A preliminary version of this paper appears in the proceedings of the *20th International Conference on Cryptology and Network Security (CANS 2021)*, © Springer 2021. This is the full version.

BlindOR: An Efficient Lattice-Based Blind Signature Scheme from OR-Proofs

Nabil Alkeilani Alkadri Patrick Harasser Christian Janson

Technische Universität Darmstadt, Germany
{nabil.alkadri, patrick.harasser, christian.janson}@tu-darmstadt.de

February 25, 2022

Abstract. An OR-proof is a protocol that enables a user to prove the possession of a witness for one of two (or more) statements, without revealing which one. Abe and Okamoto (CRYPTO 2000) used this technique to build a partially blind signature scheme whose security is based on the hardness of the discrete logarithm problem. Inspired by their approach, we present BlindOR, an efficient blind signature scheme from OR-proofs based on lattices over modules. Using OR-proofs allows us to reduce the security of our scheme from the MLWE and MSIS problems, yielding a much more efficient solution compared to previous works.

Keywords. Blind signatures · OR-proof · Lattice-based cryptography

Contents

1 Introduction

Blind signature schemes are a fundamental cryptographic primitive. First introduced by Chaum [Cha82] in the context of an anonymous e-cash system, they have since become an essential building block in many applications such as anonymous credentials [CG08, BL13], e-voting [KKS17], and blockchain protocols [HBG16]. They have been standardized as *ISO/IEC 18370*, and were deployed in real-life applications such as Microsoft's *U-Prove* technology and smart card devices produced by Gemalto.

In a blind signature scheme, a user holding a message m interacts with a signer to generate a blind signature on m under the signer's secret key. The scheme is required to satisfy two security properties called *blindness* and *one-more unforgeability* [JLO97, PS00]. Informally, the first condition means that the signer gets no information about m during the signing process, while the latter ensures that the user cannot generate signatures without interacting with the signer.

In an effort to develop practical blind signature schemes from a diverse range of assumptions (in particular, those conjectured to be secure against quantum attacks), various schemes based on lattice problems have been proposed. The first such scheme by Rückert [Rüc10] can be seen as an important step in carrying the core design of classical constructions based on the discrete logarithm problem [PS00] over to the lattice setting. The same design principle was then adopted in subsequent works, *e.g.*, by Alkeilani Alkadri *et al.* [AEB20a, AEB20b], where the scheme BLAZE and its successor BLAZE^+ have been proposed and shown to be practical.

Recently, Hauck *et al.* [HKLN20] pointed out that the proof of the one-more unforgeability property, originally by Pointcheval and Stern for a discrete logarithm based construction [PS00] and later reproposed by Rückert for his lattice-based scheme [Rüc10], has not been adapted correctly to this new setting. Indeed, the main idea of the reduction in [PS00] is to select a secret key sk and then run the forger with the related public key pk, which represents an instance of a computationally hard problem that admits more than one solution. In other words, pk is related to more than one sk, and the forger cannot distinguish which sk is used by the reduction. Note that it is crucial for the reduction to know a secret key because, unlike standard signature schemes, the signer cannot be simulated without one (otherwise the scheme would be universally forgeable [PS00]). After running the forger and obtaining an element z, the reduction rewinds the forger with the same random tape and partially different random oracle replies to obtain z'. The proof in [PS00] then uses a subtle argument to ensure that $z \neq z'$ with noticeable probability, which yields a solution to the underlying hard problem.

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

Unfortunately, as stated by the authors themselves, their work is mostly of theoretical interest. Indeed, the solution presented in [HKLN20] entails increasing the parameter sizes, so that their framework applies and yields a correct proof. In particular, the RSIS-based instantiation given in [HKLN20] has public and secret keys of size 443.75 KB and 4275 KB, respectively, and generates signatures of size 7915.52 KB. This leaves us in the regrettable position where all known (three-move) lattice-based blind signature schemes are either not backed by a correct security proof, or need impractically large parameters to achieve security.

1.1 Our Contributions

In this paper we make progress towards constructing efficient and at the same time provably secure lattice-based blind signature schemes. We present BlindOR, a new blind signature scheme based on lattices over modules. Our scheme uses the OR-technique of Cramer *et al.* [CDS94], a feature which allows us to sidestep the missing security argument pointed out in [HKLN20]. At a high level, an OR-proof is a Sigma protocol that proves the knowledge of a witness for one of two statements, without revealing which one. Therefore, the public key of our scheme consists of two statements (two instances of a hard lattice problem), and the secret key includes a witness for one of them. Consequently and for the first time, the hardness assumption underlying the public key does not have to "natively" admit multiple solutions, because the OR-technique already forces there to be more than one (and thus simulation of signatures is still possible).

In particular, the public key of BlindOR consists of two instances of the *Module Learning with Errors* (MLWE) problem, which results in a much more efficient scheme. Signing is carried out by proving the possession of the witness included in the secret key. A user interacting with the signer blinds the two transcripts generated by the signer without being able to distinguish for which instance the signer holds a witness. We capture these blinding steps in a set of algorithms and show that BlindOR is statistically blind. The one-more unforgeability of our scheme is proven in the random oracle model (ROM) [BR93], assuming the hardness of both MLWE and MSIS (the module version of SIS). The reduction creates one instance of the hard problem with a witness in order to simulate the signing oracle, and tries to solve the other instance, which is given to the reduction as input. That is, the reduction does not know a witness for its input. This is analogous to the security proof of standard lattice-based signature schemes, and hence no further conditions are required to ensure the correctness and success of the reduction with high probability. This is in contrast to previous lattice-based constructions of blind signatures, as observed in [HKLN20].

BlindOR uses techniques from prior works in order to reduce or even remove the number of restarts inherent in lattice-based schemes. More precisely, it uses the *partitioning and permutation* technique introduced in [AEB20a]. Given a hash function taking values in the challenge space of the underlying Sigma protocol, it allows to blind the hash values without having to carry out any security check or potential restart. Another advantage of this technique is that it can be used to construct OR-proofs based on lattice assumptions, because it allows to use a specified challenge space that has an abelian group structure, a crucial requirement for OR-proofs. This is in contrast to the typical challenge space used in current lattice-based schemes, which consists of polynomials from the ring $\mathbb{Z}[X]/\langle X^n + 1\rangle$ with coefficients in $\{-1, 0, 1\}$ and a given Hamming weight. We also use the *trees of commitments* technique from [AEB20b] to remove the restarts induced by the user when blinding the signature generated by the signer. We extend this technique in BlindOR to reduce the potential restarts induced by the signer when computing signatures, which must be distributed independently from the secret key.

To demonstrate the efficiency of our scheme, we propose concrete parameters for BlindOR targeting 128 bits of security. The related key and signature sizes, the communication cost, and a comparison with the corresponding metrics for the scheme proposed by Hauck *et al.* [HKLN20] are given in Table 1. In summary, although our scheme requires twice as many public key and signature parts, which is inherent to using OR-proofs, it yields smaller sizes compared to the provably secure construction from [HKLN20], resulting in a more efficient scheme overall.

We remark that the security of our scheme can easily be extended to the stronger security notions of *selective failure blindness* [CNs07] and *honest-user unforgeability* [SU17]. This is established by signing a commitment to the message instead of the message itself [FS09, SU17]. The commitment is generated by using a commitment scheme that is statistically hiding and computationally binding. However, and similarly to [HKLN20], it is still unclear how to prove the blindness property under a maliciously generated key pair [Fis06].

Table 1: A comparison between BlindOR and the scheme introduced in [HKLN20] in terms of key and signature sizes and communication cost. Numbers are given in kilobytes (KB). The related parameters are given in Table 3 and [HKLN20, Figure 9].

Scheme	Public key	Secret key	Signature	Communication
BlindOR	11.25	1.69	253.44	396.64
[HKLN20]	443.75	4275	7915.52	34037.25

1.2 Related Work

Our construction is inspired by the work of Abe and Okamoto [AO00], who used OR-proofs to build partially blind signatures with security based on the hardness of the discrete logarithm problem. Informally, a partially blind signature scheme allows to include common information *info* (*e.g.*, the date of issue) in the blind signature under some agreement between the signer and user. Unlike our construction, the public key of their scheme consists of only one instance of the hard problem. The second instance is created within the signing protocol as the image of *info* under a cryptographic hash function, which is modeled as random oracle and transforms *info* into a random public key whose secret key in unknown to anybody. Therefore, the second problem instance depends on *info*, and hence the user knows for which instance the signer holds a witness. Observe that we cannot simply convert their scheme to the lattice setting, as this would force us to use MSIS (instead of MLWE) and result in an inefficient scheme. The change to MLWE is possible because there is no common information to consider in our case.

Hauck *et al.* [HKLN20] showed that all lattice-based constructions of blind signatures from Sigma protocols (or canonical identification schemes) prior to their framework (*e.g.*, [Rüc10, PHBS19, AEB20a, AEB20b, LDS+20]) do not have a valid security argument. Furthermore, Alkeilani Alkadri *et al.* [AEB20a] showed that all two-round lattice-based blind signature schemes based on preimage sampleable trapdoor functions (*e.g.*, [CCT+11, GHWX16]) are insecure.

Fischlin and Schröder [FS10] showed that it is infeasible to find a black-box reduction from some non-interactive cryptographic assumption to the one-more unforgeability in the standard model for blind signature schemes that satisfy some specific properties. These impossibility results do not apply to our construction, since it is proven secure in the ROM.

Recently, Agrawal *et al.* [ASY21] made a step towards practical two-round lattice-based blind signatures. They improved the two-round construction of Garg *et al.* [GRS+11] which is based on general complexity assumptions, and *degraded* it to rely on the ROM. This allows them to avoid complexity leveraging, the main source of inefficiency in [GRS+11]. The construction requires a homomorphic encryption scheme with specific properties in order to evaluate the signing algorithm of a standard signature scheme. To obtain a relatively simple circuit that can be evaluated homomorphically, they provided a variant of Lyubashevsky's scheme [Lyu12] that does not use rejection sampling. This is attained by employing the so-called *noise flooding* technique [Gen09, GKPV10]. However, as pointed out by the authors, there are some challenges left before this approach becomes practical. For instance, the scheme requires the homomorphic evaluation of a specific signing algorithm that relies on the random oracle. In practice, this must be instantiated with a cryptographic hash function that can be evaluated homomorphically. Finding such a function is still an open problem. Another challenge that has to be tackled is how to handle the compatibility issue induced by the various formats of the quantities involved in the homomorphic signing. We refer to [ASY21, Section 6.3] for more details and discussions on the limitations of their construction.

In the context of blind signatures, Schnorr [Sch01] defined a computational problem called *Random inhomogenities in an Overdetermined, Solvable system of linear equations* (ROS), and showed that solving the ROS problem implies the forgeability of number-theoretic blind signature schemes such as [CP93, Oka93, PS00, AO00]. This attack was greatly improved, *e.g.*, by Wagner [Wag02], and most recently by Benhamouda *et al.* [BLL+21]. However, Hauck *et al.* [HKLN20] showed that the ROS attack cannot be applied to lattice-based blind signature schemes due to their algebraic structure. They defined a lattice variant of this problem called the *Generalized* ROS problem, and leave solving this variant in sub-exponential time as an open problem.

1.3 Outline

The remainder of the paper is structured as follows. We recall all required preliminaries in Section 2. Afterwards, we present our lattice-based blind signature scheme $\mathsf{BlindOR}$ in Section 3. We conclude the paper in Section 4.

2 Preliminaries

In this section we collect the necessary background and notation we require throughout the paper.

2.1 Notation

We denote by $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{R}$ the sets of natural numbers, integers, and real numbers, respectively. If $k \in \mathbb{N}$, we let $[k] := \{1, \dots, k\}$. For $q \in \mathbb{N}$, we write $\mathbb{Z}_q$ to denote the ring of integers modulo q with representatives in $\left[-\frac{q}{2}, \frac{q}{2}\right) \cap \mathbb{Z}$. If n is a fixed power of 2, we define the ring $R := \mathbb{Z}[X]/\langle X^n + 1\rangle$ and its quotient $R_q := R/qR$. Elements in R and R_q are denoted by regular font letters. Column vectors with coefficients in R or R_q are denoted by bold lower-case letters, while bold upper-case letters are matrices. We let $\mathbf{I}_k$ denote the identity matrix of dimension k, and $\mathbb{T}^n_\kappa$ the subset of R_q containing all polynomials with coefficients in $\{-1, 0, 1\}$ and Hamming weight κ. The ℓ_2 and ℓ_∞ norms of an element $a = \sum_{i=0}^{n-1} a_i X^i \in R$ are defined by $\|a\| := \left(\sum_{i=0}^{n-1} |a_i|^2\right)^{1/2}$ and $\|a\|_\infty := \max_i |a_i|$, respectively. Similarly, for $\mathbf{b} = (b_1, \dots, b_k)^t \in R^k$, we let $\|\mathbf{b}\| := \left(\sum_{i=1}^{k} \|b_i\|^2\right)^{1/2}$ and $\|\mathbf{b}\|_\infty := \max_i \|b_i\|_\infty$. All logarithms are to base 2.

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

We denote the security parameter by $\lambda \in \mathbb{N}$, and abbreviate probabilistic polynomial-time by PPT and deterministic polynomial-time by DPT. For a probabilistic algorithm A, we write $y \leftarrow\!\!\$\; \mathsf{A}^{\mathsf{O}}(x)$ to denote that A returns y when run on input x with access to oracle O, and $y \in \mathsf{A}^{\mathsf{O}}(x)$ if y is a possible output of $\mathsf{A}^{\mathsf{O}}(x)$. To make the randomness $r \in \mathcal{RS}_{\mathsf{A}}$ on which A is run explicit, we use the notation $y \leftarrow \mathsf{A}^{\mathsf{O}}(x; r)$. If A and B are interactive algorithms, we write $(x, y) \leftarrow\!\!\$\; \langle \mathsf{A}(a), \mathsf{B}(b) \rangle$ to denote the joint execution of A and B in an interactive protocol with private inputs a for A and b for B, as well as private outputs x for A and y for B. Accordingly, we write $\mathsf{A}^{\langle \cdot, \mathsf{B}(b) \rangle^k}(a)$ if A can invoke up to k executions of the protocol with B.

The *random oracle model* (ROM) [BR93] is a model of computation where all occurrences of a hash function are replaced by a *random oracle* H, *i.e.*, a function chosen at random from the space of all functions $\{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{H}}}$ for some $\ell_{\mathsf{H}} \in \mathbb{N}$, to which all involved parties have oracle access. This means that, for every new oracle query, H returns a truly random response from $\{0,1\}^{\ell_{\mathsf{H}}}$, and every repeated query consistently yields the same output.

2.2 Relations, Sigma Protocols, and OR-Proofs

Definition 2.1. *A* relation *is a tuple* $\mathcal{R} = (\mathcal{R}.\mathsf{PGen}, \mathcal{R}.\mathsf{RSet}, \mathcal{R}.\mathsf{Gen})$*, where:*

$\mathcal{R}.\mathsf{PGen}$ *is the* parameter generation algorithm *which, on input the security parameter* $\lambda \in \mathbb{N}$*, returns public parameters* pp*.*

```
$\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}(pp, b)$:
11:  if $b = 0$ then
12:      $d, d' \leftarrow\$ \{0,1\}$
13:      $x_d \leftarrow\$ \mathcal{R}.\mathsf{Gen}(pp, 0)$
14:      $x_{1-d} \leftarrow\$ \mathcal{R}.\mathsf{Gen}(pp, d')$
15:      return $(x_0, x_1)$
16:  else
17:      $d \leftarrow\$ \{0,1\}$
18:      $(x_0, w_0) \leftarrow\$ \mathcal{R}.\mathsf{Gen}(pp, 1)$
19:      $(x_1, w_1) \leftarrow\$ \mathcal{R}.\mathsf{Gen}(pp, 1)$
20:      $w \leftarrow w_d$
21:      return $((x_0, x_1), (d, w))$
```

Figure 1: Definition of the instance generator $\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}$ of the OR-relation on $\mathcal{R}$. Note that in line 14 we slightly abuse notation: If $d' = 1$ (*i.e.*, the instance generator creates a yes-instance), we only consider the first component of the output, and ignore the witness in the second coordinate.

