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

Nabil Alkeilani Alkadri(✉), Patrick Harasser, and Christian Janson

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

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

1 Introduction

Blind signature schemes are a fundamental cryptographic primitive. First introduced by Chaum [9] in the context of an anonymous e-cash system, they have since become an essential building block in many applications such as anonymous credentials, e-voting, and blockchain protocols. 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* [16,19]. 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 [20] can be seen as an important step in carrying the core design of classical constructions based on the discrete logarithm problem [19] over to the lattice setting. The same design principle was then adopted in subsequent works, *e.g.*, by Alkeilani Alkadri *et al.* [3,4], where the scheme BLAZE and its successor BLAZE$^+$ have been proposed and shown to be practical.

M. Conti et al. (Eds.): CANS 2021, LNCS 13099, pp. 95–115, 2021.
https://doi.org/10.1007/978-3-030-92548-2_6

Recently, Hauck *et al.* [15] pointed out that the proof of the one-more unforgeability property, originally by Pointcheval and Stern for a discrete logarithm based construction [19] and later reproposed by Rückert for his lattice-based scheme [20], has not been adapted correctly to this new setting. Indeed, the main idea of the reduction in [19] 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 [19]). 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 [19] 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 (R)SIS. The problem, as discussed in [15], 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.* [15] extended the modular framework for blind signatures from linear functions given in [14] 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 [15] entails increasing the parameter sizes, so that their framework applies and yields a correct proof. In particular, the RSIS-based instantiation given in [15] 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.

Our Contributions. In this paper we make a significant 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.* [10], a feature which allows us to sidestep the missing security argument pointed out in [15]. 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) 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 [15].

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 [3]. 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 [4] 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.* [15] 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 [15], 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* [8] and *honest-user unforgeability* [21]. This is established by signing a commitment to the message instead of the message itself [12,21]. However, and similar to [15], it is still unclear how to prove the blindness property under a maliciously generated key pair [11].

Table 1. A comparison between BlindOR and the scheme introduced in [15] 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 [15, Figure 9].

Scheme	Public key	Secret key	Signature	Communication
BlindOR	10.3	1.7	17.2	375.6
[15]	443.75	4275	7915.52	34037.25

Related Work. Our construction is inspired by the work of Abe and Okamoto [1], who used OR-proofs to build partially blind signatures with security based on the hardness of the discrete logarithm problem. 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.* [15] showed that all lattice-based constructions of blind signatures from Sigma protocols (or canonical identification schemes) prior to their framework, such as [3,20], do not have a valid security argument. Furthermore, Alkeilani Alkadri *et al.* [3] showed that all two-round lattice-based blind signature schemes based on preimage sampleable trapdoor functions are insecure.

Recently, Agrawal *et al.* [2] made a step towards practical two-round lattice-based blind signatures. They improved the two-round construction of Garg *et al.* [13] 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 [13]. 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 ROM. 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. We refer to [2, Section 6.3] for more details and discussions on the limitations of their construction.

2 Preliminaries

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 $[k] := \{1, \ldots, k\}$. 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 $\mathbb{T}_\kappa^n$ 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 $\|a\| := (\sum_{i=0}^{n-1} |a_i|^2)^{1/2}$ and $\|a\|_\infty := \max_i |a_i|$, respectively. Similarly, for $\mathbf{b} = (b_1, \ldots, b_k)^t \in R^k$, we let $\|\mathbf{b}\| := (\sum_{i=1}^{k} \|b_i\|^2)^{1/2}$ 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) [7] 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.

Relations, Sigma Protocols, and OR-Proofs

Definition 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*.*

$\mathcal{R}.\mathsf{RSet}$ *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), each 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. *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* $\mathcal{R}_{\mathsf{OR}}.\mathsf{RSet}(pp) := \{((x_0, x_1), (d, w)) \,|\, (x_d, w), (x_{1-d}, \cdot) \in \mathcal{R}.\mathsf{Gen}(pp, 1)\}$*, and whose instance generator* $\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}$ *is given in Fig. 1.*

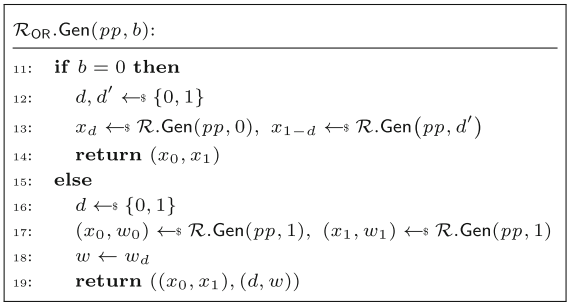

Fig. 1. Definition of the instance generator $\mathcal{R}_{\mathsf{OR}}.\mathsf{Gen}$ of the OR-relation on $\mathcal{R}$. Note that in line 13 we slightly abuse notation: If $d' = 1$, we only consider the first component of the output, and ignore the witness in the second coordinate.

