On Lattice-Based Interactive Protocols: An Approach with Less or No Aborts

Nabil Alkeilani Alkadri[1], Rachid El Bansarkhani[2], and Johannes Buchmann[1]

[1] Technische Universität Darmstadt, Germany
nabil.alkadri@tu-darmstadt.de, buchmann@cdc.informatik.tu-darmstadt.de
[2] QuantiCor Security GmbH, Germany
rachid.elbansarkhani@quanticor-security.de

Abstract. A canonical identification (CID) scheme is a 3-move protocol consisting of a commitment, challenge, and response. It constitutes the core design of many cryptographic constructions such as zero-knowledge proof systems and various types of signature schemes. Unlike number-theoretic constructions, CID in the lattice setting usually forces provers to abort and repeat the whole authentication process once the distribution of the computed response does not follow a target distribution independent from the secret key. This concept has been realized by means of rejection sampling, which makes sure that the secrets involved in a protocol are concealed after a certain number of repetitions. This however has a negative impact on the efficiency of interactive protocols because it leads to a number of communication rounds that is multiplicative in the number of aborting participants (or rejection sampling procedures). In this work we show how the CID scheme underlying many lattice-based protocols can be designed with smaller number of aborts or even without aborts. Our new technique exploits (unbalanced) binary hash trees and thus significantly reduces the communication complexity. We show how to apply this new method within interactive zero-knowledge proofs. We also present BLAZE^+: a further application of our technique to the recently proposed lattice-based blind signature scheme BLAZE (FC'20). We show that BLAZE^+ has an improved performance and communication complexity compared to BLAZE while preserving the size of keys and signatures.

Keywords: Lattice-based cryptography · Aborts · Hash trees

1 Introduction

A canonical identification (CID) scheme allows a prover $\mathcal{P}$ to prove to a verifier $\mathcal{V}$ the possession of a secret key s in the following way: $\mathcal{P}$ sends a commitment to $\mathcal{V}$, who then sends a challenge c back to $\mathcal{P}$. Upon receiving c, $\mathcal{P}$ answers with a response z. This response allows $\mathcal{V}$ to verify $\mathcal{P}$'s authenticity while not leaking any information about the secret key. In number-theoretic constructions like Schnorr's CID scheme [Sch91], the response z already hides s, since it is computed by adding a secret masking term y to the term sc, i.e., $z = y + sc$. The term y is chosen uniformly at random from a large distribution and is also used to compute the commitment. This approach has been generalized in [Lyu09] to include aborting provers for the lattice setting. In a lattice-based CID scheme y is required to be chosen from a narrow distribution (typically, Gaussian or uniform) and the so called rejection sampling procedure [vN51] is used to hide the distribution of sc. If the sum z is not accepted, a new masking term is sampled. This procedure is repeated until the sum becomes independently distributed from the secret term sc. Lattice-based CID is a fundamental building block of many cryptographic constructions including zero-knowledge protocols (e.g., [BCK+14, BDL+18]) as well as signature schemes (e.g., [Lyu12, BG14, DKL+18]) and even those with advanced functionalities such as ring signatures (e.g., [BLO18, TSS+18]), blind signatures [Rüc10, AEB20], and multisignatures [ES16].

While aborting does not affect the efficiency of constructions with one rejection sampling process like ordinary signatures, it has a significant negative impact on the performance and communication complexity of lattice-based interactive protocols with multiple rejection sampling procedures. For instance, the multisignature scheme proposed in [ES16] entails a repetition rate that grows exponentially in the number of users participating in the signing protocol. Though it is efficient for a small set of users, one would need to restart the protocol very often when instantiated with a large set because each user has to carry out rejection sampling. Another example is the blind signature scheme BLAZE [AEB20] and its predecessor introduced in [Rüc10]. In both constructions not only signers have to carry out rejection sampling and repeat the signing process $M_{\mathcal{S}}$ times until the secret key is concealed, but for maintaining blindness even users have to apply rejection sampling $M_{\mathcal{U}}$ times and request a protocol restart in case of failure. This imposes a multiplicative repetition rate $M_{\mathcal{S}} \cdot M_{\mathcal{U}}$ and an additional communication step due to the possibility of failures causing protocol restarts. In this case, a proof of failure is sent to the signer, i.e., a proof that allows the signer to verify the occurrence of a failure. Although BLAZE has been shown to be practical [AEB20], this additional step increases the time and communication complexity required to generate valid signatures and forces the use of statistically hiding and computationally binding commitments to retain security.

Therefore, masking secrets in lattice-based interactive protocols with multiple rejection sampling procedures such that aborting occurs as little as possible while maintaining efficiency and security remained a very important research question. This would improve the running time and decrease the total amount of communication required to successfully complete the protocol.

1.1 Contributions

In this work we show how to reduce the number of repetitions in lattice-based protocols by means of a tool that we call *trees of commitments*. A tree of commitments is an (unbalanced) binary hash tree of height $h \geq 1$, whose leaves are the hash values of $\ell > 1$ commitments computed from masking terms sampled during an instance of a CID-based protocol. The number ℓ is chosen such that rejection sampling succeeds for at least one masking term at a given probability bound. This allows to aggregate ℓ commitments in one tree and send only the root of the tree as a new commitment rather than ℓ commitments. The new response now further includes the authentication path of the leaf with index k $(0 \leq k < \ell)$, where at step k rejection sampling accepts for the first time after $k-1$ trials. Note that by choosing ℓ large enough we can remove aborts completely. Interestingly, only trees with small heights are required to reduce aborts to very small probabilities, e.g., $h = 3$ for a probability of at most 2^{-10}.

We demonstrate the effectiveness of using our method in interactive zero-knowledge proofs and blind signature schemes, while it could also be used for standard lattice-based signatures with aborts. More concretely, we show how to reduce the communication complexity of interactive zero-knowledge protocols by using trees of commitments in a lattice-based zero-knowledge proof of knowledge. Furthermore, we utilize trees of commitments in the blind signature scheme BLAZE [AEB20]. We call the new scheme BLAZE$^+$. In the new scheme a user constructs a tree of commitments using ℓ masking terms such that blindness is ensured at a given probability bound. More precisely, given a security level of λ bits we fix an aborting probability δ_{abort} and compute ℓ such that signatures are blind with probability of at least $1 - \delta_{\mathsf{abort}}$. For approximately 128 bits of security, our results (summarized in Table 1) show that while preserving the size of keys and signatures, the communication complexity is significantly decreased and the signing speed is improved for $\delta_{\mathsf{abort}} = 2^{-10}$. Note that choosing $\delta_{\mathsf{abort}} = 2^{-128}$ implies blindness with overwhelming probability. In this case (i.e., when $\delta_{\mathsf{abort}} = 2^{-\lambda}$) we can safely remove the last step of the protocol, hence proof of failures and the use of commitment schemes. Thus, we obtain a 3-move version of the protocol similar to the basic structure of CID. We present this version in Section 4 and the 4-move version in Appendix B, where aborts at the user side occur with probability of choice. We leave applying trees of commitments to multisignatures [ES16] as a future work.

Table 1. Comparing BLAZE^{+} (this work) with BLAZE [AEB20] at approximately 128 bits of security. The parameter δ_{abort} denotes the aborting probability by the user, and ℓ denotes the related number of masking terms. Performance is given in cycles and milliseconds (in parentheses), sizes and communication complexity in kilobytes. The corresponding parameters can be found in Table 3. Benchmarking the parameters were carried out on an Intel Core i7-6500U, operating at 2.3 GHz and 8GB of RAM.

Scheme	δ_{abort}	ℓ	**Complexity**	BS.KGen	BS.Sign	BS.Verify	**Secret key**	**Public key**	**Signature**
BLAZE^{+}	2^{-128}	71	177.8	$222,151$ (0.11)	$112,540,972$ (56.49)	$348,724$ (0.18)	0.75	3.9	6.7
BLAZE^{+}	2^{-40}	32	189.1	$222,151$ (0.11)	$56,193,762$ (28.21)	$348,724$ (0.18)	0.75	3.9	6.7
BLAZE^{+}	2^{-10}	8	189.2	$222,151$ (0.11)	$24,443,555$ (12.27)	$348,724$ (0.18)	0.75	3.9	6.6
BLAZE	0.38	1	351.6	$204,671$ (0.10)	$35,547,397$ (17.85)	$276,210$ (0.14)	0.8	3.9	6.6

Finally, we note that the impossibility results of 3-move blind signature schemes due to [FS10] do not apply to our 3-move version of BLAZE^{+}. These results show that finding black-box reductions from successful forgers to some non-interactive cryptographic assumption is infeasible in the standard model (i.e., without random oracles) for statistically blind schemes with 3 (or less) moves such that one can verify that an honest user was able to obtain a valid signature from the interaction with the (malicious) signer. In our 3-move protocol, there is no way to check that the user has obtained a valid signature, since he does not reveal the secret information that are involved in generating the signature and are required to check its validity. Furthermore, BLAZE^{+} is proven secure in the random oracle model rather than the standard model.

1.2 Techniques

We show how to reduce the number of repetitions or even remove aborts in CID-based protocols, completely. To this end, we give a brief description of the CID scheme that underlies many lattice-based constructions and was originally introduced in [Lyu09]. Let $\mathbf{A}$ be a public matrix selected uniformly at random from $\mathbb{Z}_q^{n\times m}$. The prover $\mathcal{P}$ would like to prove to a verifier $\mathcal{V}$ the possession of a secret matrix $\mathbf{S}\in\mathbb{Z}^{m\times n}$ with small entries such that $\mathbf{B}=\mathbf{AS} \pmod q$. We let χ denote some distribution over $\mathbb{Z}$. Typically, χ is either the discrete Gaussian distribution over $\mathbb{Z}$ or the uniform distribution over a small subset of $\mathbb{Z}$. The challenge space is defined by $\mathcal{C}=\{\mathbf{c}=(c_1,\ldots,c_n)\in\mathbb{Z}^n : c_i\in\{-1,0,1\}, \sum_1^n |c_i|=\kappa\}$. We let $\mathsf{RejSamp}$ denote an algorithm that carries out rejection sampling. The commitment is a vector $\mathbf{v}=\mathbf{Ay} \pmod q$, where $\mathbf{y}$ is a masking vector chosen from χ^m. For a challenge $\mathbf{c}\in\mathcal{C}$ the response is given by $\mathbf{z}=\mathbf{y}+\mathbf{Sc}$. The verifier accepts if and only if $\mathbf{v}=\mathbf{Az}-\mathbf{Bc} \pmod q$ and $\|\mathbf{z}\|_p\le B$, where B is a predefined bound and $p\in\{2,\infty\}$ depending on the distribution χ. Aborting occurs if $\mathsf{RejSamp}(\mathbf{z})$ does not accept. The protocol is always repeated by sampling a fresh $\mathbf{y}$ until $\mathsf{RejSamp}$ accepts such that $\mathbf{z}$ is statistically independent from $\mathbf{Sc}$.

Consider a lattice-based interactive protocol with $N\ge 1$ rejection sampling procedures, where each of them is repeated $x\ge 1$ times on average. The main motivation of this work is the observation that the total average number of repetitions M in such a protocol is multiplicative in N, i.e., $M=x^N$. Thus, the main question is: Can we improve it?

One can use a large enough distribution χ such that $\mathsf{RejSamp}$ accepts after a fixed number of repetitions M, e.g., $M\le 2$. This is already established in previous works as a trade-off between performance and sizes (see, e.g., [Lyu12, DDLL13, BG14, DKL+18]), but it does not solve the problem for all interactive protocols as explained above.

Our first attempt is the following. Rather than sampling one masking term $\mathbf{y}$ and repeating this process until $\mathsf{RejSamp}$ accepts, $\mathcal{P}$ generates $\ell > 1$ masking vectors $\mathbf{y}_j$ at once and computes the commitment $(\mathbf{v}_0, \ldots, \mathbf{v}_{\ell-1})$, where $\mathbf{v}_j = \mathbf{A}\mathbf{y}_j \pmod q$ and $j = 0, \ldots, \ell - 1$. The response is then $\mathbf{z}_k$, where k $(0 \leq k < \ell)$ is the first index for which $\mathsf{RejSamp}$ accepts. This reduces aborts, but the amount of exchanged data grows in ℓ. In particular, any type of lattice-based signature following this approach becomes very large. While this can be decreased by using some cryptographic hash function F and sending $\mathsf{F}(\mathbf{v}_j)$ instead of $\mathbf{v}_j$, this is still not satisfactory. An approach with some similarities has been taken in [dPL17] in a different context for zero-knowledge proofs, where all the hash values of commitments of potential masking terms are sent. We note that no tree structure for commitments has been applied in [dPL17] and furthermore the challenge size increases linearly in the number of masking terms, which is not the case in our attempt. The protocol is then repeated multiple times to achieve negligible soundness error. Thus, such an approach is still inefficient.

Our final solution to this issue is to use a *tree of commitments*: an (unbalanced) binary hash tree of height $h = \lceil \log(\ell) \rceil$, whose leaves are $\mathsf{F}(\mathbf{v}_j)$. The commitment is simply the root of the tree root, and the response is the pair $(\mathbf{z}_k, \mathsf{auth})$, where auth is the authentication path of the leaf with index k. Verification is carried out by checking that $\|\mathbf{z}_k\|_p \leq B$ and root is equal to the root of the tree associated to the leaf $\mathsf{F}(\mathbf{A}\mathbf{z}_k - \mathbf{B}\mathbf{c} \pmod q)$ and its given authentication path auth. Using a tree of commitments obviously reduces the communication complexity. It can also improve the performance of interactive protocols with multiple rejection sampling procedures as we demonstrate in this work. We note that the number of masking terms ℓ can be chosen such that the aborting probability is bounded by some given bound. In Section 3.3 we show how to optimize this number. We note that our technique may be used in [dPL17] to improve efficiency.

Finally, we briefly explain two further optimizations that can be exploited when using trees of commitments. The first one is to generate trees with randomized hashing similar to the standard of the hash-based signature scheme XMSS [HBG$^+$18]. This allows to save space and further reduce the communication complexity, since randomized hashing requires the hash function F to be only second preimage resistant rather than collision resistant. This means the output of F is required to be $\geq \lambda$ rather than $\geq 2\lambda$ bits assuming λ bits of classical security. The second optimization allows to reuse already generated, but not consumed, masking terms in subsequent executions of the protocol. This further improves the performance of the protocol, since complete subtrees of the tree can be reused. This reduces the number of new masking terms to be sampled in addition to the number of multiplications and hash computations.

1.3 Related work

In the context of analyzing the hardness of computational lattice problems, previous works such as [Gen09, BPMW16, BP18] point to techniques called "noise swallowing" or "super-polynomial noise flooding", which use Gaussian masking terms entailing a super-polynomial Gaussian parameter in order to swallow a polynomially large secret term. However, the negative impact on the efficiency is tremendous as the parameters become also super-polynomial. By generating many masking terms at once and capturing them in a tree of commitments, the secret and masking terms remain polynomially bounded while the number of repetitions is reduced. As mentioned in Section 1.2, the approach of sending hashed commitments has been used in [dPL17] for zero-knowledge proofs of small secrets, but without the use of tree structures for commitments and the other efficiency improvements. However, sending commitments in a tree structure has been suggested, e.g., in [KKW18] to reduce the communication complexity of proof systems, but not repetitions of lattice-based protocols. Our work exploits hash trees in the context of lattice-based interactive protocols with aborting participants.