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

$\mathcal{R}.\mathsf{Gen}$ *is the* instance generator algorithm *which, on input pp* $\in \mathcal{R}.\mathsf{PGen}(1^\lambda)$ *and* $b \in \{0,1\}$*, returns a pair* $(x, w) \in \mathcal{R}.\mathsf{RSet}(pp)$ *if* $b = 1$ *(where x is called a* yes-instance *for* $\mathcal{R}$ *w.r.t. pp and w a corresponding* witness*), and an element x if* $b = 0$ *(called a* no-instance *for* $\mathcal{R}$ *w.r.t. pp).*

We now define the OR-relation $\mathcal{R}_{\mathsf{OR}}$ on a relation $\mathcal{R}$. Informally, for $\lambda \in \mathbb{N}$ and public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, a yes-instance for $\mathcal{R}_{\mathsf{OR}}$ w.r.t. pp is a pair of values (x_0, x_1), where each value is a yes-instance for $\mathcal{R}$ w.r.t. pp. A witness for such an instance is a witness for one of the two coordinates, *i.e.*, a pair (d, w) with $d \in \{0,1\}$ and w a witness for x_d. In contrast, a no-instance for $\mathcal{R}_{\mathsf{OR}}$ consists of a pair (x_0, x_1), where at least one coordinate is a no-instance for $\mathcal{R}$ w.r.t. pp.

Definition 2.2. *Let* $\mathcal{R}$ *be a relation. The* OR-relation on $\mathcal{R}$ *is the relation* $\mathcal{R}_{\mathsf{OR}}$ *whose parameter generation algorithm is* $\mathcal{R}_{\mathsf{OR}}.\mathsf{PGen} := \mathcal{R}.\mathsf{PGen}$*, whose relation set is given by*

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

and whose instance generator $\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}$ *is given in Figure 1.*

We now recall the notion of a Sigma protocol for a relation $\mathcal{R}$, first introduced by Cramer [Cra97].

Definition 2.3. *Let* $\mathcal{R}$ *be a relation. A* Sigma protocol *for* $\mathcal{R}$ *is a tuple of algorithms* $\Sigma = (\Sigma.\mathsf{P}, \Sigma.\mathsf{V}, \Sigma.\mathsf{Sim}, \Sigma.\mathsf{Ext}, \Sigma.\mathsf{ComRec})$*, where:*

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

- $\Sigma.\mathsf{P}_1$ *is a PPT algorithm which, on input a set of public parameters pp and an instance-witness pair* (x, w)*, returns a message cm, called the* commitment*, and a state* $st_{\Sigma.\mathsf{P}}$.
- $\Sigma.\mathsf{P}_2$ *is a DPT algorithm which, on input a set of public parameters pp, an instance-witness pair* (x, w)*, the state information* $st_{\Sigma.\mathsf{P}}$*, and a verifier message ch, outputs a message rp, called the* response.

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

- $\Sigma.\mathsf{V}_1$ *is a PPT algorithm which, on input a set of public parameters pp, an instance x, and a prover message cm, returns a message ch (called the* challenge*) sampled uniformly at random from a finite* *abelian group $\mathcal{C}(pp)$ (called the* challenge space*), as well as a state $st_{\Sigma.\mathsf{V}} = (cm, ch)$ consisting only of the received message and the sampled challenge.*

- $\Sigma.\mathsf{V}_2$ *is a DPT algorithm which, on input a set of public parameters pp, an instance x, the state information $st_{\Sigma.\mathsf{V}} = (cm, ch)$, and a prover message rp, outputs a pair (b, int) with $b \in \{0,1\}$ and $int \in \mathbb{Z}$. We say that the verifier* accepts *the transcript if $b = 1$, and that it* rejects *if $b = 0$.*

$\Sigma.\mathsf{Sim}$ *is a PPT algorithm, called* simulator*. On input a set of public parameters pp, an instance x, and a challenge ch, it outputs a pair of messages (cm, rp).*

$\Sigma.\mathsf{Ext}$ *is a DPT algorithm, called* extractor*. On input a set of public parameters pp, an instance x, and two transcripts (cm, ch, rp) and (cm, ch', rp') such that $ch \neq ch'$ and $\Sigma.\mathsf{V}_2$ returns the same output $(1, int)$ in both cases, $\Sigma.\mathsf{Ext}$ outputs a string w such that $(x, w) \in \mathcal{R}.\mathsf{RSet}(pp)$.*

$\Sigma.\mathsf{ComRec}$ *is a DPT algorithm, called* commitment recovering *algorithm. On input a set of public parameters pp, an instance x, a challenge ch, and a response rp, it returns a message cm.*

If $\mathcal{R}$ is a relation, the Sigma protocols for $\mathcal{R}$ we consider must satisfy a few properties which we briefly describe in the following. The first one is correctness, saying that an honest protocol execution is likely to be accepted by the verifier. Next, there is a variant of the zero-knowledge property, where we require that on input an instance x and a randomly chosen challenge ch, the simulator be able to provide an authentic-looking transcript. Finally, we have soundness, saying that if the commitment recovering algorithm succeeds in finding a commitment, this commitment verifies for the given challenge and response.

We now consider the OR-combination of two Sigma protocols (*OR-proof*). It enables a prover P to show that it knows the witness of one of several statements, or that one out of many statements is true. Here, we restrict ourselves to the case where a prover holds two statements (x_0, x_1) and one witness w for x_d, with $d \in \{0,1\}$. The prover's goal is to convince the verifier that it holds a witness for one of the two statements, without revealing which one. This problem was first solved by Cramer *et al.* [CDS94], and we now recall their construction.

Let $\mathcal{R}$ be a relation and Σ_0, Σ_1 be two Sigma protocols for $\mathcal{R}$. The construction of [CDS94] (depicted in Figure 2) allows to combine Σ_0 and Σ_1 into a new Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma_0, \Sigma_1]$ for the relation $\mathcal{R}_{\mathsf{OR}}$. The key idea of the construction is that the prover $\Sigma_{\mathsf{OR}}.\mathsf{P}$ splits the challenge ch received by $\Sigma_{\mathsf{OR}}.\mathsf{V}$ into two random parts $ch = ch_0 + ch_1$, and is able to provide accepting transcripts for both statements x_0 and x_1 for the respective challenge share. In more detail, for a given security parameter $\lambda \in \mathbb{N}$, public parameters $pp \in \mathcal{R}.\mathsf{PGen}(1^\lambda)$, and instance-witness pair $((x_0, x_1), (d, w)) \in \mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}(pp, 1)$, the execution of Σ_{OR} proceeds as follows:

(a) The prover $\Sigma_{\mathsf{OR}}.\mathsf{P}_1$ starts with computing $(cm_d, st_{\Sigma_d.\mathsf{P}}) \leftarrow\!\!\$\ \Sigma_d.\mathsf{P}_1(pp, x_d, w)$ and samples a challenge $ch_{1-d} \leftarrow\!\!\$\ \mathcal{C}(pp)$. Next, it runs $(cm_{1-d}, rp_{1-d}) \leftarrow\!\!\$\ \Sigma_{1-d}.\mathsf{Sim}(pp, x_{1-d}, ch_{1-d})$ to complete the transcript of x_{1-d}. In case the simulation fails (*i.e.*, $(cm_{1-d}, rp_{1-d}) = (\perp, \perp)$), the prover re-runs the simulator. Finally, it sets $st_{\Sigma_{\mathsf{OR}}.\mathsf{P}} \leftarrow (st_{\Sigma_d.\mathsf{P}}, ch_{1-d}, rp_{1-d})$ and sends (cm_0, cm_1) to the verifier $\Sigma_{\mathsf{OR}}.\mathsf{V}_1$.

(b) Upon receiving the commitments (cm_0, cm_1), $\Sigma_{\mathsf{OR}}.\mathsf{V}_1$ samples a random challenge from the challenge space, *i.e.*, $ch \leftarrow\!\!\$\ \mathcal{C}(pp)$, and sends it to $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$. Finally, it sets its state to $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} \leftarrow (cm_0, cm_1, ch)$.

(c) After receiving the challenge ch, $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ sets $ch_d \leftarrow ch - ch_{1-d}$ and computes a response for x_d as $rp_d \leftarrow \Sigma_d.\mathsf{P}_2(pp, x_d, w, st_{\Sigma_d.\mathsf{P}}, ch_d)$. In case this computation fails (*i.e.*, $rp_d = \perp$), it also sets $rp_{1-d} \leftarrow \perp$. Otherwise, the prover sends the split challenges and responses to the verifier.

(d) After receiving (ch_0, ch_1, rp_0, rp_1) from the prover, $\Sigma_{\mathsf{OR}}.\mathsf{V}_2$ accepts if and only if the shares satisfy $ch = ch_0 + ch_1$ and both transcripts verify correctly.

```
$\Sigma_{\mathsf{OR}}.\mathsf{P}_1(pp, (x_0, x_1), (d, w))$:
  $(cm_d, st_{\Sigma_d.\mathsf{P}}) \leftarrow\$ \Sigma_d.\mathsf{P}_1(pp, x_d, w)$
  $ch_{1-d} \leftarrow\$ \mathcal{C}(pp)$
  $(cm_{1-d}, rp_{1-d}) \leftarrow\$$
        $\leftarrow\$ \Sigma_{1-d}.\mathsf{Sim}(pp, x_{1-d}, ch_{1-d})$
  if $(cm_{1-d}, rp_{1-d}) = (\perp, \perp)$ then
    restart $\Sigma_{1-d}.\mathsf{Sim}$
  $st_{\Sigma_{\mathsf{OR}}.\mathsf{P}} \leftarrow (st_{\Sigma_d.\mathsf{P}}, ch_{1-d}, rp_{1-d})$
  return $((cm_0, cm_1), st_{\Sigma_{\mathsf{OR}}.\mathsf{P}})$

$\Sigma_{\mathsf{OR}}.\mathsf{V}_1(pp, (x_0, x_1), cm_0, cm_1)$:
  $ch \leftarrow\$ \mathcal{C}(pp)$
  $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} \leftarrow (cm_0, cm_1, ch)$
  return $(ch, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$

$\Sigma_{\mathsf{OR}}.\mathsf{P}_2(pp, (x_0, x_1), (d, w), st_{\Sigma_{\mathsf{OR}}.\mathsf{P}}, ch)$:
  parse $st_{\Sigma_{\mathsf{OR}}.\mathsf{P}} = (st_{\Sigma_d.\mathsf{P}}, ch_{1-d}, rp_{1-d})$
  $ch_d \leftarrow ch - ch_{1-d}$
  $rp_d \leftarrow \Sigma_d.\mathsf{P}_2(pp, x_d, w, st_{\Sigma_d.\mathsf{P}}, ch_d)$
  if $rp_d = \perp$ then
    $(ch_0, ch_1, rp_0, rp_1) \leftarrow (\perp, \perp, \perp, \perp)$
  return $(ch_0, ch_1, rp_0, rp_1)$

$\Sigma_{\mathsf{OR}}.\mathsf{Sim}(pp, (x_0, x_1), ch)$:
  $ch_0 \leftarrow\$ \mathcal{C}(pp)$
  $ch_1 \leftarrow ch - ch_0$
  $(cm_0, rp_0) \leftarrow\$ \Sigma_0.\mathsf{Sim}(pp, x_0, ch_0)$
  $(cm_1, rp_1) \leftarrow\$ \Sigma_1.\mathsf{Sim}(pp, x_1, ch_1)$
  if $((cm_0, rp_0) = (\perp, \perp)) \vee$
        $((cm_1, rp_1) = (\perp, \perp))$ then
    $(cm_0, cm_1) \leftarrow (\perp, \perp)$
    $(ch_0, ch_1, rp_0, rp_1) \leftarrow (\perp, \perp, \perp, \perp)$
  return $((cm_0, cm_1), (ch_0, ch_1, rp_0, rp_1))$
```

```
$\Sigma_{\mathsf{OR}}.\mathsf{V}_2(pp, (x_0, x_1), st_{\Sigma_{\mathsf{OR}}.\mathsf{V}}, (ch_0, ch_1, rp_0, rp_1))$:
  parse $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} = (cm_0, cm_1, ch)$
  if $ch \neq ch_0 + ch_1$ then
    return $(0, (-1, -1))$
  $st_{\Sigma_0.\mathsf{V}} \leftarrow (cm_0, ch_0)$
  $st_{\Sigma_1.\mathsf{V}} \leftarrow (cm_1, ch_1)$
  $(b_0, int_0) \leftarrow \Sigma_0.\mathsf{V}_2(pp, x_0, st_{\Sigma_0.\mathsf{V}}, rp_0)$
  $(b_1, int_1) \leftarrow \Sigma_1.\mathsf{V}_2(pp, x_1, st_{\Sigma_1.\mathsf{V}}, rp_1)$
  if $(b_0 = 0) \vee (b_1 = 0)$ then
    return $(0, (-1, -1))$
  return $(1, (int_0, int_1))$

$\Sigma_{\mathsf{OR}}.\mathsf{Ext}(pp, (x_0, x_1), Tran, Tran')$:
  parse $Tran = ((cm_0, cm_1), ch, (ch_0, ch_1, rp_0, rp_1))$
  parse $Tran' = ((cm_0, cm_1), ch', (ch'_0, ch'_1, rp'_0, rp'_1))$
  if $ch_0 \neq ch'_0$ then
    $w \leftarrow \Sigma_0.\mathsf{Ext}(pp, x_0, (cm_0, ch_0, rp_0),$
        $(cm_0, ch'_0, rp'_0))$
    return $(0, w)$
  else
    $w \leftarrow \Sigma_1.\mathsf{Ext}(pp, x_1, (cm_1, ch_1, rp_1),$
        $(cm_1, ch'_1, rp'_1))$
    return $(1, w)$

$\Sigma_{\mathsf{OR}}.\mathsf{ComRec}(pp, (x_0, x_1), ch, (ch_0, ch_1, rp_0, rp_1))$:
  if $ch \neq ch_0 + ch_1$ then
    return $(\perp, \perp)$
  $cm_0 \leftarrow \Sigma_0.\mathsf{ComRec}(pp, x_0, ch_0, rp_0)$
  $cm_1 \leftarrow \Sigma_1.\mathsf{ComRec}(pp, x_1, ch_1, rp_1)$
  if $(cm_0 = \perp) \vee (cm_1 = \perp)$ then
    $(cm_0, cm_1) \leftarrow (\perp, \perp)$
  return $(cm_0, cm_1)$
```

Figure 2: A description of the protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma_0, \Sigma_1]$ introduced in [CDS94].

For the remainder of the paper, we are interested in the situation where a Sigma protocol is combined with itself, *i.e.*, we obtain a new Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ for the relation $\mathcal{R}_{\mathsf{OR}}$. One can show that this protocol inherits many properties of Σ, such as correctness and special honest-verifier zero-knowledge. An important property of Σ_{OR} is that it is witness indistinguishable, meaning that the verifier does not learn which particular witness was used to generate the proof.

2.3 Blind Signatures

In the following we define the syntax of a blind signature scheme. We follow the exposition of Hauck *et al.* [HKLN20], where the interaction between signer and user consists of three moves, since this is sufficient for our setting. Note that one can easily extend the formalism to more than three moves and hence also cover schemes that require more interaction, such as [Rüc10, AEB20a].

Definition 2.4. *A* blind signature scheme *is a tuple of polynomial-time algorithms* $\mathsf{BS} = (\mathsf{BS.PGen}, \mathsf{BS.KGen}, \mathsf{BS.S}, \mathsf{BS.U}, \mathsf{BS.Verify})$ *where:*

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

$\mathsf{BS.KGen}$ *is a PPT* key generation algorithm *that, on input a set of public parameters $pp \in \mathsf{BS.PGen}(1^\lambda)$, returns a public/secret key pair (pk, sk).*

$\mathsf{BS.S}$ *is an interactive algorithm, called* signer, *that consists of two algorithms $\mathsf{BS.S} = (\mathsf{BS.S}_1, \mathsf{BS.S}_2)$, where:*

- *The PPT algorithm $\mathsf{BS.S}_1$ takes as input a set of public parameters pp and a key pair (pk, sk), and returns the signer message s_1 and a state st_S.*
- *The DPT algorithm $\mathsf{BS.S}_2$ takes as input a set of public parameters pp, a key pair (pk, sk), the state information st_S, and the user message u_1, and returns the next signer message s_2.*

$\mathsf{BS.U}$ *is an interactive algorithm, called* user, *that consists of two algorithms $\mathsf{BS.U} = (\mathsf{BS.U}_1, \mathsf{BS.U}_2)$, where:*

- *The PPT algorithm $\mathsf{BS.U}_1$ takes as input a set of public parameters pp, a public key pk, a message $m \in \mathcal{M}(pp)$, and a signer message s_1, and returns a user message u_1 and a state st_U.*
- *The DPT algorithm $\mathsf{BS.U}_2$ takes as input a set of public parameters pp, a public key pk, a message m, a user state st_U, and a signer message s_2, and outputs a signature sig. We let $sig = \perp$ denote failure.*

$\mathsf{BS.Verify}$ *is a DPT* verification algorithm *that, upon receiving a set of public parameters pp, a public key pk, a message m, and a signature sig as input, outputs 1 if the signature is valid and 0 otherwise.*

We require blind signature schemes to satisfy the correctness property. More formally, we say that BS is $\mathsf{corr}_\mathsf{BS}$-*correct* w.r.t. $pp \in \mathsf{BS.PGen}(1^\lambda)$ if, for every key pair $(sk, pk) \in \mathsf{BS.KGen}(pp)$ and every message $m \in \mathcal{M}(pp)$, we have

$$\Pr\left[\begin{matrix}(view, sig) \leftarrow\!\!\$\ \langle \mathsf{BS.S}(pp, pk, sk), \mathsf{BS.U}(pp, pk, m)\rangle, \\ \mathsf{BS.Verify}(pp, pk, m, sig) = 1\end{matrix}\right] \geq 1 - \mathsf{corr}_\mathsf{BS}\,.$$

The security of blind signatures is captured by the notions blindness and one-more unforgeability [JLO97, PS00]. We start with blindness, which prevents a malicious signer to learn information about signed messages.