Definition 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.* [10], 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 [10] 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.

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.

Blind Signatures. We define blind signatures following the exposition of Hauck *et al.* [15], where the interaction between signer and user consists of three moves.

Definition 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*. We assume that the set* 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:*

- *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:*

- *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.*

Let $pp \in \mathsf{BS.PGen}(1^\lambda)$. We say that BS is $\mathsf{corr}_{\mathsf{BS}}$*-correct* w.r.t. pp if $\mathsf{BS.Verify}$ validates honestly signed messages under honestly created keys with probability at least $1 - \mathsf{corr}_{\mathsf{BS}}$. The security of blind signatures is defined by the notions blindness and one-more unforgeability [16,19].

Definition 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* $\mathbf{Adv}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp) := 2 \cdot |\Pr[\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}(pp) = 1] - \frac{1}{2}| \leq \varepsilon$*, where the game* $\mathbf{Exp}^{\mathrm{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ *is depicted in Fig. 2.* BS *is* ε-statistically blind *if it is* (t, ε)*-blind for every* t*.*

Definition 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* $\mathbf{Adv}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp) := \Pr[\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}(pp) = 1] \leq \varepsilon$*, where the game* $\mathbf{Exp}^{\mathrm{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$ *is depicted in Fig. 2.*

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

11: $b \leftarrow\!\!\$\ \{0,1\}$
12: $(pk, sk) \leftarrow\!\!\$\ \mathsf{BS.KGen}(pp)$
13: $(m_0, m_1, st_{\mathsf{find}}) \leftarrow\!\!\$\ \mathsf{S}^*(\mathsf{find}, pp, pk, sk)$
$\langle \cdot, \mathsf{BS.U}(pp,pk,m_b)\rangle^1,$
14: $st_{\mathsf{issue}} \leftarrow\!\!\$\ \mathsf{S}^{*\langle \cdot, \mathsf{BS.U}(pp,pk,m_{1-b})\rangle^1}(\mathsf{issue}, st_{\mathsf{find}})$
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

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

31: $(pk, sk) \leftarrow\!\!\$\ \mathsf{BS.KGen}(pp)$
32: $\big((m_1, sig_1), \ldots, (m_l, sig_l)\big) \leftarrow\!\!\$$
$\leftarrow\!\!\$\ \mathsf{U}^{*\langle \mathsf{BS.S}(pp,pk,sk),\cdot\rangle^\infty}(pp, pk)$
33: $k \leftarrow$ No. successful signing invocations
34: **if** $\forall i, j \in [l],\ i \neq j$:
$(m_i \neq m_j) \wedge$
$(\mathsf{BS.Verify}(pp, pk, m_i, sig_i) = 1) \wedge$
$(k + 1 = l)$ **then**
35: **return** 1
36: **return** 0

Fig. 2. Definition of the experiments $\mathbf{Exp}^{\text{Blind}}_{\mathsf{BS},\mathsf{S}^*}$ and $\mathbf{Exp}^{\text{OMUF}}_{\mathsf{BS},\mathsf{U}^*}$.

Lattices and Gaussians

Definition 7. *Let $\mathcal{L} \subset \mathbb{R}^m$ be a lattice, $\sigma \in \mathbb{R}_{>0}$, and $\mathbf{c} \in \mathbb{R}^m$. The* discrete Gaussian distribution over $\mathcal{L}$ with standard deviation σ and center $\mathbf{c}$ *is the probability distribution $D_{\mathcal{L},\sigma,\mathbf{c}}$ which assigns to every $\mathbf{x} \in \mathcal{L}$ 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 $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\mathcal{L}) := \sum_{\mathbf{x}\in\mathcal{L}} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. We will omit the subscript $\mathbf{c}$ when $\mathbf{c} = \mathbf{0}$.*

Next we recall a special version of the rejection sampling lemma related to the discrete Gaussian distribution [18, Theorem 4.6].

Lemma 8. *Let $T \in \mathbb{R}_{>0}$, and define $V := \{\mathbf{v} \in \mathbb{Z}^m \,|\, \|\mathbf{v}\| \leq T\}$. 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 $\exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2}) \leq M$, 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 $\bot$ otherwise.
(b) $\mathbf{v} \leftarrow\!\!\$\ h$, $\mathbf{z} \leftarrow\!\!\$\ D_{\mathbb{Z}^m,\sigma}$, output $(\mathbf{z}, \mathbf{v})$ with probability $1/M$, and $\bot$ otherwise.