1.4 Outline

In Section 2 we review the relevant background. In Section 3 we define trees of commitments and show how they can be utilized in lattice-based canonical identification schemes and hence, in interactive zero-knowledge protocols. In Section 4 we demonstrate the practical relevance of our new technique by introducing a new blind signature scheme that we call BLAZE^+.

2 Preliminaries

2.1 Basic Notation

We let $\mathbb{N}, \mathbb{Z}, \mathbb{R}$ denote the set of natural numbers, integers, and real numbers, respectively. We denote column vectors with bold lower-case letters and matrices with bold upper-case letters. The identity matrix of dimension n is denoted by $\mathbf{I}_n$. For any positive integer q we write $\mathbb{Z}_q$ to denote the set of integers in the range $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$. The Euclidean norm ($\ell_2$-norm) of a vector $\mathbf{v}$ with entries v_i is defined as $\|\mathbf{v}\| = (\sum_i |v_i|^2)^{1/2}$, and its ℓ_∞-norm as $\|\mathbf{v}\|_\infty = \max_i |v_i|$. We define the ring $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ and its quotient $R_q = R/qR$, where n is a power of 2. We assume that R is an integral domain. A ring element $a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} \in R_q$ is denoted by $\hat{a}$ and it corresponds to a vector $\mathbf{a} \in \mathbb{Z}_q^n$ via coefficient embedding, hence $\|\hat{a}\| = \|\mathbf{a}\|$ and $\|\hat{a}\|_\infty = \|\mathbf{a}\|_\infty$. We write $\hat{\mathbf{a}} = (\hat{a}_1, \ldots, \hat{a}_k) \in R_q^k$ to denote a vector of ring elements and $\hat{\mathbf{A}}$ for a matrix with entries from R_q. The norms of $\hat{\mathbf{a}}$ are defined by $\|\hat{\mathbf{a}}\| = (\sum_{i=1}^k \|\hat{a}_i\|^2)^{1/2}$ and $\|\hat{\mathbf{a}}\|_\infty = \max_i \|\hat{a}_i\|_\infty$. We let $\mathbb{T}_\kappa^n$ denote the set of all $(n-1)$-degree polynomials with coefficients from $\{-1, 0, 1\}$ and Hamming weight κ. All logarithms in this work are to base 2, i.e., $\log(\cdot) = \log_2(\cdot)$. We always denote the security parameter by $\lambda \in \mathbb{N}$. A function $f : \mathbb{N} \longrightarrow \mathbb{R}$ is called *negligible* if there exists an $n_0 \in \mathbb{N}$ such that for all $n > n_0$, it holds $f(n) < \frac{1}{p(n)}$ for any polynomial p. With $\mathrm{negl}(\lambda)$ we denote a negligible function in λ. A probability is called overwhelming if it is at least $1 - \mathrm{negl}(\lambda)$. The *statistical distance* between two distributions X, Y over a countable domain D is defined by $\Delta(X, Y) = \frac{1}{2}\sum_{n \in D} |X(n) - Y(n)|$. The distributions X, Y are called *statistically close* if $\Delta(X, Y) = \mathrm{negl}(\lambda)$. We write $x \leftarrow D$ to denote that x is sampled according to a distribution D. We let $x \leftarrow_\$ S$ denote choosing x uniformly random from a finite set S.

2.2 Cryptographic Primitives

This section defines canonical identification and blind signature schemes and related security notions.

A canonical identification (CID) scheme is a 3-move interactive protocol of the following form: A prover $\mathcal{P}$ initiates the protocol by sending a commitment message y to a verifier $\mathcal{V}$. Upon receiving y, $\mathcal{V}$ sends a uniform random challenge c to $\mathcal{P}$. Afterwards, a response z is sent from $\mathcal{P}$ back to $\mathcal{V}$, which then allows $\mathcal{V}$ to make a deterministic decision about $\mathcal{P}$'s authenticity. The tuple (y, c, z) represents a protocol transcript. A formal definition follows.

Definition 1 (Canonical Identification Scheme). *A canonical identification scheme with commitment space $\mathcal{Y}$, challenge space $\mathcal{C}$, and response space $\mathcal{Z}$ is defined as a tuple of the following polynomial-time algorithms:*

- $\mathsf{KG}(1^\ell)$ *is a key generation algorithm that outputs a pair of keys $(\mathsf{pk}, \mathsf{sk})$ from some key space $\mathcal{K}$, where pk is a public key and sk is a secret key.*
- $\mathsf{P} = (\mathsf{P}_1(\mathsf{sk}), \mathsf{P}_2(\mathsf{sk}, y, c, \mathsf{st}))$ *is a prover algorithm consisting of two algorithms: P_1 takes as input a secret key sk and returns a commitment $y \in \mathcal{Y}$ and a state st, whereas P_2 on input sk, y, a challenge $c \in \mathcal{C}$, and st, outputs a response $z \in \mathcal{Z} \cap \{\perp\}$, where the symbol $\perp \notin \mathcal{Z}$ indicates failure.*

```
Game $\mathsf{Forge}_{\mathsf{BS},\mathcal{U}^*}(\lambda)$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{BS.KGen}(1^\lambda)$
$\mathsf{H} \leftarrow \mathcal{H}(1^\lambda)$
$((\mu_1,\sigma_1),\ldots,(\mu_l,\sigma_l)) \leftarrow \mathcal{U}^{*\mathsf{H}(\cdot),\langle\mathcal{S}(\mathsf{sk}),\cdot\rangle^\infty}(\mathsf{pk})$
$k :=$ number of successful signing invocations
if $\big(\mu_i \neq \mu_j$ for all $1 \leq i < j \leq l \ \wedge\ \mathsf{BS.Verify}(\mathsf{pk},\mu_i,\sigma_i) = 1, \forall i \in [l] \ \wedge\ k+1 = l\big)$ then
    return 1
return 0
```

Fig. 1. The security game of one-more unforgeability of blind signatures.

- *$\mathsf{V}(\mathsf{pk},y,c,z)$ is a verification algorithm that takes as input a public key pk and a transcript (y,c,z), and outputs 1 if it is valid and 0 otherwise.*

Any CID scheme must satisfy the correctness property. It states that the algorithm V always (or with overwhelming probability) validates honestly generated transcripts, i.e., for all $\ell \in \mathbb{N}$, all $(\mathsf{pk},\mathsf{sk})$, and all honestly generated transcripts (y,c,z), it holds that $\Pr[\mathsf{V}(\mathsf{pk},y,c,z) = 1 \mid z \neq \bot] \geq 1-\delta$, where $\delta = \mathrm{negl}(\ell)$ is called the correctness error. The standard security notion of CID schemes is impersonation under the active or passive attack model. In the active attack model, any adversary $\mathcal{A}$ interacting with a prover as a verifier must not be able to extract any useful information. 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 [AABN02], impersonation under passive attacks is stronger than the active attack model.

Definition 2 (Blind Signature Scheme). *A blind signature scheme BS is a tuple of polynomial-time algorithms BS=($BS.KGen$,$BS.Sign$,$BS.Verify$) such that:*

- *$BS.KGen(1^\lambda)$ is a key generation algorithm that outputs a pair of keys (pk,sk), where pk is a public key and sk is a secret key.*
- *$BS.Sign(sk, pk, \mu)$ is an interactive protocol between a signer $\mathcal{S}$ and a user $\mathcal{U}$. The input of $\mathcal{S}$ is a secret key sk, whereas the input of $\mathcal{U}$ is a public key pk and a message $\mu \in \mathcal{M}$, where $\mathcal{M}$ is the message space. The output of $\mathcal{S}$ is a view $\mathcal{V}$ (interpreted as a random variable) and the output of $\mathcal{U}$ is a signature σ, i.e., $(\mathcal{V},\sigma) \leftarrow \langle\mathcal{S}(sk),\mathcal{U}(pk,\mu)\rangle$. We write $\sigma = \bot$ to denote failure.*
- *$BS.Verify(pk,\mu,\sigma)$ is a verification algorithm that outputs 1 if the signature σ is valid and 0 otherwise.*

Blind signature schemes require the completeness property, i.e., $\mathsf{BS.Verify}$ always (or with overwhelming probability) validates honestly signed messages under honestly created keys. Security of blind signatures is captured by two security notions: blindness and one-more unforgeability [JLO97, PS00]. The former prevents a malicious signer to learn information about user's messages (see [AEB20] for a formal definition). The latter ensures that each completed execution of $\mathsf{BS.Sign}$ yields at most one signature.

Definition 3 (One-More Unforgeability). *Let $\mathcal{H}$ be a family of random oracles. A blind signature scheme BS is called $(t, q_{Sign}, q_H, \varepsilon)$-one-more unforgeable in the random oracle model if for any adversarial user $\mathcal{U}^*$ running in time at most t and making at most q_{Sign} signing and q_H hash queries, the game $Forge_{BS,\mathcal{U}^*}(\lambda)$ depicted in Figure 1 outputs 1 with probability $\Pr[Forge_{BS,\mathcal{U}^*}(\lambda) = 1] \leq \varepsilon$. The scheme is strongly $(t, q_{Sign}, q_H, \varepsilon)$-one-more unforgeable if the condition $\mu_i \neq \mu_j$ in the game changes to $(\mu_i,\sigma_i) \neq (\mu_j,\sigma_j)$ for all $1 \leq i < j \leq l$.*

2.3 Lattices and Gaussians

Let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_k\} \in \mathbb{R}^{m \times k}$ be a set of linearly independent vectors for $k \leq m$. The m-dimensional *lattice* $\mathcal{L}$ of rank k generated by $\mathbf{B}$ is given by $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} \mid \mathbf{x} \in \mathbb{Z}^k\} \subset \mathbb{R}^m$.

The *discrete Gaussian distribution* $D_{\mathcal{L},\sigma,\mathbf{c}}$ over a lattice $\mathcal{L}$ with standard deviation $\sigma > 0$ and center $\mathbf{c} \in \mathbb{R}^n$ is defined as follows: For every $\mathbf{x} \in \mathcal{L}$ the probability of $\mathbf{x}$ is $D_{\mathcal{L},\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathcal{L})$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(\frac{-\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\mathcal{L}) = \sum_{\mathbf{x}\in\mathcal{L}} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. The subscript $\mathbf{c}$ is taken to be $\mathbf{0}$ when omitted. We recall a lemma that gives a tail bound on discrete Gaussians and a rejection sampling lemma.

Lemma 1 ([Lyu12, Lemma 4.4]). *For any $t, \eta > 0$ we have*

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

Lemma 2 ([Lyu12, Theorem 4.6, Lemma 4.7]). *Let $V \subseteq \mathbb{Z}^m$ with elements having norms bounded by T, $\sigma = \omega(T\sqrt{\log m})$, and $h : V \to \mathbb{R}$ be a probability distribution. Then there exists a constant $M = O(1)$ such that $\forall \boldsymbol{v} \in V : \Pr[D_{\mathbb{Z}^m,\sigma}(\boldsymbol{z}) \leq M \cdot D_{\mathbb{Z}^m,\sigma,\boldsymbol{v}}(\boldsymbol{z});\ \boldsymbol{z} \leftarrow D_{\mathbb{Z}^m,\sigma}] \geq 1-\varepsilon$, where $\varepsilon = 2^{-\omega(\log m)}$. Furthermore, the following two algorithms are within statistical distance $\delta = \varepsilon/M$.*

1. *$\boldsymbol{v} \leftarrow h$, $\boldsymbol{z} \leftarrow D_{\mathbb{Z}^m,\sigma,\boldsymbol{v}}$, output $(\boldsymbol{z}, \boldsymbol{v})$ with probability $\frac{D_{\mathbb{Z}^m,\sigma}(\boldsymbol{z})}{M \cdot D_{\mathbb{Z}^m,\sigma,\boldsymbol{v}}(\boldsymbol{z})}$.*
2. *$\boldsymbol{v} \leftarrow h$, $\boldsymbol{z} \leftarrow D_{\mathbb{Z}^m,\sigma}$, output $(\boldsymbol{z}, \boldsymbol{v})$ with probability $1/M$.*

Moreover, the probability that the first algorithm outputs something is at least $(1-\varepsilon)/M$. If $\sigma = \alpha T$ for any positive α, then $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$ with $\varepsilon = 2^{-100}$.

We let $\mathsf{RejSamp}(x)$ denote an algorithm that carries out rejection sampling on input x. The algorithm outputs 1 if it accepts and 0 otherwise. To specify the randomness r used within the algorithm we write $\mathsf{RejSamp}(x; r)$. Next we define the lattice problems related to this work.

Definition 4 (Module Short Integer Solution (MSIS) Problem). *Let n, q, k_1, k_2 be positive integers and β a positive real. Given a uniform random matrix $\hat{\mathbf{A}} \in R_q^{k_1 \times k_2}$, the Hermite Normal Form of MSIS asks to find a non-zero vector $\hat{\mathbf{x}} \in R^{k_1+k_2}$ such that $[\mathbf{I}_{k_1}\ \ \hat{\mathbf{A}}] \cdot \hat{\mathbf{x}} = \mathbf{0} \pmod q$, where $\|\hat{\mathbf{x}}\| \leq \beta$.*

Definition 5 (Module Learning With Errors (MLWE) Problem). *Let n, q, k_1, k_2 be positive integers and $\hat{\mathbf{A}}$ be a matrix chosen uniformly at random from $R_q^{k_1 \times k_2}$. Given $(\hat{\mathbf{A}}, \hat{\mathbf{b}})$, the decision MLWE problem asks to distinguish (with non-negligible advantage) whether $\hat{\mathbf{b}}$ were chosen from the uniform distribution over $R_q^{k_1}$ or from the following distribution: Given $\hat{\mathbf{s}} \leftarrow \chi^{k_2}$ and $\hat{\mathbf{e}} \leftarrow \chi^{k_1}$, output the vector $\hat{\mathbf{A}}\hat{\mathbf{s}} + \hat{\mathbf{e}} \pmod q$, where χ is an error distribution (typically, either $D_{\mathbb{Z}^n,\sigma}$ or the uniform distribution over a small subset of R_q).*

The MLWE problem [LS15] generalizes LWE [Reg05] and RLWE [LPR10]. More precisely, by setting $k_1 = 1$ in the definition above we obtain the ring version 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] and its special versions SIS [Ajt96] and RSIS [Mic02].

