_0^*, \mathbf{c}_1^*$ using the algorithm Cha to obtain the blind challenge parts $\mathbf{c}_0, \mathbf{c}_1$. Finally, two blind responses $(\mathbf{z}_0, auth_0)$ and $(\mathbf{z}_1, auth_1)$ are computed by running the algorithm Rsp twice. The blind signature is given by $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$.

Verification. Given (pp, pk, m, sig), $\mathsf{BS.Verify}$ computes two commitments $root_0, root_1$ by running the algorithm Rec twice. The signature is rejected if at least one execution of Rec returns $\perp$. Otherwise, sig is accepted if and only if $\mathbf{c}_0 \cdot \mathbf{c}_1 = \mathsf{H}(root_0, root_1, m)$.

The following theorem shows the correctness of $\mathsf{BlindOR}$.

Theorem 5.21. *Let* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ *be two cryptographic hash functions. The blind signature scheme* $\mathsf{BlindOR}$ *is* $\mathrm{corr}_{\mathsf{BS}}$*-correct w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$*, where*

$$\mathrm{corr}_{\mathsf{BS}} = \delta^* + 2\varepsilon^* + 2\delta + 2\varepsilon.$$

Table 5.3: A review of the parameters of BlindOR.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 2p + 1 \pmod{4p}$, $n \geq p > 1$, $p = 2^{p'}$, $p' \in \mathbb{N}_{>0}$, $q^{1/p} > 2$
σ'	Standard deviation of $\mathbf{s}$	$\sigma' > 0$
κ	Specifies the set $\mathbb{T}^\kappa$	$\lvert\mathbb{T}^\kappa\rvert = (2n)^\kappa \geq 2^\lambda$
ω, ℓ	No. masking vectors $\mathbf{y}^{(i)}, \mathbf{e}^{(k)}$	$\omega \in \mathbb{N}_{>0}$, $\ell \in \mathbb{N}_{>1}$
σ^*	Standard deviation in Σ	$\sigma^* = \alpha^*\sqrt{\kappa}B_s$, $S = \exp\left(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\right)$, $\alpha^* > 0$, $\left(1 - \frac{1-2^{-100}}{S}\right)^\omega \leq \delta^*$, $\delta^* > 0$
σ	Standard deviation in BS.U	$\sigma = \alpha B_{z^*}$, $U = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$, $\alpha > 0$, $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell \leq \delta$, $\delta > 0$
h	Tree height	$h = \lceil \log(\omega\ell) \rceil$
M	No. restarts of BS	$M = 1/(1 - \delta^*)$
B_s	Bound of $\lVert\mathbf{s}\rVert$ in sk	$B_s = \eta'\sigma'\sqrt{(k_1 + k_2)n}$, $\eta' > 0$
B_{z^*}	Bound of $\lVert\mathbf{z}\rVert$ in Σ	$B_{z^*} = \eta^*\sigma^*\sqrt{(k_1 + k_2)\kappa n}$, $\eta^* > 0$
B_z	Bound of $\lVert\mathbf{z}\rVert$ in BS.U	$B_z = \eta\sigma\sqrt{(k_1 + k_2)\kappa n}$, $\eta > 0$
ℓ_F	Output length of function F	$\ell_\mathsf{F} \geq 2\lambda$

The term δ^* defines the probability that $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ returns a response $rp^* = \perp$, ε^* is the probability that $\Sigma.\mathsf{V}_2$ returns $(0, i)$, δ is the probability that IterateRej *returns $(\perp, \perp)$, and ε is the probability that* Rec *returns $\perp$.*

Proof. By the correctness property of the underlying Sigma protocol Σ, the signer returns a response $rp^* = \perp$ with probability at most δ^*, and the algorithm $\Sigma_{\mathsf{OR}}.\mathsf{V}_2$ returns $(0, (i_0, i_1))$ with probability at most $2\varepsilon^*$ (see the proof of Theorem 5.20). Furthermore, the user returns $sig = \perp$ if at least one of both executions of the algorithm Rsp within $\mathsf{BS.U}_2$ returns a blind response $(\mathbf{z}, auth) = (\perp, \perp)$. This event occurs with probability at most 2δ because of the following: For each $b \in \{0, 1\}$, the algorithm Com generates ℓ masking vectors from $D_{\mathbb{Z}^n,\sigma}^{(k_1+k_2)\kappa}$. By Lemma 2.8, $\mathsf{Rej}(pp, \mathbf{z}_b, \mathbf{z}'_b) = 1$ with probability $(1 - 2^{-100})/U$, where $U = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$ and $\ell = 1$. Therefore, when $\ell \geq 1$, IterateRej returns $(\mathbf{z}_b, k_b) = (\perp, \perp)$ with probability $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell$, where ℓ is chosen such that $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell \leq \delta$.

Let $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$ be a blind signature generated by BlindOR on some message. Then, both vectors $\mathbf{z}_0$ and $\mathbf{z}_1$ follow the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{(k_1+k_2)\kappa}$, and by Lemma 2.6, each of the norms $\lVert\mathbf{z}_0\rVert$ and $\lVert\mathbf{z}_1\rVert$ is greater than B_z with probability

$$\varepsilon = \eta^{(k_1+k_2)\kappa n} \exp\left(\frac{(k_1 + k_2)\kappa n}{2}(1 - \eta^2)\right).$$

By a suitable choice of $\eta > 0$, we obtain $\lVert\mathbf{z}_0\rVert \leq B_z$ and $\lVert\mathbf{z}_1\rVert \leq B_z$ each with probability

at least $1-\varepsilon$. Moreover, for each $b \in \{0,1\}$ we have $root_b = \mathsf{RootCalc}(\mathbf{w}_b, auth_b)$, where $\mathbf{w}_b = (\mathbf{w}_{1,b}, \ldots, \mathbf{w}_{\kappa,b})$ and for all $j \in [\kappa]$ we have

$$\begin{aligned}\mathbf{w}_{j,b} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,b} - \mathbf{b}_b c_{j,b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{e}_{j,b}^{(k_b)} + \mathbf{z}_{j,b}^* p_{j,b}) - \mathbf{b}_b c_{j,b} \\ &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}_{j,b}^{(k_b)} + \mathbf{v}_{j,b}^{*\,(i_b)} p_{j,b} = \mathbf{v}_{j,b}^{(i_b,k_b)} \pmod q.\end{aligned}$$

Therefore, $\mathsf{BS.Verify}$ rejects *sig* if at least one of both executions of Rec within $\mathsf{BS.Verify}$ returns $\perp$, *i.e.*, if $root_0 = \perp$ or $root_1 = \perp$. This event occurs with probability at most 2ε. Hence, the correctness error of $\mathsf{BlindOR}$ is given by $\mathrm{corr}_{\mathsf{BS}} = \delta^* + 2\varepsilon^* + 2\delta + 2\varepsilon$. □

5.5.4 Security Analysis

In this section we prove that $\mathsf{BlindOR}$ satisfies both the statistical blindness and computational one-more unforgeability in the ROM.

Theorem 5.22. *Let* $\mathsf{F} : \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H} : \{0,1\}^* \to \mathbb{T}^\kappa$ *be two cryptographic hash functions modeled as random oracles. The blind signature scheme* $\mathsf{BlindOR}$ *is* ε*-statistically blind w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM, where* $\varepsilon = \max\{2^{-100}/U, (2n)^{-\kappa}\}$.

Proof. The proof is carried out in a similar way to the proof of Theorem 5.12. For completeness, we provide the proof in the following. Let S^* be an adversarial signer in the blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$. Then, S^* selects two messages m_0, m_1 and interacts with the honest user $\mathsf{BS.U}$ twice. We show that after both interactions, the messages output by the user, *i.e.*, two blind vectors of the form $\mathbf{c}^* \in \mathbb{T}^\kappa$ and two blind signatures of the form $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$, are independently distributed and do not leak any information about the signed messages and the respective interaction.