Definition 2.5. *Let BS be a blind signature scheme, $\lambda \in \mathbb{N}$ and $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that BS is (t, ε)*-blind w.r.t. *pp if, for every adversarial signer S^* running in time at most t and working in modes* find, issue, *and* guess, *we have*

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

where the game $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ is depicted in Figure 3. The scheme BS is called ε-statistically blind *if it is (t, ε)-blind for every t.*

```
$\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp)$:
11: $b \leftarrow\!\!\$ \{0,1\}$
12: $(pk, sk) \leftarrow\!\!\$ \mathsf{BS.KGen}(pp)$
13: $(m_0, m_1, st_{\mathsf{find}}) \leftarrow\!\!\$ \mathsf{S}^*(\mathsf{find}, pp, pk, sk)$
14: $st_{\mathsf{issue}} \leftarrow\!\!\$ \mathsf{S}^{*\langle\cdot,\mathsf{BS.U}(pp,pk,m_b)\rangle^1,\langle\cdot,\mathsf{BS.U}(pp,pk,m_{1-b})\rangle^1}(\mathsf{issue}, st_{\mathsf{find}})$
15: $sig_b \leftarrow \mathsf{BS.U}(pp, pk, m_b)$
16: $sig_{1-b} \leftarrow \mathsf{BS.U}(pp, pk, m_{1-b})$
17: if $(sig_0 = \bot) \vee (sig_1 = \bot)$ then
18:     $(sig_0, sig_1) \leftarrow (\bot, \bot)$
19: $b^* \leftarrow\!\!\$ \mathsf{S}^*(\mathsf{guess}, sig_0, sig_1, st_{\mathsf{issue}})$
20: if $b = b^*$ then
21:     return 1
22: return 0

$\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp)$:
31: $(pk, sk) \leftarrow\!\!\$ \mathsf{BS.KGen}(pp)$
32: $\big((m_1, sig_1), \dots, (m_l, sig_l)\big) \leftarrow\!\!\$ \mathsf{U}^{*\langle\mathsf{BS.S}(pp,pk,sk),\cdot\rangle^\infty}(pp, pk, \cdot)$
33: $k \leftarrow$ number of successful signing invocations
34: if $\big((\forall 1 \le i < j \le l)(m_i \ne m_j)\big) \wedge \big((\forall i \in [l])(\mathsf{BS.Verify}(pp, pk, m_i, sig_i) = 1)\big) \wedge (k+1 = l)$ then
35:     return 1
36: return 0
```

Figure 3: Definition of the experiments $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ and $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$. Here, lines 15 and 16 mean that sig_b and sig_{1-b} are assigned the signatures obtained by BS.U when S^* interacts with its first (*i.e.*, where BS.U wants to sign m_b) and second (*i.e.*, where BS.U wants to sign m_{1-b}) oracle, respectively.

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

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

Definition 2.6. *Let* BS *be a blind signature scheme, $\lambda \in \mathbb{N}$ and $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that* BS *is $(t, q_{\mathsf{Sign}}, \varepsilon)$-*one-more unforgeable w.r.t. *pp if, for every adversarial user U^* running in time at most t and making at most q_{Sign} signing queries, we have*

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

where the game $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$ is depicted in Figure 3.

In the game $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$, U^* interacts with the honest signer BS.S, and has to output $k+1$ valid pairs $(m_1, sig_1), \dots, \big(m_{k+1}, sig_{k+1}\big)$ after at most k successful interactions with BS.S.

2.4 Lattices and Gaussians

Definition 2.7. *Let $k, m \in \mathbb{Z}_{>0}$ with $k \leq m$, and let $\mathbf{B} \in \mathbb{R}^{m \times k}$ be a matrix of rank k. The m-*dimensional lattice $\mathcal{L}$ of rank k generated by $\mathbf{B}$ *is given by $\mathcal{L}(\mathbf{B}) := \left\{ \mathbf{B}\mathbf{x} \mid \mathbf{x} \in \mathbb{Z}^k \right\} \subseteq \mathbb{R}^m$. In the following we will leave the matrix $\mathbf{B}$ implied and omit it from the notation, since it plays no further role.*

Definition 2.8. *Let $\mathcal{L} \subseteq \mathbb{R}^m$ be a lattice, $\sigma \in \mathbb{R}_{>0}$, and $\mathbf{c} \in \mathbb{R}^m$. The* discrete Gaussian distribution over $\mathcal{L}$ with standard deviation σ and center $\mathbf{c}$ *is the probability distribution $D_{\mathcal{L},\sigma,\mathbf{c}}$ which assigns to every $\mathbf{x} \in \mathcal{L}$ the probability of occurrence given by $D_{\mathcal{L},\sigma,\mathbf{c}}(\mathbf{x}) := \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathcal{L})$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp\left(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2}\right)$ and $\rho_{\sigma,\mathbf{c}}(\mathcal{L}) := \sum_{\mathbf{x} \in \mathcal{L}} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. We will omit the subscript $\mathbf{c}$ when $\mathbf{c} = \mathbf{0}$.*

The next lemma, due to Lyubashevsky [Lyu12, Lemma 4.4], gives a tail bound on Gaussian distributed random variables.

Lemma 2.9. *Let $t, \eta \in \mathbb{R}_{>0}$ and $m \in \mathbb{Z}_{>0}$. Then we have:*

(a) $\Pr_{x \leftarrow_\$ D_{\mathbb{Z},\sigma}}[|x| > t\sigma] \leq 2\exp\left(-t^2/2\right)$. *This implies that* $\Pr_{\mathbf{x} \leftarrow_\$ D_{\mathbb{Z}^m,\sigma}}[\|\mathbf{x}\|_\infty > t\sigma] \leq 2m\exp\left(-t^2/2\right)$.

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

Next we recall a special version of the rejection sampling lemma related to the discrete Gaussian distribution [Lyu12, Theorem 4.6].

Lemma 2.10. *Let $T \in \mathbb{R}_{>0}$, and define $V := \{\mathbf{v} \in \mathbb{Z}^m \mid \|\mathbf{v}\| \leq T\}$. Let $\sigma := \alpha T$ for some $\alpha \in \mathbb{R}_{>0}$, and let $h \colon V \to \mathbb{R}$ be a probability distribution. Then there exists a constant $M \in \mathbb{R}_{>0}$ such that $\exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right) \leq M$, and such that the following two algorithms are within statistical distance of at most $2^{-100}/M$:*

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

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

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

We let Rej denote an algorithm that carries out rejection sampling on $\mathbf{z}$, where $\mathbf{z} \leftarrow_\$ D_{\mathbb{Z}^m,\sigma,\mathbf{v}}$, with $\mathbf{v} \in \mathbb{Z}^m$ such that $\|\mathbf{v}\| \leq T$, and $\sigma = \alpha T$. It outputs 1 if $\mathbf{z}$ is accepted and 0 if rejected. By Lemma 2.10, the output 1 indicates that the distribution of $\mathbf{z}$ is within statistical distance of at most $2^{-100}/M$ from the Gaussian distribution $D_{\mathbb{Z}^m,\sigma}$, where $\exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right) \leq M$. The algorithm Rej returns 1 with probability $\approx 1/M$, and hence the expected number of restarts necessary to return 1 is M.

We will use the following result due to Lyubashevsky and Seiler [LS18, Corollary 1.2], which shows that for a suitable choice of the modulus q, all polynomials with small norms are invertible in R_q.

Lemma 2.11. *Let $n \geq p > 1$ be powers of 2 and $q = 2p + 1 \pmod{4p}$ be a prime. Then $X^n + 1$ factors as*

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

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

```
$\mathbf{Exp}_{\mathsf{A}^*}^{\mathsf{MSIS}}(pp)$:
parse $pp = (n, q, k_1, k_2, \beta)$
$\mathbf{A} \leftarrow\!\!\$\; R_q^{k_1 \times k_2}$
$\mathbf{x} \leftarrow\!\!\$\; \mathsf{A}^*(pp, \mathbf{A})$
if $(\mathbf{x} \in R^{k_1+k_2}) \wedge (\mathbf{0} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{x} \pmod q) \wedge$
        $\wedge (\mathbf{x} \neq \mathbf{0}) \wedge (\|\mathbf{x}\| \leq \beta)$ then
  return 1
return 0
```

```
$\mathbf{Exp}_{\mathsf{D}^*}^{\mathsf{D\text{-}MLWE}}(pp)$:
parse $pp = (n, q, k_1, k_2, \sigma, \mathbf{A})$
$b \leftarrow\!\!\$\; \{0, 1\}$
$\mathbf{b} \leftarrow\!\!\$\; R_q^{k_1}$
if $b = 1$ then
  $\mathbf{s} \leftarrow\!\!\$\; D_{\mathbb{Z}^n, \sigma}^{k_1+k_2}$
  $\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
$b^* \leftarrow\!\!\$\; \mathsf{D}^*(pp, \mathbf{b})$
if $b = b^*$ then
  return 1
return 0
```

Figure 4: Definition of the experiments $\mathbf{Exp}_{\mathsf{A}^*}^{\mathsf{MSIS}}$ and $\mathbf{Exp}_{\mathsf{D}^*}^{\mathsf{D\text{-}MLWE}}$.

Finally, we recall the definitions of the two lattice problems relevant to our work, the Module Short Integer Solution (MSIS) and the decisional Module Learning With Errors (D-MLWE) problems. In both cases, we assume that there is an algorithm that, on input 1^λ, generates a set of public parameters pp. Note that D-MLWE can be defined w.r.t. an arbitrary distribution; here we only focus on the case where the witness is sampled from the Gaussian distribution.

Definition 2.12. *Let $pp = (n, q, k_1, k_2, \beta)$, where $n, q, k_1, k_2 \in \mathbb{Z}_{>0}$, and $\beta \in \mathbb{R}_{>0}$. We say that the Hermite normal form of the module short integer solution problem (*MSIS*) is* (t, ε)-hard w.r.t. *pp if, for every algorithm A^* running in time at most t, we have*

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

where the game $\mathbf{Exp}_{\mathsf{A}^*}^{\mathsf{MSIS}}$ is depicted in Figure 4.

Definition 2.13. *Let $pp = (n, q, k_1, k_2, \sigma, \mathbf{A})$, where $n, q, k_1, k_2 \in \mathbb{Z}_{>0}$, $\sigma \in \mathbb{R}_{>0}$, and $\mathbf{A} \in R_q^{k_1 \times k_2}$. We say that the decisional module learning with errors problem (*D-MLWE*) is* (t, ε)-hard w.r.t. *pp if, for every algorithm A^* running in time at most t, we have $\mathbf{Adv}_{\mathsf{A}^*}^{\mathsf{D\text{-}MLWE}}(pp) := 2 \cdot \left|\Pr\left[\mathbf{Exp}_{\mathsf{A}^*}^{\mathsf{D\text{-}MLWE}}(pp) = 1\right] - \frac{1}{2}\right| \leq \varepsilon$, where the game $\mathbf{Exp}_{\mathsf{A}^*}^{\mathsf{D\text{-}MLWE}}$ is depicted in Figure 4.*

The MLWE problem, defined by Langlois and Stehlé [LS15], generalizes the problems 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]. In [LS15], it was shown that both MSIS and MLWE are as hard as the *Shortest Independent Vectors problem* (SIVP) in the worst-case, defined on module lattices. Given a lattice $\mathcal{L}$ of dimension m, the SIVP problem asks to find m linearly independent lattice vectors $(\mathbf{x}_1, \ldots, \mathbf{x}_m)$ such that $\max\{\|\mathbf{x}_1\|, \ldots, \|\mathbf{x}_m\|\}$ is within a factor γ of its optimal value, where $\gamma > 0$ is called the approximation factor. For any $\gamma \in O(1)$, SIVP is NP-hard [BS99], and it is conjectured that there is no polynomial-time (quantum) algorithm that achieves an approximation factor γ bounded by any polynomial in m (see, *e.g.*, [LS15]).

2.5 Partitioning and Permutation

Partitioning and permutation is a technique introduced by Alkeilani Alkadri *et al.* [AEB20a] in the context of lattice-based blind signatures. It allows the user to mask the challenge returned by some cryptographic hash function, so that blindness is satisfied at the first user stage, without having to apply rejection sampling or restart the algorithm $\mathsf{BS.U}_1$. As a result, smaller parameters can be chosen for the scheme, which reduces the signature size and speeds up the signing process. It also allows to use cryptographic hash functions that output challenges from $\mathbb{T}^n_\kappa$ instead of polynomials with unspecified Hamming weight.

Definition 2.14. *We define by* $\mathbb{T} := \left\{(-1)^b \cdot X^i \,\middle|\, b \in \{0,1\}, i \in \mathbb{Z}\right\}$ *the set of signed permutation polynomials which represent a rotation multiplied by a sign.*

We will need the following simple result due to [AEB20a, Lemma 3]:

Lemma 2.15. *The set* $\mathbb{T}$ *defines a group with respect to multiplication in the ring* R*. The inverse of any* $p = (-1)^b \cdot X^i \in \mathbb{T}$ *is given by* $p^{-1} = (-1)^{1-b} \cdot X^{n-i} \in \mathbb{T}$.

The partitioning and permutation technique works as follows. If $c \in \mathbb{T}^n_\kappa$ is a challenge, the user splits c into shares $c_1, \dots, c_\kappa \in \mathbb{T}$ such that $c = \sum_{j=1}^{\kappa} c_j$ and each c_j is the jth non-zero entry of c at exactly the same position. Then, for each $j \in [\kappa]$, c_j is masked by computing $c_j^* = p_j^{-1} \cdot c_j$, where the p_j are chosen uniformly at random from $\mathbb{T}$. The masked challenge sent to the signer is then $\mathbf{c}^* = (c_1^*, \dots, c_\kappa^*) \in \mathbb{T}^\kappa$.

The following lemma, due to [AEB20a, Lemma 4], shows that the partitions c_j^* are independently distributed from c_j, for all $j \in [\kappa]$.

Lemma 2.16. *Let* $c \in \mathbb{T}^n_\kappa$ *and* $c_1, \dots, c_\kappa$ *be a partition of* c *such that* $c = \sum_{j=1}^{\kappa} c_j$ *and each* c_j *contains the* j*th non-zero entry of* c *at the* j*th position. Furthermore, let* $c_j^* = p_j^{-1} c_j$ *for random signed rotations* $p_1, \dots, p_\kappa \in \mathbb{T}$*. Then,* $c_j^*, c_j \in \mathbb{T}$*, and we have:*

$$\Pr_{p_j \leftarrow_\$ \mathbb{T}}\left[(c_1^*, \dots, c_\kappa^*) = \left(p_1^{-1} c_1, \dots, p_\kappa^{-1} c_\kappa\right) \,\middle|\, c\right] = \Pr_{\substack{p_j \leftarrow_\$ \mathbb{T} \\ c \leftarrow_\$ \mathbb{T}^n_\kappa}}\left[(c_1^*, \dots, c_\kappa^*) = \left(p_1^{-1} c_1, \dots, p_\kappa^{-1} c_\kappa\right)\right] = (2n)^{-\kappa}.$$

In order to construct lattice-based OR-proofs, we will directly use the abelian group $\mathbb{T}^\kappa$ as the challenge space rather than the set $\mathbb{T}^n_\kappa$, since it does not define a group structure.

2.6 Trees of Commitments

Trees of commitments is a technique introduced by Alkeilani Alkadri *et al.* [AEB20b] in order to reduce or even remove the number of restarts inherent in lattice-based (interactive) protocols when masking secrets using rejection sampling. When masking a secret (or secret-related) term x in lattice-based cryptographic protocols, rejection sampling is usually applied on the sum $z = y + x$ in order to make sure that z is independently distributed from x, where y is a masking term that has been generated and committed to in an earlier phase of the protocol. If rejection sampling accepts, *i.e.*, if $\mathsf{Rej}(z) = 1$, then x and z are independently distributed and z already hides x. Otherwise, z may leak information about x, and hence the protocol is restarted with a fresh y in order to retain the security of the protocol. This affects the efficiency of interactive protocols with multiple rejection sampling procedures in a negative way. More precisely, assume that a protocol involves N rejection sampling procedures and each one is repeated M_i times on average, where $i = 1, \dots, N$. Then, the average total number of restarts is $\prod_{i=1}^{N} M_i$, which is multiplicative in N. This number can be reduced or even removed by using trees of commitments.