3 How to Reduce Aborts in Lattice-Based Protocols

In this section we show how aborting in lattice-based protocols can be reduced or even be removed at all. As stated in Section 1, when the number of rejection sampling procedures N in an interactive CID-based

```
$\mathsf{HashTree}(v_0, \ldots, v_{\ell-1})$
$h \leftarrow \lceil \log(\ell) \rceil$
$\mathsf{tree} := \varnothing$
for $(j = 0, \ldots, \ell - 1)$ do
    $v_0[j] \leftarrow \mathsf{F}(v_j)$
    $\mathsf{tree} \leftarrow \mathsf{tree} \cup \{v_0[j]\}$
for $(j = \ell, \ldots, 2^h - 1)$ do
    $v_0[j] \leftarrow \mathsf{F}(j)$
    $\mathsf{tree} \leftarrow \mathsf{tree} \cup \{v_0[j]\}$
for $(i = 1, \ldots, h)$ do
    for $(j = 0, \ldots, 2^{h-i} - 1)$ do
        $v_i[j] \leftarrow \mathsf{F}(v_{i-1}[2j], v_{i-1}[2j+1])$
        $\mathsf{tree} \leftarrow \mathsf{tree} \cup \{v_i[j]\}$
$\mathsf{root} \leftarrow v_h[0]$
return $(\mathsf{root}, \mathsf{tree})$
```

```
$\mathsf{BuildAuth}(k, \mathsf{tree}, h)$
$\mathsf{tree} := (v_i[j])_{i,j}$, $0 \leq i \leq h, 0 \leq j < 2^{h-i}$, $v_i[j] \in \{0,1\}^\omega$
for $(i = 0, \ldots, h - 1)$ do
    $s \leftarrow \lfloor k/2^i \rfloor$
    $\mathsf{bit} \leftarrow s \bmod 2$
    if $(\mathsf{bit} = 1)$ then
        $\mathbf{a}_i \leftarrow v_i[s-1]$
    else
        $\mathbf{a}_i \leftarrow v_i[s+1]$
$\mathsf{auth} := (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$
return $\mathsf{auth}$
```

```
$\mathsf{RootCalc}(v, \mathsf{auth})$
$\mathsf{auth} := (k, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$, $\mathbf{a}_i \in \{0,1\}^\omega$, $i = 0, \ldots, h-1$
$\mathbf{b}_0 \leftarrow \mathsf{F}(v)$
for $(i = 1, \ldots, h)$ do
    $s \leftarrow \lfloor k/2^{i-1} \rfloor$
    $\mathsf{bit} \leftarrow s \bmod 2$
    if $(\mathsf{bit} = 1)$ then
        $\mathbf{b}_i \leftarrow \mathsf{F}(\mathbf{a}_{i-1}, \mathbf{b}_{i-1})$
    else
        $\mathbf{b}_i \leftarrow \mathsf{F}(\mathbf{b}_{i-1}, \mathbf{a}_{i-1})$
$\mathsf{root} := \mathbf{b}_i$
return $\mathsf{root}$
```

Fig. 2. A description of the algorithms HashTree, BuildAuth, and RootCalc.

protocol grows, the total number of repetitions becomes multiplicative in N, e.g., [Rüc10,ES16,AEB20], and a large amount of communication is required to successfully complete the protocol. Consider the CID protocol sketched in Section 1. If rejection sampling fails, a new masking term is sampled, hence a new commitment has to be computed and sent in order to receive a new challenge c. Suppose that c does not change for certain number of masking terms and related commitments, which are sent in an aggregated form while any successfully computed response can be verified and related to the corresponding commitment. In this case repetition does not have to occur often or even not at all. We realize this concept by means of *tree of commitments*: a method by which different commitments belong to one challenge in an aggregated form and only the valid response and its related commitment will be revealed. Masking terms that are rejected or not consumed during rejection sampling remain hidden and will never be revealed.

3.1 Trees of Commitments

In this section we describe trees of commitments. We first define some relevant functions and algorithms. For a positive integer $\omega \geq 2\lambda$, we let $\mathsf{F} : \{0,1\}^* \to \{0,1\}^\omega$ be a collision resistant hash function. We define the algorithms related to binary hash trees in a way that fits to our purposes. In Figure 2 we formally give one possible description of these algorithms.

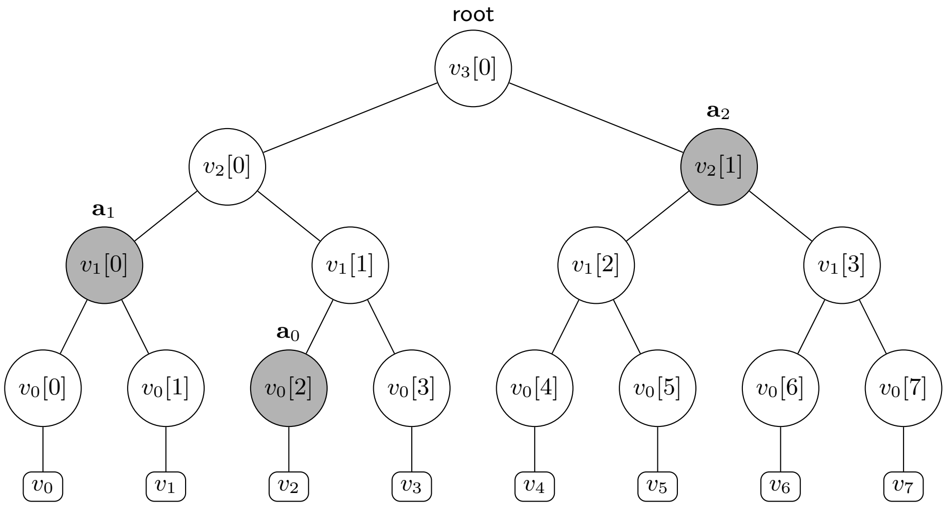

Fig. 3. A tree of commitments of height $h = 3$ and $\ell = 8$ commitments. Assume that the first time $\mathsf{RejSamp}$ accepts at step $k = 3$ ($0 \leq k < \ell$), then the gray colored nodes represent the authentication path required to compute the root starting from v_3.

$\mathsf{HashTree}$: An algorithm that computes an (unbalanced) binary hash tree of height $h \geq 1$. Its input consists of $\ell \leq 2^h$ commitments $v_0, \ldots, v_{\ell-1}$ whose hash values are the leaves of the tree, i.e., $(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{HashTree}(v_0, \ldots, v_{\ell-1})$, where root is the root of the tree and tree is a sequence of all other nodes.

$\mathsf{BuildAuth}$: An algorithm that on input an index k, a sequence of nodes tree, and a height h outputs the corresponding authentication path auth including the index k, i.e., $\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k, \mathsf{tree}, h)$.

$\mathsf{RootCalc}$: An algorithm that computes the root of a hash tree given a commitment v and its authentication path auth, i.e., $\mathsf{root} \leftarrow \mathsf{RootCalc}(v, \mathsf{auth})$.

In the following we define trees of commitments. The leaves are the hash values of commitments v_j, i.e., $v_0[j] = \mathsf{F}(v_j)$ for $0 \leq j < \ell$. The inner nodes of height i are denoted by $v_i[j]$, where $0 < i \leq h$, $0 \leq j < 2^{h-i}$. They are typically computed as $v_i[j] = \mathsf{F}(v_{i-1}[2j] \parallel v_{i-1}[2j+1])$. The root is the only node of height h, i.e., $v_h[0] = \mathsf{root}$. A formal definition follows.

Definition 6 (Tree of Commitments). *Let v_j be commitments of $\ell > 1$ secrets y_j, where $0 \leq j < \ell$. A tree of commitments is an (unbalanced) binary hash tree of height $h = \lceil \log(\ell) \rceil$, whose leaves are the hash values of v_j, i.e., $\mathsf{F}(v_j)$. The root constitutes an aggregated commitment root, and auth is the authentication path of the commitment v_k generated using the secret y_k, where $0 \leq k < \ell$.*

Next we define trees of commitments for lattice-based canonical identification (CID) schemes. Figure 3 illustrates such a tree of height $h = 3$.

Definition 7 (Tree of Commitments for CID). *Let CID be a lattice-based canonical identification scheme. Let v_j be commitments of CID generated using $\ell > 1$ masking terms y_j ($0 \leq j < \ell$). A tree of commitments for CID is an (unbalanced) binary hash tree of height $h = \lceil \log(\ell) \rceil$, whose leaves are the hash values of v_j, i.e., $\mathsf{F}(v_j)$, and its root constitutes an aggregated commitment root to ℓ masking terms for up to ℓ repetitions within CID for the same challenge c. A response is composed of (z_k, auth), where $z_k = y_k + sc$ and y_k is the first masking term for which rejection sampling succeeds (i.e., $\mathsf{RejSamp}(z_k) = 1$ for $0 \leq k < \ell$), and auth is the authentication path of the commitment v_k generated by use of the masking term y_k.*

```
$\mathcal{P}(\hat{\mathbf{A}} \in R_q^{k_1 \times k_2}, (\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2) \in D_{\mathbb{Z}^n,\sigma'}^{k_2} \times D_{\mathbb{Z}^n,\sigma'}^{k_1})$                $\mathcal{V}(\hat{\mathbf{A}}, \hat{\mathbf{b}} = \hat{\mathbf{A}}\hat{\mathbf{s}}_1 + \hat{\mathbf{s}}_2 \pmod q)$
$\hat{\mathbf{y}}_1^{(0)}, \ldots, \hat{\mathbf{y}}_1^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n,\sigma}^{k_2}$
$\hat{\mathbf{y}}_2^{(0)}, \ldots, \hat{\mathbf{y}}_2^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n,\sigma}^{k_1}$
for $(k = 0, \ldots, \ell - 1)$ do
    $\hat{\mathbf{v}}^{(k)} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{y}}_2^{(k)} \pmod q$
$(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{HashTree}(\hat{\mathbf{v}}^{(0)}, \ldots, \hat{\mathbf{v}}^{(\ell-1)})$
                                        $\xrightarrow{\mathsf{root}}$
                                                                        $\hat{c} \leftarrow_{\$} \mathbb{T}_\kappa^n$
                                        $\xleftarrow{\hat{c}}$
$k \leftarrow 0$
while $(k < \ell)$ do
    $\hat{\mathbf{z}}_1 \leftarrow \hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{s}}_1\hat{c}$
    $\hat{\mathbf{z}}_2 \leftarrow \hat{\mathbf{y}}_2^{(k)} + \hat{\mathbf{s}}_2\hat{c}$
    if $(\mathsf{RejSamp}(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) = 0)$ then
        $k \leftarrow k + 1$
if $(k \geq \ell)$ then
    restart
$\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k, \mathsf{tree}, h)$
                                        $\xrightarrow{(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \mathsf{auth})}$
                                                        $\hat{\mathbf{w}} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$
                                                        accept if $\|(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)\| \leq B$ and
                                                            $\mathsf{root} = \mathsf{RootCalc}(\hat{\mathbf{w}}, \mathsf{auth})$
```

Fig. 4. Canonical identification based on MLWE and MSIS using trees of commitments.

3.2 Canonical Identification Using Trees of Commitments

Figure 4 describes a variant of the CID protocol briefly explained in Section 1. The variant shown here is based on MLWE and MSIS and utilizes trees of commitments. Using the Fiat-Shamir transform [FS86] we obtain a digital signature scheme. In Appendix A we give a formal description of this signature scheme and prove its correctness and security. By the equivalence results of [AABN02], we deduce the soundness property of the CID protocol described in this section as well as its security against impersonation under passive attacks. More concretely, the main goal of providing the signature scheme and its security proof in Appendix A is to show how trees of commitments can also be used in lattice-based Fiat-Shamir signatures, and to establish the security of the CID protocol shown in this section based on the results of [AABN02].

We can choose ℓ such that at least one of the masking pairs $(\hat{\mathbf{y}}_1^{(k)}, \hat{\mathbf{y}}_2^{(k)})$ (see Figure 4) hides $\hat{\mathbf{S}}\hat{c}$ with probability of at least $1 - \delta_{\mathsf{abort}}$ for a given bound δ_{abort}, where $\hat{\mathbf{S}} = (\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2)$. This can be established as follows. Since the entries of the masking pairs are chosen from $D_{\mathbb{Z}^n,\sigma}$, the probability of successfully outputting $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)$ with only one masking pair is $\approx 1/M$, where M is the expected number of repetitions (see Lemma 2). Consequently, one of the ℓ masking pairs conceals the secret key with probability $1-(1-1/M)^\ell$. Hence, by choosing ℓ satisfying $(1-1/M)^\ell \leq \delta_{\mathsf{abort}}$, the protocol aborts with probability at most δ_{abort}. For instance, to obtain a probability negligible in λ we have to select ℓ such that $(1 - 1/M)^\ell \leq 2^{-\lambda}$, which allows to completely eliminate aborts.

Let us consider an illustrative example. Suppose that we set $\delta_{\mathsf{abort}} = 2^{-10}$ and use masking pairs with entries sampled from $D_{\mathbb{Z}^n,\sigma}$, where $\sigma = \alpha\|\hat{\mathbf{S}}\hat{c}\|$ and $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$ (Lemma 2). Then, by setting $\alpha = 23$ we need only $\ell = 8$ masking pairs in order to hide $\hat{\mathbf{S}}\hat{c}$ with probability at least 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 4 hash values and a pair of Gaussian vectors with $\sigma = 23 \cdot \|\hat{\mathbf{S}}\hat{c}\|$, i.e., $(\mathsf{root}, \hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \mathsf{auth} = (\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2))$. The choice of $\alpha = 23$ increases σ in this example and hence the size of $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)$ by at most 1.1 bits per integer entry in comparison to $\alpha = 11$, which is a typical choice (see, e.g., [DLL+17]) that induces a repetition rate of $M \approx 3$ and a communication complexity consisting of 3 vectors from $R_q^{k_1}$ and 3 Gaussian vectors with $\sigma = 11 \cdot \|\hat{\mathbf{S}}\hat{c}\|$. Note that in order to hide $\hat{\mathbf{S}}\hat{c}$ with probability $1 - 2^{-10}$ using a single masking pair we need to set $\alpha > 2^{13.6}$, which increases the size of the response to at least 10.1 bits per integer entry when compared with $\alpha = 11$. Hence, a larger modulus q is required and the communication complexity increases to a vector from $R_q^{k_1}$ and a Gaussian vector with a very large σ, i.e., $\sigma > 2^{13.6} \cdot \|\hat{\mathbf{S}}\hat{c}\|$.

Furthermore, we can improve the performance of protocols employing trees of commitments as follows. For subsequent executions of the protocol we can reuse the masking pairs that were sampled in previous executions but were not consumed during rejection sampling. For instance, consider the tree in Figure 3, where the first time RejSamp accepts at step $k = 3$. For the next protocol run we can simply reuse the whole subtree with root $\mathbf{a}_2 = v_2[1]$ such that we only need to compute a new subtree of height $h - 1$ and combine both subtrees to obtain a new tree of height h. This decreases the number of new masking terms to be sampled and reduces the number of hash computations and multiplications modulo q. We can also lower the security requirement of the hash function F following the standard of the hash-based signature scheme XMSS [HBG+18] and using randomized tree hashing. This allows to generate trees of commitments, where F is required to be only second preimage resistant rather than collision resistant. This reduces the size of the authentication path to one half of its original size.