Moreover, the probability that the first algorithm returns a value different from $\bot$ 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.

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 ($\mathsf{D\text{-}MLWE}$) 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 $\mathsf{D\text{-}MLWE}$ 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.

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

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

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

21: **parse** $pp = (n, q, k_1, k_2, \sigma, \mathbf{A})$
22: $b \leftarrow\!\!\$\ \{0,1\},\ \mathbf{b} \leftarrow\!\!\$\ R_q^{k_1}$
23: **if** $b = 1$ **then**
24: $\mathbf{s} \leftarrow\!\!\$\ D^{k_1+k_2}_{\mathbb{Z}^n,\sigma},\ \mathbf{b} \leftarrow [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{s} \pmod q$
25: $b^* \leftarrow\!\!\$\ \mathsf{D}^*(pp, \mathbf{b})$
26: **if** $b = b^*$ **then**
27: **return** 1
28: **return** 0

Fig. 3. Definition of the experiments $\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}$ and $\mathbf{Exp}^{\mathsf{D\text{-}MLWE}}_{\mathsf{D}^*}$.

Definition 9. *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 $\mathbf{Adv}^{\mathsf{MSIS}}_{\mathsf{A}^*}(pp) := \Pr[\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}(pp) = 1] \leq \varepsilon$, where the game $\mathbf{Exp}^{\mathsf{MSIS}}_{\mathsf{A}^*}$ is depicted in Fig. 3.*

Definition 10. *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 $\mathbf{A} \leftarrow\!\!\$\ R_q^{k_1 \times k_2}$. 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 $\mathbf{Adv}^{\mathsf{D\text{-}MLWE}}_{\mathsf{A}^*}(pp) := 2 \cdot |\Pr[\mathbf{Exp}^{\mathsf{D\text{-}MLWE}}_{\mathsf{A}^*}(pp) = 1] - \frac{1}{2}| \leq \varepsilon$, where the game $\mathbf{Exp}^{\mathsf{D\text{-}MLWE}}_{\mathsf{A}^*}$ is depicted in Fig. 3.*

Additional Preliminaries. In the full version of this paper [5] we provide a description of the partitioning and permutation technique [3], trees of commitments technique [4], and a minor modified version of the general forking lemma [6], which is used in the security proof of BlindOR. Here, we only give the required definitions.

We define by $\mathbb{T} := \{(-1)^b \cdot X^i | b \in \{0,1\},\ i \in \mathbb{Z}\}$ the set of signed permutation polynomials, which represent a rotation multiplied by a sign. The set $\mathbb{T}$ has an abelian group structure with respect to multiplication in R. The inverse of any $p = (-1)^b \cdot X^i \in \mathbb{T}$ is given by $p^{-1} = (-1)^{1-b} \cdot X^{n-i} \in \mathbb{T}$. When constructing OR-proofs, we will use the abelian group $\mathbb{T}^\kappa$ as the challenge space rather than $\mathbb{T}^n_\kappa$, since the latter does not have a group structure.

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:

HashTree is an algorithm that computes an (unbalanced) binary hash tree of height $h \geq 1$. On input $\ell \leq 2^h$ strings $v_0, \ldots, v_{\ell-1}$, it returns a pair $(root, tree)$, where *root* is the root of the hash tree whose leaves are hashes of $v_0, \ldots, v_{\ell-1}$, and *tree* is the sequence of all the other nodes in the tree.

BuildAuth is an algorithm that, on input an index $0 \leq k \leq \ell - 1$, a sequence of nodes *tree*, and a height h, returns the authentication path *auth* for k.

RootCalc is an algorithm that computes the root of a hash tree given a leaf and its authentication path.

3 BlindOR: a New Blind Signature Scheme

Sigma Protocol. 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 [17]. 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 $B_1' \subseteq B_1$. This relaxation is known to be sufficient for many cryptographic applications, *e.g.*, digital signatures [18]. Here we extend this line of applications to blind signatures.