A tree of commitments is an (unbalanced) binary hash tree, whose leaves are the hash values of commitments to ℓ masking terms y_j, $0 \leq j \leq \ell - 1$, that are generated at once in a protocol run. These commitments are then aggregated to one, the root of the tree. In a subsequent phase of the protocol only the sum $z = y_k + x$ together with *auth* are revealed, where y_k is the first masking term for which rejection sampling accepts and *auth* is the authentication path of the index k, $0 \leq k < \ell$.

Let $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ be a cryptographic hash function, where $\ell_\mathsf{F} \geq 2\lambda$ for F to be collision resistant. We consider the following algorithms:

$\mathsf{HashTree}$: 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.*, $(root, tree) \leftarrow \mathsf{HashTree}(v_0, \ldots, v_{\ell-1})$, 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.*, $auth \leftarrow \mathsf{BuildAuth}(k, tree, h)$.

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

2.7 Forking Lemma

In this section we recall the forking lemma, which constitutes a crucial tool for proving the security of cryptographic schemes in the random oracle model.

The lemma considers the situation where an algorithm A runs on input an instance *ist* and random values $h_1, \ldots, h_q$ belonging to a finite set $\mathcal{C}$, and returns a pair (idx, out), where $0 \leq idx \leq q$ and the output *out* is related to h_{idx}. Here, we convene that $idx = 0$ means that A has failed to compute an output related to any of the values $h_1, \ldots, h_q$. At a high level, the forking lemma gives a lower bound on the probability that A, if run twice on the same instance *ist* and randomness, but partially different values $h_1, \ldots, h_q$, will return answers related to values h_{idx} and h'_{idx} with the same index *idx*, where h_{idx} and h'_{idx} were sampled in the first and second run, respectively.

The forking lemma was first introduced by Pointcheval and Stern [PS00], who used a specific version of this result in their security proofs of certain digital signature and blind signature schemes. The lemma was later generalized by Bellare and Neven [BN06] and reformulated as a purely probabilistic statement. We remark that, in recent work, Hauck *et al.* [HKL19] proved an even more general version of the forking lemma, which gives a lower bound on the forking probability even if the outputs of A are required to satisfy some additional (probabilistic) constraints.

In this paper, we will need a minor modified version of the general forking lemma [BN06], where the algorithm A being forked returns a second index as part of the side output. In this version of the lemma, forking succeeds only if both the index identifying the random oracle query that the output is related to and the index in the side output coincide across both runs. The proof of this version of the lemma is virtually identical to the standard proof by Bellare and Neven [BN06]. We provide this proof for the sake of completeness.

Lemma 2.17. *Let $q, \ell \in \mathbb{Z}_{\geq 1}$, let IG be a probabilistic algorithm and $\mathcal{C}$ be a finite set of size $|\mathcal{C}| \geq 2$. Suppose that A is a probabilistic algorithm that, on input ist $\in \mathsf{IG}$ and $h_1, \ldots, h_q \in \mathcal{C}$, outputs a tuple (idx_1, idx_2, out), with $0 \leq idx_1 \leq q$ and $0 \leq idx_2 < \ell$. Define the* accepting probability *and the* forking probability *of A as $acc = \Pr\left[\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IG},\mathcal{C},\mathsf{A}} = 1\right]$ and $frk = \Pr\left[\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IG},\mathcal{C},\mathsf{A}} = 1\right]$, respectively, where $\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$ and $\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$ are given in Figure 5. Then*

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

Proof. For any output *ist* returned by IG with non-zero probability, define

$$acc(ist) := \Pr\left[\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IG},\mathcal{C},\mathsf{A}} = 1 \,\middle|\, \mathsf{IG} = ist\right], \qquad frk(ist) := \Pr\left[\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IG},\mathcal{C},\mathsf{A}} = 1 \,\middle|\, \mathsf{IG} = ist\right].$$

```
$\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$:
    $ist \leftarrow\!\$\ \mathsf{IG}$
    $h_1,\dots,h_q \leftarrow\!\$\ \mathcal{C}$
    $(idx_1, idx_2, out) \leftarrow\!\$\ \mathsf{A}(ist, h_1,\dots,h_q)$
    if $1 \le idx_1 \le q$ then
        return 1
    return 0

$\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$:
    $ist \leftarrow\!\$\ \mathsf{IG}$
    $(b, out, out') \leftarrow\!\$\ \mathsf{Frk}_{\mathcal{C},\mathsf{A}}(ist)$
    return $b$

$\mathsf{Frk}_{\mathcal{C},\mathsf{A}}(ist)$:
    $r \leftarrow\!\$\ \mathcal{RS}_{\mathsf{A}}$
    $h_1,\dots,h_q \leftarrow\!\$\ \mathcal{C}$
    $(idx_1, idx_2, out) \leftarrow \mathsf{A}(ist, h_1,\dots,h_q; r)$
    if $idx_1 = 0$ then
        return $(0,\bot,\bot)$
    $h'_{idx_1},\dots,h'_q \leftarrow\!\$\ \mathcal{C}$
    $(idx'_1, idx'_2, out') \leftarrow \mathsf{A}(ist, h_1,\dots,h_{idx_1-1}, h'_{idx_1},\dots,h'_q; r)$
    if $(idx_1 = idx'_1) \wedge (idx_2 = idx'_2) \wedge (h_{idx_1} \neq h'_{idx_1})$ then
        return $(1, out, out')$
    return $(0,\bot,\bot)$
```

Figure 5: Definition of the experiments $\mathbf{Exp}^{\mathrm{Acc}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$ and $\mathbf{Exp}^{\mathrm{Frk}}_{\mathsf{IG},\mathcal{C},\mathsf{A}}$, as well as the forking algorithm $\mathsf{Frk}_{\mathcal{C},\mathsf{A}}$.

We show that the inequality holds for every such *ist*. The statement then follows by averaging over all possible *ist*. Observe that

$$\begin{aligned}
frk(ist) &= \Pr\big[(idx_1 \ge 1) \wedge \big(idx_1 = idx'_1\big) \wedge \big(idx_2 = idx'_2\big) \wedge \big(h_{idx_1} \neq h'_{idx_1}\big)\big] \\
&\ge \Pr\big[(idx_1 \ge 1) \wedge \big(idx_1 = idx'_1\big) \wedge \big(idx_2 = idx'_2\big)\big] - \Pr\big[(idx_1 \ge 1) \wedge \big(h_{idx_1} = h'_{idx_1}\big)\big] \\
&= \Pr\big[(idx_1 \ge 1) \wedge \big(idx_1 = idx'_1\big) \wedge \big(idx_2 = idx'_2\big)\big] - \frac{\Pr[idx_1 \ge 1]}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \Pr\big[(idx_1 = i) \wedge \big(idx'_1 = i\big) \wedge (idx_2 = j) \wedge \big(idx'_2 = j\big)\big] - \frac{acc(ist)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \sum_{(\bar{r},\bar{h}_1,\dots,\bar{h}_{i-1})} \Pr\left[\begin{matrix}(idx_1 = i) \wedge \big(idx'_1 = i\big) \wedge \\ \wedge\,(idx_2 = j) \wedge \big(idx'_2 = j\big)\end{matrix} \,\middle|\, \begin{matrix}(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \\ \wedge \dots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\end{matrix}\right] \\
&\qquad\qquad \cdot \Pr\big[(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \dots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\big] - \frac{acc(ist)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \sum_{(\bar{r},\bar{h}_1,\dots,\bar{h}_{i-1})} \Pr\left[(idx_1 = i) \wedge (idx_2 = j) \,\middle|\, \begin{matrix}(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \\ \wedge \dots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\end{matrix}\right]^2 \\
&\qquad\qquad \cdot \Pr\big[(r = \bar{r}) \wedge \big(h_1 = \bar{h}_1\big) \wedge \dots \wedge \big(h_{i-1} = \bar{h}_{i-1}\big)\big] - \frac{acc(ist)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \mathop{\mathrm{E}}_{(r,h_1,\dots,h_{i-1})}\Big[\Pr[(idx_1 = i) \wedge (idx_2 = j) \mid r \wedge h_1 \wedge \dots \wedge h_{i-1}]^2\Big] - \frac{acc(ist)}{|\mathcal{C}|} \\
&\ge \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \mathop{\mathrm{E}}_{(r,h_1,\dots,h_{i-1})}[\Pr[(idx_1 = i) \wedge (idx_2 = j) \mid r \wedge h_1 \wedge \dots \wedge h_{i-1}]]^2 - \frac{acc(ist)}{|\mathcal{C}|} \\
&= \sum_{i=1}^{q}\sum_{j=0}^{\ell-1} \Pr[(idx_1 = i) \wedge (idx_2 = j)]^2 - \frac{acc(ist)}{|\mathcal{C}|}
\end{aligned}$$

$$\geq \frac{1}{q\ell}\left(\sum_{i=1}^{q}\sum_{j=0}^{\ell-1}\Pr[(idx_1 = i)\wedge(idx_2 = j)]\right)^2 - \frac{acc(ist)}{|\mathcal{C}|}$$
$$\geq \frac{acc(ist)^2}{q\ell} - \frac{acc(ist)}{|\mathcal{C}|}\,.$$

The critical (in)equalities given above follow from the conditional independence given $r, h_1, \ldots, h_{i-1}$ and by applying Jensen's inequality twice. $\square$

3 BlindOR: A New Blind Signature Scheme

In this section we present BlindOR: our new lattice-based construction of blind signatures from OR-proofs. In Section 3.1, we present the Sigma protocol underlying BlindOR. Then, in Section 3.2 we present our scheme BlindOR, and analyze its security properties of blindness and one-more unforgeability in Section 3.3. Finally, in Section 3.4 we propose concrete parameters of the scheme targeting 128 bits of security.

3.1 Sigma Protocol

In this section we present the lattice-based Sigma protocol that is used as a building block in BlindOR.

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

BlindOR is built on a variant of the Sigma protocol introduced in [Lyu09], so we briefly recall this construction before presenting our modified protocol. Given a public matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$ and an instance $\mathbf{b} \in R_q^{k_1}$, the prover holds a witness $(\mathbf{s}, 1) \in B_1' \times B_2 \subseteq R^{k_1+k_2} \times R_q$ such that $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod{q}$. An execution of the protocol allows him to prove knowledge of a witness $(\bar{\mathbf{s}}, \bar{c}) \in B_1 \times B_2$, with $B_1' \subseteq B_1 \subseteq R^{k_1+k_2}$, such that $\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod{q}$. The commitment message is given by $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y} \pmod{q}$, where $\mathbf{y}$ is a masking vector with small entries. Upon receiving a challenge $c \in \mathbb{T}_\kappa^n$, the response is computed as $\mathbf{z} = \mathbf{y} + \mathbf{s}c$, and is sent to the verifier only if it follows a specified distribution, typically the Gaussian distribution $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$ for some $\sigma > 0$ or the uniform distribution over a small subset of $R^{k_1+k_2}$. This ensures that $\mathbf{y}$ masks the secret-related term $\mathbf{s}c$ and that $\mathbf{z}$ is independently distributed from $\mathbf{s}$. If $\mathbf{z}$ does not follow the target distribution, the prover restarts the protocol with a fresh $\mathbf{y}$. The verifier accepts if $\mathbf{v} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod{q}$ and if $\|\mathbf{z}\|_p$ is bounded by some predefined value. Note that $p \in \{2, \infty\}$, depending on the distribution of $\mathbf{z}$.

We now turn our attention to our modified Sigma protocol, built on top of the protocol recalled above, and start by introducing the relation $\mathcal{R}$ it is associated to. The algorithm $\mathcal{R}.\mathsf{PGen}$ generates a set of public parameters of the form

$$pp = (1^\lambda, n, k_1, k_2, q, \omega, \kappa, \sigma', \sigma^*, S, B_s, B_{z^*}, \delta^*, \mathbf{A}) \leftarrow\!\!\$\; \mathcal{R}.\mathsf{PGen}(1^\lambda)\,,$$

subject to the constraints given in Table 2, where the matrix $\mathbf{A} \in R_q^{k_1 \times k_2}$ follows the uniform distribution[1]. In Table 3 we propose a concrete tuple of such parameters targeting 128 bits of security. The relation set is then given by

$$\mathcal{R}.\mathsf{RSet}(pp) := \left\{ \left(\mathbf{b}, (\bar{\mathbf{s}}, \bar{c})\right) \in R_q^{k_1} \times (R^{k_1+k_2} \times \overline{\mathcal{C}}) \;\middle|\; (\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod q) \wedge (\|\bar{\mathbf{s}}\| \leq 2B_{z^*}) \right\}, \tag{1}$$

where

$$\overline{\mathcal{C}} = \left\{ c - c' \mid c, c' \in \mathbb{T},\ c \neq c' \right\},$$

and the instance generator is given in Figure 6. The actual witness the prover possesses is of the form $(\mathbf{s}, 1)$, where $\|\mathbf{s}\| \leq B_s < B_{z^*}$ and $\mathbf{b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$. The challenge space of the protocol is $\mathbb{T}^\kappa$, and its other algorithms are given in Figures 6 and 7.

At a high level, the protocol can be seen as a generalized version of the one given in [Lyu09] and briefly recalled above. In particular, it is optimized to work for BlindOR. Rather than computing only one commitment to a masking vector in $\Sigma.\mathsf{P}_1$, the prover computes commitments to $\omega \geq 1$ such vectors and sends them to the verifier all at once. Choosing $\omega > 1$ allows to reduce the number of restarts, since the chance of masking the secret-related term without restarting the protocol is increased. More concretely, increasing ω allows to compute a response such that there is no need to trigger a protocol restart with some fixed probability. The masking vectors are chosen according to the Gaussian distribution $D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$. Upon receiving the challenge $\mathbf{c} \in \mathbb{T}^\kappa$, the prover sends the first response $\mathbf{z}$ for which rejection sampling accepts, *i.e.*, for the masking vector $\mathbf{y}^{(i)}$ such that $\mathsf{Rej}(pp, \mathbf{z}) = 1$ and i is chosen from the uniform distribution over the set $T \subseteq \{0, \ldots, \omega - 1\}$. The random choice of the index i ensures that the simulator $\Sigma.\mathsf{Sim}$ returns $(\mathbf{v}, \mathbf{z}) \neq (\perp, \perp)$ with the same probability as the prover. Note that each of the ω commitments consists of κ components, where κ defines the challenge space $\mathbb{T}^\kappa$. This allows to use the partitioning and permutation technique in BlindOR. To verify a transcript $(\mathbf{v}, \mathbf{c}, \mathbf{z})$, the verifier first finds out which of the ω commitments is related to the response. The index i of the corresponding commitment is part of the verifier's output.

Theorem 3.1. *Given the parameters in Table 2, the protocol depicted in Figures 6 and 7 is a Sigma protocol for relation $\mathcal{R}$ given in Equation (1).*

Proof. Note that the prover generates ω commitments in one protocol run and sends them to the verifier at once. By Lemma 2.10, the prover responds with probability $(1 - 2^{-100})/S$ after an average number of S restarts, where $S = \exp\left(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\right)$ and $\omega = 1$. Therefore, when $\omega \geq 1$, the prover sends a response $\mathbf{z} \neq \perp$ with probability $1 - \left(1 - \frac{1-2^{-100}}{S}\right)^\omega$ after at most $M = 1/(1 - \delta^*)$ restarts, where ω is chosen such that $\left(1 - \frac{1-2^{-100}}{S}\right)^\omega \leq \delta^*$. This means that the prover returns a response $\mathbf{z} = \perp$ with probability at most δ^*, and after $M = 1/(1 - \delta^*)$ restarts, the prover returns a response $\mathbf{z} \neq \perp$.