3.3 Optimizing the Number of Masking Terms

The previous section shows how to reduce the overall repetition rate of lattice-based protocols with multiple rejection sampling procedures. In this section we show how to minimize the height of the tree of commitments when using Gaussian distributed masking terms. This improves the performance of interactive protocols significantly. A similar approach can be taken for masking terms sampled from other distributions such as the uniform distribution.

Lemma 3. *Let $\epsilon = 2^{-\omega(\log n)}$ and M be the repetition rate of sampling masking terms from $D_{\mathbb{Z}^n,\sigma}$ such that rejection sampling succeeds. Let δ_{abort} be the desired aborting probability. Then, the number of masking terms ℓ required to conceal a secret-related term with norm bounded by T and with probability at most $1 - \delta_{\mathsf{abort}}$ is minimized by solving the following optimization problem:*

$$\min(\ell) \text{ conditioned on } (1 - \frac{1-\epsilon}{M})^{\ell} \leq \delta_{\mathsf{abort}} \ .$$

Proof. Given a fixed δ_{abort} we can write ℓ as a function in M using the inequality given above. In particular, if $\sigma = \alpha T$ for $\alpha > 0$, then $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$, $\epsilon = 2^{-100}$, and the probability of aborting using only one masking term is given by $1 - (1 - \epsilon)/M$ (see Lemma 2). Hence, ℓ can also be considered as a function in α, i.e.,

$$\ell(\alpha) = \log(\delta_{\mathsf{abort}}) / \log\left(1 - \frac{1 - 2^{-100}}{\exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})}\right) \ .$$

Note that increasing α directly reduces ℓ. Therefore, this problem translates to finding a local minimum of the function $\ell(\alpha)$ within a given range of α, which can be solved using Lagrange optimization. □

The above lemma shows that reducing the number of masking terms ℓ for a fixed aborting probability δ_{abort} increases σ, hence the size of the responses (or signatures). In Table 2 we exhibit examples for various values of ℓ and $\sigma = \alpha T$ given δ_{abort} and $T = 500$.

Table 2. Values for the required number of Gaussian masking terms ℓ and the bit length of the standard deviation $\sigma = \alpha T = \alpha \cdot 500$ at given aborting probabilities.

Aborting probability δ_{abort}	2^{-128}	2^{-100}	2^{-80}	2^{-40}	2^{-40}	2^{-10}	2^{-10}
Number of masking terms ℓ	64	63	62	31	16	16	8
Height of the binary hash tree h	6	6	6	5	4	4	3
Parameter α	42	30	23	23	62	12	23
Bit length of the standard deviation σ	15	14	14	14	15	13	14

4 Applications

As mentioned in Section 1, there are various advanced lattice-based constructions that are based on canonical identification (CID) and thus may benefit from using trees of commitments as described in Section 3. Our approach can also be applied to interactive zero-knowledge proof systems in a straightforward way. For instance, the scheme depicted in Figure 4 can be seen as a zero-knowledge proof of knowledge of RLWE secrets with reduced communication complexity.

As a further practical application, we exploit trees of commitments within the blind signature scheme BLAZE [AEB20] resulting in major efficiency gains. The signing protocol of BLAZE consists of 4 moves between a signer $\mathcal{S}$ and a user $\mathcal{U}$. It can be aborted due to 2 rejection sampling procedures; the first one is carried out by $\mathcal{S}$ in order to hide the secret key and the second one by $\mathcal{U}$ to ensure blindness. In case the latter fails, $\mathcal{U}$ must send $\mathcal{S}$ a proof of failure in order to restart the signing protocol. This is why the last move is needed in the protocol as opposed to the standard 3-move structure of the CID scheme underlying BLAZE. Due to the possibility of failures the user must also use a statistically hiding and computationally binding commitment scheme in order to hide the message from the signer.

In the following we redesign BLAZE such that signatures can be generated in 3 moves. We call the new scheme BLAZE$^+$. In particular, we are able to completely remove the rejection sampling procedure carried out by $\mathcal{U}$. This is accomplished by generating enough masking terms at once such that blindness is achieved with overwhelming probability. This allows to safely eliminate the last move in the protocol and hence the need for proof of failures. Consequently, statistically hiding and computationally binding commitments concealing the message from $\mathcal{S}$ are not required anymore. We also describe a 4-move version of BLAZE$^+$ in Appendix B. In that version aborts at the user side occur with probability of choice. We note that a similar approach may be applied at the signer side.

In addition to the functions defined in Section 3 we need some additional tools. Let $\mathsf{H} : \{0,1\}^* \to \mathbb{T}_\kappa^n$ be a public hash function modeled as a random oracle. Further, let E be a public function that expands given strings to any desired length. Sampling from $D_{\mathbb{Z},s}^n$ using randomness ρ is denoted by $D_{\mathbb{Z},s}^n(\rho)$. We let $\hat{\mathbb{T}} = \{\pm x^i : i \in [n]\} \subset R_q$. Let Compress and Decompress be functions for (de)compressing Gaussian elements (see [AEB20] for description). Next we describe the new blind signature scheme BLAZE$^+$. The respective algorithms are formalized in Figure 5.

Key Generation.
As in BLAZE, the algorithm BS.KGen generates an instance of RSIS (see Figure 5). It's secret vector $\hat{\mathbf{s}}$ is sampled from $D_{\mathbb{Z}^n,\sigma}^{m+1}$. However, BLAZE$^+$ employs an additional condition on $\hat{\mathbf{s}}$, which can also be used in BLAZE. More concretely, the ℓ_2-norm of $\hat{\mathbf{s}}$ is bounded by $\gamma\sigma\sqrt{(m+1)n}$. This condition represents a trade-off between the speed of generating keys and the size of signatures, since the standard deviation s^* of masking terms used by the signer is a multiple of $\|\hat{\mathbf{s}}\|$. Therefore, a smaller γ decreases s^*, but reduces the success probability of passing the given condition (see Lemma 1). Note that $\hat{\mathbf{s}}$ can also be sampled from the uniform distribution over a subset of R^{m+1}, in which the coefficients of each polynomial from R are in the set $\{-d, \ldots, 0, \ldots, d\}$, where $d \in \mathbb{Z}_{>0}$.

```
$\boxed{\mathsf{BS.KGen}(1^\lambda)}$
1: $\mathsf{seed} \leftarrow_\$ \{0,1\}^\lambda$
2: $\hat{\mathbf{a}}' \in R_q^m \leftarrow \mathsf{E}(\mathsf{seed})$
3: $\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
4: $\hat{\mathbf{s}} \leftarrow D_{\mathbb{Z}^n,\sigma}^{m+1}$
5: if $(\|\hat{\mathbf{s}}\| > \gamma\sigma\sqrt{(m+1)n})$ then
6:     goto 4
7: $\hat{b} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{s}} \pmod q$
8: $\mathsf{sk} := (\mathsf{seed}, \hat{\mathbf{s}}),\ \mathsf{pk} := (\mathsf{seed}, \hat{b})$
9: return $(\mathsf{sk}, \mathsf{pk})$
```

```
$\boxed{\mathsf{BS.Verify}(\mathsf{pk}, \mu, (\hat{\mathbf{z}}, \hat{c}, \mathsf{auth}))}$
1: $\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed})$
2: $\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
3: $\hat{\mathbf{z}} \leftarrow \mathsf{Decompress}(\hat{\mathbf{z}})$
4: $\hat{w} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{z}} - \hat{b}\hat{c} \pmod q$
5: $\mathsf{root} \leftarrow \mathsf{RootCalc}(\hat{w}, \mathsf{auth})$
6: if $\big(\|\hat{\mathbf{z}}\| \le B\ \wedge\ \hat{c} = \mathsf{H}(\mathsf{root}, \mu)\big)$ then
7:     return 1
8: return 0
```

```
$\boxed{\mathsf{BS.Sign}(\mathsf{sk}, \mathsf{pk}, \mu)}$

Signer $\mathcal{S}(\mathsf{sk})$                                  User $\mathcal{U}(\mathsf{pk}, \mu)$
$\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed}),\ \hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
$\hat{\mathbf{y}}_1^*, \ldots, \hat{\mathbf{y}}_\kappa^* \leftarrow D_{\mathbb{Z}^n,s^*}^{m+1}$
for $(j = 1, \ldots, \kappa)$ do
    $\hat{y}_j \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{y}}_j^* \pmod q$
        $\xrightarrow{\hat{\mathbf{y}} := (\hat{y}_1, \ldots, \hat{y}_\kappa)}$
                                                   $\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed}),\ \hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}'],\ \hat{p}_1, \ldots, \hat{p}_\kappa \leftarrow_\$ \hat{\mathbb{T}}$
                                                   $\rho \leftarrow_\$ \{0,1\}^\lambda,\ \hat{\mathbf{e}}^{(0)}, \ldots, \hat{\mathbf{e}}^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n,s}^{m+1}(\rho)$
                                                   $\hat{y} \leftarrow \sum_1^\kappa \hat{p}_j \hat{y}_j \pmod q$
                                                   for $(k = 0, \ldots, \ell-1)$ do
                                                       $\hat{t}^{(k)} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{e}}^{(k)} + \hat{y} \pmod q$
                                                   $(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{HashTree}(\hat{t}^{(0)}, \ldots, \hat{t}^{(\ell-1)})$
                                                   $\hat{c} \leftarrow \mathsf{H}(\mathsf{root}, \mu),\ \hat{c} := \sum_1^\kappa \hat{c}_j,\ \ \hat{c}_j \in \hat{\mathbb{T}}$
                                                   for $(j = 1, \ldots, \kappa)$ do
                                                       $\hat{c}_j^* \leftarrow \hat{p}_j^{-1} \cdot \hat{c}_j$
                                                           $\xleftarrow{\hat{\mathbf{c}}^* := (\hat{c}_1^*, \ldots, \hat{c}_\kappa^*)}$
for $(j = 1, \ldots, \kappa)$ do
    $\hat{\mathbf{z}}_j^* \leftarrow \hat{\mathbf{y}}_j^* + \hat{\mathbf{s}}\hat{c}_j^*$
if $(\mathsf{RejSamp}(\hat{\mathbf{z}}_1^*, \ldots, \hat{\mathbf{z}}_\kappa^*) = 0)$ then
    restart
        $\xrightarrow{\hat{\mathbf{z}}^* := (\hat{\mathbf{z}}_1^*, \ldots, \hat{\mathbf{z}}_\kappa^*)}$
                                                   $\hat{\mathbf{v}} \leftarrow \sum_1^\kappa \hat{p}_j \hat{\mathbf{z}}_j^*$
                                                   if $(\|\hat{\mathbf{v}}\| > \eta s^* \sqrt{(m+1)\kappa n})$ then
                                                       abort (occurs with probability $2^{-\lambda}$)
                                                   $k \leftarrow 0$
                                                   while $(k < \ell)$ do
                                                       $\hat{\mathbf{z}} \leftarrow \hat{\mathbf{e}}^{(k)} + \hat{\mathbf{v}}$
                                                       if $(\mathsf{RejSamp}(\hat{\mathbf{z}}) = 0)$ then
                                                           $k \leftarrow k + 1$
                                                   if $(k \ge \ell)$ then
                                                       abort (occurs with probability $2^{-\lambda}$)
                                                   $\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k, \mathsf{tree}, h)$
                                                   $\hat{\mathbf{z}} \leftarrow \mathsf{Compress}(\hat{\mathbf{z}})$
                                                   return $(\mu, (\hat{\mathbf{z}}, \hat{c}, \mathsf{auth}))$
```

Fig. 5. A formal description of the new blind signature scheme BLAZE^+.

Signing.
The signing algorithm is similar to that of BLAZE [AEB20]. The difference is that in BLAZE^+ the user $\mathcal{U}$ generates $\ell > 1$ masking vectors $\hat{\mathbf{e}}^{(0)}, \ldots, \hat{\mathbf{e}}^{(\ell-1)}$ chosen from $D_{\mathbb{Z}^n,s}^{m+1}$. These vectors are then

used to compute $\hat{t}^{(k)} = \hat{\mathbf{a}} \cdot \hat{\mathbf{e}}^{(k)} + \hat{y} \pmod q$, which are needed to generate a tree of commitments of height $h = \lceil \log(\ell) \rceil$ via the algorithm $\mathsf{HashTree}$. We note that generating $\hat{\mathbf{e}}^{(k)}$ and $\hat{\mathbf{a}} \cdot \hat{\mathbf{e}}^{(k)} \pmod q$, for $k = 0, \ldots, \ell - 1$, can be precomputed by $\mathcal{U}$ before starting the protocol with $\mathcal{S}$. The sum $\hat{t}^{(k)}$ containing $\hat{y}$ and the construction of the tree cannot be carried out in advance, since $\hat{y}$ is computed from the commitment sent by $\mathcal{S}$ (see Figure 5). We also note that $\hat{\mathbf{e}}^{(k)}$ can be reused when $\mathcal{S}$ restarts the protocol, since $\hat{\mathbf{e}}^{(k)}$ are not revealed and $\hat{\mathbf{c}}^*$ is always fresh. After receiving the vector $\hat{\mathbf{z}}^*$, $\mathcal{U}$ computes $\hat{\mathbf{z}}$ and the authentication path auth associated to the first index $k < \ell$ for which the vector $\hat{\mathbf{e}}^{(k)}$ ensures blindness. Note that ℓ is chosen such that this happens with probability at least $1 - 2^{-\lambda}$, i.e., $\mathcal{U}$ outputs a valid signature with overwhelming probability. Also note that for each signature a new root is generated.

Verification.
Verifying a signature is straightforward as described in Figure 5.

In the following we give the main security statements of this section comprising completeness, blindness, and strong one-more unforgeability of BLAZE^+. The proofs of both correctness and blindness directly follow from [AEB20] and are hence omitted. In particular, proving blindness requires to show that the exchanged messages during protocol execution together with the user's output does not leak any information about the message being signed. In comparison to BLAZE, the authentication path auth, which is a part of the signature in BLAZE^+ is the only additional information exchanged between the signer and the user. Obviously, auth does not give any information about the signed message.

Theorem 1 (Completeness). *Let $\alpha^*, \alpha, \gamma, \eta > 0$, $s^* = \alpha^* \gamma \sigma \sqrt{(m+1)\kappa n}$, $s = \eta \alpha s^* \sqrt{(m+1)\kappa n}$, and $B = \eta s \sqrt{(m+1)n}$. Further, let $(1 - \frac{1-2^{-100}}{U})^\ell \leq 2^{-\lambda}$, where $U = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$. After at most M repetitions, any blind signature produced by BLAZE^+ is validated with probability of at least $1 - 2^{-\lambda}$, where $M = \exp(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}})$.*

Theorem 2 (Blindness). *The scheme BLAZE^+ is statistically blind. The statistical distance between two executions of its signing protocol is given by $2^{-100}/M$.*

Next, we prove the strong one-more unforgeability of BLAZE^+. In the proof we assume that in the unforgeability game (see Definition 3) any forgery output by the adversary $\mathcal{A}$ is considered valid if and only if its associated root was not queried to the signing oracle by $\mathcal{A}$, i.e., a forgery must contain a new root that is distinct from the roots queried to the signing oracle. We note that our 3-move protocol achieves completeness with overwhelming probability, i.e., each completed interaction yields a valid blind signature, where users do not have to request a protocol restart. Therefore, the one-more unforgeability proof does not need to consider aborts at the user side as opposed to BLAZE.