The vectors $\mathbf{c}^*, \mathbf{c}_0, \mathbf{c}_1$ are uniformly distributed over $\mathbb{T}^\kappa$. Hence, they do not leak any information. Moreover, Lemma 5.7 ensures that the vector $\mathbf{c}^*$ is independently distributed from $\mathbf{c} = \mathbf{c}_0 \cdot \mathbf{c}_1$, and S^* can link both elements only with probability $(2n)^{-\kappa}$ over guessing. Furthermore, the authentication paths $auth_0, auth_1$ include hash values that are uniformly distributed over $\{0,1\}^{\ell_\mathsf{F}}$. By Lemma 2.8, the blind vectors $\mathbf{z}_0$ and $\mathbf{z}_1$ are independently distributed from $\mathbf{z}_0'$ and $\mathbf{z}_1'$, respectively. Therefore, both vectors $\mathbf{z}_0$ and $\mathbf{z}_1$ completely mask $\mathbf{z}_0'$ and $\mathbf{z}_1'$, and hence $\mathbf{z}_0^*$ and $\mathbf{z}_1^*$, respectively. By the same lemma, $\mathbf{z}_0$ and $\mathbf{z}_1$ are within statistical distance of $2^{-100}/U$ from the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{(k_1+k_2)\kappa}$.

Finally, if a protocol restart is triggered by S^*, then $\mathsf{BS.U}$ generates fresh random elements. Therefore, the protocol restarts are independent of each other, and hence S^* does not get any information about the message being signed. □

Theorem 5.23. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ *be cryptographic hash functions modeled as random oracles. The blind signature scheme* $\mathsf{BlindOR}$ *is* $(t, q_{\mathsf{Sign}}, q_\mathsf{F}, q_\mathsf{H}, \varepsilon)$*-one-more unforgeable w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM if* MLWE^2 *is* (t', ε')*-hard w.r.t.* $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$ *and* MSIS^2 *is* (t'', ε'')*-hard w.r.t.* $pp'' = (n, k_1, k_2+1, q, 2\sqrt{B_z^2+1})$*, where*

$$t' \approx t \;\wedge\; \varepsilon' \approx \varepsilon \;\wedge\; t'' \approx 2t \;\wedge\; \varepsilon'' \approx \left(\frac{1}{2} - \varepsilon'\right) \cdot \left(\frac{1}{q_{\mathsf{Sign}}+1}\right) \cdot acc \cdot \left(\frac{acc}{(q_{\mathsf{Sign}}+1)\omega\ell} - \frac{1}{|\mathbb{T}^\kappa|}\right),$$

and $acc = \left(\varepsilon - \frac{q_\mathsf{F}^2+q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{q_{\mathsf{Sign}}+1}{|\mathbb{T}^\kappa|}\right) \Big/ q_\mathsf{H}^{q_{\mathsf{Sign}}+1}$.

$\mathsf{O}_\mathsf{F}(\mathbf{x})$:

1: **if** $\exists(\mathbf{x}, \mathsf{O}_\mathsf{F}(\mathbf{x})) \in \mathrm{L}_\mathsf{F}$ **then**
2: **return** $\mathsf{O}_\mathsf{F}(\mathbf{x})$
3: $\mathsf{O}_\mathsf{F}(\mathbf{x}) \leftarrow\!\!\$\ \{0,1\}^{\ell_\mathsf{F}}$
4: **if** $\exists(\mathbf{x}', \mathsf{O}_\mathsf{F}(\mathbf{x}')) \in \mathrm{L}_\mathsf{F} \colon (\mathbf{x} \neq \mathbf{x}') \wedge (\mathsf{O}_\mathsf{F}(\mathbf{x}) = \mathsf{O}_\mathsf{F}(\mathbf{x}'))$ **then**
5: **return** $\perp$
6: **if** $\exists(\mathbf{y}, \mathsf{O}_\mathsf{F}(\mathbf{y})) \in \mathrm{L}_\mathsf{F} \colon \mathbf{y} = \mathsf{O}_\mathsf{F}(\mathbf{x})$ **then**
7: **return** $\perp$
8: $\mathrm{L}_\mathsf{F} \leftarrow \mathrm{L}_\mathsf{F} \cup \{(\mathbf{x}, \mathsf{O}_\mathsf{F}(\mathbf{x}))\}$
9: **return** $\mathsf{O}_\mathsf{F}(\mathbf{x})$

$\mathsf{O}_\mathsf{H}(root_0, root_1, m)$:

11: **if** $\exists((root_0, root_1, m), \mathsf{O}_\mathsf{H}(root_0, root_1, m)) \in \mathrm{L}_\mathsf{H}$ **then**
12: **return** $\mathsf{O}_\mathsf{H}(root_0, root_1, m)$
13: $\mathsf{O}_\mathsf{H}(root_0, root_1, m) \hookleftarrow C$ // picks the first unused answer from C
14: $\mathrm{L}_\mathsf{H} \leftarrow \mathrm{L}_\mathsf{H} \cup \{((root_0, root_1, m), \mathsf{O}_\mathsf{H}(root_0, root_1, m))\}$
15: **return** $\mathsf{O}_\mathsf{H}(root_0, root_1, m)$

Figure 5.16: A description of the random oracles O_F and O_H.

Proof. Let A^* be a forger against BlindOR, *i.e.*, A^* runs in time t and can win the experiment $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{A}^*}$ given in Definition 5.4 with probability ε. First, we observe that the hardness of MLWE^2 w.r.t. pp' is required to protect against key recovery attacks, *i.e.*, being able to determine the yes-instance of MLWE^2 included in the public key $pk = (\mathbf{b}_0, \mathbf{b}_1)$ allows A^* to compute the secret key, and hence forgeries. Therefore, in what follows we assume the hardness of MLWE^2 w.r.t. pp', and construct a reduction algorithm R that solves MSIS^2 w.r.t. pp''.

Given pp'' and a uniformly random matrix $\mathbf{A}' \in R_q^{k_1 \times (k_2+1)}$, the reduction R chooses a bit $d \in \{0,1\}$ uniformly at random, and writes $\mathbf{A}' = [\mathbf{A} \mid \mathbf{b}_{1-d}] \in R_q^{k_1 \times k_2} \times R_q^{k_1}$. After that, R generates the remaining public parameters of BlindOR as specified in Table 5.3 to obtain the set of public parameters pp given in Equation (5.6). Then, R selects $\mathbf{c}_1, \ldots, \mathbf{c}_{q_\mathsf{H}}$ according to the uniform distribution over $\mathbb{T}^\kappa$, and sets $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_{q_\mathsf{H}}\}$. Afterwards, R runs the instance generator algorithm $\mathcal{R}.\mathsf{IGen}(pp, 1)$ depicted in Figure 5.11 to obtain the yes-instance and its corresponding witness $(\mathbf{b}_d, \mathbf{s})$. Then, R sets $pk = (\mathbf{b}_0, \mathbf{b}_1)$, $sk = (d, \mathbf{s})$, and runs the adversary A^* on input (pp, pk). The random oracle and signing queries that A^* make are answered by R as follows:

Random oracle query. The reduction R maintains a list L_H initialized by the empty set. It stores pairs of queries to the oracle O_H and their answers. If O_H was previously queried on some input, then R looks up its entry in L_H and returns its answer $\mathbf{c} \in \mathbb{T}^\kappa$. Otherwise, it picks the first unused $\mathbf{c} \in C$ and updates the list. Furthermore, R initializes an empty list L_F in order to store pairs of queries to the oracle O_F and

their answers. The queries to $\mathsf{O_F}$ are answered by R in a way that excludes collisions and chains. Excluding collisions rules out queries $\mathbf{x} \neq \mathbf{x}'$ such that $\mathsf{O_F}(\mathbf{x}) = \mathsf{O_F}(\mathbf{x}')$, while excluding chains guarantees that the query $\mathsf{O_F}(\mathsf{O_F}(\mathbf{x}))$ will not be made before the query $\mathsf{O_F}(\mathbf{x})$. This ensures that each node output by the algorithm $\mathsf{HashTree}$ has a unique preimage, and prevents spanning hash trees with cycles. Simulating $\mathsf{O_F}$ this way is within statistical distance of at most $\frac{q_\mathsf{F}^2+q_\mathsf{F}}{2^{\ell_\mathsf{F}}}$ from an oracle that allows the existence of collisions and chains. The description of $\mathsf{O_H}$ and $\mathsf{O_F}$ is given in Figure 5.16.

Signature query. Upon receiving a signature query from A^*, R runs the signing protocol of $\mathsf{BlindOR}$ acting as the signer. During interaction, R updates both lists $\mathrm{L_F}$ and $\mathrm{L_H}$ accordingly. Note that the environment of A^* is perfectly simulated and signatures are generated with the same probability as in the real execution of the signing protocol.