Given an honestly created transcript $(\mathbf{v}, \mathbf{c}, \mathbf{z})$, its response $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$ is distributed as $D_{\mathbb{Z}^n,\sigma^*}^{(k_1+k_2)\kappa}$. By Lemma 2.9, $\|\mathbf{z}\| > B_{z^*}$ with probability at most

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

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

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

[1]In practice, the matrix $\mathbf{A}$ is deterministically generated from a uniformly random seed using an extendable output function such as SHAKE.

```
$\mathcal{R}.\mathsf{Gen}(pp, b)$:
if $b = 0$ then
    $\mathbf{b} \leftarrow_\$ R_q^{k_1}$
    return $\mathbf{b}$
if $b = 1$ then
    repeat $\mathbf{s} \leftarrow_\$ D_{\mathbb{Z}^n,\sigma'}^{k_1+k_2}$ until $\|\mathbf{s}\| \leq B_s$
    $\mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
    return $(\mathbf{b}, \mathbf{s})$
```

```
$\Sigma.\mathsf{P}_1(pp, \mathbf{b}, \mathbf{s})$:
for $i = 0$ to $\omega - 1$ do
    for $j = 1$ to $\kappa$ do
        $\mathbf{y}_j \leftarrow_\$ D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$
        $\mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
    $\mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
    $\mathbf{y}^{(i)} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
$\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
$st_{\Sigma.\mathsf{P}} \leftarrow (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\omega-1)})$
return $(\mathbf{v}, st_{\Sigma.\mathsf{P}})$
```

```
$\Sigma.\mathsf{V}_1(pp, \mathbf{b}, \mathbf{v})$:
$\mathbf{c} = (c_1, \ldots, c_\kappa) \leftarrow_\$ \mathbb{T}^\kappa$
$st_{\Sigma.\mathsf{V}} \leftarrow (\mathbf{v}, \mathbf{c})$
return $(\mathbf{c}, st_{\Sigma.\mathsf{V}})$
```

```
$\Sigma.\mathsf{Ext}(pp, \mathbf{b}, (\mathbf{v}, \mathbf{c}, \mathbf{z}), (\mathbf{v}, \mathbf{c}', \mathbf{z}'))$:
parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
parse $\mathbf{c}' = (c'_1, \ldots, c'_\kappa)$
parse $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
parse $\mathbf{z}' = (\mathbf{z}'_1, \ldots, \mathbf{z}'_\kappa)$
Pick $j \in [\kappa]$ such that $c_j \neq c'_j$
$\bar{\mathbf{s}} \leftarrow \mathbf{z}_j - \mathbf{z}'_j$
$\bar{c} \leftarrow c_j - c'_j$
return $(\bar{\mathbf{s}}, \bar{c})$
```

```
$\Sigma.\mathsf{P}_2(pp, \mathbf{b}, \mathbf{s}, st_{\Sigma.\mathsf{P}}, \mathbf{c})$:
parse $st_{\Sigma.\mathsf{P}} = (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\omega-1)})$
parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
$T := \{0, \ldots, \omega - 1\}$
while $T \neq \emptyset$ do
    $i \leftarrow_\$ T$
    $T \leftarrow T \setminus \{i\}$
    parse $\mathbf{y}^{(i)} = (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
    for $j = 1$ to $\kappa$ do
        $\mathbf{z}_j \leftarrow \mathbf{y}_j + \mathbf{s}c_j$
    $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
    if $\mathsf{Rej}(pp, \mathbf{z}) = 1$ then
        return $\mathbf{z}$
return $\perp$
```

```
$\Sigma.\mathsf{V}_2(pp, \mathbf{b}, st_{\Sigma.\mathsf{V}}, \mathbf{z})$:
if $\|\mathbf{z}\| > B_{z^*}$ then
    return $(0, -1)$
parse $st_{\Sigma.\mathsf{V}} = (\mathbf{v}, \mathbf{c})$
parse $\mathbf{v} = (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
parse $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
for $j = 1$ to $\kappa$ do
    $\mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
for $i = 0$ to $\omega - 1$ do
    $int \leftarrow 0$
    parse $\mathbf{v}^{(i)} = (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
    for $j = 1$ to $\kappa$ do
        if $\mathbf{w}_j = \mathbf{v}_j$ then
            $int = int + 1$
    if $int = \kappa$ then
        return $(1, i)$
return $(0, -1)$
```

Figure 6: The lattice-based Sigma protocol underlying BlindOR. The commitment recovering algorithm and simulator are given in Figure 7. Note that the prover restarts the protocol if $\Sigma.\mathsf{P}_2$ returns $\perp$.

```
Σ.Sim(pp, b, c):
    return $(\bot, \bot)$ with probability $\delta^*$
    parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
    $i \leftarrow\$ \{0, \ldots, \omega - 1\}$
    for $j = 1$ to $\kappa$ do
        $\mathbf{z}_j \leftarrow\$ D^{k_1+k_2}_{\mathbb{Z}^n, \sigma^*}$
        $\mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
    $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
    $\mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
    for $k = 0$ to $\omega - 1$ do
        if $k = i$ then
            continue
        for $j = 1$ to $\kappa$ do
            $\mathbf{y}_j \leftarrow\$ D^{k_1+k_2}_{\mathbb{Z}^n, \sigma^*}$
            $\mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{y}_j \pmod q$
        $\mathbf{v}^{(k)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
    $\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
    return $(\mathbf{v}, \mathbf{z})$

Σ.ComRec(pp, b, c, z):
    if $\|\mathbf{z}\| > B_{z^*}$ then
        return $\bot$
    parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
    parse $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
    for $j = 1$ to $\kappa$ do
        $\mathbf{v}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
    $\mathbf{v}^{(0)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
    for $i = 1$ to $\omega - 1$ do
        $\mathbf{v}^{(i)} \leftarrow (\mathbf{0}, \ldots, \mathbf{0}) \in (R_q^{k_1})^\kappa$
    $\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
    return $\mathbf{v}$
```

Figure 7: The commitment recovering algorithm and simulator of the lattice-based Sigma protocol given in Figure 6.

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

Next, we show that $\Sigma.\mathsf{Ext}$ returns a witness in $\mathcal{R}$. Assume we have two correctly verified transcripts $(\mathbf{v}, \mathbf{c}, \mathbf{z})$ and $(\mathbf{v}, \mathbf{c}', \mathbf{z}')$, where $\mathbf{c} \neq \mathbf{c}'$ and $\Sigma.\mathsf{V}_2$ returns the same output $(1, i)$ when given both transcripts as input, where $i \in \{0, \ldots, \omega - 1\}$. This means that, for all $j \in [\kappa]$, we have

$$(\|\mathbf{z}\| \leq B_{z^*}) \wedge \left(\left\|\mathbf{z}'\right\| \leq B_{z^*}\right) \wedge \left([\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b}c_j = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_j - \mathbf{b}c'_j \pmod q\right).$$

Since $\mathbf{c} \neq \mathbf{c}'$, there is at least one index $j \in [\kappa]$ such that $c_j \neq c'_j$. We set $\bar{\mathbf{s}} = \mathbf{z}_j - \mathbf{z}'_j$ and $\bar{c} = c_j - c'_j$ to obtain $\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod q$. Note that $\left\|\mathbf{z}_j - \mathbf{z}'_j\right\| \leq 2B_{z^*}$ and $\bar{c} \in \overline{\mathcal{C}}$. Moreover, by Lemma 2.11, $\bar{c}$ is invertible in R_q if q satisfies $\|\bar{c}\| = 2 < q^{1/p}$, where $q = 2p + 1 \pmod{4p}$ and p is a power of 2 such that $n \geq p > 1$. This implies that $\mathbf{b}\bar{c} \neq \mathbf{0} \pmod q$, and we can extract a witness $(\bar{\mathbf{s}}, \bar{c})$ in $\mathcal{R}$ as described in Figure 6.

We observe that the protocol is also special honest-verifier zero-knowledge. By Lemma 2.10, the distribution of $\mathbf{z}$ does not depend on the witness. Furthermore, real and simulated transcripts are statistically indistinguishable and within statistical distance of at most $2^{-100}/M$. Finally, the witness indistinguishability property directly follows from the base Sigma protocol due to [Lyu09]. □

We remark that when constructing the Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ as depicted in Figure 2, where Σ is the protocol introduced above, we must consider the group operation defined on the challenge space $\mathbb{T}^\kappa$. More precisely, $\Sigma_{\mathsf{OR}}.\mathsf{P}_1$ samples $\mathbf{c}_{1-d} = (c_{1,1-d}, \ldots, c_{\kappa,1-d}) \leftarrow\$ \mathbb{T}^\kappa$ and then runs $\Sigma_{1-d}.\mathsf{Sim}$ on $\mathbf{c}_{1-d}$. Upon receiving a challenge $\mathbf{c} = (c_1, \ldots, c_\kappa)$ from $\Sigma_{\mathsf{OR}}.\mathsf{V}_1$, $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ computes $\mathbf{c}_d = (c_1 c^{-1}_{1,1-d}, \ldots, c_\kappa c^{-1}_{\kappa,1-d})$ and runs $\Sigma_d.\mathsf{P}_2$ on $\mathbf{c}_d$. Therefore, we have $\mathbf{c} = \mathbf{c}_d \cdot \mathbf{c}_{1-d} = (c_{1,d}c_{1,1-d}, \ldots, c_{\kappa,d}c_{\kappa,1-d})$.

3.2 Description of BlindOR

In this section we introduce our blind signature scheme $\mathsf{BlindOR}$.

Let BS be a blind signature scheme as defined in Section 2.3. Recall how signing and verification of such a scheme works. The signer computes and sends a commitment cm^* to the user. The user blinds cm^* to obtain a blind commitment cm and computes a challenge ch, which is generated by evaluating a hash function H on input (cm, m), *i.e.*, $ch = \mathsf{H}(cm, m)$ with m being a message. After that, the user unblinds ch to obtain a challenge ch^* and sends it to the signer. The signer computes a response rp^* and sends it back to the user. Finally, the user blinds rp^* to obtain a blind response rp and outputs $sig = (ch, rp)$. Note that we call ch a blind challenge, since it is a part of the blind signature. Verifying the validity of sig is established by computing a commitment cm corresponding to ch and rp, and then checking if ch matches $\mathsf{H}(cm, m)$. Observe that while the steps carried out by the signer are actually what a prover in a Sigma protocol does when proving the possession of a witness for a statement, the steps performed by the user consist of blinding the transcript (cm^*, ch^*, rp^*) during interaction. In $\mathsf{BlindOR}$, we capture these blinding steps by algorithms $\mathsf{Com}, \mathsf{Cha}$, and Rsp, which we describe next.

For the remainder of this section we let Σ be the Sigma protocol depicted in Figures 6 and 7. Furthermore, let $h = \lceil \log(\omega\ell) \rceil$ and define the following bijective mapping:

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

The mapping IntIdx converts the pair (i, k) into a unique positive integer. This is used in $\mathsf{BlindOR}$ to build authentication paths via the algorithm $\mathsf{BuildAuth}$. Let pp be a set of public parameters for $\mathsf{BlindOR}$ and $x = \mathbf{b} \in R_q^{k_1}$ be an instance for $\mathcal{R}$. We define the following algorithms, which are formally described in Figure 8:

Com is a PPT algorithm that, on input pp, the statement x, and a commitment $cm^* = \mathbf{v}^*$ generated by $\Sigma.\mathsf{P}_1$, returns a blind commitment $cm = root$ and a state $(\mathbf{p}, tree, \mathbf{e})$.

Cha is a DPT algorithm that, on input pp, a randomness $\mathbf{p} \in \mathbb{T}^\kappa$, a challenge $ch^* = \mathbf{c}^* \in \mathbb{T}^\kappa$, and an auxiliary bit $b \in \{0, 1\}$, returns a challenge $ch = \mathbf{c} \in \mathbb{T}^\kappa$. Observe that b determines if $\mathbf{c}^*$ will be blinded using $\mathbf{p}$ or its inverse with respect to the group operation defined on $\mathbb{T}^\kappa$.

Rsp is a DPT algorithm that, on input pp, a state $(\mathbf{p}, tree, \mathbf{e})$, a response $rp^* = \mathbf{z}^*$ generated by $\Sigma.\mathsf{P}_2$, and an integer $i \in \{0, \ldots, \omega - 1\}$, returns a blind response $rp = (\mathbf{z}, auth)$, where $rp = (\perp, \perp)$ is possible.

Rec is a DPT algorithm that, on input pp, the statement x, a challenge ch, and a response rp, returns a commitment cm, where $cm = \perp$ is possible.

Note that the blinding algorithms depicted in Figure 8 can be seen as a generalized version of the blinding steps implicitly presented in the lattice-based blind signature scheme BLAZE^+ [AEB20b]. Unlike BLAZE^+, the algorithms shown in Figure 8 are defined for lattices over modules rather than over rings. This complies with the module structure of Σ and allows for more flexibility when choosing concrete parameters. Furthermore, these blinding algorithms employ the partitioning and permutation technique, which allows to use the abelian group $\mathbb{T}^\kappa$ as a challenge space rather than the set $\mathbb{T}^n_\kappa$, which does not have a group structure. Moreover, the algorithm Com blinds ω commitments $\mathbf{v}^{*(0)}, \ldots, \mathbf{v}^{*(\omega-1)}$ rather than only one commitment generated by $\Sigma.\mathsf{P}_1$. More precisely, the trees of commitments technique employed in BLAZE^+ is extended to further include ω commitments created by the prover. These ω commitments are then combined with ℓ commitments generated within Com to compute the root related to a tree of $\omega\ell$ commitments. We require ℓ to be chosen large enough so that Rsp returns a blind response $(\mathbf{z}, auth) = (\perp, \perp)$ with probability close to zero, *e.g.*, 2^{-80}. This is crucial for $\mathsf{BlindOR}$ since otherwise, we would need an extra move between the signer and user, which would allow the user to request a restart of the signing protocol in case the

```
Com(pp, b, v*):
11:  parse $\mathbf{v}^* = (\mathbf{v}^{*(0)}, \ldots, \mathbf{v}^{*(\omega-1)})$
12:  $\mathbf{p} = (p_1, \ldots, p_\kappa) \leftarrow\!\!\$\ \mathbb{T}^\kappa$
13:  for $i = 0$ to $\omega - 1$ do
14:    parse $\mathbf{v}^{*(i)} = (\mathbf{v}_1^*, \ldots, \mathbf{v}_\kappa^*)$
15:    for $k = 0$ to $\ell - 1$ do
16:      for $j = 1$ to $\kappa$ do
17:        $\mathbf{e}_j^{(k)} \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$
18:        $\mathbf{v}_j^{(i,k)} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}_j^{(k)} + \mathbf{v}_j^* p_j \pmod q$
19:      $\mathbf{e}^{(k)} \leftarrow (\mathbf{e}_1^{(k)}, \ldots, \mathbf{e}_\kappa^{(k)})$
20:      $\mathbf{v}^{(i,k)} \leftarrow (\mathbf{v}_1^{(i,k)}, \ldots, \mathbf{v}_\kappa^{(i,k)})$
21:  $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{v}^{(0,0)}, \ldots, \mathbf{v}^{(\omega-1,\ell-1)})$
22:  $\mathbf{e} \leftarrow (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
23:  return $(root, (\mathbf{p}, tree, \mathbf{e}))$

Cha(pp, p, c*, b):
31:  parse $\mathbf{p} = (p_1, \ldots, p_\kappa)$
32:  parse $\mathbf{c}^* = (c_1^*, \ldots, c_\kappa^*)$
33:  if $b = 0$ then
34:    $\mathbf{c} \leftarrow (c_1^* p_1^{-1}, \ldots, c_\kappa^* p_\kappa^{-1})$
35:  else
36:    $\mathbf{c} \leftarrow (c_1^* p_1, \ldots, c_\kappa^* p_\kappa)$
37:  return $\mathbf{c}$

Rsp(pp, (p, tree, e), z*, i):
41:  $(\mathbf{z}, k) \leftarrow \mathsf{IterateRej}(pp, \mathbf{e}, \mathbf{z}^*, \mathbf{p})$
42:  if $(\mathbf{z}, k) = (\bot, \bot)$ then
43:    return $(\bot, \bot)$
44:  $auth \leftarrow \mathsf{BuildAuth}(\mathsf{IntIdx}(i, k), tree, h)$
45:  return $(\mathbf{z}, auth)$

IterateRej(pp, e, z*, p):
51:  parse $\mathbf{e} = (\mathbf{e}^{(0)}, \ldots, \mathbf{e}^{(\ell-1)})$
52:  parse $\mathbf{z}^* = (\mathbf{z}_1^*, \ldots, \mathbf{z}_\kappa^*)$
53:  parse $\mathbf{p} = (p_1, \ldots, p_\kappa)$
54:  for $k = 0$ to $\ell - 1$ do
55:    parse $\mathbf{e}^{(k)} = (\mathbf{e}_1^{(k)}, \ldots, \mathbf{e}_\kappa^{(k)})$
56:    for $j = 1$ to $\kappa$ do
57:      $\mathbf{z}_j \leftarrow \mathbf{e}_j^{(k)} + \mathbf{z}_j^* p_j$
58:    $\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
59:    if $\mathsf{Rej}(pp, \mathbf{z}) = 1$ then
60:      return $(\mathbf{z}, k)$
61:  return $(\bot, \bot)$

Rec(pp, b, c, z, auth):
71:  if $\|\mathbf{z}\| > B_z$ then
72:    return $\bot$
73:  parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
74:  parse $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
75:  for $j = 1$ to $\kappa$ do
76:    $\mathbf{w}_j \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_j - \mathbf{b} c_j \pmod q$
77:  $\mathbf{w} \leftarrow (\mathbf{w}_1, \ldots, \mathbf{w}_\kappa)$
78:  $root \leftarrow \mathsf{RootCalc}(\mathbf{w}, auth)$
79:  return $root$
```

Figure 8: A formal description of algorithms $\mathsf{Com}, \mathsf{Cha}, \mathsf{Rsp}$, and Rec.

algorithm $\mathsf{IterateRej}$ returns $(\bot, \bot)$. This extra move would increase the communication complexity and force the signer to carry out almost all computations performed by the user before triggering a protocol restart. Moreover, this extra move would not allow the use of Gaussian distributed masking vectors $\mathbf{e}$ since a blind signature could be correctly verified even if rejection sampling does not accept. This would enable the user to request a protocol restart and obtain two different signatures. The advantage of using the Gaussian distribution for masking is that it allows to generate blind signatures with a size smaller than signatures generated using masking vectors that are uniformly distributed over a small subset of R.