Theorem 3 (Unforgeablility). *The scheme BLAZE^+ is strongly one-more unforgeable in the random oracle model (ROM) if F is a collision resistant hash function and RSIS is hard. More precisely, suppose that F is collision resistant and it is hard to find a vector $\hat{\mathbf{x}} \neq \mathbf{0}$ satisfying $[1 \ \hat{\mathbf{a}}'] \cdot \hat{\mathbf{x}} = 0 \pmod q$ and $\|\hat{\mathbf{x}}\| \leq \beta$ for $\beta = 2(B + \eta\sigma\sqrt{(m+1)\kappa n})$, then BLAZE^+ is strongly one-more unforgeable in the ROM.*

Proof. We assume that there exists a forger $\mathcal{A}$ that wins the one-more unforgeability game given in Definition 3 with probability $\varepsilon_{\mathcal{A}}$. We construct a reduction algorithm $\mathcal{D}$ that finds collisions in the hash function F or computes a vector $\hat{\mathbf{x}} \neq \mathbf{0}$ as described in the theorem statement with probability $\varepsilon_{\mathcal{D}}$ as given below.

Setup. The input of $\mathcal{D}$ is a uniformly random vector $\hat{\mathbf{a}}' \in R_q^m$ and a hash function F. It also has access to an oracle $\mathcal{O}_{\mathsf{F}}$ for F. The reduction $\mathcal{D}$ samples a vector $\hat{\mathbf{s}}$ from $D_{\mathbb{Z}^n,\sigma}^{m+1}$ and randomly selects answers for random oracle queries $\{\hat{h}_1, \ldots, \hat{h}_{q_{\mathsf{H}}}\}$. Then, it runs the forger $\mathcal{A}$ with public key $(\hat{\mathbf{a}}, \hat{b})$, where $\hat{\mathbf{a}} = [1\ \hat{\mathbf{a}}']$ and $\hat{b} = \hat{\mathbf{a}} \cdot \hat{\mathbf{s}} \pmod q$.

Random Oracle Query. The reduction $\mathcal{D}$ maintains a list L_{H}, which includes pairs of random oracle queries and their answers. If H was previously queried on some input, then $\mathcal{D}$ looks up its entry in L_{H} and returns its answer $\hat{c} \in \mathbb{T}^n_\kappa$. Otherwise, it returns the first unused $\hat{c}$ and updates the list.

Hash Query. Hash queries to F sent by $\mathcal{A}$ are forwarded to the oracle $\mathcal{O}_{\mathsf{F}}$. The reduction $\mathcal{D}$ also maintains a list L_{F}, which includes pairs of hash queries to F and their answers as well as the structure of the trees.

Blind Signature Query. Upon receiving signature queries from the forger $\mathcal{A}$ as a user, $\mathcal{D}$ interacts as a signer with $\mathcal{A}$ according to the signing protocol (see Figure 5).

Output. After $k \le q_{\mathsf{Sign}}$ successful executions of the signing protocol, $\mathcal{A}$ outputs $k+1$ distinct messages and their valid signatures $(\mu_1, \mathsf{sig}_1), \ldots, (\mu_{k+1}, \mathsf{sig}_{k+1})$. Then, one of the following two cases applies.

Case 1. $\mathcal{D}$ finds two signatures of messages $\mu, \mu' \in \{\mu_1, \ldots, \mu_{k+1}\}$ with the same random oracle answer $\hat{c}$. In this case the verification algorithm yields $\mathsf{H}(\mathsf{root}, \mu) = \mathsf{H}(\mathsf{root}', \mu')$. If $\mu \neq \mu'$ or $\mathsf{root} \neq \mathsf{root}'$, then a second preimage of $\hat{c}$ has been found by $\mathcal{A}$. If $\mu = \mu'$ and $\mathsf{root} = \mathsf{root}'$, then both signatures were generated using the same hash tree. This does not follow the unforgeability game because the output of $\mathcal{A}$ does not include a valid forgery.

Case 2. If all signatures output by $\mathcal{A}$ have distinct random oracle answers, then $\mathcal{D}$ guesses an index $i \in [k+1]$ such that $\hat{c}_i = \hat{h}_j$ for some $j \in [q_{\mathsf{H}}]$. Then, it records the pair $(\mu_i, (\hat{\mathbf{z}}_i, \hat{c}_i, \mathsf{auth}_i))$ and invokes $\mathcal{A}$ again with the same random tape and the random oracle queries $\{\hat{h}_1, \ldots, \hat{h}_{j-1}, \hat{h}'_j, \ldots, \hat{h}'_{q_{\mathsf{H}}}\}$, where $\{\hat{h}'_j, \ldots, \hat{h}'_{q_{\mathsf{H}}}\}$ are fresh random elements. After the second invocation, the output of $\mathcal{A}$ includes a pair $(\mu'_i, (\hat{\mathbf{z}}'_i, \hat{c}'_i, \mathsf{auth}'_i))$. By the General Forking Lemma [BN06], $\mathcal{A}$ outputs a forgery containing $\hat{c}'_i$ with probability $\varepsilon_{\mathsf{fork}}$ (see below), where $\hat{c}_i \neq \hat{c}'_i$ and $\mathsf{root} = \mathsf{root}'$. Let $\hat{w} = \hat{\mathbf{a}} \cdot \hat{\mathbf{z}}_i - \hat{b}\hat{c}_i \pmod q$ and $\hat{w}' = \hat{\mathbf{a}} \cdot \hat{\mathbf{z}}'_i - \hat{b}\hat{c}'_i \pmod q$. Then, one of the following holds:

1. $\hat{\mathbf{z}}_i \neq \hat{\mathbf{z}}'_i$ and $\mathsf{auth}_i = \mathsf{auth}'_i$. If $\hat{w} = \hat{w}'$, then $\hat{\mathbf{a}}(\hat{\mathbf{z}}_i - \hat{\mathbf{z}}'_i) - \hat{b}(\hat{c}_i - \hat{c}'_i) = 0 \pmod q$. Therefore, by setting $\hat{b} = \hat{\mathbf{a}} \cdot \hat{\mathbf{s}} \pmod q$ we obtain $\hat{\mathbf{a}} \cdot \hat{\mathbf{x}} = 0 \pmod q$, where $\hat{\mathbf{x}} = \hat{\mathbf{z}}_i - \hat{\mathbf{z}}'_i - \hat{\mathbf{s}}(\hat{c}_i - \hat{c}'_i)$. Since both signatures are valid, we have $\|\hat{\mathbf{z}}_i\| \le B$ and $\|\hat{\mathbf{z}}'_i\| \le B$. Moreover we have $\|\hat{\mathbf{s}}(\hat{c}_i - \hat{c}'_i)\| \le 2\eta\sigma\sqrt{(m+1)\kappa n}$. Hence, $\|\hat{\mathbf{x}}\| \le 2(B + \eta\sigma\sqrt{(m+1)\kappa n})$. This constitutes a solution to RSIS with norm bound β. If $\hat{w} \neq \hat{w}'$, then a collision in F has been found in the leaves of the hash tree.
2. $\hat{\mathbf{z}}_i \neq \hat{\mathbf{z}}'_i$ and $\mathsf{auth}_i \neq \mathsf{auth}'_i$. If $\hat{w} = \hat{w}'$, then we have a solution to RSIS (as in **1.**). If $\hat{w} \neq \hat{w}'$, then we consider two cases: If $\hat{w}, \hat{w}'$ belong to different hash trees, then a collision has occurred in F similar to [Mer89], i.e., there exists an index $j \in \{0, \ldots, h-1\}$ such that $\mathbf{a}_j \neq \mathbf{a}'_j$, where $\mathbf{a}_j \in \mathsf{auth}_i$, $\mathbf{a}'_j \in \mathsf{auth}'_i$ and $\mathsf{RootCalc}(\hat{w}, \mathsf{auth}_i) = \mathsf{root} = \mathsf{RootCalc}(\hat{w}', \mathsf{auth}'_i)$. If $\hat{w}, \hat{w}'$ belong to the same hash tree, then $\mathcal{D}$ keeps the pair $(\mu_i, (\hat{\mathbf{z}}_i, \hat{c}_i, \mathsf{auth}_i))$ and invokes $\mathcal{A}$ at most ℓ times with the same random tape and the random oracle queries $\{\hat{h}_1, \ldots, \hat{h}_{j-1}, \hat{h}^{(t)}_j, \ldots, \hat{h}^{(t)}_{q_{\mathsf{H}}}\}$ (where $t \in \{0, \ldots, \ell-1\}$) until we obtain two forgeries such that the associated leaves have the same index in the tree and the same hash value. If $\hat{w} = \hat{w}'$, then we have a solution to RSIS (as in **1.**). Otherwise, we have a collision in F.
3. $\hat{\mathbf{z}}_i = \hat{\mathbf{z}}'_i$ and $\mathsf{auth}_i = \mathsf{auth}'_i$. If $\hat{w} = \hat{w}'$, then $\hat{\mathbf{a}} \cdot \hat{\mathbf{s}}(\hat{c}_i - \hat{c}'_i) = \hat{b}(\hat{c}_i - \hat{c}'_i) = 0 \pmod q$ and $\|\hat{\mathbf{s}}(\hat{c}_i - \hat{c}'_i)\| \le \beta$. Since $\hat{c}_i \neq \hat{c}'_i$ and R is an integral domain, then $\hat{\mathbf{s}}(\hat{c}_i - \hat{c}'_i) \in R^{m+1} \backslash \{\mathbf{0}\}$. This constitutes a solution to RSIS. Note that if $\hat{b}$ is invertible in R_q, then we obtain $\hat{c}_i - \hat{c}'_i = 0 \pmod q$. This contradicts $\hat{c}_i \neq \hat{c}'_i$. Moreover, if $(\hat{c}_i - \hat{c}'_i)$ is invertible in R_q, then $\hat{b} = 0 \pmod q$, which is not the case. If $\hat{w} \neq \hat{w}'$, then a collision has occurred in F (in the leaves of the hash tree).
4. $\hat{\mathbf{z}}_i = \hat{\mathbf{z}}'_i$ and $\mathsf{auth}_i \neq \mathsf{auth}'_i$. If $\hat{w} = \hat{w}'$, then we have a solution to RSIS (as in **3.**). If $\hat{w} \neq \hat{w}'$, then one of the cases considered in **2.** (for $\hat{w} \neq \hat{w}'$) applies.

Table 3. Parameters for BLAZE$^+$ and BLAZE targeting approximately 128 bits of security. The performance, sizes, and communication complexity corresponding to these parameters are given in Table 1.

Scheme	**Parameters**														
	δ_{abort}	ℓ	h	n	m	q	σ	γ	κ	α^*	α	s^*	s	$M_{\mathcal{S}}$	$M_{\mathcal{U}}$	M
BLAZE$^+$	2^{-128}	71	7	1024	1	$\approx 2^{31}$	0.5	1.01	16	19	33	1736.9	12450734	1.9	1	1.9
BLAZE$^+$	2^{-40}	32	5	1024	1	$\approx 2^{31}$	0.5	1.01	16	28	22	2559.6	12232099	1.5	1	1.5
BLAZE$^+$	2^{-10}	8	3	1024	1	$\approx 2^{31}$	0.5	1.01	16	28	22	2559.6	12232099	1.5	1	1.5
BLAZE	0.38	1	0	1024	1	$\approx 2^{31}$	0.5	1.2	16	20	25	2172.2	11796306	1.8	1.6	2.9
BLAZE$^+$	2^{-128}	71	7	1024	3	$\approx 2^{31}$	9.6	1.01	16	19	33	47161.3	478102394	1.9	1	1.9
BLAZE$^+$	2^{-40}	32	5	1024	3	$\approx 2^{31}$	9.6	1.01	16	28	22	69500.9	469714882	1.5	1	1.5
BLAZE$^+$	2^{-10}	8	3	1024	3	$\approx 2^{31}$	9.6	1.01	16	28	22	69500.9	469714882	1.5	1	1.5
BLAZE	0.38	1	0	1024	3	$\approx 2^{31}$	9.6	1.2	16	20	25	54067.2	380633088	1.8	1.6	2.9

The reduction $\mathcal{D}$ retries at most $q_{\mathsf{H}}^{(k+1)}$ times with different random tape and random oracle queries.

Analysis. First, we note that the environment of $\mathcal{A}$ is perfectly simulated by $\mathcal{D}$ and signatures are generated with the same probability as in the real execution of the signing protocol. Next, one of the $k+1$ pairs output by $\mathcal{A}$ is by assumption not generated during the execution of the signing protocol. The probability of correctly guessing the index i corresponding to this pair is $1/(k+1)$, where there are q_{H}^{k+1} index pairs (i, j) such that $\hat{c}_i = \hat{h}_j$. Therefore, one of the q_{H}^{k+1} reruns of $\mathcal{A}$ yields the correct index pair (i, j). The probability that $\hat{c}_i$ was a random oracle query made by $\mathcal{A}$ is at least $1 - 1/|\mathbb{T}_\kappa^n|$. Thus, the probability that $\hat{c}_i = \hat{h}_j$ is at least $\varepsilon_{\mathcal{A}} - 1/|\mathbb{T}_\kappa^n|$. By the General Forking Lemma with at most $\ell = O(1)$ rewindings and distinct $\hat{h}_j^{(t)}, \ldots, \hat{h}_{q_{\mathsf{H}}}^{(t)}$, we have

$$\varepsilon_{\mathsf{fork}} \geq \left(\varepsilon_{\mathcal{A}} - \frac{1}{|\mathbb{T}_\kappa^n|}\right) \cdot \left(\left(\frac{\varepsilon_{\mathcal{A}} - 1/|\mathbb{T}_\kappa^n|}{q_{\mathsf{Sign}} + q_{\mathsf{H}}}\right)^{\ell} - \tau\right), \text{ where } \tau = 1 - \prod_{t=1}^{\ell} \frac{|\mathbb{T}_\kappa^n| - t}{|\mathbb{T}_\kappa^n|} \leq 1 - \left(\frac{|\mathbb{T}_\kappa^n| - \ell}{|\mathbb{T}_\kappa^n|}\right)^{\ell} \leq \frac{\ell^2}{|\mathbb{T}_\kappa^n|} \,.$$

Since $\hat{\mathbf{s}}$ is not uniquely defining $\hat{b}$ when $(m+1)\log(2d) > \log(q)$ (see, e.g., [MR07]) for a sufficiently large d that is related to the size of the coefficients of $\hat{\mathbf{s}}$, $\mathcal{A}$ does not know which $\hat{\mathbf{s}}$ is being used to construct $\hat{\mathbf{x}}$. Hence, $\hat{\mathbf{x}} \neq \mathbf{0}$ with probability at least $1/2$ (see, e.g., [Rüc10, LPR13]). This can be easily shown, e.g., when the coefficients of $\hat{\mathbf{s}}$ are uniformly distributed over $\{-d, \ldots, 0, \ldots, d\}$. The success probability of $\mathcal{D}$ is given by $\varepsilon_{\mathcal{D}} \geq \dfrac{\varepsilon_{\mathsf{fork}}}{2(k+1)}$, which is non-negligible if $\varepsilon_{\mathcal{A}}$ is non-negligible. □

Parameters. Table 3 shows our proposed parameters for BLAZE$^+$, which are selected for approximately 128 bits of security. The table also reviews the parameters of BLAZE proposed in [AEB20] for the same security level. Table 1 gives the related communication complexity, performance, and sizes of keys and signatures. For the sake of comparison with BLAZE, we choose $m = 1$ and $m = 3$ for a practical scheme instantiation. Note that the choice of e.g., $m = 1$ implies an instantiation that is based on RLWE rather than RSIS and hence, it is not covered by the security proof as indicated in [AEB20]. This is because the secret key has insufficient entropy or does not satisfy the condition $(m+1)\log(2d) > \log(q)$ (see the proof of Therorem 3). It seems that using RLWE does not reduce the security of the scheme, but rather using RSIS appears to be more an artifact of the proof technique.