BlindOR is built on a variant of the Sigma protocol introduced in [17], 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 $\mathbf{v} = [\mathbf{I}_{k_1} | \mathbf{A}] \cdot \mathbf{z} - \mathbf{b}c \pmod q$ 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

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

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. In Table 3 we propose a concrete tuple of such parameters targeting 128 bits of security. The relation set is then given by

$$\mathcal{R}.\mathsf{RSet}(pp) := \Big\{ (\mathbf{b}, (\bar{\mathbf{s}}, \bar{\mathbf{c}})) \in R_q^{k_1} \times (R^{k_1+k_2} \times R_q^{\kappa}) \;\Big|\; (\mathbf{b}\bar{c} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \bar{\mathbf{s}} \pmod q) \\ \wedge (\bar{\mathbf{c}} = (\bar{c}_1, \ldots, \bar{c}_\kappa) \in \overline{\mathcal{C}}) \wedge (\bar{c} = \sum_{j=1}^{\kappa} \bar{c}_j) \wedge (\|\bar{\mathbf{s}}\| \leq 2B_z) \Big\}\ , \qquad (1)$$

where $\overline{\mathcal{C}} = \{\mathbf{c} - \mathbf{c}' = (c_1 - c'_1, \ldots, c_\kappa - c'_\kappa) \mid \mathbf{c}, \mathbf{c}' \in \mathbb{T}^\kappa,\ \mathbf{c} \neq \mathbf{c}'\}$, and the instance generator is given in Fig. 4. 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 Fig. 4.

At a high level, the protocol can be seen as a generalized version of the one given in [17] 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 $D^{k_1+k_2}_{\mathbb{Z}^n,\sigma^*}$. Upon receiving the challenge $\mathbf{c} \in \mathbb{T}^\kappa$, the prover sends the first response $\mathbf{z}$ for which rejection sampling accepts, *i.e.*, for the masking vector $\mathbf{y}^{(i)}$ such that $\mathsf{Rej}(pp, \mathbf{z}) = 1$ 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 11. *Given the parameters in Table 2, the protocol depicted in Fig. 4 is a Sigma protocol for relation $\mathcal{R}$ given in Eq. (1).*

The proof is provided in the full version of this paper [5]. We remark that when constructing the Sigma protocol $\Sigma_{\mathsf{OR}} = \mathsf{OR}[\Sigma, \Sigma]$, 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 $\mathbf{c}_{1-d} = (c_{1,1-d}, \ldots, c_{\kappa,1-d}) \hookleftarrow \mathbb{T}^\kappa$ 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 $\mathbf{c} = \mathbf{c}_d \cdot \mathbf{c}_{1-d} = (c_{1,d}c_{1,1-d}, \ldots, c_{\kappa,d}c_{\kappa,1-d})$.

Description of BlindOR. Let BS be a blind signature scheme as defined in Sect. 2. 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 $sig = (ch, rp)$. 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

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

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** $\|\mathbf{s}\| \leq B_s$
  $\mathbf{b} \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{s} \pmod q$
  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** κ **do**
    $\mathbf{y}_j \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$
    $\mathbf{v}_j \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{y}_j \pmod q$
  $\mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
  $\mathbf{y}^{(i)} \leftarrow (\mathbf{y}_1, \ldots, \mathbf{y}_\kappa)$
$\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
$st_{\Sigma.\mathsf{P}} \leftarrow (\mathbf{y}^{(0)}, \ldots, \mathbf{y}^{(\omega-1)})$
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 $(\mathbf{c}, st_{\Sigma.\mathsf{V}})$

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

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** κ **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 $\perp$

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

parse $\mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
parse $\mathbf{z}' = (\mathbf{z}'_1, \ldots, \mathbf{z}'_\kappa)$
$\bar{\mathbf{s}} \leftarrow \sum_{j=1}^{\kappa} (\mathbf{z}_j - \mathbf{z}'_j)$
$\bar{\mathbf{c}} \leftarrow \mathbf{c} - \mathbf{c}'$
return $(\bar{\mathbf{s}}, \bar{\mathbf{c}})$

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

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** κ **do**
  $\mathbf{w}_j \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
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** κ **do**
    if $\mathbf{w}_j = \mathbf{v}_j$ **then**
      $int = int + 1$
  if $int = \kappa$ **then**
    return $(1, i)$
return $(0, -1)$

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

return $(\perp, \perp)$ with probability δ^*
parse $\mathbf{c} = (c_1, \ldots, c_\kappa)$
$i \leftarrow\!\!\$\ \{0, \ldots, \omega - 1\}$
for $j = 1$ **to** κ **do**
  $\mathbf{z}_j \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$
  $\mathbf{v}_j \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
$\mathbf{z} \leftarrow (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa),\ \mathbf{v}^{(i)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
for $k = 0$ **to** $\omega - 1$ **do**
  if $k = i$ **then**
    continue
  for $j = 1$ **to** κ **do**
    $\mathbf{y}_j \leftarrow\!\!\$\ D_{\mathbb{Z}^n,\sigma^*}^{k_1+k_2}$
    $\mathbf{v}_j \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{y}_j \pmod q$
  $\mathbf{v}^{(k)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
$\mathbf{v} \leftarrow (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)})$
return $(\mathbf{v}, \mathbf{z})$