Next, we give a detailed description of our scheme $\mathsf{BlindOR}$. The respective algorithms are formalized in Figure 9. Let $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$ and $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_{\mathsf{F}}}$, $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ be cryptographic hash

```
$\mathsf{BS.KGen}(pp)$:
    $((\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s})) \leftarrow_\$ \mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}(pp, 1)$
    $pk \leftarrow (\mathbf{b}_0, \mathbf{b}_1)$
    $sk \leftarrow (d, \mathbf{s})$
    **return** $(pk, sk)$

$\mathsf{BS.S}_1(pp, pk, sk)$:
    **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
    **parse** $sk = (d, \mathbf{s})$
    $(\mathbf{v}_0^*, \mathbf{v}_1^*, st_{\mathsf{S}}) \leftarrow_\$ \Sigma_{\mathsf{OR}}.\mathsf{P}_1(pp, (\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s}))$
    **return** $(\mathbf{v}_0^*, \mathbf{v}_1^*, st_{\mathsf{S}})$

$\mathsf{BS.U}_1(pp, pk, m, \mathbf{v}_0^*, \mathbf{v}_1^*)$:
    **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
    $(root_0, (\mathbf{p}_0, tree_0, \mathbf{e}_0)) \leftarrow_\$ \mathsf{Com}(pp, \mathbf{b}_0, \mathbf{v}_0^*)$
    $(root_1, (\mathbf{p}_1, tree_1, \mathbf{e}_1)) \leftarrow_\$ \mathsf{Com}(pp, \mathbf{b}_1, \mathbf{v}_1^*)$
    $\mathbf{c} \leftarrow \mathsf{H}(root_0, root_1, m)$
    $\mathbf{c}^* \leftarrow \mathsf{Cha}(pp, \mathbf{p}_0 \cdot \mathbf{p}_1, \mathbf{c}, 0)$
    $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}} \leftarrow (\mathbf{v}_0^*, \mathbf{v}_1^*, \mathbf{c}^*)$
    $st_{\mathsf{U}} \leftarrow (\mathbf{p}_0, \mathbf{p}_1, tree_0, tree_1, \mathbf{e}_0, \mathbf{e}_1, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$
    **return** $(\mathbf{c}^*, st_{\mathsf{U}})$

$\mathsf{BS.S}_2(pp, pk, sk, st_{\mathsf{S}}, \mathbf{c}^*)$:
    **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
    **parse** $sk = (d, \mathbf{s})$
    $(\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*) \leftarrow$
            $\leftarrow \Sigma_{\mathsf{OR}}.\mathsf{P}_2(pp, (\mathbf{b}_0, \mathbf{b}_1), (d, \mathbf{s}), st_{\mathsf{S}}, \mathbf{c}^*)$
    **if** $(\mathbf{z}_0^*, \mathbf{z}_1^*) = (\perp, \perp)$ **then**
        **return** $\perp$
    **return** $(\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$

$\mathsf{BS.U}_2(pp, pk, m, st_{\mathsf{U}}, \mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$:
    **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
    **parse** $st_{\mathsf{U}} = (\mathbf{p}_0, \mathbf{p}_1, tree_0, tree_1, \mathbf{e}_0, \mathbf{e}_1, st_{\Sigma_{\mathsf{OR}}.\mathsf{V}})$
    $(b, (i_0, i_1)) \leftarrow \Sigma_{\mathsf{OR}}.\mathsf{V}_2(pp, (\mathbf{b}_0, \mathbf{b}_1),$
            $st_{\Sigma_{\mathsf{OR}}.\mathsf{V}}, \mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$
    **if** $b = 0$ **then**
        **return** $\perp$
    $\mathbf{c}_0 \leftarrow \mathsf{Cha}(pp, \mathbf{p}_0, \mathbf{c}_0^*, 1)$
    $\mathbf{c}_1 \leftarrow \mathsf{Cha}(pp, \mathbf{p}_1, \mathbf{c}_1^*, 1)$
    $(\mathbf{z}_0, auth_0) \leftarrow \mathsf{Rsp}(pp, (\mathbf{p}_0, tree_0, \mathbf{e}_0), \mathbf{z}_0^*, i_0)$
    $(\mathbf{z}_1, auth_1) \leftarrow \mathsf{Rsp}(pp, (\mathbf{p}_1, tree_1, \mathbf{e}_1), \mathbf{z}_1^*, i_1)$
    **if** $(\mathbf{z}_0 = \perp) \vee (\mathbf{z}_1 = \perp)$ **then**
        **return** $\perp$
    $sig \leftarrow (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$
    **return** $(m, sig)$

$\mathsf{BS.Verify}(pp, pk, m, sig)$:
    **parse** $pk = (\mathbf{b}_0, \mathbf{b}_1)$
    **parse** $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$
    $root_0 \leftarrow \mathsf{Rec}(pp, \mathbf{b}_0, \mathbf{c}_0, \mathbf{z}_0, auth_0)$
    $root_1 \leftarrow \mathsf{Rec}(pp, \mathbf{b}_1, \mathbf{c}_1, \mathbf{z}_1, auth_1)$
    **if** $(root_0 = \perp) \vee (root_1 = \perp)$ **then**
        **return** 0
    $\mathbf{c} \leftarrow \mathsf{H}(root_0, root_1, m)$
    **if** $\mathbf{c} \neq \mathbf{c}_0 \cdot \mathbf{c}_1$ **then**
        **return** 0
    **return** 1
```

Figure 9: A formal description of the lattice-based blind signature scheme BlindOR. The description of BS.PGen is given in the text. Note that if $(\mathbf{z}_0^*, \mathbf{z}_1^*) = (\perp, \perp)$, then the signer restarts the protocol.

functions, where $\ell_{\mathsf{F}} \geq 2\lambda$ and $\mathbb{T}^{\kappa}$ is the challenge space of Σ. The first function is used to build tress of commitments while the second one hashes the blind commitments together with the message being signed to obtain a blind challenge.

Parameter generation. On input λ, the algorithm BS.PGen generates and returns a set of public parameters pp of the following form:

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

The description of the parameters is summarized in Table 2. The matrix $\mathbf{A}$ is chosen from the uniform distribution over $R_q^{k_1 \times k_2}$. We remark that pp includes the public parameters of the relation $\mathcal{R}$ for which Σ is defined, *i.e.*, $\mathsf{BS.PGen}$ may first run $\mathcal{R}.\mathsf{PGen}(1^\lambda)$ and then generates the remaining parameters of the scheme. For simplicity, the input of the algorithms of Σ includes pp.

Key generation. Given a set of public parameters pp, the algorithm $\mathsf{BS.KGen}$ runs $\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}$ on input $(pp, 1)$ to obtain a pair of instances $(\mathbf{b}_0, \mathbf{b}_1)$ for the relation $\mathcal{R}$ and a witness $\mathbf{s}$ for one of them, *i.e.*, $(d, \mathbf{s})$, where $d \in \{0, 1\}$. The algorithm returns the public key $pk = (\mathbf{b}_0, \mathbf{b}_1)$ and the secret key $sk = (d, \mathbf{s})$.

Signing. Let m be the message being signed. The signing protocol is initiated by the signer, which plays the role of the prover of $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$. The user in turn blinds the transcripts created by the signer via the blinding algorithms $\mathsf{Com}, \mathsf{Cha}$, and Rsp. The interaction works as follows:

On input (pp, pk, sk), $\mathsf{BS.S}_1$ runs $\Sigma_{\mathsf{OR}}.\mathsf{P}_1$ to obtain two commitments $\mathbf{v}_0^*, \mathbf{v}_1^*$, which are sent to the user.

Given $(pp, pk, m, \mathbf{v}_0^*, \mathbf{v}_1^*)$, $\mathsf{BS.U}_1$ computes two blind commitments $root_0$ and $root_1$ using Com. Afterwards, the hash function H is evaluated on input $(root_0, root_1, m)$ to obtain the blind challenge $\mathbf{c}$. This $\mathbf{c}$ is then "unblinded" to $\mathbf{c}^*$ via Cha, which uses the randomness $\mathbf{p}_0 \cdot \mathbf{p}_1 = (p_{1,0}p_{1,1}, \ldots, p_{\kappa,0}p_{\kappa,1})$. The unblinded challenge $\mathbf{c}^*$ is then sent to the signer.

Upon receiving $\mathbf{c}^*$, $\mathsf{BS.S}_2$ runs $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ to obtain $(\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$. The signing protocol is restarted if $(\mathbf{z}_0^*, \mathbf{z}_1^*) = (\perp, \perp)$. Otherwise, the output of $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ is sent to the user.

Given $(\mathbf{c}_0^*, \mathbf{c}_1^*, \mathbf{z}_0^*, \mathbf{z}_1^*)$, $\mathsf{BS.U}_2$ verifies the validity of the transcripts $(\mathbf{v}_0^*, \mathbf{c}_0^*, \mathbf{z}_0^*)$, $(\mathbf{v}_1^*, \mathbf{c}_1^*, \mathbf{z}_1^*)$ by running $\Sigma_{\mathsf{OR}}.\mathsf{V}_2$. Then, it blinds $\mathbf{c}_0^*, \mathbf{c}_1^*$ using Cha to obtain $\mathbf{c}_0, \mathbf{c}_1$. Finally, two blind responses $(\mathbf{z}_0, auth_0)$, $(\mathbf{z}_1, auth_1)$ are computed using Rsp. The blind signature is given by $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$.

Verification. On input $(pp, pk = (\mathbf{b}_0, \mathbf{b}_1), m, sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1))$, $\mathsf{BS.Verify}$ first computes two commitments $root_0$ and $root_1$ via Rec on input $(pp, \mathbf{b}_0, \mathbf{c}_0, \mathbf{z}_0, auth_0)$ and $(pp, \mathbf{b}_1, \mathbf{c}_1, \mathbf{z}_1, auth_1)$, respectively. The signature is rejected if at least one of the commitments is equal to $\perp$. Otherwise, sig is accepted if $ch_0 \cdot ch_1 = \mathsf{H}(root_0, root_1, m)$.

Next, we state and prove the correctness of $\mathsf{BlindOR}$.

Theorem 3.2. *Given the parameters in Table 2,* $\mathsf{BlindOR}$ *is* $\mathsf{corr}_{\mathsf{BS}}$*-correct w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$*, where* $\mathsf{corr}_{\mathsf{BS}} = \delta^* + 2\varepsilon^* + 2\delta + 2\varepsilon$*,* δ^* *is the probability that algorithm* $\Sigma_{\mathsf{OR}}.\mathsf{P}_2$ *returns* $\perp$*,* ε^* *is the probability that algorithm* $\Sigma.\mathsf{V}_2$ *returns* $(0, i)$*,* δ *is the probability that algorithm* Rsp *returns* $\perp$*, and* ε *is the probability that* Rec *returns* $\perp$*.*

Proof. By the correctness of the underlying Sigma protocol, the signer returns $\perp$ with probability at most δ^*, and the algorithm $\Sigma_{\mathsf{OR}}.\mathsf{V}_2$ returns $(0, (i_0, i_1))$ with probability at most $2\varepsilon^*$ (see Theorem 3.1). Furthermore, the user returns $sig = \perp$ if at least one execution of Rsp in $\mathsf{BS.U}_2$ returns $(\perp, \perp)$. This event occurs with probability at most 2δ. This is because for each $b \in \{0, 1\}$, Com generates ℓ masking vectors from $D_{\mathbb{Z}^n, \sigma}^{(k_1+k_2)\kappa}$, and by Lemma 2.10, $\mathsf{Rej}(pp, \mathbf{z}_b) = 1$ with probability $(1 - 2^{-100})/U$, where $U = \exp\left(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\right)$ and $\ell = 1$. Therefore, when $\ell \geq 1$, $\mathsf{IterateRej}$ returns $(\mathbf{z}_b, k_b) = (\perp, \perp)$ with probability $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell$, where ℓ is chosen such that $\left(1 - \frac{1-2^{-100}}{U}\right)^\ell \leq \delta$.

Let $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$ be a blind signature generated by $\mathsf{BlindOR}$ on some message. Then, $\mathbf{z}_0, \mathbf{z}_1$ are distributed according to $D_{\mathbb{Z}^n, \sigma}^{(k_1+k_2)\kappa}$ and by Lemma 2.9, both $\|\mathbf{z}_0\|, \|\mathbf{z}_1\|$ are greater than B_z

Table 2: A review of the parameters and sizes of keys and signatures of BlindOR. The signature size is computed without compression. In practice however, an integer $z \in D_{\mathbb{Z},\sigma}$ is optimally compressed via Huffman encoding, which requires $\approx \tau + 2.25$ bits on average, where $\sigma \approx 2^{\tau}$ [DLL+17].

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{Z}_{\geq 1}$
q	Modulus	prime, $q = 2p + 1 \pmod{4p}$, $n \geq p > 1$, $p = 2^{p'}$, $p' \in \mathbb{Z}_{\geq 1}$, $q^{1/p} > 2$
ω, ℓ	No. masking vectors	$\omega, \ell \in \mathbb{Z}_{\geq 1}$
h	Tree height	$h = \lceil \log(\omega\ell) \rceil$
κ	Specifies the set $\mathbb{T}^{\kappa}$	$\lvert\mathbb{T}^{\kappa}\rvert = (2n)^{\kappa} \geq 2^{\lambda}$
σ'	Standard deviation of in sk	$\sigma' > 0$
σ^*	Standard deviation in Σ	$\sigma^* = \alpha^* \sqrt{\kappa} B_s$, $S = \exp\big(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}}\big)$, $\Big(1 - \frac{1-2^{-100}}{S}\Big)^{\omega} \leq \delta^*$, $\delta^* > 0$
σ	Standard deviation in BS.U	$\sigma = \alpha B_{z^*}$, $U = \exp\big(\frac{12}{\alpha} + \frac{1}{2\alpha^2}\big)$, $\Big(1 - \frac{1-2^{-100}}{U}\Big)^{\ell} \leq \delta$, $\delta > 0$
M	No. restarts of BS.S	$M = 1/(1 - \delta^*)$
B_s	Bound of $\lVert\mathbf{s}\rVert$ in sk	$B_s = \eta'\sigma'\sqrt{(k_1 + k_2)n}$, $\eta' > 0$
B_{z^*}	Bound of $\lVert\mathbf{z}\rVert$ in Σ	$B_{z^*} = \eta^*\sigma^*\sqrt{(k_1 + k_2)\kappa n}$, $\eta^* > 0$
B_z	Bound of $\lVert\mathbf{z}\rVert$ in BS.U	$B_z = \eta\sigma\sqrt{(k_1 + k_2)\kappa n}$, $\eta > 0$
ℓ_{F}	Output length of F	$\ell_{\mathsf{F}} \geq 2\lambda$

Public key size (bit)	$2k_1 n \lceil \log q \rceil$
Secret key size (bit)	$(k_1 + k_2)n\lceil \log(t\sigma' + 1)\rceil + 1$, $t > 0$
Signature size (bit)	$2\kappa(1 + \lceil \log n \rceil) + 2(k_1 + k_2)\kappa n \lceil \log(t\sigma + 1)\rceil + 2h(\ell_{\mathsf{F}} + 1)$

with probability at most

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

By a suitable choice of $\eta > 0$ we obtain $\lVert\mathbf{z}_0\rVert \leq B_z$ and $\lVert\mathbf{z}_1\rVert \leq B_z$ each with probability at least $1 - \varepsilon$. Moreover, for each $b \in \{0, 1\}$ we have $root_b = \mathsf{RootCalc}(\mathbf{w}_b, auth_b)$, where $\mathbf{w}_b = (\mathbf{w}_{1,b}, \ldots, \mathbf{w}_{\kappa,b})$ and for all $j \in [\kappa]$ we have

$$\begin{aligned}\mathbf{w}_{j,b} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,b} - \mathbf{b}_b c_{j,b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot (\mathbf{e}_{j,b}^{(k_b)} + \mathbf{z}^*_{j,b} p_{j,b}) - \mathbf{b}_b c_{j,b} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{e}_{j,b}^{(k_b)} + {\mathbf{v}^*_{j,b}}^{(i_b)} p_{j,b} \\ &= \mathbf{v}_{j,b}^{(i_b,k_b)} \pmod{q}.\end{aligned}$$

Therefore, BS.Verify returns 0 if at least one execution of Rec within BS.Verify returns $\perp$, *i.e.*, if $root_0 = \perp$ or $root_1 = \perp$. This occurs with probability 2ε, and hence the correctness error of BlindOR is at most $\mathsf{corr}_{\mathsf{BS}} = \delta^* + 2\varepsilon^* + 2\delta + 2\varepsilon$. □

3.3 Security Analysis

In this section we prove that BlindOR satisfies the statistical blindness and computational one-more unforgeability in the ROM. We start with the blindness property.