Acknowledgements

We thank the anonymous reviewers of ACISP'20 for their valuable comments. This work has been partially supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297.

References

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 *Advances in Cryptology - EUROCRYPT 2002*, pages 418–433. Springer, 2002. 6, 10, 18

AEB20. Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. BLAZE: Practical lattice-based blind signatures for privacy-preserving applications. In *Financial Cryptography and Data Security - FC 2020*. Springer, 2020. Full version: http://eprint.iacr.org/2019/1167. 1, 2, 3, 6, 8, 12, 13, 14, 16, 24

Ajt96. Miklós Ajtai. Generating hard instances of lattice problems. In *ACM symposium on Theory of computing - STOC 1996*, pages 99–108. ACM, 1996. 7

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 *Advances in Cryptology–ASIACRYPT 2014*, pages 551–572. Springer, 2014. 1

BDL$^+$18. Carsten Baum, Ivan Damgård, Vadim Lyubashevsky, Sabine Oechsner, and Chris Peikert. More efficient commitments from structured lattice assumptions. In *Security and Cryptography for Networks - SCN 2018*, pages 368–385. Springer, 2018. 1

BG14. Shi Bai and Steven D Galbraith. An improved compression technique for signatures based on learning with errors. In *Cryptographers' Track at the RSA Conference*, pages 28–47. Springer, 2014. 1, 3

BLO18. Carsten Baum, Huang Lin, and Sabine Oechsner. Towards practical lattice-based one-time linkable ring signatures. In *Information and Communications Security - ICICS 2018*, pages 303–322. Springer, 2018. 1

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM conference on Computer and communications security - CCS 2006*, pages 390–399. ACM, 2006. 15, 22

BP18. Zvika Brakerski and Renen Perlman. Order-LWE and the hardness of Ring-LWE with entropic secrets. Cryptology ePrint Archive, Report 2018/494, 2018. https://eprint.iacr.org/2018/494. 4

BPMW16. Florian Bourse, Rafaël Del Pino, Michele Minelli, and Hoeteck Wee. FHE circuit privacy almost for free. In *Advances in Cryptology - CRYPTO 2016*, pages 62–89. Springer, 2016. 4

DDLL13. Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal Gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013. 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. *Transactions on Cryptographic Hardware and Embedded Systems - TCHES 2018*, (1):238–268, 2018. 1, 3

DLL$^+$17. Leo Ducas, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS–Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. Version: 20170627:201152, http://eprint.iacr.org/2017/633. 11

dPL17. Rafaël del Pino and Vadim Lyubashevsky. Amortization with fewer equations for proving knowledge of small secrets. In *Advances in Cryptology - CRYPTO 2017*, pages 365–394. Springer, 2017. 4

ES16. Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In *Cryptology and Network Security, CANS 2016*, pages 140–155, 2016. 1, 2, 8

FS86. Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology–CRYPTO 86*, pages 186–194. Springer, 1986. 10, 18

FS10. Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In *Advances in Cryptology–EUROCRYPT 2010*, pages 197–215. Springer, 2010. 3

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing - STOC 2009*, pages 169–178. ACM, 2009. 4

HBG$^+$18. Andreas Hülsing, Denis Butin, Stefan Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, May 2018. 4, 11

JLO97. Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures. In *Advances in Cryptology - CRYPTO 1997*, pages 150–164. Springer, 1997. 6

KKW18. Jonathan Katz, Vladimir Kolesnikov, and Xiao Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *ACM Conference on Computer and Communications Security - CCS 2018*, pages 525–537. ACM, 2018. 4

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010*, pages 1–23. Springer, 2010. 7

LPR13. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for Ring-LWE cryptography. In *Advances in Cryptology–EUROCRYPT 2013*, pages 35–54. Springer, 2013. 16

LS15. Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs Codes Cryptography*, 75(3):565–599, 2015. 7

Lyu09. Vadim Lyubashevsky. Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In *Advances in Cryptology–ASIACRYPT 2009*, pages 598–616. Springer, 2009. 1, 3

Lyu12. Vadim Lyubashevsky. Lattice signatures without trapdoors. In *Advances in Cryptology–EUROCRYPT 2012*, pages 738–755. Springer, 2012. 1, 3, 7

Mer89. Ralph C. Merkle. A certified digital signature. In *Advances in Cryptology - CRYPTO '89*, pages 218–238. Springer, 1989. 15, 22

Mic02. Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *Proceedings of the 43rd Symposium on Foundations of Computer Science FOCS*, pages 356–365. IEEE, 2002. 7

MR07. Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM Journal on Computing*, 37(1):267–302, 2007. 16

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000. 6

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *ACM symposium on Theory of computing*, pages 84–93. ACM, 2005. 7

Rüc10. Markus Rückert. Lattice-based blind signatures. In *Advances in Cryptology–ASIACRYPT 2010*, pages 413–430. Springer, 2010. 1, 2, 8, 16

Sch91. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991. 1

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 *Information Security and Privacy - ACISP 2018, Proceedings*, pages 558–576. Springer, 2018. 1

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. 1

A Lattice-Based Signatures Using Trees of Commitments

In this section we present an ordinary lattice-based signature scheme. It is constructed by applying the Fiat-Shamir transform [FS86] to the CID scheme described in Section 3.2 (Figure 4). Its signing algorithm generates a tree of commitments with enough masking terms such that it outputs valid signatures with a probability of choice. More precisely, with a desired aborting probability signatures do not leak information about the secret key using at least one of the generated masking terms. The goal of the scheme introduced below is to show how trees of commitments can also be utilized in lattice-based ordinary signatures with a proof of security. In particular, we omit exploiting other standard tools and techniques from prior works on lattice-based Fiat-Shamir signatures towards a practical construction. On the other hand, the security proof of the signature scheme we provide in this section allows us to establish the security of the CID protocol introduced in Section 3.2 by using the equivalence results of [AABN02]. We first define signature schemes and their security.

Definition 8 (Signature Scheme). *Let λ be a security parameter. A signature scheme Sig with key space $\mathcal{K}$, message space $\mathcal{M}$, and signature space $\mathcal{S}$ is a tuple of polynomial-time algorithms $(\mathsf{KGen}, \mathsf{Sign}, \mathsf{Verify})$ such that*

- $\mathsf{KGen}(1^\lambda)$ *is a key generation algorithm that outputs a key pair* $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{K}$*, where* pk *is a public (verification) key and* sk *is a secret (signing) key.*

```
Game $\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda)$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$
$\mathsf{H} \leftarrow \mathcal{H}(1^\lambda)$
$\mathcal{Q} := \varnothing$
$(\mu^*, s^*) \leftarrow \mathcal{A}^{\mathsf{H}(\cdot), \mathcal{O}(\mathsf{sk}, \cdot)}(\mathsf{pk})$
if $\big(\mu^* \notin \mathcal{Q} \;\wedge\; \mathsf{Verify}(\mathsf{pk}, \mu^*, s^*) = 1\big)$ then
    return 1
return 0
```

```
$\mathcal{O}(\mathsf{sk}, \mu)$
$\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\mu\}$
$s \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu)$
return $s$
```

Fig. 6. The security game EUF-CMA of signature schemes.

- *$\mathsf{Sign}(\mathsf{sk}, \mu)$ is a signing algorithm that takes as input a secret key sk and a message $\mu \in \mathcal{M}$. It outputs a signature $s \in \mathcal{S}$.*
- *$\mathsf{Verify}(\mathsf{pk}, \mu, s)$ is a verification algorithm that takes as input a public key pk, a message μ with its signature s. It outputs 1 if s is valid and 0 otherwise.*

A signature scheme requires that Verify always (or with overwhelming probability) validates correctly signed messages, i.e., for all $\lambda \in \mathbb{N}$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(1^\lambda)$, $\mu \in \mathcal{M}$, and all $s \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu)$, it holds $\Pr[\mathsf{Verify}(\mathsf{pk}, \mu, s) = 1] \geq 1 - \operatorname{negl}(\lambda)$. Security of signature schemes is captured by the security notion *existential unforgeability under adaptive chosen-message attacks* (EUF-CMA).

Definition 9 (EUF-CMA Security). *Let $\mathcal{H}$ be a family of random oracles. A signature scheme Sig is called $(t, q_{\mathsf{Sign}}, q_H, \varepsilon)$-EUF-CMA in the random oracle model if for any adversary $\mathcal{A}$ running in time at most t and making at most q_{Sign} signature queries and at most q_H random oracle queries to $H \leftarrow \mathcal{H}(1^\lambda)$, the game $\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda)$ depicted in Figure 6 outputs 1 with probability at most ε, i.e., $\Pr[\mathsf{EUF\text{-}CMA}_{\mathcal{A}}(\lambda) = 1] \leq \varepsilon$. The scheme is strongly $(t, q_{\mathsf{Sign}}, q_H, \varepsilon)$-EUF-CMA if the condition $\mu^* \notin \mathcal{Q}$ changes to $(\mu^*, s^*) \notin \{(\mu_1, s_1), \ldots, (\mu_q, s_q)\}$, where $\mathcal{Q} = \{\mu_1, \ldots, \mu_q\}$ and $q \leq q_{\mathsf{Sign}}$.*

Next, we describe the new signature scheme. The relevant functions and algorithms are already defined in Section 3 and 4. The respective algorithms are formalized in Figure 7.

Key Generation.

On input the security parameter 1^λ, the algorithm samples a k_2-dimensional vector $\hat{\mathbf{s}}_1$ with entries distributed according to $D_{\mathbb{Z}^n, \sigma'}$ and a k_1-dimensional vector $\hat{\mathbf{s}}_2$ from $D_{\mathbb{Z}^n, \sigma'}$. It also selects a uniform random matrix $\hat{\mathbf{A}}$ from $R_q^{k_1 \times k_2}$. The secret key sk consists of the pair $(\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2)$, while the public key pk is given by $(\hat{\mathbf{A}}, \hat{\mathbf{b}} = \hat{\mathbf{A}}\hat{\mathbf{s}}_1 + \hat{\mathbf{s}}_2 \pmod q)$.

Signing.

Given the secret key sk, the matrix $\hat{\mathbf{A}}$, and a message μ, the algorithm starts by sampling ℓ pairs of masking vectors $(\hat{\mathbf{y}}_1^{(k)}, \hat{\mathbf{y}}_2^{(k)})$ from $D_{\mathbb{Z}^n, \sigma}^{k_2} \times D_{\mathbb{Z}^n, \sigma}^{k_1}$, where $k = 0, \ldots, \ell - 1$. Afterwards, the vectors $\hat{\mathbf{v}}^{(k)} = \hat{\mathbf{A}}\hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{y}}_2^{(k)} \pmod q$ are computed and used to generate a tree of commitments of height $h = \lceil \log(\ell) \rceil$, i.e., $(\mathsf{root}, \mathsf{tree}) = \mathsf{HashTree}(\hat{\mathbf{v}}^{(0)}, \ldots, \hat{\mathbf{v}}^{(\ell-1)})$. The function H is then called on input (root, μ) to compute a polynomial $\hat{c}$. After that, the vectors $\hat{\mathbf{z}}_1 = \hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{s}}_1\hat{c}$, $\hat{\mathbf{z}}_2 = \hat{\mathbf{y}}_2^{(k)} + \hat{\mathbf{s}}_2\hat{c}$ are computed and RejSamp is applied on $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)$ starting from $k = 0$ until it outputs 1 for some $k < \ell$. The authentication path of the vector $\hat{\mathbf{v}}^{(k)}$ is then built, i.e., $\mathsf{auth} = \mathsf{BuildAuth}(k, \mathsf{tree}, h)$, and the algorithm outputs the signature $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth})$. The signing algorithm restarts if RejSamp outputs 0 for all $k = 0, \ldots, \ell - 1$.

Verification.

On input $(\mathsf{pk}, \mu, (\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth}))$, the algorithm computes the vector $\hat{\mathbf{w}} = \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$

```
KGen(1^λ)
1: $\hat{\mathbf{s}}_1 \leftarrow D_{\mathbb{Z}^n,\sigma'}^{k_2}$
2: $\hat{\mathbf{s}}_2 \leftarrow D_{\mathbb{Z}^n,\sigma'}^{k_1}$
3: $\hat{\mathbf{A}} \leftarrow_\$ R_q^{k_1 \times k_2}$
4: $\hat{\mathbf{b}} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{s}}_1 + \hat{\mathbf{s}}_2 \pmod q$
5: $\mathsf{sk} := (\hat{\mathbf{s}}_1, \hat{\mathbf{s}}_2)$
6: $\mathsf{pk} := (\hat{\mathbf{A}}, \hat{\mathbf{b}})$
7: return $(\mathsf{sk}, \mathsf{pk})$
```

```
Verify(pk, μ, (ẑ1, ẑ2, ĉ, auth))
1: $\hat{\mathbf{w}} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$
2: $\mathsf{root} \leftarrow \mathsf{RootCalc}(\hat{\mathbf{w}}, \mathsf{auth})$
3: if $\left(\|(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)\| \le B \ \wedge\ \mathsf{H}(\mathsf{root}, \mu) = \hat{c}\right)$ then
4:     return 1
5: return 0
```

```
Sign(sk, Â, μ)
1: $\hat{\mathbf{y}}_1^{(0)}, \ldots, \hat{\mathbf{y}}_1^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n,\sigma}^{k_2}$
2: $\hat{\mathbf{y}}_2^{(0)}, \ldots, \hat{\mathbf{y}}_2^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n,\sigma}^{k_1}$
3: for $(k = 0, \ldots, \ell - 1)$ do
4:     $\hat{\mathbf{v}}^{(k)} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{y}}_2^{(k)} \pmod q$
5: $(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{HashTree}(\hat{\mathbf{v}}^{(0)}, \ldots, \hat{\mathbf{v}}^{(\ell-1)})$
6: $\hat{c} \leftarrow \mathsf{H}(\mathsf{root}, \mu)$
7: $k \leftarrow 0$
8: while $(k < \ell)$ do
9:     $\hat{\mathbf{z}}_1 \leftarrow \hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{s}}_1\hat{c}$
10:    $\hat{\mathbf{z}}_2 \leftarrow \hat{\mathbf{y}}_2^{(k)} + \hat{\mathbf{s}}_2\hat{c}$
11:    if $(\mathsf{RejSamp}(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) = 0)$ then
12:        $k \leftarrow k + 1$
13: if $(k \ge \ell)$ then
14:     goto 1
15: $\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k, \mathsf{tree}, h)$
16: return $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth})$
```

Fig. 7. A formal description of a standard (non-optimized) signature scheme using trees of commitments.

in addition to the root of the hash tree corresponding to $\hat{\mathbf{w}}$ and its authentication path auth, i.e., $\mathsf{root} = \mathsf{RootCalc}(\hat{\mathbf{w}}, \mathsf{auth})$. The algorithm accepts if and only if the Euclidean norm of $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)$ is smaller than some predefined bound B and the output of H on input (root, μ) is equal to $\hat{c}$.