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

if $\|\mathbf{z}\| > B_z$ **then**
  return $\perp$
parse $\mathbf{c} = (c_1, \ldots, c_\kappa),\ \mathbf{z} = (\mathbf{z}_1, \ldots, \mathbf{z}_\kappa)$
for $j = 1$ **to** κ **do**
  $\mathbf{v}_j \leftarrow \left[\mathbf{I}_{k_1} \mid \mathbf{A}\right] \cdot \mathbf{z}_j - \mathbf{b}c_j \pmod q$
$\mathbf{v}^{(0)} \leftarrow (\mathbf{v}_1, \ldots, \mathbf{v}_\kappa)$
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 $\mathbf{v}$

Fig. 4. The Sigma protocol underlying BlindOR. Prover restarts Σ if $\Sigma.\mathsf{P}_2$ returns $\perp$.

a witness for a statement, the steps performed by the user consist of blinding the transcript (cm^*, ch^*, rp^*) during interaction. In BlindOR, we capture these blinding steps by algorithms Com, Cha, and Rsp, which we describe next.

For the remainder of this section we let Σ be the Sigma protocol depicted in Fig. 4. Furthermore, let $h = \lceil \log(\omega\ell) \rceil$ and define the bijective mapping $\mathsf{IntIdx}\colon \{0,\ldots,\omega-1\} \times \{0,\ldots,\ell-1\} \to \{0,\ldots,\omega\ell-1\}$, $(i,k) \mapsto k + i\ell$. IntIdx converts the pair (i,k) to a unique positive integer. This is used in BlindOR to build authentication paths via the algorithm BuildAuth. Let pp be a set of public parameters for 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 Fig. 5:

Com is a PPT algorithm that, on input pp, 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 using 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 Fig. 5 can be seen as a generalized version of the blinding steps implicitly presented in the lattice-based blind signature scheme BLAZE$^+$ [4]. Unlike BLAZE$^+$, the algorithms shown in Fig. 5 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 $\mathbb{T}_\kappa^n$, 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 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 algorithm IterateRej returns $(\perp, \perp)$. 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

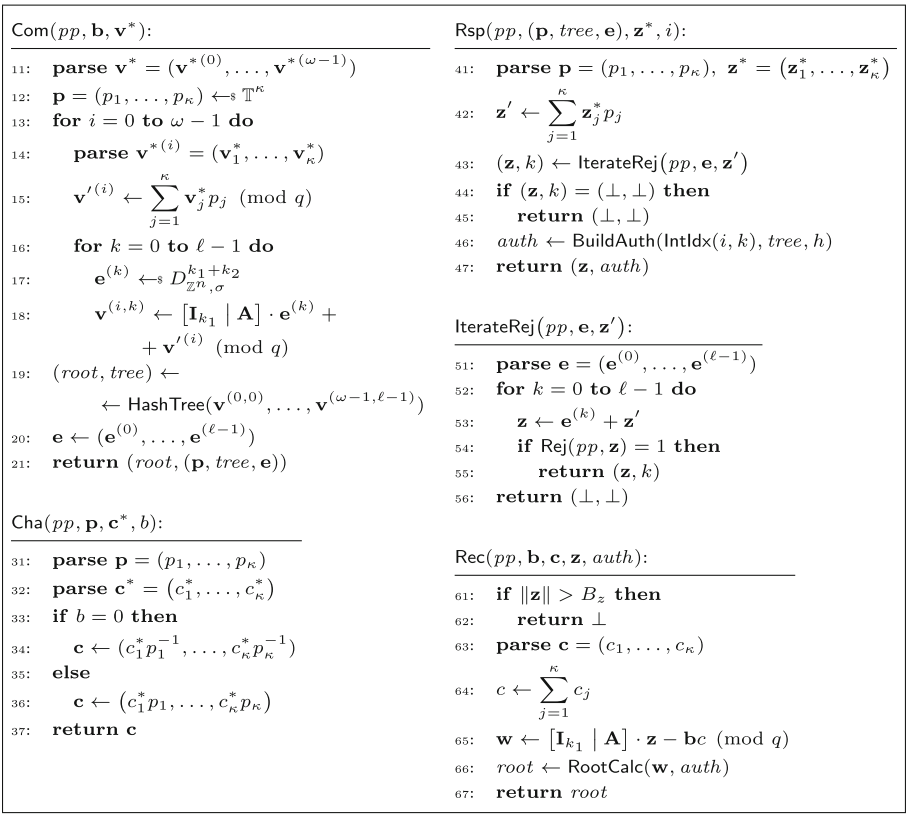

Fig. 5. A formal description of algorithms Com, Cha, Rsp, and Rec.