After q_Sign successful interactions, A^* returns $q_\mathsf{Sign}+1$ pairs of distinct messages and their valid blind signatures, *i.e.*, $(m_1, sig_1), \ldots, (m_{q_\mathsf{Sign}+1}, sig_{q_\mathsf{Sign}+1})$, where one of these pairs is not generated during the interaction between R and A^*. If $\mathsf{O_H}$ was not programmed or queried during invocation of A^*, then A^* produces a valid blind signature that includes a vector $\mathbf{c} \in \mathbb{T}^\kappa$ with probability $1/|\mathbb{T}^\kappa|$. Therefore, the probability that A^* succeeds in a forgery such that all $q_\mathsf{Sign}+1$ blind signatures correspond to random oracle queries made by A^* is at least $\varepsilon - \frac{q_\mathsf{Sign}+1}{|\mathbb{T}^\kappa|}$.

Suppose that the output of A^* includes two pairs (m, sig) and (m', sig') with the same random oracle answer $\mathbf{c} \in \mathbb{T}^\kappa$. This means we have $\mathsf{O_H}(root_0, root_1, m) = \mathsf{O_H}(root'_0, root'_1, m')$. If $(m \neq m') \vee (root_0 \neq root'_0) \vee (root_1 \neq root'_1)$, then a second preimage of $\mathbf{c}$ has been found by A^*. This occurs with probability at most $1/|\mathbb{T}^\kappa|$. On the other hand, the $q_\mathsf{Sign}+1$ messages are not pairwise distinct if $(m = m') \wedge (root_0 = root'_0) \wedge (root_1 = root'_1)$. Hence, we may assume for the remainder of the proof that all $q_\mathsf{Sign}+1$ blind signatures output by A^* include distinct random oracle answers of the form $\mathbf{c} \in \mathbb{T}^\kappa$.

Afterwards, R guesses an index $i^* \in [q_\mathsf{Sign}+1]$ such that $\mathbf{c}_{i^*} = \mathbf{c}_{j^*}$ for some $j^* \in [q_\mathsf{H}]$, *i.e.*, $\mathbf{c}_{i^*}$ is related to one of the $q_\mathsf{Sign}+1$ signatures output by A^* and $\mathbf{c}_{j^*} \in C$. Then, R records the pair $(m_{i^*}, sig_{i^*} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1))$ and invokes A^* again with the same random tape and the set of random oracle answers $C' = \{\mathbf{c}_1, \ldots, \mathbf{c}_{j^*-1}, \mathbf{c}'_{j^*}, \ldots, \mathbf{c}'_{q_\mathsf{H}}\}$, where $\mathbf{c}'_{j^*}, \ldots, \mathbf{c}'_{q_\mathsf{H}} \in \mathbb{T}^\kappa$ are freshly generated by R. After rewinding, A^* returns $q_\mathsf{Sign}+1$ pairs of distinct messages and their valid blind signatures. The potential two valid forgeries (before and after rewinding) output by A^* at index i^* have the form

$$(m, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)) \text{ and } (m', (\mathbf{c}'_0, \mathbf{c}'_1, \mathbf{z}'_0, \mathbf{z}'_1, auth'_0, auth'_1)),$$

where $\mathbf{c}_i = (c_{1,i}, \ldots, c_{\kappa,i})$, $\mathbf{c}'_i = (c'_{1,i}, \ldots, c'_{\kappa,i})$, and $i \in \{0,1\}$. By the verification algorithm of $\mathsf{BlindOR}$ we obtain

$$\begin{aligned}
\mathbf{w}_{j,1-d} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,1-d} - \mathbf{b}_{1-d} c_{j,1-d} \pmod q, \text{ for all } j \in [\kappa] \\
\mathbf{w}'_{j,1-d} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_{j,1-d} - \mathbf{b}_{1-d} c'_{j,1-d} \pmod q, \text{ for all } j \in [\kappa] \\
\mathbf{w}_{1-d} &= (\mathbf{w}_{1,1-d}, \ldots, \mathbf{w}_{\kappa,1-d}), \ \mathbf{w}'_{1-d} = \big(\mathbf{w}'_{1,1-d}, \ldots, \mathbf{w}'_{\kappa,1-d}\big), \\
root_{1-d} &= \mathsf{RootCalc}(\mathbf{w}_{1-d}, auth_{1-d}), \ root'_{1-d} = \mathsf{RootCalc}\big(\mathbf{w}'_{1-d}, auth'_{1-d}\big), \\
\mathbf{c}_0 \cdot \mathbf{c}_1 &= \mathbf{c} = \mathsf{O_H}(root_0, root_1, m), \ \mathbf{c}'_0 \cdot \mathbf{c}'_1 = \mathbf{c}' = \mathsf{O_H}(root'_0, root'_1, m').
\end{aligned}$$

By the forking lemma (Lemma 2.18) we have

$$(\mathbf{c} \neq \mathbf{c}') \wedge (root_0 = root'_0) \wedge (root_1 = root'_1) \wedge (m = m') \wedge \big(k_{1-d} = k'_{1-d}\big),$$

where $k_{1-d}, k'_{1-d} \in \{0, \ldots, \omega\ell - 1\}$ are the indices included in $auth_{1-d}$, $auth'_{1-d}$, respectively. Furthermore, answering the hash queries to the oracle $\mathsf{O_F}$ as described in Figure 5.16 ensures that both $auth_{1-d}, auth'_{1-d}$ include the same sequence of hash values, and hence we have

$$\big(auth_{1-d} = auth'_{1-d}\big) \wedge \big(\mathbf{w}_{1-d} = \mathbf{w}'_{1-d}\big).$$

If $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$, then for at least one index $j \in [\kappa]$ we have $c_{j,1-d} \neq c'_{j,1-d}$. Thus, we obtain

$$[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,1-d} - \mathbf{b}_{1-d} c_{j,1-d} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_{j,1-d} - \mathbf{b}_{1-d} c'_{j,1-d} \pmod q.$$

This gives the non-trivial solution $\big[\mathbf{z}_{j,1-d} - \mathbf{z}'_{j,1-d} \;\big|\; c'_{j,1-d} - c_{j,1-d}\big]^\top$ to MSIS w.r.t. pp'' and the matrix $[\mathbf{I}_{k_1} \mid \mathbf{A} \mid \mathbf{b}_{1-d}] = [\mathbf{I}_{k_1} \mid \mathbf{A}']$.

Finally, we analyze the success probability of the reduction R. The probability that R answers the correct sequence of $q_{\mathsf{Sign}} + 1$ random oracle queries to $\mathsf{O_H}$ that are used by A^* in the forgery is at least $1/q_{\mathsf{H}}^{q_{\mathsf{Sign}}+1}$. Since one of the $q_{\mathsf{Sign}} + 1$ pairs output by A^* is by assumption not generated during the interaction with R, the probability of correctly guessing the index i^* corresponding to this pair is $1/(q_{\mathsf{Sign}} + 1)$. The success probability of the forking (*cf.* Lemma 2.18) is given by

$$frk \geq acc \cdot \left(\frac{acc}{(q_{\mathsf{Sign}} + 1)\omega\ell} - \frac{1}{|\mathbb{T}^\kappa|}\right), \text{ where } acc = \left(\varepsilon - \frac{q_{\mathsf{F}}^2 + q_{\mathsf{F}}}{2^{\ell_{\mathsf{F}}}} - \frac{q_{\mathsf{Sign}} + 1}{|\mathbb{T}^\kappa|}\right) \Big/ q_{\mathsf{H}}^{q_{\mathsf{Sign}}+1}.$$

By Lemma 5.24, the probability that $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$ is given by $\frac{1}{2} - \varepsilon'$. This deduces the probability ε'' that is given in the theorem statement. □

Lemma 5.24. *Assume that after rewinding the forger A^* as done by the reduction algorithm R given in the proof of Theorem 5.23, the two forgeries output by A^* satisfy $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$ with probability $1/2 + \varepsilon'$, where $d \in \{0, 1\}$ corresponds to the yes-instance included in the public key and ε' is noticeably greater than 0. Then, there exists a distinguisher D^* that runs A^* in a black-box manner in order to win the experiment $\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}$ depicted in Figure 2.2 with advantage ε'.*