Theorem 3.3. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ *be two hash functions modeled as random oracles. Given the parameters in Table 2,* $\mathsf{BlindOR}$ *is* ε*-statistically blind w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM, where* $\varepsilon = \max\{(2n)^{-\kappa}, 2^{-100}/U\}$.

Proof. Let S^* be an adversarial signer in the blindness experiment $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ defined in Figure 3. Then, S^* selects two messages m_0, m_1 and interacts with the honest user twice. The goal is to show that after both interactions, the messages output by the user, *i.e.*, two blind challenges of the form $\mathbf{c}^* \in \mathbb{T}^\kappa$ together with two blind signatures of the form $sig = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)$, are independently distributed and do not leak any information about the signed messages and the respective interaction.

The authentication paths $auth_0, auth_1$ include hash values that are uniformly distributed over $\{0,1\}^{\ell_\mathsf{F}}$. The challenge $\mathbf{c}^*$ as well as the signature part $(\mathbf{c}_0, \mathbf{c}_1)$ are uniformly distributed over $\mathbb{T}^\kappa$, and hence they do not leak any information. Moreover, Lemma 2.16 ensures that $\mathbf{c}^*$ is independently distributed from $\mathbf{c} = \mathbf{c}_0 \cdot \mathbf{c}_1$, and S^* can link $\mathbf{c}$ to the correct $\mathbf{c}^*$ only with probability $(2n)^{-\kappa}$ over guessing. The blind vectors $\mathbf{z}_0, \mathbf{z}_1$ have the form $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$, where $\mathbf{z}_j = \mathbf{e}_j + \mathbf{z}_j^* p_j$ (see Figure 8). By Lemma 2.10, both vectors completely mask the terms $\mathbf{z}_j^* p_j$ and are independently distributed within statistical distance of $2^{-100}/U$ from $D^{(k_1+k_2)\kappa}_{\mathbb{Z}^n,\sigma}$.

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

Next, we move to the one-more unforgeability property.

Theorem 3.4. *Let* $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$ *and* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}^\kappa$ *be two hash functions modeled as random oracles. Given the parameters in Table 2,* $\mathsf{BlindOR}$ *is* $(t, q_{\mathsf{Sign}}, q_\mathsf{F}, q_\mathsf{H}, \varepsilon)$*-one-more unforgeable w.r.t.* $pp \in \mathsf{BS.PGen}(1^\lambda)$ *in the ROM if* $\mathsf{D}\text{-}\mathsf{MLWE}$ *is* (t', ε')*-hard w.r.t.* $pp_{\mathsf{MLWE}} = (n, q, k_1, k_2, \sigma', \mathbf{A})$ *and* MSIS *is* (t'', ε'')*-hard w.r.t.* $pp_{\mathsf{MSIS}} = (n, q, k_1, k_2 + 1, 2\sqrt{B_z^2 + 1})$*. More precisely, if there exists a forger* A^* *against* $\mathsf{BlindOR}$ *w.r.t.* pp *that returns* $q_{\mathsf{Sign}} + 1$ *blind signatures in time* t *and with probability* ε*, and after making* q_F *and* q_H *queries to* F *and* H*, respectively, then* A^* *can be used to solve* $\mathsf{D}\text{-}\mathsf{MLWE}$ *w.r.t.* pp_{MLWE} *in time* $t' \approx t$ *and advantage* $\varepsilon' \approx \varepsilon$*, or* A^* *can be used to solve* MSIS *w.r.t.* pp_{MSIS} *in time* $t'' \approx 2t$ *and probability*

$$\varepsilon'' \approx \left(\frac{1}{2} - \varepsilon'\right) \cdot \left(\frac{1}{q_{\mathsf{Sign}} + 1}\right) \cdot acc \cdot \left(\frac{acc}{(q_{\mathsf{Sign}} + 1)\omega\ell} - \frac{1}{|\mathbb{T}^\kappa|}\right),$$

where $acc = \left(\varepsilon - \frac{q_\mathsf{F}^2 + q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{q_{\mathsf{Sign}}+1}{|\mathbb{T}^\kappa|}\right) \Big/ q_\mathsf{H}^{q_{\mathsf{Sign}}+1}$.

Proof. First we observe that the hardness of $\mathsf{D}\text{-}\mathsf{MLWE}$ is required to protect against key recovery attacks, *i.e.*, being able to determine the yes-instance of MLWE included in the public key $pk = (\mathbf{b}_0, \mathbf{b}_1)$ allows to compute the secret key, and hence forgeries. Therefore, in what follows we assume the hardness of $\mathsf{D}\text{-}\mathsf{MLWE}$ w.r.t. pp_{MLWE}, and construct a reduction algorithm R that solves MSIS w.r.t. pp_{MSIS} as given in the theorem statement.

Given pp_{MSIS} and a uniformly random matrix $\mathbf{A}' \in R_q^{k_1 \times (k_2+1)}$, reduction R chooses a bit $d \leftarrow\!\!\$\ \{0,1\}$, and writes $\mathbf{A}' = [\mathbf{A} \mid \mathbf{b}_{1-d}] \in R_q^{k_1 \times k_2} \times R_q^{k_1}$. Then, it generates the remaining public parameters pp of $\mathsf{BlindOR}$, and sets $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_{q_\mathsf{H}}\}$, where $\mathbf{c}_1, \ldots, \mathbf{c}_{q_\mathsf{H}} \leftarrow\!\!\$\ \mathbb{T}^\kappa$. Afterwards, R runs $\mathcal{R}.\mathsf{Gen}(pp, 1)$ to obtain $(\mathbf{b}_d, \mathbf{s})$. Then, R sets $pk = (\mathbf{b}_0, \mathbf{b}_1)$, $sk = (d, \mathbf{s})$, and runs A^* on input (pp, pk). The random oracle and signing queries that A^* make are answered by R as follows:

Random oracle query. R maintains a list L_H initialized as the empty set. It stores pairs of queries to H and their answers. If H was previously queried on some input, then R looks up its entry in L_H and returns its answer $\mathbf{c}$. Otherwise, it picks the first unused $\mathbf{c} \in C$ and updates the list. Furthermore, R initializes an empty list L_F to store pairs of queries to F and their answers. The queries to F are

```
F(x):
11:  if $(\mathbf{x}, \mathsf{F}(\mathbf{x})) \in L_{\mathsf{F}}$ then
12:      return $\mathsf{F}(\mathbf{x})$
13:  $\mathsf{F}(\mathbf{x}) \leftarrow\!\$\ \{0,1\}^{\ell_{\mathsf{F}}}$
14:  if $\exists(\mathbf{x}', \mathsf{F}(\mathbf{x}')) \in L_{\mathsf{F}} \colon (\mathbf{x} \neq \mathbf{x}') \wedge (\mathsf{F}(\mathbf{x}) = \mathsf{F}(\mathbf{x}'))$ then
15:      return $\perp$
16:  if $\exists(\mathbf{y}, \mathsf{F}(\mathbf{y})) \in L_{\mathsf{F}} \colon \mathbf{y} = \mathsf{F}(\mathbf{x})$ then
17:      return $\perp$
18:  $L_{\mathsf{F}} \leftarrow L_{\mathsf{F}} \cup \{(\mathbf{x}, \mathsf{F}(\mathbf{x}))\}$
19:  return $\mathsf{F}(\mathbf{x})$

H(root_0, root_1, m):
21:  if $\exists((root_0, root_1, m), \mathsf{H}(root_0, root_1, m)) \in L_{\mathsf{H}}$ then
22:      return $\mathsf{H}(root_0, root_1, m)$
23:  $\mathsf{H}(root_0, root_1, m) \hookleftarrow C$    // picks the first unused answer from $C$
24:  $L_{\mathsf{H}} \leftarrow L_{\mathsf{H}} \cup \{((root_0, root_1, m), \mathsf{H}(root_0, root_1, m))\}$
25:  return $\mathsf{H}(root_0, root_1, m)$
```

Figure 10: A description of the random oracles F and H.

answered in a way that excludes collisions and chains. Excluding collisions rules out queries $\mathbf{x} \neq \mathbf{x}'$ such that $\mathsf{F}(\mathbf{x}) = \mathsf{F}(\mathbf{x}')$, and excluding chains guarantees that the query $\mathsf{F}(\mathsf{F}(\mathbf{x}))$ will not be made before the query $\mathsf{F}(\mathbf{x})$. This ensures that each node output by HashTree has a unique preimage, and prevents spanning hash trees with cycles. Simulating F this way is within statistical distance of at most $\frac{q_{\mathsf{F}}^2 + q_{\mathsf{F}}}{2^{\ell_{\mathsf{F}}}}$ from an oracle that allows collisions and chains. The description of the oracles F and H is given in Figure 10.

Signature query. Upon receiving a signature query from A^*, R runs the signing protocol of BlindOR. Furthermore, R updates both lists L_{H} and L_{F} accordingly. Note that the environment of A^* is perfectly simulated and signatures are generated with the same probability as in the real execution of the signing protocol.

After q_{Sign} successful invocations, A^* returns $q_{\mathsf{Sign}} + 1$ pairs of distinct messages and their signatures, where one of these pairs is not generated during the interaction. If H was not programmed or queried during invocation of A^*, then A^* produces a $\mathbf{c} \in \mathbb{T}^{\kappa}$ that validates correctly with probability $1/|\mathbb{T}^{\kappa}|$. Therefore, the probability that A^* succeeds in a forgery such that all $q_{\mathsf{Sign}} + 1$ signatures correspond to random oracle queries made by A^* is at least $\varepsilon - \frac{q_{\mathsf{Sign}}+1}{|\mathbb{T}^{\kappa}|}$.

Suppose that the output of A^* includes two pairs (m, sig) and (m', sig') with the same $\mathbf{c} \in \mathbb{T}^{\kappa}$. This means we have $\mathsf{H}(root_0, root_1, m) = \mathsf{H}(root_0', root_1', m')$. If $(m \neq m') \vee (root_0 \neq root_0') \vee (root_1 \neq root_1')$, then a second preimage of $\mathbf{c}$ has been found by A^*. This occurs with probability at most $1/|\mathbb{T}^{\kappa}|$. On the other hand, the $q_{\mathsf{Sign}} + 1$ messages are not pairwise distinct if $(m = m') \wedge (root_0 = root_0') \wedge (root_1 = root_1')$. Hence, we may assume for the remainder of the proof that all $q_{\mathsf{Sign}} + 1$ blind signatures output by A^* include distinct random oracle answers of the form $\mathbf{c} \in \mathbb{T}^{\kappa}$.

Afterwards, R guesses an index $i^* \in [q_{\mathsf{Sign}} + 1]$ such that $\mathbf{c}_{i^*} = \mathbf{c}_{j^*}$ for some $j^* \in [q_{\mathsf{H}}]$. Then, R records the pair $(m_{i^*}, sig_{i^*} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1))$ and invokes A^* again with the same random tape and the random oracle queries $C' = \{\mathbf{c}_1, \dots, \mathbf{c}_{j^*-1}, \mathbf{c}'_{j^*}, \dots, \mathbf{c}'_{q_{\mathsf{H}}}\}$, where $\mathbf{c}'_{j^*}, \dots, \mathbf{c}'_{q_{\mathsf{H}}} \in \mathbb{T}^\kappa$ are freshly generated by R. After rewinding, A^* returns $q_{\mathsf{Sign}} + 1$ pairs of distinct messages and their valid signatures. The potential two valid forgeries (before and after rewinding) output by A^* at index i^* have the form

$$(m, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, \mathit{auth}_0, \mathit{auth}_1)) \text{ and } (m', (\mathbf{c}'_0, \mathbf{c}'_1, \mathbf{z}'_0, \mathbf{z}'_1, \mathit{auth}'_0, \mathit{auth}'_1)),$$

where $\mathbf{c}_i = (c_{1,i}, \dots, c_{\kappa,i})$ and $\mathbf{c}'_i = (c'_{1,i}, \dots, c'_{\kappa,i})$, $i \in \{0,1\}$. By the verification algorithm we obtain

$$\begin{aligned} \mathbf{w}_{j,1-d} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,1-d} - \mathbf{b}_{1-d} c_{j,1-d} \pmod q, & \mathbf{w}'_{j,1-d} &= [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_{j,1-d} - \mathbf{b}_{1-d} c'_{j,1-d} \pmod q, \\ \mathbf{w}_{1-d} &= (\mathbf{w}_{1,1-d}, \dots, \mathbf{w}_{\kappa,1-d}), & \mathbf{w}'_{1-d} &= \left(\mathbf{w}'_{1,1-d}, \dots, \mathbf{w}'_{\kappa,1-d}\right), \\ \mathit{root}_{1-d} &= \mathsf{RootCalc}(\mathbf{w}_{1-d}, \mathit{auth}_{1-d}), & \mathit{root}'_{1-d} &= \mathsf{RootCalc}\left(\mathbf{w}'_{1-d}, \mathit{auth}'_{1-d}\right), \\ \mathbf{c}_0 \cdot \mathbf{c}_1 &= \mathbf{c} = \mathsf{H}(\mathit{root}_0, \mathit{root}_1, m), & \mathbf{c}'_0 \cdot \mathbf{c}'_1 &= \mathbf{c}' = \mathsf{H}\left(\mathit{root}'_0, \mathit{root}'_1, m'\right). \end{aligned}$$

By the forking lemma (see Lemma 2.17) we have

$$\mathit{root}_0 = \mathit{root}'_0, \qquad \mathit{root}_1 = \mathit{root}'_1, \qquad m = m', \qquad \mathbf{c} \neq \mathbf{c}', \qquad k_{1-d} = k'_{1-d},$$

where $k_{1-d}, k'_{1-d} \in \{0, \dots, \omega\ell - 1\}$ are the indices included in auth_{1-d} and auth'_{1-d}, respectively. Observe that simulating the hash queries to F as shown in Figure 10 ensures that both auth_{1-d} and auth'_{1-d} include the same sequence of hash values, and hence $\mathit{auth}_{1-d} = \mathit{auth}'_{1-d}$ and $\mathbf{w}_{1-d} = \mathbf{w}'_{1-d}$. If $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$, then for at least one index $j \in [\kappa]$ we have $c_{j,1-d} \neq c'_{j,1-d}$. Thus, we obtain

$$[\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{j,1-d} - \mathbf{b}_{1-d} c_{j,1-d} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_{j,1-d} - \mathbf{b}_{1-d} c'_{j,1-d} \pmod q.$$

This gives the non-trivial solution $\left[\mathbf{z}_{j,1-d} - \mathbf{z}'_{j,1-d} \,\middle|\, c'_{j,1-d} - c_{j,1-d}\right]^\top$ to MSIS w.r.t. pp_{MSIS} and the matrix $[\mathbf{I}_{k_1} \mid \mathbf{A} \mid \mathbf{b}_{1-d}] = [\mathbf{I}_{k_1} \mid \mathbf{A}']$.