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 describe BlindOR. 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 hash functions, where $\ell_\mathsf{F} \geq 2\lambda$ and $\mathbb{T}^\kappa$ is the challenge space of Σ. The algorithm BS.PGen generates and returns a set of public parameters $pp = (1^\lambda, n, k_1, k_2, q, \omega, \ell, h, \kappa, \sigma', \sigma^*, \sigma, S, M, B_s, B_{z^*}, B_z, \ell_\mathsf{F}, \mathbf{A})$. 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.*, 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. The remaining algorithms of BlindOR are formalized in Fig. 6.

In Table 3, we propose concrete parameters for BlindOR targeting 128 bits of security. Next, we state the correctness, blindness, and one-more unforgeability

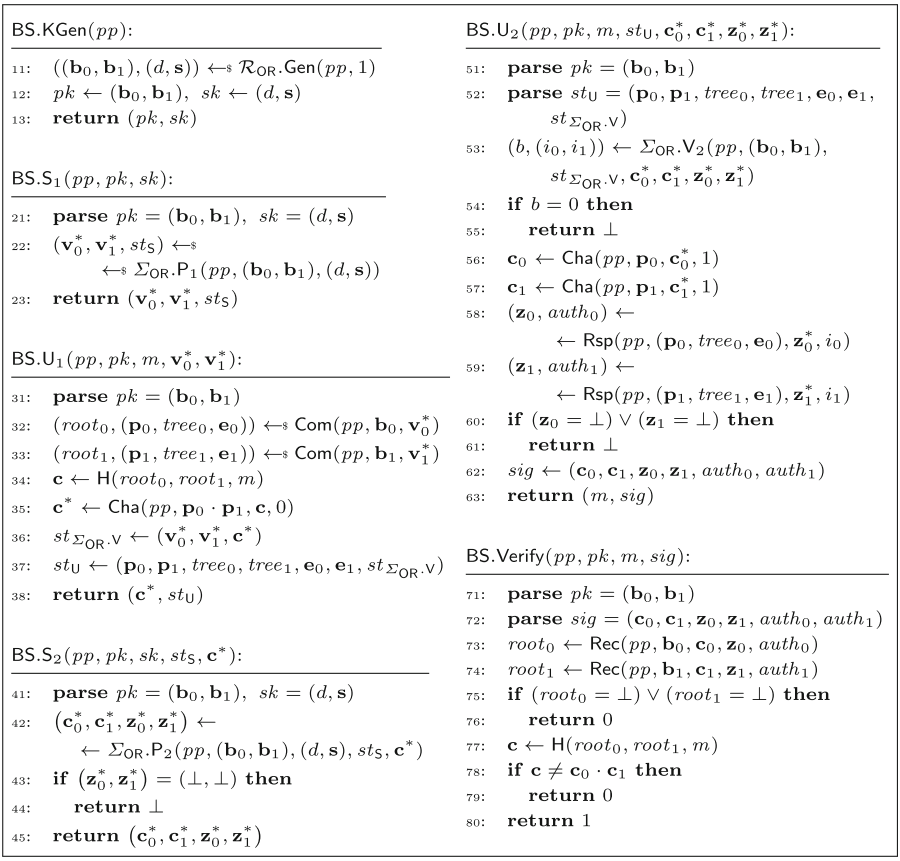

Fig. 6. A formal description of BlindOR. Signer restarts the protocol if $(\mathbf{z}_0^*, \mathbf{z}_1^*) = (\perp, \perp)$.

of BlindOR. We provide the description of the parameter selection as well as the proof of correctness in the full version of this paper [5].

Theorem 12. *Given the parameters in Table 2,* BlindOR *is* $\mathsf{corr}_{\mathsf{BS}}$-correct *w.r.t. pp, 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* $\perp$.

Theorem 13. *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,* BlindOR *is* ε*-statistically blind w.r.t. pp 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 Fig. 2. 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

Table 2. A review of the parameters of BlindOR.