Proof. The input of D^* is the set of public parameters $pp' = (n, k_1, k_2, q, \sigma', \mathbf{A})$ and a vector $\mathbf{b}' \in R_q^{k_1}$. As indicated in the experiment $\mathbf{Exp}^{\mathsf{MLWE}}_{\mathsf{D}^*}$, the goal of D^* is to decide if the instance $\mathbf{b}'$ is either uniformly random (no-instance), or it has the form $\mathbf{b}' = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}' \pmod q$, *i.e.*, if $\mathbf{b}'$ is a yes-instance with witness $\mathbf{s}'$. Similar to the reduction R, D^* sets $\mathbf{b}_{1-d} = \mathbf{b}'$ and creates the yes-instance $\mathbf{b}_d$ with the corresponding witness $\mathbf{s}$. Then, D^* runs the forger on input $(pp, pk = (\mathbf{b}_0, \mathbf{b}_1))$. The signing and random oracle queries are answered by D^* as performed by R. After rewinding, we have $\mathbf{c} \neq \mathbf{c}'$ due to the forking lemma, where $\mathbf{c} = \mathbf{c}_0 \cdot \mathbf{c}_1$ and $\mathbf{c}' = \mathbf{c}'_0 \cdot \mathbf{c}'_1$. If $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$, then D^* returns $b^* = 0$ indicating that its input is a no-instance. Otherwise, D^* returns a randomly chosen bit b^*.

If the input $\mathbf{b}_{1-d}$ of D^* is a no-instance, then by assumption we have $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$ with probability $1/2 + \varepsilon'$. Thus, D^* returns $b^* = 0$. However, if the input of D^* is a yes-instance, then the forger has obtained a public key consisting of two yes-instances. By the witness indistinguishability property of the underlying OR-protocol, the forger A^* cannot guess which coordinate D^* is using to sign. Therefore, D^* returns a randomly chosen bit b^* in case $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$. □

Table 5.4: Overview of the sizes (in bits) of keys and signatures as well as the communication cost of each move of our schemes BLAZE, BLAZE$^+$, and BlindOR. We set $m = (k_1 + k_2)n$.

	BLAZE	BLAZE$^+$	BlindOR
Public key	$\ell_{seed} + k_1 n\lceil \log q \rceil$	$\ell_{seed} + k_1 n\lceil \log q \rceil$	$\ell_{seed} + 2k_1 n\lceil \log q \rceil$
Secret key	$m\lceil \log(2s+1) \rceil$	$m\lceil \log(t\sigma'+1) \rceil$	$1 + m\lceil \log(t\sigma'+1) \rceil$
Signature	$\ell_{str} + \kappa(1+\lceil \log n \rceil) + m\lceil \log(2z+1) \rceil$	$\kappa(1+\lceil \log n \rceil) + m\lceil \log(t\sigma+1) \rceil + h(\ell_{\mathsf{F}}+1)$	$2\kappa(1+\lceil \log n \rceil) + 2\kappa m\lceil \log(t\sigma+1) \rceil + 2h(\ell_{\mathsf{F}}+1)$
1$^{\text{th}}$ move	$\kappa k_1 n\lceil \log q \rceil$	$\kappa k_1 n\lceil \log q \rceil$	$\omega\kappa k_1 n\lceil \log q \rceil$
2$^{\text{th}}$ move	$\kappa(1+\lceil \log n \rceil)$	$\kappa(1+\lceil \log n \rceil)$	$\kappa(1+\lceil \log n \rceil)$
3$^{\text{th}}$ move	$\kappa m\lceil \log(2z^*+1) \rceil$	$\kappa m\lceil \log(t\sigma^*+1) \rceil$	$2\kappa(1+\lceil \log n \rceil) + 2\kappa m\lceil \log(t\sigma^*+1) \rceil$
4$^{\text{th}}$ move	$\ell_\tau + \ell_{seed} + 2\kappa(1+\lceil \log n \rceil)$	-	-

5.6 Concrete Parameters

In this section we propose concrete parameters for our blind signature schemes BLAZE, BLAZE$^+$, and BlindOR. The parameters target 128 bits of security. We provide a practical comparison between the schemes, and show which of them is the most suitable one in terms of sizes of keys/signatures, communication cost, performance, and security.

First, we provide in Table 5.4 the theoretical sizes of keys/signatures and the communication cost of each move of our schemes. In the following we explain how these sizes were obtained. To this end, we let $k, x \in \mathbb{Z}_{>0}$, $\sigma, t \in \mathbb{R}_{>0}$, $\ell \in \mathbb{N}_{>1}$, and $h = \lceil \log(\ell) \rceil$. Obviously, the size of any k-dimensional vector from R_q is given by $kn\lceil \log q \rceil$ bits. By the definition of the set S_x (*cf.* Section 2.2), any element belonging to S_x^k occupies $k\lceil \log(2x+1) \rceil$ bits. The size of any k-dimensional vector $\mathbf{v}$ following the Gaussian distribution $D^k_{\mathbb{Z}^n,\sigma}$ can be computed according to Lemma 2.6. We fixed the probability 2^{-80} and obtained $t = 11.3$ so that $\Pr[\|\mathbf{v}\|_\infty > t\sigma] \leq 2^{-80}$. Any vector $\mathbf{c} = (c_1, \ldots, c_\kappa) \in \mathbb{T}^\kappa$ can be described with $\kappa(1+\lceil \log n \rceil)$ bits, where each coefficient of $\mathbf{c}$ is of the form $\pm X^i$ for $i \in \{0, \ldots, n-1\}$, *i.e.*, we need one bit for the sign and $\lceil \log n \rceil$ bits for the degree i. An authentication path $auth = (k', \mathbf{a}_0, \ldots, \mathbf{a}_{\ell-1})$ requires $h(\ell_{\mathsf{F}}+1)$ bits, where the index k' lies in the set $\{0, \ldots, \ell-1\}$ and the remaining elements are hash values under the hash function F. Thus, the bit length of k' is at most h, and each hash value has a length of ℓ_{F} bits. For using our schemes in practice, we assume that the implementation of the key generation algorithm BS.KGen includes the selection of a uniformly random seed from $\{0,1\}^{\ell_{seed}}$. This seed is used to generate the public matrix $\mathbf{A}$ in a deterministic way via the function Expand, which allows to save space by storing the seed instead of $\mathbf{A}$. Therefore, the seed is included in the public key pk, and hence we extend the size of pk to ℓ_{seed} bits (*cf.* Table 5.4). The signature size of BLAZE$^+$ and BlindOR is computed without compression. In practice however, a Gaussian integer $z \in D_{\mathbb{Z},\sigma}$ is optimally compressed via Huffman encoding, which requires approximately $\tau + 2.25$ bits on average [DLL+17b], where $\sigma \approx 2^\tau$.

Table 5.5: Concrete parameters for BLAZE, BLAZE$^+$, and BlindOR. The parameters target 128 bits of security. The description of the parameters is summarized in Table 5.1, Tables 5.2 and 5.3. We set $\ell_{seed} = \ell_{str} = \ell_\tau = 128$, and $\ell_{\mathsf{F}} = 256$. The table also gives the corresponding sizes (in kilobytes) of keys/signatures and communication cost required to obtain a valid blind signature.

Parameter	BLAZE	BLAZE$^+$	BlindOR
(n, k_1, k_2)	$(256, 8, 7)$	$(256, 5, 4)$	$(256, 5, 4)$
q	$2^{45} - 2^9 + 1$	$2^{32} - 2^{20} + 1$	$\approx 2^{34}$
κ	23	23	15
s	4	-	-
σ'	-	4	4
(y, z^*)	$(2^{19}, 524284)$	-	-
(α^*, σ^*)	-	$(13.5, 12680)$	$(8.3, 6296)$
(e, z)	$(2^{42}, 4398034452572)$	-	-
(α, σ)	-	$(11.5, 37931560)$	$(21.5, 26171185)$
(ω, ℓ, h)	-	$(1, 126, 7)$	$(2, 63, 7)$
$(M_{\mathsf{S}}, M_{\mathsf{U}}, M)$	$(2, 1.02, 2.04)$	$(2.4, 1, 2.4)$	$(2.4, 1, 2.4)$
(η', η^*, η)	-	$(1.02, 1.03, 1.16)$	$(1.02, 1.04, 1.04)$
Public key size (KB)	11.27	5.02	10.64
Secret key size (KB)	1.88	1.69	1.69
Signature size (KB)	20.20	8.40	245.16
Communication cost (KB)	967.86	392.50	526.01

We present our concrete parameters in Table 5.5 including the corresponding sizes of keys and signatures as well as the total amount of communication cost that is required to generate valid blind signatures. In the following we give some insights of how these parameters were selected.