The following two theorems show the correctness and security of the above described scheme.

Theorem 4. *Let $\alpha, \eta, \sigma' > 0$, $\sigma = \alpha\eta\sigma'\sqrt{\kappa(k_1 + k_2)n}$, and $B = \eta\sigma\sqrt{(k_1 + k_2)n}$. Any signature generated by the scheme depicted in Figure 7 is verified with probability at least $1 - 2^{-\lambda}$. A signature is generated with probability $1 - \varepsilon$, where $\varepsilon = (1 - \frac{1-2^{-100}}{M})^\ell$, $\ell \in \mathbb{Z}_{\ge 1}$, and $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$.*

Proof. For an honestly generated signature $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth})$ the pair $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)$ is distributed according to $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$ and bounded by $\eta\sigma\sqrt{(k_1 + k_2)n} = B$ with probability $1 - \eta^{(k_1+k_2)n} \cdot \exp(\frac{(k_1+k_2)n}{2}(1 - \eta^2))$ due to Lemma 1. Therefore, choosing η such that this probability $\le 2^{-\lambda}$ ensures that the condition $\|(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)\| \le B$ is satisfied with probability $1 - 2^{-\lambda}$. Furthermore, the honestly generated authentication path auth together with the fact that

$$\hat{\mathbf{w}} = \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} = \hat{\mathbf{A}}\hat{\mathbf{y}}_1^{(k)} + \hat{\mathbf{y}}_2^{(k)} = \hat{\mathbf{v}}^{(k)} \pmod q$$

ensure that the algorithm RootCalc computes the correct root of the hash tree. Therefore, the input of H in Verify is equal to the input of H during signing, hence both outputs equal to $\hat{c}$ and the second condition is satisfied.

Finally, we justify the acceptance probability $1 - \varepsilon$ of a correctly generated signature. Rather than subsequently generating a pair of masking terms $(\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2)$ from $D_{\mathbb{Z}^n,\sigma}^{k_2} \times D_{\mathbb{Z}^n,\sigma}^{k_1}$ and applying rejection sampling, the signing algorithm generates ℓ pairs at once. By Lemma 2, the probability that one pair $(\hat{\mathbf{y}}_1^{(k)}, \hat{\mathbf{y}}_2^{(k)})$, for $k = 0, \ldots, \ell - 1$, masks $\hat{\mathbf{u}} = (\hat{\mathbf{s}}_1\hat{c}, \hat{\mathbf{s}}_2\hat{c})$ is given by

$$D_{\mathbb{Z}^{(k_1+k_2)n},\sigma}(\mathbf{w}^{(k)})/(M \cdot D_{\mathbb{Z}^{(k_1+k_2)n},\sigma,\mathbf{u}}(\mathbf{w}^{(k)})),$$

where $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$ is the expected number of repetitions, $\sigma = \alpha\|\mathbf{u}\|$, and $\mathbf{w}^{(k)}, \mathbf{u}$ are the vector representations of $(\hat{\mathbf{y}}_1^{(k)}, \hat{\mathbf{y}}_2^{(k)}) + \hat{\mathbf{u}}$ and $\hat{\mathbf{u}}$, respectively. Note that by choosing η as described above

Algorithm 1 Simulation of signing queries from the adversary $\mathcal{A}$.

```
$\hat{\mathbf{y}}_1^{(0)},\dots,\hat{\mathbf{y}}_1^{(\ell-1)} \leftarrow D^{k_2}_{\mathbb{Z}^n,\sigma}$
$\hat{\mathbf{y}}_2^{(0)},\dots,\hat{\mathbf{y}}_2^{(\ell-1)} \leftarrow D^{k_1}_{\mathbb{Z}^n,\sigma}$
$k \leftarrow_{\$} \{0,\dots,\ell-1\}$
$(\hat{\mathbf{z}}_1,\hat{\mathbf{z}}_2) \leftarrow (\hat{\mathbf{y}}_1^{(k)},\hat{\mathbf{y}}_2^{(k)})$
$\hat{\mathbf{v}}^{(k)} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$
for $(i = 0,\dots,k-1,k+1,\dots,\ell-1)$ do
    $\hat{\mathbf{v}}^{(i)} \leftarrow \hat{\mathbf{A}}\hat{\mathbf{y}}_1^{(i)} + \hat{\mathbf{y}}_2^{(i)} \pmod q$
$(\mathsf{root},\mathsf{tree}) \leftarrow \mathsf{HashTree}(\hat{\mathbf{v}}^{(0)},\dots,\hat{\mathbf{v}}^{(\ell-1)})$
$\hat{c} := \mathsf{H}(\mathsf{root},\mu)$
$\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k,\mathsf{tree},h)$
return $(\hat{\mathbf{z}}_1,\hat{\mathbf{z}}_2,\hat{c},\mathsf{auth})$
```

the norm $\|\mathbf{u}\|$ is bounded by $\eta\sigma'\sqrt{\kappa(k_1+k_2)n}$ with probability $1-2^{-\lambda}$. The acceptance probability of rejection sampling using one making pair is at least $(1-2^{-100})/M$ following Lemma 2. Thus, the probability that the distribution of $\hat{\mathbf{u}}$ is not concealed by neither of the ℓ pairs $(\hat{\mathbf{y}}_1^{(k)},\hat{\mathbf{y}}_2^{(k)})$ is given by $\varepsilon = (1-\frac{1-2^{-100}}{M})^{\ell}$. □

Remark 1. By choosing ℓ large enough such that the probability $\varepsilon \leq 2^{-\lambda}$, aborting can completely be removed and signatures are generated without repetition with probability of at least $1-2^{-\lambda}$.

Theorem 5. *The signature scheme depicted in Figure 7 is strongly EUF-CMA in the random oracle model (ROM) if F is a collision resistant hash function and both MLWE and MSIS are hard. More precisely, suppose that F is collision resistant, $(\hat{\mathbf{A}},\hat{\mathbf{b}})$ is indistinguishable from uniform, and it is hard to find a vector $(\hat{\mathbf{v}}_1,\hat{\mathbf{v}}_2,\hat{v}_3) \neq \mathbf{0}$ satisfying $\hat{\mathbf{v}}_1 + \hat{\mathbf{A}}\hat{\mathbf{v}}_2 + \hat{\mathbf{b}}\hat{v}_3 = \mathbf{0} \pmod q$ such that $\|(\hat{\mathbf{v}}_1,\hat{\mathbf{v}}_2)\| \leq 2B$ and $\|\hat{v}_3\|_\infty \leq 2$, then the scheme is strongly EUF-CMA in the ROM.*

Proof. We assume that there exists an adversary $\mathcal{A}$, which is able to forge signatures with probability $\varepsilon_{\mathcal{A}}$. We construct a reduction algorithm $\mathcal{D}$ that finds collisions in F or computes $(\hat{\mathbf{v}}_1,\hat{\mathbf{v}}_2,\hat{v}_3) \neq \mathbf{0}$ as described in the theorem statement with probability $\varepsilon_{\mathcal{D}} \geq \varepsilon_{\mathsf{fork}}$, where $\varepsilon_{\mathsf{fork}}$ is given below. The reduction $\mathcal{D}$ has access to an oracle $\mathcal{O}_{\mathsf{F}}$ for F.

Setup. The input of $\mathcal{D}$ is a function F and a matrix $[\mathbf{I}_{k_1}\ \hat{\mathbf{A}}']$, where $\hat{\mathbf{A}}' \leftarrow_{\$} R_q^{k_1\times(k_2+1)}$. The reduction $\mathcal{D}$ writes $\hat{\mathbf{A}}' = [\hat{\mathbf{A}}\ \hat{\mathbf{b}}] \in R_q^{k_1\times k_2} \times R_q^{k_1}$ and then runs the forger $\mathcal{A}$ with public key $(\hat{\mathbf{A}},\hat{\mathbf{b}})$. Assuming the hardness of MLWE any public key generated by KGen is indistinguishable from the uniform distribution over $R_q^{k_1\times k_2} \times R_q^{k_1}$.

Random oracle query. The reduction $\mathcal{D}$ maintains a list L_{H}, which includes pairs of random oracle queries and their answers. If H was previously queried on some input, then $\mathcal{D}$ looks up its entry in L_{H} and returns its answer $\hat{c} \in \mathbb{T}_\kappa^n$. Otherwise, it selects a new $\hat{c} \leftarrow_{\$} \mathbb{T}_\kappa^n$ and updates the list.

Hash Query. Hash queries to F sent by $\mathcal{A}$ are forwarded to the oracle $\mathcal{O}_{\mathsf{F}}$. The reduction $\mathcal{D}$ also maintains a list L_{F}, which includes pairs of hash queries to F and their answers as well as the structure of the trees.

Signature query. Upon receiving a signature query from $\mathcal{A}$, the reduction $\mathcal{D}$ runs Algorithm 1 in order to generate a signature and sends it back to $\mathcal{A}$. Note that $\mathcal{D}$ queries $\mathcal{O}_{\mathsf{F}}$ in order to generate binary hash trees using HashTree. By Lemma 2, simulating the computation of $(\hat{\mathbf{z}}_1,\hat{\mathbf{z}}_2)$ by $\mathcal{D}$ (without having the secret key) is statistically indistinguishable from generating them as in a real execution of the signing algorithm.

Output. After invocation of $\mathcal{A}$, it outputs a valid pair of message and its corresponding signature $(\mu,(\hat{\mathbf{z}}_1,\hat{\mathbf{z}}_2,\hat{c},\mathsf{auth}))$ with probability $\varepsilon_{\mathcal{A}}$. We note that a forgery is considered to be valid even if $\mathcal{A}$ succeeds in changing any part of a signature queried from the signing oracle. If H was not programmed or queried during invocation of $\mathcal{A}$, then $\mathcal{A}$ produces a $\hat{c}$ that validates correctly with probability $1/|\mathbb{T}_\kappa^n|$.

Therefore, the probability that $\mathcal{A}$ succeeds in a forgery and $\hat{c}$ corresponds to one of the random oracle queries $\hat{c}_j$ (for some j) is at least $\varepsilon_{\mathcal{A}} - 1/|\mathbb{T}_\kappa^n|$. Then, one of the following two cases applies.

Case 1. If $\hat{c}$ was included in a response $(\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2', \hat{c}, \mathsf{auth}')$ to a signing query made by $\mathcal{A}$ for a message μ', then $\mathcal{D}$ knows its corresponding hash value root'. In this case we have $\mathsf{H}(\mathsf{root}, \mu) = \hat{c} = \mathsf{H}(\mathsf{root}', \mu')$. If $\mu \neq \mu'$ or $\mathsf{root} \neq \mathsf{root}'$, then a second preimage of $\hat{c}$ has been found by $\mathcal{A}$. If $\mu = \mu'$ and $\mathsf{root} = \mathsf{root}'$, then we have $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \mathsf{auth}) \neq (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2', \mathsf{auth}')$ according to the unforgeability game of Definition 9. Let $\hat{\mathbf{w}} = \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$ and $\hat{\mathbf{w}}' = \hat{\mathbf{A}}\hat{\mathbf{z}}_1' + \hat{\mathbf{z}}_2' - \hat{\mathbf{b}}\hat{c}' \pmod q$. Then, one of the following holds:

1. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) \neq (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} = \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then we know that $\|(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2)\| \leq B$ and $\|(\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')\| \leq B$. Therefore, we have $(\hat{\mathbf{z}}_2 - \hat{\mathbf{z}}_2') + \hat{\mathbf{A}}(\hat{\mathbf{z}}_1 - \hat{\mathbf{z}}_1') = \mathbf{0} \pmod q$ and $\|(\hat{\mathbf{z}}_1 - \hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2 - \hat{\mathbf{z}}_2')\| \leq 2B$. This constitutes a solution to MSIS. If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then a collision in F has been found in the leaves of the hash tree.
2. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) \neq (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} \neq \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then we have a solution to MSIS (as in **1.**). If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then we consider two cases: If $\hat{\mathbf{w}}, \hat{\mathbf{w}}'$ belong to different hash trees, then a collision has occurred in F similar to [Mer89], i.e., there exists an index $i \in \{0, \ldots, h-1\}$ such that $\mathbf{a}_i \neq \mathbf{a}_i'$, where $\mathbf{a}_i \in \mathsf{auth}$, $\mathbf{a}_i' \in \mathsf{auth}'$ and $\mathsf{RootCalc}(\hat{\mathbf{w}}, \mathsf{auth}) = \mathsf{root} = \mathsf{RootCalc}(\hat{\mathbf{w}}', \mathsf{auth}')$. If $\hat{\mathbf{w}}, \hat{\mathbf{w}}'$ belong to the same hash tree, then $\mathcal{A}$ must have found a collision such that $\mathsf{F}(\hat{\mathbf{w}}) = \mathsf{F}(\hat{\mathbf{x}})$, where $\hat{\mathbf{x}} \in L_{\mathsf{F}}$.
3. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) = (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} \neq \mathsf{auth}'$. This yields a collision in F such that $\mathsf{F}(\hat{\mathbf{w}}) = \mathsf{F}(\hat{\mathbf{x}})$, where $\hat{\mathbf{x}} \in L_{\mathsf{F}}$.