Parameter	Description	Bounds
n, k_1, k_2	Dimension	$n = 2^{n'}$, $n', k_1, k_2 \in \mathbb{Z}_{\geq 1}$
q	Modulus	prime, $q = 2p+1 \pmod{4p}$, $n \geq p > 1$, $p = 2^{p'}$, $p' \in \mathbb{Z}_{\geq 1}$, $q^{1/p} > 2\kappa$
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	$\sigma^* = \alpha^*\sqrt{\kappa}B_s$, $S = \exp(\frac{12}{\alpha^*} + \frac{1}{2\alpha^{*2}})$, $(1 - \frac{1-2^{-100}}{S})^\omega \leq \delta^*$, $\delta^* > 0$
σ	Standard deviation in BS.U	$\sigma = \alpha\eta\frac{B_{z^*}}{\eta^*}$, $U = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$, $(1 - \frac{1-2^{-100}}{U})^\ell \leq \delta$, $\delta > 0$
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	$B_z = \eta\sigma\sqrt{(k_1+k_2)n}$, $\eta > 0$
ℓ_F	Output length of F	$\ell_\mathsf{F} \geq 2\lambda$

Table 3. Concrete parameters of BlindOR targeting 128 bits of security.

n	k_1	k_2	q	ω	ℓ	h	κ	σ'	α^*	σ^*	α	σ	M	ℓ_F
256	5	4	$\approx 2^{33}$	1	8	3	15	4	11	8344	41	71230016	3	384

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, [3, Lemma 4] 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 $\mathbf{z} = \mathbf{e} + \sum_{j=1}^{\kappa} \mathbf{z}_j^* p_j$ (see Fig. 5). By Lemma 8, both vectors completely mask $\sum_{j=1}^{\kappa} \mathbf{z}_j^* p_j$ and are independently distributed within statistical distance of $2^{-100}/U$ from $D_{\mathbb{Z}^n,\sigma}^{k_1+k_2}$.

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

Theorem 14. *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,* BlindOR *is* $(t, q_{\mathsf{Sign}}, q_\mathsf{F}, q_\mathsf{H}, \varepsilon)$*-one-more unforgeable w.r.t. pp in the ROM if* D-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.* $pp_{\mathsf{MSIS}} = (n, q, k_1, k_2+1, 2\sqrt{B_z^2 + \kappa^2})$*. More precisely, if there exists a forger* A^*

against BlindOR *w.r.t. pp that returns* $q_{\mathsf{Sign}}+1$ *blind signatures in time* t *and with probability* ε*, and after making* $q_{\mathsf{F}}, q_{\mathsf{H}}$ *queries to* F, H*, respectively, then* A^* *can be used to solve* D-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*

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

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

Proof. First we observe that the hardness of D-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 D-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 matrix $\mathbf{A}' \in R_q^{k_1 \times (k_2+1)}$, 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 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 by 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 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.

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.

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 $\varepsilon - \frac{q_{\mathsf{Sign}}+1}{|\mathbb{T}^\kappa|}$.

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 $C' = \{\mathbf{c}_1, \ldots, \mathbf{c}_{j^*-1}, \mathbf{c}'_{j^*}, \ldots, \mathbf{c}'_{q_\mathsf{H}}\}$, where $\mathbf{c}'_{j^*}, \ldots, \mathbf{c}'_{q_\mathsf{H}} \in \mathbb{T}^\kappa$ are freshly generated by R. After rewinding, A^* returns $q_\mathsf{Sign}+1$ pairs of distinct messages and their valid signatures. The potential two valid forgeries (before and after rewinding) output by A^* at index i^* have the form

$$(m, (\mathbf{c}_0, \mathbf{c}_1, \mathbf{z}_0, \mathbf{z}_1, auth_0, auth_1)) \text{ and } (m', (*)\mathbf{c}'_0, \mathbf{c}'_1, \mathbf{z}'_0, \mathbf{z}'_1, auth'_0, auth'_1) \ ,$$

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

$$\begin{gathered}
\mathbf{w}_{1-d} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}_{1-d} - \mathbf{b}_{1-d} c_{1-d} \pmod q \ , \\
\mathbf{w}'_{1-d} = [\mathbf{I}_{k_1} \mid \mathbf{A}] \cdot \mathbf{z}'_{1-d} - \mathbf{b}_{1-d} c'_{1-d} \pmod q \ , \\
root_{1-d} = \mathsf{RootCalc}(\mathbf{w}_{1-d}, auth_{1-d}), \ root'_{1-d} = \mathsf{RootCalc}(\mathbf{w}'_{1-d}, auth'_{1-d}) \ , \\
\mathbf{c}_0 \cdot \mathbf{c}_1 = \mathbf{c} = \mathsf{H}(root_0, root_1, m), \ \mathbf{c}'_0 \cdot \mathbf{c}'_1 = \mathbf{c}' = \mathsf{H}(root'_0, root'_1, m') \ ,
\end{gathered}$$