- In BLAZE and BLAZE$^+$, the modulus q is a prime such that $q = 1 \pmod{2n}$. This allows for efficient polynomial multiplication. In BlindOR, the modulus is given by the prime $q = 17179868353$, which satisfies $q = 2p + 1 \pmod{4p}$ so that the polynomial $X^n + 1$ underlying the ring R_q has $p = 32$ irreducible factors (*cf.* Lemma 2.1) and each polynomial $\bar{c} \in \bar{\mathcal{C}}$ having the form given in Equation (5.5) is invertible in R_q. This guarantees that the reduction algorithm given in the proof of Theorem 5.23 is able to compute a non-zero solution to MSIS w.r.t. pp''.

- The values of the parameters y and e are computed by fixing the average number of restarts M_{S} and M_{U}, respectively. In order to facilitate sampling elements from the uniform distribution over $S_x^{k_1+k_2}$, where $x \in \{s, y, e\}$, the parameters s, y, e are set to power of two.

Table 5.6: Performance of an implementation of BLAZE$^+$ on an Intel Core i7-9750H processor operating at 2.6 GHz.

Performance	Key generation	Signing	Verification
Average milliseconds	0.8	117	0.8
Average cycles	2,117,012	303,906,073	2,058,038

- The parameters ω, α^*, and σ^* ensure that the rejection sampling procedure carried out by the signer accepts with probability $1/M$. However, the choice of the parameters ℓ, α, and σ allows the user to compute a valid blind signature with probability at least $1 - 2^{-80}$, *i.e.*, the algorithm IterateRej returns $(\perp, \perp)$ with probability at most 2^{-80}. Assuming that the signing algorithm will generate at most 2^{64} signatures, the latter probability is sufficient and we believe that it is not worthwhile to increase the parameter ℓ in order to obtain a probability greater than $1 - 2^{-80}$. This is because increasing ℓ significantly affects the performance of the signing protocol of BLAZE$^+$ and BlindOR.

- The values of the parameters η', η^*, and η are computed according to Lemma 2.6. More concretely, with the given value of η' the key generation algorithm succeeds in sampling a Gaussian vector $\mathbf{s}$ satisfying $\|\mathbf{s}\| \leq B_s$ with probability 0.75. This probability reduces B_s, and hence the size of blind signatures. The values of η^* and η ensure that $\|\mathbf{z}^*\| \leq B_{z^*}$ and $\|\mathbf{z}\| \leq B_z$ with probability at least $1 - 2^{-80}$, respectively.

We remark that the communication cost provided in Table 5.5 gives the total amount of communication including the number of restarts that is required to obtain valid blind signatures. In BLAZE, this amount is computed as

$$M \cdot (|\text{Move}_1| + |\text{Move}_2| + |\text{Move}_3|) + (M - 1) \cdot |\text{Move}_4|,$$

while in BLAZE$^+$ and BlindOR it is computed as

$$M \cdot (|\text{Move}_1| + |\text{Move}_2|) + |\text{Move}_3|,$$

where for $i \in \{1, 2, 3, 4\}$, the term $|\text{Move}_i|$ denotes the length of the bit string related to the i^{th} move of the signing protocol.

At the time of writing this thesis, we are working on optimized implementations of our schemes [A5], where only a working implementation of BLAZE$^+$ is ready, so that it can be used to evaluate the performance of the parameters given in Table 5.5. We provide these performance measures in Table 5.6. We expect the performance evaluation of BlindOR to be twice as large as the evaluation given for BLAZE$^+$. Observe that due to using OR-proofs, the operations carried out in BlindOR are twice the operations involved in BLAZE$^+$. In particular, the values of the parameters n, k_1, k_2, σ' are identical, and the number of restarts induced by the signer is the same, and hence both schemes have the same total number of restarts.

Finally, we provide a practical comparison between our schemes. As Table 5.5 shows, BLAZE$^+$ reduces the sizes of public key and signatures in addition to the communication complexity to a factor of at least 1/2, when comparing with BLAZE. In addition to the performance, using OR-proofs in BlindOR requires a public key and signatures of sizes that are at least twice as large as those required for BLAZE$^+$. Hence, if provable security is the top priority, then BlindOR is the right option to use at the cost of larger sizes of public key and signatures as well as lower performance. Otherwise, BLAZE$^+$ seems to be the most suitable choice for applications, where smaller sizes and reduced communication complexity are in the first place. We expect BLAZE$^+$ to be even faster than BLAZE. This is due to the larger dimension and modulus, *i.e.*, larger values for k_1, k_2, and q, that are required for BLAZE to achieve the desired security level. We note that we can choose parameters for BLAZE having a dimension identical to BLAZE$^+$, *i.e.*, we can reduce the value of the pair (k_1, k_2) in BLAZE from $(8, 7)$ to $(5, 4)$. However, this would require to use a prime modulus of length at least 61 bits. This modulus would significantly affect the performance of the scheme, and in particular, the polynomial multiplication and modular reduction algorithm.

5.7 Cryptanalysis of Two-Round Blind Signature Schemes

In this section we present two attacks on previous two-round lattice-based constructions of blind signatures. More concretely, we show that after two interactions with the signer, any user can simply compute the secret key of the lattice-based blind signature scheme given in [ZTZ+17]. Then, we explain how the instance of the SIS problem underlying the earlier identity-based blind signature schemes [CCT+11, ZM14, ZH16, GHWX16, GHW+17] can be solved by any user after only one execution of the signing protocol.

5.7.1 Key Recovery of a Blind Signature Proposal

In this section we describe a key recovery attack on the blind signature scheme proposed in [ZTZ+17]. We sketch its key generation and signing protocol, and only explain the elements required for our analysis.

The secret key is an $(n \times n)$-matrix $\mathbf{S}$ with coefficients from $\{-1, 0, 1\}$, and the public key consists of two $(n \times n)$-matrices $(\mathbf{P}, \mathbf{H})$, where $\mathbf{P} = \lfloor 2\rho(\mathbf{S}) + 1 \rfloor \cdot \mathbf{I}_n$, $\rho(\mathbf{S})$ is the spectral radius of $\mathbf{S}$, and $\mathbf{H}$ is the Hermite normal form of $\mathbf{P} - \mathbf{S}$. Signing is performed as follows:

1. The user sends an n-dimensional vector $\mathbf{u}$ to the signer, where $\mathbf{u}$ is computed using some random elements and the message being signed.
2. The signer sends $\mathbf{z}' = \mathbf{u} - \lfloor \mathbf{u}\mathbf{P}^{-1} \rceil (\mathbf{P} - \mathbf{S})$ back to the user.
3. The user outputs $\mathbf{z} = \mathbf{z}'\mathbf{T}^{-1} - \mathbf{e}$ as a part of the signature, where $\mathbf{T}, \mathbf{e}$ are included in the vector $\mathbf{u}$.

The secret key $\mathbf{S}$ can be computed as follows: The user selects two random vectors $\mathbf{u}_1, \mathbf{u}_2$ such that $\mathbf{x} = \lfloor \mathbf{u}_1\mathbf{P}^{-1} \rceil - \lfloor \mathbf{u}_2\mathbf{P}^{-1} \rceil$ is invertible. Then, it initiates the signing protocol twice by sending $\mathbf{u}_1$ and $\mathbf{u}_2$ in the first and second invocation, respectively. After receiving $\mathbf{z}'_1, \mathbf{z}'_2$, the secret key can be computed as $\mathbf{S} = (\mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{u}_1 + \mathbf{u}_2 + \mathbf{x}\mathbf{P})\mathbf{x}^{-1}$.

5.7.2 Forgeability of Two-Round ID-Based Blind Signature Schemes

In this section we show a design flaw in the previous two-round lattice-based identity-based blind signature schemes proposed in [CCT+11, ZM14, ZH16, GHWX16, GHW+17]. They all follow a specified framework that employs the preimage sampleable trapdoor function introduced by Gentry *et al.* [GPV08]. More concretely, the framework defines a public random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with a secret trapdoor. This trapdoor is a short basis for the lattice whose vectors are given by the set $\{\mathbf{e} \in \mathbb{Z}^m \mid \mathbf{0} = \mathbf{Ae} \pmod q\}$. Signing is carried out as follows:

1. The user sends an n-dimensional vector $\mathbf{y} = \mathbf{Ax} + \mathbf{c}t \pmod q$ to the signer, where $\mathbf{x}$ is an m-dimensional Gaussian vector, $\mathbf{c} \in \mathbb{Z}_q^n$ is a hash value of the message being signed, and t is a small integer.