Next, we analyze the success probability of R. The probability that R answers the correct sequence of $q_{\mathsf{Sign}} + 1$ random oracle queries to H that are used by A^* in the forgery is at least $1/q_{\mathsf{H}}^{q_{\mathsf{Sign}}+1}$. Since one of the $q_{\mathsf{Sign}} + 1$ pairs output by A^* is by assumption not generated during the interaction with R, the probability of correctly guessing the index i^* corresponding to this pair is $1/(q_{\mathsf{Sign}} + 1)$. Forking succeeds with probability

$$\mathit{frk} \geq \mathit{acc} \cdot \left(\frac{\mathit{acc}}{(q_{\mathsf{Sign}} + 1)\omega\ell} - \frac{1}{|\mathbb{T}^\kappa|} \right), \text{ where } \mathit{acc} = \left(\varepsilon - \frac{q_{\mathsf{F}}^2 + q_{\mathsf{F}}}{2^{\ell_{\mathsf{F}}}} - \frac{q_{\mathsf{Sign}} + 1}{|\mathbb{T}^\kappa|} \right) \Big/ q_{\mathsf{H}}^{q_{\mathsf{Sign}}+1}.$$

By Lemma 3.5, the probability that $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$ is given by $\frac{1}{2} - \varepsilon'$. This results in the probability ε'' that is given in the theorem statement. □

Lemma 3.5. *Assume that after rewinding the forger* A^* *as done by the reduction* R *given in Theorem 3.4, the two forgeries output by* A^* *satisfy* $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$ *with probability* $1/2 + \varepsilon'$*, where* d *corresponds to the yes-instance of* MLWE *included in the public key and* ε' *is noticeably greater than* 0*. Then, there exists a distinguisher* D^* *that uses* A^* *to win the experiment* $\mathbf{Exp}_{\mathsf{D}^*}^{\mathsf{D\text{-}MLWE}}$ *with the advantage* ε'*.*

Proof. The input of D^* is $pp_{\mathsf{MLWE}} = (n, k_1, k_2, q, \sigma', \mathbf{A})$ and a vector $\mathbf{b}' \in R_q^{k_1}$. As indicated in the experiment $\mathbf{Exp}_{\mathsf{D}^*}^{\mathsf{D\text{-}MLWE}}$, the goal of D^* is to decide if the instance $\mathbf{b}'$ is either uniformly random (no-instance), or it has the form $\mathbf{b}' = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s}' \pmod q$, *i.e.*, if $\mathbf{b}'$ is a yes-instance with witness $\mathbf{s}'$. Similar to the reduction R, D^* sets $\mathbf{b}_{1-d} = \mathbf{b}'$ and creates the yes-instance $\mathbf{b}_d$ with the corresponding witness $\mathbf{s}$. Then, D^* runs the forger on input $(pp, pk = (\mathbf{b}_0, \mathbf{b}_1))$. The signing and random oracle queries are answered by D^* as done by R. After rewinding, we have $\mathbf{c} \neq \mathbf{c}'$ due to the forking lemma, where $\mathbf{c} = \mathbf{c}_0 \cdot \mathbf{c}_1$ and

Table 3: Concrete parameters of BlindOR for 128 bits of security. The related sizes of keys/signatures and communication cost are given in Table 1.

n	k_1	k_2	q	ω	ℓ	h	κ	σ'	α^*	σ^*	α	σ	M	ℓ_{F}
256	5	4	$\approx 2^{36}$	1	8	3	15	4	11	8344	41	66142158	3	384

$\mathbf{c}' = \mathbf{c}'_0 \cdot \mathbf{c}'_1$. If $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$, then D^* returns $b^* = 0$ indicating that its input is a no-instance. Otherwise, D^* returns a randomly chosen bit b^*.

If the input $\mathbf{b}_{1-d}$ of D^* is a no-instance, then by assumption we have $\mathbf{c}_{1-d} = \mathbf{c}'_{1-d}$ with probability $1/2+\varepsilon'$. Thus, D^* returns $b^* = 0$. However, if the input of D^* is a yes-instance, then the forger has obtained a public key consisting of two yes-instances. By the witness indistinguishability property of the underlying OR-protocol, the forger A^* cannot guess which coordinate D^* is using to sign. Therefore, D^* returns a randomly chosen bit b^* in case $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$. □

3.4 Concrete Parameters

In this section we propose concrete parameters for BlindOR targeting 128 bits of security. The parameters are given in Table 3. In the following we highlight some key points that we considered during the parameter selection. Then, we explain the hardness estimation of the instances of D-MLWE and MSIS underlying BlindOR.

The concrete value of the modulus is given by the 36-bit prime number $q = 68719473217$. By Lemma 2.11, the polynomial $X^n + 1$ underlying the ring R_q has $p = 32$ irreducible factors, and each polynomial $\bar{c} \in \overline{\mathcal{C}}$ is invertible in R_q, where $\overline{\mathcal{C}} = \{c - c' \mid c, c' \in \mathbb{T},\ c \neq c'\}$. This guarantees that in the proof of the one-more unforgeability property the reduction computes a non-zero solution $\mathbf{x}$ to MSIS. We set $\kappa = 15$ so that $|\mathbb{T}^\kappa| > 2^{128}$, which is large enough for 128 bits of security. The parameters ω, α^*, and σ^* ensure that the rejection sampling procedure carried out by the signer accepts with probability 1/3, *i.e.*, the signer triggers a protocol restart at most twice. However, the choice of the parameters ℓ, α, and σ makes sure that the user computes a blind signature with probability at least $1 - 2^{-15}$. Since the security reduction to the one-more unforgeability property of BlindOR allows for poly-logarithmic number of signatures per public key, the latter probability is sufficiently large, and we believe that it is not worthwhile to increase the parameter ℓ in order to obtain a probability of, *e.g.*, at least $1 - 2^{-40}$. This is because increasing ℓ significantly affects the performance of the scheme. For the parameter σ', the instance of D-MLWE underlying the public key is hard enough for the target security level. The parameter η' included in the bound B_s (*cf.* Table 2) is given by $\eta' = 1.02$, which guarantees that the Gaussian vector $\mathbf{s}$ in sk is bounded by B_s with probability at least 0.75 (*cf.* Lemma 2.9), *i.e.*, the algorithm BS.KGen is restarted with probability at most 0.25. Furthermore, the parameters η^* and η included in B_{z^*} and B_z, respectively, are set to $\eta^* = \eta = 1.04$. This ensures that the response sent by the signer as well as the blind signature are both verified with probability at least $1 - 2^{-80}$. The cryptographic hash function F used for building trees of commitments is considered to output hash values of length $\ell_{\mathsf{F}} = 3\lambda = 384$.

Estimating the hardness of D-MLWE w.r.t. $pp_{\mathsf{MLWE}} = (n, q, k_1, k_2, \sigma', \mathbf{A})$, was carried out using the well known LWE estimator [APS15]. The hardness of solving MSIS, w.r.t. $pp_{\mathsf{MSIS}} = (n, q, k_1, k_2 + 1, \beta)$, where $\beta = 2\sqrt{B_z^2 + 1}$, is equivalent to finding a non-trivial vector bounded by β in the lattice

$$\left\{ \mathbf{x} \in \mathbb{Z}^{(k_1+k_2+1)n} \;\middle|\; \mathbf{0} = [\mathbf{I}_{k_1} \mid \mathbf{A} \mid \mathbf{b}] \cdot \mathbf{x} \pmod{q} \right\}.$$

The best known algorithm for finding short non-trivial vectors is the Block–Korkine–Zolotarev algorithm (BKZ) [SE94], which was improved in practice in [CN11]. As a subroutine, BKZ uses an algorithm for the shortest vector problem (SVP) in lattices of dimension b, where b is called the *block size*. The best known algorithm for SVP with no memory restrictions is due to [BDGL16], and takes time approximately $2^{0.292b}$. The time required by BKZ to run with block size b on an m-dimensional lattice $\mathcal{L}$ is given by (see, *e.g.*, [BDGL16])

$$8m \cdot 2^{0.292b+16.4} . \tag{2}$$

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

$$\delta = \left(b(\pi b)^{\frac{1}{b}}/(2\pi e)\right)^{\frac{1}{2(b-1)}} . \tag{3}$$

Therefore, in order to compute the time required by BKZ to solve MSIS w.r.t. pp_{MSIS}, we first determine δ by setting $\beta = \delta^m \det(\mathcal{L})^{1/m}$, where $\beta = 2\sqrt{B_z^2+1}$ and $m = (k_1+k_2+1)n$. After that, we compute the minimum block size b required to achieve δ by using Equation (3). Then, we put the resulted b in Equation (2) to obtain the time required by BKZ to solve MSIS w.r.t. pp_{MSIS}.

Table 1 gives a simple comparison between BlindOR and the lattice-based construction of blind signatures presented by Hauck *et al.* [HKLN20] in terms of the sizes of keys and signatures. We remark that the Hermite delta related to our parameters is given by $\delta = 1.004$, while $\delta = 1.005$ for the parameters proposed in [HKLN20].

4 Conclusion

In this paper we have presented BlindOR, a lattice-based construction of blind signatures from OR-proofs. Although this construction doubles the number of public key and signature parts, it offers small sizes compared to the literature. This comes from the fact that we can reduce the one-more unforgeability property of our construction from both the MLWE and the MSIS problems.

Similar to previous works, our construction allows a poly-logarithmic number of signatures per public key. This restriction is inherited from the proof technique due to Pointcheval and Stern [PS00] as well as Abe and Okamoto [AO00]. Therefore, it would be interesting to investigate extending the work of Pointcheval [Poi98] to the lattice setting in order to increase the number of signatures to a polynomial amount per public key at the cost of increasing the communication complexity and generating signatures in a sequential manner. A further interesting direction for future work is to prove the security of our scheme in the quantum random oracle model due to Boneh *et al.* [BDF+11] as well as investigating constructions based on sequential OR-proofs [AOS02, FHJ20].

Acknowledgments

We thank Marc Fischlin for helpful discussions. We are grateful to Shuichi Katsumata for pointing out a missing argument in an earlier version of the proof of Theorem 3.4. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297.

References

[AEB20a] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. BLAZE: Practical lattice-based blind signatures for privacy-preserving applications. In Joseph Bonneau and Nadia Heninger, editors, *FC 2020*, volume 12059 of *LNCS*, pages 484–502. Springer, Heidelberg, February 2020. (Cited on pages 3, 4, 5, 10, 13, and 14.)

[AEB20b] Nabil Alkeilani Alkadri, Rachid El Bansarkhani, and Johannes Buchmann. On lattice-based interactive protocols: An approach with less or no aborts. In Joseph K. Liu and Hui Cui, editors, *ACISP 20*, volume 12248 of *LNCS*, pages 41–61. Springer, Heidelberg, November / December 2020. (Cited on pages 3, 4, 5, 14, and 21.)

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. (Cited on page 13.)

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286. Springer, Heidelberg, August 2000. (Cited on pages 5 and 30.)

[AOS02] Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. In Yuliang Zheng, editor, *ASIACRYPT 2002*, volume 2501 of *LNCS*, pages 415–432. Springer, Heidelberg, December 2002. (Cited on page 30.)

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015. (Cited on page 29.)

[ASY21] Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Towards practical and round-optimal lattice-based threshold and blind signatures. Cryptology ePrint Archive, Report 2021/381, 2021. `https://eprint.iacr.org/2021/381`. (Cited on page 5.)

[BCK$^+$14] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 551–572. Springer, Heidelberg, December 2014. (Cited on page 17.)

[BDF$^+$11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 41–69. Springer, Heidelberg, December 2011. (Cited on page 30.)

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In Robert Krauthgamer, editor, *27th SODA*, pages 10–24. ACM-SIAM, January 2016. (Cited on page 30.)

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013. (Cited on page 3.)

[BLL$^+$21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021. (Cited on page 6.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006. (Cited on page 15.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993. (Cited on pages 4 and 6.)

[BS99] Johannes Blömer and Jean-Pierre Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *31st ACM STOC*, pages 711–720. ACM Press, May 1999. (Cited on page 13.)

[CCT$^+$11] Liang Chen, Yongquan Cui, Xueming Tang, Dongping Hu, and Xin Wan. Hierarchical ID-based blind signature from lattices. In Yuping Wang, Yiu-ming Cheung, Ping Guo, and Yingbin Wei, editors, *CIS 2011*, pages 803–807. IEEE Computer Society, December 2011. (Cited on page 5.)

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Heidelberg, August 1994. (Cited on pages 4, 8, and 9.)

[CG08] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 345–356. ACM Press, October 2008. (Cited on page 3.)

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982. (Cited on page 3.)

[Che13] Yuanmi Chen. *Réduction de réseau et sécurité concrete du chiffrement completement homomorphe.* PhD thesis, ENS-Lyon, France, 2013. (Cited on page 30.)

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20. Springer, Heidelberg, December 2011. (Cited on pages 29 and 30.)

[CNs07] Jan Camenisch, Gregory Neven, and abhi shelat. Simulatable adaptive oblivious transfer. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 573–590. Springer, Heidelberg, May 2007. (Cited on page 4.)

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993. (Cited on page 5.)

[Cra97] Ronald Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols.* PhD thesis, Universiteit van Amsterdam, The Netherlands, 1997. (Cited on page 7.)

[DLL$^+$17] Léo Ducas, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehle. CRYSTALS – Dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Report 2017/633, 2017. https://eprint.iacr.org/2017/633. (Cited on page 25.)

[FHJ20] Marc Fischlin, Patrick Harasser, and Christian Janson. Signatures from sequential-OR proofs. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020*, volume 12107 of *LNCS*, pages 212–244. Springer, Heidelberg, May 2020. (Cited on page 30.)

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Heidelberg, August 2006. (Cited on page 4.)

[FS09] Marc Fischlin and Dominique Schröder. Security of blind signatures under aborts. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 297–316. Springer, Heidelberg, March 2009. (Cited on page 4.)

[FS10] Marc Fischlin and Dominique Schröder. On the impossibility of three-move blind signature schemes. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 197–215. Springer, Heidelberg, May / June 2010. (Cited on page 5.)

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, USA, 2009. (Cited on page 5.)

[GHWX16] Wen Gao, Yupu Hu, Baocang Wang, and Jia Xie. Identity-based blind signature from lattices in standard model. In Kefei Chen, Dongdai Lin, and Moti Yung, editors, *Inscrypt 2016*, volume 10143 of *LNCS*, pages 205–218. Springer, Heidelberg, November 2016. (Cited on page 5.)

[GKPV10] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In Andrew Chi-Chih Yao, editor, *ICS 2010*, pages 230–240. Tsinghua University Press, January 2010. (Cited on page 5.)

[GRS$^+$11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 630–648. Springer, Heidelberg, August 2011. (Cited on page 5.)

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pages 43–60. Springer, Heidelberg, February 2016. (Cited on page 3.)

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019*, volume 11478 of *LNCS*, pages 345–375. Springer, Heidelberg, May 2019. (Cited on pages 3 and 15.)

[HKLN20] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020*, volume 12171 of *LNCS*, pages 500–529. Springer, Heidelberg, August 2020. (Cited on pages 3, 4, 5, 6, 10, and 30.)

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164. Springer, Heidelberg, August 1997. (Cited on pages 3 and 10.)

[KKS17] Mahender Kumar, Chittaranjan Padmanabha Katti, and Prem Chandra Saxena. A secure anonymous e-voting system using identity-based blind signature scheme. In Rudrapatna K. Shyamasundar, Virendra Singh, and Jaideep Vaidya, editors, *ICISS 2017*, volume 10717 of *LNCS*, pages 29–49. Springer, Heidelberg, December 2017. (Cited on page 3.)

[LDS$^+$20] Huy Quoc Le, Dung Hoang Duong, Willy Susilo, Ha Thanh Nguyen Tran, Viet Cuong Trinh, Josef Pieprzyk, and Thomas Plantard. Lattice blind signatures with forward security. In Joseph K. Liu and Hui Cui, editors, *ACISP 20*, volume 12248 of *LNCS*, pages 3–22. Springer, Heidelberg, November / December 2020. (Cited on page 5.)

[LN17] Vadim Lyubashevsky and Gregory Neven. One-shot verifiable encryption from lattices. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 293–323. Springer, Heidelberg, April / May 2017. (Cited on page 17.)

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23. Springer, Heidelberg, May / June 2010. (Cited on page 13.)

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptogr.*, 75(3):565–599, 2015. (Cited on page 13.)

[LS18] Vadim Lyubashevsky and Gregor Seiler. Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018*, volume 10820 of *LNCS*, pages 204–224. Springer, Heidelberg, April / May 2018. (Cited on page 12.)

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Heidelberg, December 2009. (Cited on pages 17, 18, and 20.)

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012. (Cited on pages 5, 12, and 17.)

[Mic02] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *43rd FOCS*, pages 356–365. IEEE Computer Society Press, November 2002. (Cited on page 13.)

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, August 1993. (Cited on page 5.)

[PHBS19] D. Papachristoudis, D. Hristu-Varsakelis, F. Baldimtsi, and G. Stephanides. Leakage-resilient lattice-based partially blind signatures. Cryptology ePrint Archive, Report 2019/1452, 2019. `https://eprint.iacr.org/2019/1452`. (Cited on page 5.)

[Poi98] David Pointcheval. Strengthened security for blind signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 391–405. Springer, Heidelberg, May / June 1998. (Cited on page 30.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *J. Cryptology*, 13(3):361–396, June 2000. (Cited on pages 3, 5, 10, 15, and 30.)

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. (Cited on page 13.)

[Rüc10] Markus Rückert. Lattice-based blind signatures. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 413–430. Springer, Heidelberg, December 2010. (Cited on pages 3, 5, and 10.)

[Sch01] Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Heidelberg, November 2001. (Cited on page 5.)

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994. (Cited on page 29.)

[SU17] Dominique Schröder and Dominique Unruh. Security of blind signatures revisited. *J. Cryptology*, 30(2):470–494, April 2017. (Cited on page 4.)

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Heidelberg, August 2002. (Cited on page 5.)