Case 2. If $\hat{c}$ was a response to a random oracle query made by $\mathcal{A}$, then the reduction $\mathcal{D}$ records the pair $(\mu, (\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth}))$ and invokes $\mathcal{A}$ again with the same random tape and the random oracle queries $\{\hat{c}_1, \ldots, \hat{c}_{j-1}, \hat{c}_j', \ldots, \hat{c}_{q_{\mathsf{Sign}}+q_{\mathsf{H}}}'\}$, where $\{\hat{c}_j', \ldots, \hat{c}_{q_{\mathsf{Sign}}+q_{\mathsf{H}}}'\}$ are fresh random elements and $q_{\mathsf{Sign}}, q_{\mathsf{H}}$ denotes the maximum number of signing and random oracle queries made by $\mathcal{A}$, respectively. By the General Forking Lemma [BN06], the forger $\mathcal{A}$ outputs a signature $(\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2', \hat{c}', \mathsf{auth}')$ of the message μ with probability $\varepsilon_{\mathsf{fork}}$ (see below), where $\hat{c} \neq \hat{c}'$ and $\mathsf{root} = \mathsf{root}'$. Let $\hat{\mathbf{w}} = \hat{\mathbf{A}}\hat{\mathbf{z}}_1 + \hat{\mathbf{z}}_2 - \hat{\mathbf{b}}\hat{c} \pmod q$ and $\hat{\mathbf{w}}' = \hat{\mathbf{A}}\hat{\mathbf{z}}_1' + \hat{\mathbf{z}}_2' - \hat{\mathbf{b}}\hat{c}' \pmod q$. Then, one of the following holds:

1. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) \neq (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} = \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$ then $(\hat{\mathbf{z}}_2 - \hat{\mathbf{z}}_2') + \hat{\mathbf{A}}(\hat{\mathbf{z}}_1 - \hat{\mathbf{z}}_1') + \hat{\mathbf{b}}(\hat{c}' - \hat{c}) = \mathbf{0} \pmod q$, where $\|(\hat{\mathbf{z}}_1 - \hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2 - \hat{\mathbf{z}}_2')\| \leq 2B$ and $\|\hat{c}' - \hat{c}\|_\infty \leq 2$. This constitutes a solution to MSIS. If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then a collision in F has been found in the leaves of the hash tree.
2. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) \neq (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} \neq \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then we have a solution to MSIS (as in **1.**). If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then we consider two cases: If $\hat{\mathbf{w}}, \hat{\mathbf{w}}'$ belong to different hash trees, then a collision has occurred in F, i.e., there exists an index $i \in \{0, \ldots, h-1\}$ such that $\mathbf{a}_i \neq \mathbf{a}_i'$, where $\mathbf{a}_i \in \mathsf{auth}$, $\mathbf{a}_i' \in \mathsf{auth}'$ and $\mathsf{RootCalc}(\hat{\mathbf{w}}, \mathsf{auth}) = \mathsf{root} = \mathsf{RootCalc}(\hat{\mathbf{w}}', \mathsf{auth}')$. If $\hat{\mathbf{w}}, \hat{\mathbf{w}}'$ belong to the same hash tree, then $\mathcal{D}$ keeps the pair $(\mu, (\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2, \hat{c}, \mathsf{auth}))$ and invokes $\mathcal{A}$ at most ℓ times with the same random tape and the random oracle queries $\{\hat{c}_1, \ldots, \hat{c}_{j-1}, \hat{c}_j^{(t)}, \ldots, \hat{c}_{q_{\mathsf{Sign}}+q_{\mathsf{H}}}^{(t)}\}$ (where $t \in \{0, \ldots, \ell-1\}$) until we obtain two forgeries such that the associated leaves have the same index in the tree and the same hash value. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then we have a solution to MSIS (as in **1.**). Otherwise, we have a collision in F.
3. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) = (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} = \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then $\hat{\mathbf{b}}(\hat{c} - \hat{c}') = \mathbf{0} \pmod q$. Since $\hat{c} \neq \hat{c}'$ and R is an integral domain, the vector $(\mathbf{0}, \mathbf{0}, \hat{c} - \hat{c}')$ constitutes a solution to MSIS. Note that if $\hat{\mathbf{b}}$ is invertible, then we obtain $\hat{c}_i - \hat{c}_i' = 0 \pmod q$. This contradicts $\hat{c}_i \neq \hat{c}_i'$. Moreover, if $(\hat{c}_i - \hat{c}_i')$ is invertible in R_q, then we obtain $\hat{\mathbf{b}} = \mathbf{0} \pmod q$, which is not the case. If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then a collision has occurred in F (in the leaves of the hash tree).
4. $(\hat{\mathbf{z}}_1, \hat{\mathbf{z}}_2) = (\hat{\mathbf{z}}_1', \hat{\mathbf{z}}_2')$ and $\mathsf{auth} \neq \mathsf{auth}'$. If $\hat{\mathbf{w}} = \hat{\mathbf{w}}'$, then we have a solution to MSIS (as in **3.**). If $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$, then one of the cases considered in **2.** (for $\hat{\mathbf{w}} \neq \hat{\mathbf{w}}'$) applies.

By the General Forking Lemma with at most $\ell = O(1)$ rewindings and distinct $\hat{c}_j^{(t)}, \ldots, \hat{c}_{q_{\mathsf{Sign}}+q_{\mathsf{H}}}^{(t)}$, we have

$$\varepsilon_{\mathsf{fork}} \geq \left(\varepsilon_{\mathcal{A}} - \frac{1}{|\mathbb{T}_\kappa^n|}\right) \cdot \left(\left(\frac{\varepsilon_{\mathcal{A}} - 1/|\mathbb{T}_\kappa^n|}{q_{\mathsf{Sign}} + q_{\mathsf{H}}}\right)^\ell - \tau\right), \text{ where } \tau = 1 - \prod_{t=1}^{\ell} \frac{|\mathbb{T}_\kappa^n| - t}{|\mathbb{T}_\kappa^n|} \leq 1 - \left(\frac{|\mathbb{T}_\kappa^n| - \ell}{|\mathbb{T}_\kappa^n|}\right)^\ell \leq \frac{\ell^2}{|\mathbb{T}_\kappa^n|}\ .$$

□

```
$\mathsf{BS.Sign}(\mathsf{sk}, \mathsf{pk}, \mu)$

Signer $\mathcal{S}(\mathsf{sk}, \mathsf{pk})$                          User $\mathcal{U}(\mathsf{pk}, \mu)$

$\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed})$, $\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
$\hat{\mathbf{y}}_1^*, \ldots, \hat{\mathbf{y}}_\kappa^* \leftarrow D_{\mathbb{Z}^n, s^*}^{m+1}$
for $(j = 1, \ldots, \kappa)$ do
    $\hat{y}_j \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{y}}_j^* \pmod q$
        $\hat{\mathbf{y}} := (\hat{y}_1, \ldots, \hat{y}_\kappa)$
  ──────────────────────────────→

$\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed})$, $\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
$\mathbf{r}, \mathbf{r}', \rho, \rho' \leftarrow_\$ \{0,1\}^\lambda$, $\hat{p}_1, \ldots, \hat{p}_\kappa \leftarrow_\$ \hat{\mathbb{T}}$
$\tau \leftarrow \mathsf{Com}(\mu; \mathbf{r})$, $\tau' \leftarrow \mathsf{Com}(\rho'; \mathbf{r}')$
$\hat{\mathbf{e}}^{(0)}, \ldots, \hat{\mathbf{e}}^{(\ell-1)} \leftarrow D_{\mathbb{Z}^n, s}^{m+1}(\rho)$
$\hat{y} \leftarrow \sum_1^\kappa \hat{p}_j \hat{y}_j \pmod q$
for $(k = 0, \ldots, \ell - 1)$ do
    $\hat{t}^{(k)} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{e}}^{(k)} + \hat{y} \pmod q$
$(\mathsf{root}, \mathsf{tree}) \leftarrow \mathsf{HashTree}(\hat{t}^{(0)}, \ldots, \hat{t}^{(\ell-1)})$
$\hat{c} \leftarrow \mathsf{H}(\mathsf{root}, \tau', \tau)$, $\hat{c} := \sum_1^\kappa \hat{c}_j,\ \hat{c}_j \in \hat{\mathbb{T}}$
for $(j = 1, \ldots, \kappa)$ do
    $\hat{c}_j^* \leftarrow \hat{p}_j^{-1} \cdot \hat{c}_j$
        $\hat{\mathbf{c}}^* := (\hat{c}_1^*, \ldots, \hat{c}_\kappa^*)$
  ←──────────────────────────────

for $(j = 1, \ldots, \kappa)$ do
    $\hat{\mathbf{z}}_j^* \leftarrow \hat{\mathbf{y}}_j^* + \hat{\mathbf{s}}\hat{c}_j^*$
if $(\mathsf{RejSamp}(\hat{\mathbf{z}}_1^*, \ldots, \hat{\mathbf{z}}_\kappa^*) = 0)$ then
    restart
        $\hat{\mathbf{z}}^* := (\hat{z}_{1,1}^*, \ldots, \hat{z}_{\kappa,2}^*)$
  ──────────────────────────────→

$\hat{\mathbf{v}} \leftarrow \sum_1^\kappa \hat{p}_j \hat{\mathbf{z}}_j^*$
if $(\|\hat{\mathbf{v}}\| > \eta s^* \sqrt{(m+1)\kappa n})$ then
    abort (occurs with probability $2^{-\lambda}$)
$(\rho_0, \ldots, \rho_{\ell-1}) \leftarrow \mathsf{E}(\rho')$
$k \leftarrow 0$
while $(k < \ell)$ do
    $\hat{\mathbf{z}} \leftarrow \hat{\mathbf{e}}^{(k)} + \hat{\mathbf{v}}$
    if $(\mathsf{RejSamp}(\hat{\mathbf{z}}; \rho_k) = 0)$ then
        $k \leftarrow k + 1$
if $(k \geq \ell)$ then
    $\mathsf{result} \leftarrow (\tau, \rho, \rho', \mathbf{r}', \hat{p}_1, \ldots, \hat{p}_\kappa, \hat{c})$
$\mathsf{result} \leftarrow \mathsf{ok}$
              $\mathsf{result}$
  ←──────────────────────────────

if $(\mathsf{result} \neq \mathsf{ok})$ then
    if $(\mathsf{Proof}(\mathsf{pk}, \hat{\mathbf{y}}, \hat{\mathbf{c}}^*, \hat{\mathbf{z}}^*, \mathsf{result}) = 1)$ then
        restart

if $(\mathsf{result} = \mathsf{ok})$
    $\mathsf{auth} \leftarrow \mathsf{BuildAuth}(k, \mathsf{tree}, h)$
    $\hat{\mathbf{z}} \leftarrow \mathsf{Compress}(\hat{\mathbf{z}})$
    return $(\mu, (\tau', \mathbf{r}, \hat{\mathbf{z}}, \hat{c}, \mathsf{auth}))$
```

Fig. 8. The signing algorithm of the 4-move version of BLAZE^+. The algorithm Proof is described in Figure 9.

B The 4-Move Version of BLAZE$^+$

In this section we present a 4-move version of BLAZE$^+$. Similar to BLAZE [AEB20], in this version the user $\mathcal{U}$ requests a protocol restart from the signer $\mathcal{S}$ if the computed signature does not follow the correct distribution (blindness may not be satisfied). In order to check this, $\mathcal{U}$ carries out rejection sampling. After that, $\mathcal{U}$ sends $\mathcal{S}$ either an ok message or a proof of failure, which allows $\mathcal{S}$ to verify the invalidity of the computed signature and restarts the signing protocol.

Let $\mathsf{Com} : \{0,1\}^* \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a statistically hiding and computationally binding commitment function. The key generation algorithm is identical to that of the 3-move version (see Figure 5). Signing is described in Figure 8. The verification algorithm as well as the algorithm that verifies the proof of failure is given in Figure 9. The proof of blindness directly follows from [AEB20]. The proof of one-more unforgeability is carried out similar to the 3-move version (Theorem 3) in addition to making use of the one-more unforgeability proof of BLAZE, which further shows that under the RSIS assumption users cannot obtain a valid signature after an aborted interaction with the signer.

```
Proof(pk, $\hat{\mathbf{y}}$, $\hat{\mathbf{c}}^*$, $\hat{\mathbf{z}}^*$, result)
$\hat{\mathbf{y}} := (\hat{y}_1, \ldots, \hat{y}_\kappa)$, $\hat{\mathbf{c}}^* := (\hat{c}_1^*, \ldots, \hat{c}_\kappa^*)$, $\hat{\mathbf{z}}^* := (\hat{\mathbf{z}}_1^*, \ldots, \hat{\mathbf{z}}_\kappa^*)$
result $:= (\tau, \rho, \rho', \mathbf{r}', \hat{p}_1, \ldots, \hat{p}_\kappa, \hat{c})$
$\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed})$, $\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$, $\tau' \leftarrow \mathsf{Com}(\rho'; \mathbf{r}')$, $(\rho_0, \ldots, \rho_{\ell-1}) \leftarrow \mathsf{E}(\rho')$
$\hat{\mathbf{e}}^{(0)}, \ldots, \hat{\mathbf{e}}^{(\ell-1)} \leftarrow D^{m+1}_{\mathbb{Z}^n,s}(\rho)$
$\hat{y} \leftarrow \sum_1^\kappa \hat{p}_j \hat{y}_j \pmod q$
for $(k = 0, \ldots, \ell-1)$ do
    $\hat{t}^{(k)} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{e}}^{(k)} + \hat{y} \pmod q$
(root, tree) $\leftarrow$ HashTree($\hat{t}^{(0)}, \ldots, \hat{t}^{(\ell-1)}$)
$\hat{\mathbf{v}} \leftarrow \sum_1^\kappa \hat{p}_j \hat{\mathbf{z}}_j^*$
for $(k = 0, \ldots, \ell-1)$ do
    $\hat{\mathbf{z}}^{(k)} \leftarrow \hat{\mathbf{e}}^{(k)} + \hat{\mathbf{v}}$
    auth $\leftarrow$ BuildAuth($k$, tree, $h$)
    $\hat{w}^{(k)} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{z}}^{(k)} - \hat{b}\hat{c} \pmod q$
    if $\left(\hat{c} \neq \mathsf{H}(\mathsf{RootCalc}(\hat{w}^{(k)}, \mathsf{auth}), \tau', \tau) \vee \mathsf{RejSamp}(\hat{\mathbf{z}}^{(k)}; \rho_k) = 1\right)$ then
        return 0
if $\left(\sum_1^\kappa \hat{p}_j \hat{c}_j^* = \hat{c} = \mathsf{H}(\mathsf{root}, \tau', \tau)\right)$ then
    return 1
else return 0

BS.Verify(pk, $\mu$, $(\tau', \mathbf{r}, \hat{\mathbf{z}}, \hat{c}, \mathsf{auth})$)
$\hat{\mathbf{a}}' \leftarrow \mathsf{E}(\mathsf{seed})$
$\hat{\mathbf{a}} \leftarrow [1\ \hat{\mathbf{a}}']$
$\hat{\mathbf{z}} \leftarrow \mathsf{Decompress}(\hat{\mathbf{z}})$
$\hat{w} \leftarrow \hat{\mathbf{a}} \cdot \hat{\mathbf{z}} - \hat{b}\hat{c} \pmod q$
root $\leftarrow$ RootCalc($\hat{w}$, auth)
if $\left(\|\hat{\mathbf{z}}\| \leq B \ \wedge\ \hat{c} = \mathsf{H}(\mathsf{root}, \tau', \mathsf{Com}(\mu; \mathbf{r}))\right)$ then
    return 1
return 0
```

Fig. 9. The verification algorithm as well as the algorithm that verifies the proof of failure of the 4-move version of BLAZE$^+$.