2. The signer samples a preimage vector $\mathbf{e}$ such that $\mathbf{y} = \mathbf{Ae} \pmod q$ and sends it back to the user.

3. The user outputs the signature $\mathbf{z} = (\mathbf{e} - \mathbf{x})t^{-1} \pmod q$. We note that two of the proposals we analyze here consider t as an invertible Gaussian $(n \times n)$-matrix, although the signature $\mathbf{z}$ cannot be obtained by multiplying an m-dimensional vector with an $(n \times n)$-matrix.

Verification is carried out by checking that $\mathbf{c} = \mathbf{Az} \pmod q$. Apparently, it is assumed that the signing protocol is stateful in order to prevent re-querying attacks. That is, the signer has a local storage to return the same preimage of previous signing queries. Nevertheless, any user can simply send $\mathbf{y} = \mathbf{Ax} \pmod q$ and let the signer return a preimage $\mathbf{x}'$ of $\mathbf{y}$. Thus, we obtain $\mathbf{0} = \mathbf{A}(\mathbf{x} - \mathbf{x}') \pmod q$, where $\mathbf{x} - \mathbf{x}'$ is a short and non-zero vector with high probability, since collisions always exist.

6 Conclusion

In this chapter we conclude this thesis and discuss possible research directions for future work.

We started by presenting the first post-quantum construction of signature schemes with re-randomizable public keys. The construction uses lattice-based Fiat-Shamir signature schemes as a building block and provides security in the quantum random oracle model (QROM). We showed that our scheme can be used to build post-quantum deterministic wallets, and described how it can be instantiated with the ordinary signature scheme qTESLA [ABB+20]; a second-round candidate of the NIST post-quantum standardization process. We further presented alternative approaches for re-randomizing keys in the lattice setting including a scheme that can use either Gaussian or uniformly distributed secret keys, and allows the re-randomization algorithms RSig.RandSK and RSig.RandPK to synchronize on some state.

With focus on signature schemes with advanced functionalities, we went on to analyze lattice-based (interactive) protocols with multiple rejection sampling procedures. We observed that such protocols require a number of restarts that is multiplicative in the number of the involved rejection sampling procedures. To solve this problem, we presented the tree of commitments technique, which allows to reduce the number of restarts, or even remove the restarts, inherent in lattice-based schemes. Using this technique we constructed a canonical identification scheme that is a variant of the one given in [Lyu09] and has a reduced amount of communication complexity. We also demonstrated the practical relevance of our technique by utilizing it in blind signature schemes.

Finally, we studied lattice-based blind signatures, and presented the new schemes BLAZE, BLAZE$^+$, and BlindOR. We demonstrated the efficiency of our proposals and the ability to use them in real-life privacy-preserving applications. For applications where provable security is critical, we argued that BlindOR is the right scheme to use. Otherwise, BLAZE$^+$ is the most suitable choice for applications where smaller sizes and reduced communication complexity are in the first place. Furthermore, our argumentation shows that BLAZE$^+$ seems to outperform BLAZE, so that BLAZE$^+$ is also the best choice for applications having performance as the top priority. We showed that the existing lattice-based constructions of blind signatures following the hash-and-sign approach and using preimage sampleable trapdoor functions [GPV08] are insecure.

Future research. In this thesis we focused on designing practical lattice-based constructions of two advanced signature schemes, *i.e.*, signature schemes with re-randomizable keys and blind signature schemes. However, there is still various types of lattice-based signatures that have to be pushed towards practice. For instance, multi-signatures [IN83], aggregate signatures [BGLS03], threshold signatures [DF90], and threshold ring signatures [BSS02]. We believe that our tree of commitments technique can be a crucial tool in the design of at least multi-, threshold, and threshold ring signature schemes based on lattices.

Similar to previous works on blind signatures, *e.g.* [PS00,HKLN20], the security reduction to the one-more unforgeability property of our blind signature scheme BlindOR allows to generate a small number of signatures per public key. This restriction is inherited from the proof technique due to Pointcheval and Stern [PS00] as well as Abe and Okamoto [AO00]. Pointcheval [Poi98] studied the Okamoto-Schnorr blind signature scheme [Oka93,PS00] and showed how to increase the number of signatures to a polynomial amount per public key at the cost of increasing the communication complexity of the signing protocol and generating signatures in a sequential manner. His work was recently extended and generalized by Katz *et al.* [KLR21] to include number-theoretic schemes based on RSA, factoring, and discrete logarithm assumptions in the random oracle model (ROM). It would be interesting to investigate the ability to extend these works to the lattice setting.

While the security of our signature scheme with re-randomizable public keys is proven in the QROM, the blind signature scheme BlindOR is proven secure in the ROM only. Therefore, a further important extension is to prove the security of BlindOR in the QROM. This may be established by proving the one-more unforgeability property via a security reduction in the QROM from both lattice problems MLWE and SelfTargetMSIS rather than MLWE and MSIS. As mentioned in Chapter 2, Kiltz *et al.* [KLS18] gave reasonable arguments to assume that the hardness of SelfTargetMSIS is equivalent to the hardness of MSIS in the QROM.

Finally, instead of parallel OR-proofs due to Cramer *et al.* [CDS94], one could alternatively investigate building lattice-based blind signatures with security in the QROM based on sequential OR-proofs due to Abe *et al.* [AOS02]. Sequential OR-proofs were studied, for example, by Fischlin *et al.* [FHJ20] and shown to provide ordinary signatures that are secure in the QROM.

Bibliography

[AABN02] Michel Abdalla, Jee Hea An, Mihir Bellare, and Chanathip Namprempre. From identification to signatures via the Fiat-Shamir transform: Minimizing assumptions for security and forward-security. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 418–433, Amsterdam, The Netherlands, April 28 – May 2, 2002. Springer, Heidelberg, Germany. (cited on pages 4, 25, 39, 41, 42, and 47.)

[ABB+20] Erdem Alkim, Paulo S. L. M. Barreto, Nina Bindel, Juliane Krämer, Patrick Longa, and Jefferson E. Ricardini. The lattice-based digital signature scheme qTESLA. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 20: 18th International Conference on Applied Cryptography and Network Security, Part I*, volume 12146 of *Lecture Notes in Computer Science*, pages 441–460, Rome, Italy, October 19–22, 2020. Springer, Heidelberg, Germany. (cited on pages 2, 3, 4, 22, 26, 27, 37, 38, 39, and 105.)

[ACdMT05] Giuseppe Ateniese, Daniel H. Chou, Breno de Medeiros, and Gene Tsudik. Sanitizable signatures. In Sabrina De Capitani di Vimercati, Paul F. Syverson, and Dieter Gollmann, editors, *ESORICS 2005: 10th European Symposium on Research in Computer Security*, volume 3679 of *Lecture Notes in Computer Science*, pages 159–177, Milan, Italy, September 12–14, 2005. Springer, Heidelberg, Germany. (cited on pages 2 and 21.)

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT'96*, volume 1163 of *Lecture Notes in Computer Science*, pages 244–251, Kyongju, Korea, November 3–7, 1996. Springer, Heidelberg, Germany. (cited on pages 5 and 79.)

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended ab-

stract). In *28th Annual ACM Symposium on Theory of Computing*, pages 99–108, Philadephia, PA, USA, May 22–24, 1996. ACM Press. (cited on page 12.)

[Alk15] Nabil Alkeilani Alkadri. Post-quantum commitment schemes. Master's thesis, Technische Universität Darmstadt, 2015. http://tubiblio.ulb.tu-darmstadt.de/104187/. (cited on page 62.)

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 271–286, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany. (cited on pages 5, 59, 79, and 106.)

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 415–432, Queenstown, New Zealand, December 1–5, 2002. Springer, Heidelberg, Germany. (cited on page 106.)

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015. (cited on pages 14 and 33.)

[ASY21] Shweta Agrawal, Damien Stehle, and Anshu Yadav. Towards practical and round-optimal lattice-based threshold and blind signatures. Cryptology ePrint Archive, Report 2021/381, 2021. https://eprint.iacr.org/2021/381. (cited on page 5.)

[BCK+14] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014, Part I*, volume 8873 of *Lecture Notes in Computer Science*, pages 551–572, Kaoshiung, Taiwan, R.O.C., December 7–11, 2014. Springer, Heidelberg, Germany. (cited on page 85.)

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. (cited on page 16.)

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 10–24, Arlington, VA, USA, January 10–12, 2016. ACM-SIAM. (cited on page 14.)

[BF11] Dan Boneh and David Mandell Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011: 14th International Conference on Theory and Practice of Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 1–16, Taormina, Italy, March 6–9, 2011. Springer, Heidelberg, Germany. (cited on page 10.)

[BFLS10] Christina Brzuska, Marc Fischlin, Anja Lehmann, and Dominique Schröder. Unlinkability of sanitizable signatures. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010: 13th International Conference on Theory and Practice of Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 444–461, Paris, France, May 26–28, 2010. Springer, Heidelberg, Germany. (cited on pages 2 and 21.)

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014*, volume 8366 of *Lecture Notes in Computer Science*, pages 28–47, San Francisco, CA, USA, February 25–28, 2014. Springer, Heidelberg, Germany. (cited on pages 4, 27, 38, 39, and 44.)

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology – EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 416–432, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany. (cited on page 106.)

[BL13a] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013: 20th Conference on Computer and Communications Security*, pages 1087–1098, Berlin, Germany, November 4–8, 2013. ACM Press. (cited on pages 4 and 53.)

[BL13b] Foteini Baldimtsi and Anna Lysyanskaya. On the security of one-witness blind signature schemes. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part II*, volume 8270 of *Lecture Notes in Computer Science*, pages 82–99, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. (cited on page 55.)

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 33–53, Zagreb, Croatia, October 17–21, 2021. Springer, Heidelberg, Germany. (cited on page 60.)

[BLO18] Carsten Baum, Huang Lin, and Sabine Oechsner. Towards practical lattice-based one-time linkable ring signatures. In David Naccache, Shouhuai Xu, Sihan Qing, Pierangela Samarati, Gregory Blanc, Rongxing Lu, Zonghua Zhang, and Ahmed Meddahi, editors, *ICICS 18: 20th International Conference on Information and Communication Security*, volume 11149 of *Lecture Notes in Computer Science*, pages 303–322, Lille, France, October 29–31, 2018. Springer, Heidelberg, Germany. (cited on pages 4 and 39.)

[Blu81] Manuel Blum. Coin flipping by telephone. In Allen Gersho, editor, *Advances in Cryptology – CRYPTO'81*, volume ECE Report 82-04, pages 11–15, Santa Barbara, CA, USA, 1981. U.C. Santa Barbara, Dept. of Elec. and Computer Eng. (cited on pages 59 and 61.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. (cited on page 17.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. (cited on pages 5 and 16.)

[BS99] Johannes Blömer and Jean-Pierre Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *31st Annual ACM Symposium on Theory of Computing*, pages 711–720, Atlanta, GA, USA, May 1–4, 1999. ACM Press. (cited on page 12.)

[BSS02] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 465–480, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. (cited on page 106.)

[CCT+11] Liang Chen, Yongquan Cui, Xueming Tang, Dongping Hu, and Xin Wan. Hierarchical id-based blind signature from lattices. In *International Conference on Computational Intelligence and Security, CIS 2011*, pages 803–807. IEEE Computer Society, 2011. (cited on pages 4, 54, 103, and 104.)

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *Advances in Cryptology – CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Heidelberg, Germany. (cited on pages 5, 55, 79, 83, 84, 85, and 106.)

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology – CRYPTO'82*, pages 199–203, Santa Barbara, CA, USA, 1982. Plenum Press, New York, USA. (cited on pages 4 and 53.)

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrete du chiffrement completement homomorphe*. PhD thesis, ENS-Lyon, France, 2013. (cited on page 14.)

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 1–20, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany. (cited on page 14.)

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 573–590, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany. (cited on page 59.)

[Com11] Gemalto Digital Security Company. Integration of Gemalto's smart card security with Microsoft U-Prove. https://www.securetechalliance.org/gemalto-integrates-smart-card-security-with-microsoft-u-prove, 2011. (cited on page 53.)

[Coo10] Howard A. Schmidt (National Cybersecurity Coordinator). National strategy for trusted identities in cyberspace. Cyberwar Resources Guide, Item #163, 2010. http://www.projectcyw-d.org/resources/items/show/163. (cited on page 53.)

[Cor07] Microsoft Corporation. Microsoft's Open Specification Promise. https://docs.microsoft.com/en-us/openspecs/dev_center/ms-devcentlp/1c24c7c8-28b0-4ce1-a47d-95fe1ff504bc, 2007. (cited on page 53.)

[Cra96] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, CWI and University of Amsterdam, 1996. (cited on page 81.)

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany. (cited on page 106.)

[DFL19] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 651–668. ACM Press, November 11–15, 2019. (cited on pages 2, 21, 24, 33, 34, and 35.)

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir transformation in the quantum random-oracle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 356–383, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. (cited on page 27.)

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. (cited on page 3.)

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839. (cited on pages 2, 4, 26, 27, 36, 37, 39, and 44.)

[DLL+17a] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, Version: 20170627:201152, 2017. http://eprint.iacr.org/2017/633. (cited on page 11.)

[DLL+17b] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. https://eprint.iacr.org/2017/633. (cited on page 100.)

[DPP94] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *Advances in Cryptology – CRYPTO'93*, volume 773 of *Lecture Notes in Computer Science*, pages 250–265, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Heidelberg, Germany. (cited on pages 59 and 61.)

[ES16] Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In Sara Foresti and Giuseppe Persiano, editors, *CANS 16: 15th International Conference on Cryptology and Network Security*, volume 10052 of *Lecture Notes in Computer Science*, pages 140–155, Milan, Italy, November 14–16, 2016. Springer, Heidelberg, Germany. (cited on pages 3, 4, 39, and 40.)

[FHJ20] Marc Fischlin, Patrick Harasser, and Christian Janson. Signatures from sequential-OR proofs. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 212–244, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. (cited on page 106.)

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *Advances in Cryptology – CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 60–77, Santa Barbara, CA, USA, August 20–24, 2006. Springer, Heidelberg, Germany. (cited on page 59.)

[FKM+16] Nils Fleischhacker, Johannes Krupp, Giulio Malavolta, Jonas Schneider, Dominique Schröder, and Mark Simkin. Efficient unlinkable sanitizable signatures from signatures with re-randomizable keys. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 9614 of *Lecture Notes in Computer Science*, pages 301–330, Taipei, Taiwan, March 6–9, 2016. Springer, Heidelberg, Germany. (cited on pages 2, 3, 21, 22, 23, 24, 34, 35, 36, and 37.)

[Fra15] Pedro Franco. *Understanding Bitcoin: Cryptography, engineering and economics*. John Wiley & Sons, Chichester, UK, 2015. (cited on page 21.)

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany. (cited on pages 3, 25, and 42.)

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd Annual ACM Symposium on Theory of Computing*, pages 416–426, Baltimore, MD, USA, May 14–16, 1990. ACM Press. (cited on page 83.)

[FS09] Marc Fischlin and Dominique Schröder. Security of blind signatures under aborts. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 297–316, Irvine, CA, USA, March 18–20, 2009. Springer, Heidelberg, Germany. (cited on page 59.)

[FS10] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 197–215, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. (cited on page 59.)

[Gam20] Jay Gambetta. IBM's roadmap for scaling quantum technology. https://research.ibm.com/blog/ibm-quantum-roadmap, 2020. (cited on page 1.)

[GHW+17] Wen Gao, Yupu Hu, Baocang Wang, Jia Xie, and Momeng Liu. Identity-based blind signature from lattices. *Wuhan University Journal of Natural Sciences*, 22(4):355–360, 2017. (cited on pages 4, 54, 103, and 104.)

[GHWX16] Wen Gao, Yupu Hu, Baocang Wang, and Jia Xie. Identity-based blind signature from lattices in standard model. In *Information Security and Cryptology - Inscrypt 2016*, pages 205–218. Springer, 2016. (cited on pages 4, 54, 103, and 104.)

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *ICS 2010: 1st Innovations in Computer Science*, pages 230–240, Tsinghua University, Beijing, China, January 5–7, 2010. Tsinghua University Press. (cited on pages 4, 36, and 44.)

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988. (cited on page 15.)

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press. (cited on pages 4, 10, 104, and 105.)

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, volume 6841 of *Lecture Notes in Computer Science*, pages 630–648, Santa Barbara, CA, USA, August 14–18, 2011. Springer, Heidelberg, Germany. (cited on page 5.)

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *Lecture Notes in Computer Science*, pages 43–60, Christ Church, Barbados, February 26, 2016. Springer, Heidelberg, Germany. (cited on pages 4 and 53.)

[HBG+18] Andreas Hülsing, Denis Butin, Stefan Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018. (cited on page 52.)

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 345–375, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany. (cited on pages 5 and 78.)

[HKLN20] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 500–529, Santa Barbara,

CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany. (cited on pages 5, 53, 55, 59, 60, 78, 79, 80, and 106.)

[IN83] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, (71):1–8, 1983. (cited on page 106.)

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *Advances in Cryptology – CRYPTO'97*, volume 1294 of *Lecture Notes in Computer Science*, pages 150–164, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany. (cited on page 57.)

[KKS17] Mahender Kumar, Chittaranjan Padmanabha Katti, and Prem Chandra Saxena. A secure anonymous e-voting system using identity-based blind signature scheme. In *International Conference on Information Systems Security, ICISS 2017*, pages 29–49. Springer, 2017. (cited on pages 4 and 53.)

[KKW18] Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 525–537, Toronto, ON, Canada, October 15–19, 2018. ACM Press. (cited on page 4.)

[KLR21] Jonathan Katz, Julian Loss, and Michael Rosenberg. Boosting the security of blind signature schemes. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021*, volume 13093, pages 468–492, Cham, 2021. Springer International Publishing. (cited on page 106.)

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 552–586, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. (cited on pages 13, 14, 27, 42, and 106.)

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 33–61, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. (cited on page 42.)

[Laa16] Thijs Laarhoven. *Search problems in cryptography*. PhD thesis, Eindhoven University of Technology, 2016. (cited on page 14.)

[LN17] Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances*

in Cryptology – EUROCRYPT 2017, Part I, volume 10210 of *Lecture Notes in Computer Science*, pages 293–323, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany. (cited on page 85.)

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany. (cited on page 12.)

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015. (cited on page 12.)

[LS18] Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part I*, volume 10820 of *Lecture Notes in Computer Science*, pages 204–224, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany. (cited on page 8.)

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In Ronald Cramer, editor, *PKC 2008: 11th International Workshop on Theory and Practice in Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 162–179, Barcelona, Spain, March 9–12, 2008. Springer, Heidelberg, Germany. (cited on page 54.)

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 598–616, Tokyo, Japan, December 6–10, 2009. Springer, Heidelberg, Germany. (cited on pages 3, 4, 40, 43, 45, 47, 85, 87, 89, and 105.)

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany. (cited on pages 4, 10, 39, 44, and 85.)

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting post-quantum Fiat-Shamir. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 326–355, Santa Barbara, CA, USA, August 18–22, 2019. Springer, Heidelberg, Germany. (cited on page 27.)

[Mic02] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd*

Annual Symposium on Foundations of Computer Science, pages 356–365, Vancouver, BC, Canada, November 16–19, 2002. IEEE Computer Society Press. (cited on page 12.)

[Mic11] Daniele Micciancio. The geometry of lattice cryptography. In Alessandro Aldini and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design VI - FOSAD Tutorial Lectures*, volume 6858 of *Lecture Notes in Computer Science*, pages 185–210. Springer, 2011. (cited on page 9.)

[Mos18] Michele Mosca. Cybersecurity in an era with quantum computers: Will we be ready? *IEEE Security & Privacy*, 16(5):38–41, 2018. (cited on page 1.)

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual Symposium on Foundations of Computer Science*, pages 372–381, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press. (cited on page 10.)

[MSM+16] Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the schnorr signature scheme and DSA against related-key attacks. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15: 18th International Conference on Information Security and Cryptology*, volume 9558 of *Lecture Notes in Computer Science*, pages 20–35, Seoul, Korea, November 25–27, 2016. Springer, Heidelberg, Germany. (cited on pages 21, 28, and 29.)

[Nat17] National Institute of Standards and Technology (NIST). Post-Quantum Cryptography Standardization. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2017. (cited on page 2.)

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 31–53, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany. (cited on page 106.)

[Paq13] Christian Paquin. U-Prove technology overview v1.1 (revision 2). Microsoft Corporation, 2013. https://www.microsoft.com/en-us/research/publication/u-prove-technology-overview-v1-1-revision-2/. (cited on pages 53 and 55.)

[Poi98] David Pointcheval. Strengthened security for blind signatures. In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 391–405, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. (cited on page 106.)

[PotEU09] European Parliament and Council of the European Union. Directive 2009/136/EC. *Official Journal of the European Union*, 2009. (cited on page 53.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. (cited on pages 57, 59, 78, and 106.)

[PSM17] Albrecht Petzoldt, Alan Szepieniec, and Mohamed Saied Emam Mohamed. A practical multivariate blind signature scheme. In Aggelos Kiayias, editor, *FC 2017: 21st International Conference on Financial Cryptography and Data Security*, volume 10322 of *Lecture Notes in Computer Science*, pages 437–454, Sliema, Malta, April 3–7, 2017. Springer, Heidelberg, Germany. (cited on page 53.)

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93, Baltimore, MA, USA, May 22–24, 2005. ACM Press. (cited on page 12.)

[Rüc10] Markus Rückert. Lattice-based blind signatures. In Masayuki Abe, editor, *Advances in Cryptology – ASIACRYPT 2010*, volume 6477 of *Lecture Notes in Computer Science*, pages 413–430, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany. (cited on pages 3, 4, 5, 39, 53, 54, 55, 61, and 78.)

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, January 1991. (cited on pages 3, 22, and 39.)

[Sch01] Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01: 3rd International Conference on Information and Communication Security*, volume 2229 of *Lecture Notes in Computer Science*, pages 1–12, Xian, China, November 13–16, 2001. Springer, Heidelberg, Germany. (cited on page 59.)

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. (cited on page 14.)

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. (cited on page 1.)

[SU17] Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. *Journal of Cryptology*, 30(2):470–494, April 2017. (cited on page 59.)

[TSS+18] Wilson Abel Alberto Torres, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, Veronika Kuchta, Nandita Bhattacharjee, Man Ho Au, and Jacob Cheng. Post-quantum one-time linkable ring signature and application to ring confidential transactions in blockchain (lattice RingCT v1.0). In Willy Susilo and

Guomin Yang, editors, *ACISP 18: 23rd Australasian Conference on Information Security and Privacy*, volume 10946 of *Lecture Notes in Computer Science*, pages 558–576, Wollongong, NSW, Australia, July 11–13, 2018. Springer, Heidelberg, Germany. (cited on page 39.)

[Unr17] Dominique Unruh. Post-quantum security of Fiat-Shamir. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 65–95, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany. (cited on page 27.)

[vN51] John von Neumann. Various techniques used in connection with random digits. In *Monte Carlo Method*, pages 36–38. National Bureau of Standards Applied Mathematics Series, 12, 1951. (cited on page 3.)

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 288–303, Santa Barbara, CA, USA, August 18–22, 2002. Springer, Heidelberg, Germany. (cited on page 60.)

[ZH16] Yanhua Zhang and Yupu Hu. Forward-secure identity-based shorter blind signature from lattices. *American Journal of Networks and Communications*, 5(2):17–26, 2016. (cited on pages 4, 54, 103, and 104.)

[ZM14] Lili Zhang and Yanqin Ma. A lattice-based identity-based proxy blind signature scheme in the standard model. *Mathematical Problems in Engineering*, 2014, 2014. (cited on pages 4, 54, 103, and 104.)

[ZTZ+17] Hongfei Zhu, Yu-an Tan, Xiaosong Zhang, Liehuang Zhu, Changyou Zhang, and Jun Zheng. A round-optimal lattice-based blind signature scheme for cloud services. *Future Generation Computer Systems*, 73:106–114, 2017. (cited on pages 4, 54, and 103.)

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit - abgesehen von den in ihr ausdrücklich genannten Hilfen - selbständig verfasst habe.

Darmstadt, January 2022 _____