By the forking lemma (see the full version [5]) we have $root_0 = root'_0$, $root_1 = root'_1$, $m = m'$, $\mathbf{c} \neq \mathbf{c}'$, and $k_{1-d} = k'_{1-d}$, where $k_{1-d}, k'_{1-d} \in \{0, \ldots, \omega\ell - 1\}$ are the indices included in $auth_{1-d}, auth'_{1-d}$, respectively. Observe that simulating the hash queries to F as described above ensures that both $auth_{1-d}, auth'_{1-d}$ include the same sequence of hash values, and hence $auth_{1-d} = auth'_{1-d}$ and $\mathbf{w}_{1-d} = \mathbf{w}'_{1-d}$. If $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$, then we have

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

where $c_{1-d} = \sum_{j=1}^{\kappa} c_{j,1-d}$ and $c'_{1-d} = \sum_{j=1}^{\kappa} c'_{j,1-d}$. In this case, R runs $\Sigma.\mathsf{Ext}$ on input $(pp, \mathbf{b}_{1-d}, (\mathbf{v}, \mathbf{c}, \mathbf{z}), (\mathbf{v}, \mathbf{c}', \mathbf{z}'))$, where

$$\begin{gathered}
\mathbf{v} = (\mathbf{v}^{(0)}, \ldots, \mathbf{v}^{(\omega-1)}), \ \mathbf{v}^{(0)} = (\mathbf{w}_{1-d}, \mathbf{0}, \ldots, \mathbf{0}) \in (R_q^{k_1})^\kappa, \\
\mathbf{v}^{(i)} = (\mathbf{0}, \ldots, \mathbf{0}) \in (R_q^{k_1})^\kappa \text{ for all } i \in [\omega - 1], \ \mathbf{z} = (\mathbf{z}_{1-d}, \mathbf{0}, \ldots, \mathbf{0}) \in (R_q^{k_1})^\kappa, \\
\mathbf{z}' = (\mathbf{z}'_{1-d}, \mathbf{0}, \ldots, \mathbf{0}) \in (R_q^{k_1})^\kappa, \ \|\mathbf{z}_{1-d}\| \leq B_z, \ \|\mathbf{z}'_{1-d}\| \leq B_z.
\end{gathered}$$

The output of $\Sigma.\mathsf{Ext}$ is the pair $(\mathbf{z}_{1-d} - \mathbf{z}'_{1-d}, \mathbf{c}_{1-d} - \mathbf{c}'_{1-d})$, which gives the non-trivial solution $[\mathbf{z}_{1-d} - \mathbf{z}'_{1-d} \,|\, c'_{1-d} - c_{1-d}]^\top$ 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 $1/(q_\mathsf{Sign}+1)$. The success probability of the forking is given by $frk \geq acc \cdot (\frac{acc}{(q_\mathsf{Sign}+1)\omega\ell} - \frac{1}{|\mathbb{T}^\kappa|})$, where $acc = (\varepsilon - \frac{q_\mathsf{F}^2 + q_\mathsf{F}}{2^{\ell_\mathsf{F}}} - \frac{q_\mathsf{Sign}+1}{|\mathbb{T}^\kappa|}) \Big/ q_\mathsf{H}^{q_\mathsf{Sign}+1}$. By Lemma 15, the probability that $\mathbf{c}_{1-d} \neq \mathbf{c}'_{1-d}$ is given by $\frac{1}{2} - \varepsilon'$. This deduces the probability ε'' that is given in the theorem statement. □

Lemma 15. *Assume that after rewinding the forger* A^* *by the reduction* R *given in Theorem 14, 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* $\mathbf{Exp}^{\mathsf{D\text{-}MLWE}}_{\mathsf{D}^*}$ *with the advantage* ε'*.*

The proof is provided in the full version of this paper [5].

Acknowledgments. We thank Marc Fischlin for helpful discussions. This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297.

References

1. Abe, M., Okamoto, T.: Provably secure partially blind signatures. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 271–286. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-44598-6_17
2. Agrawal, S., Yadav, A.L.: Towards practical and round-optimal lattice-based threshold and blind signatures. Cryptology ePrint Archive, Report 2021/381
3. Alkeilani Alkadri, N., El Bansarkhani, R., Buchmann, J.: BLAZE: practical lattice-based blind signatures for privacy-preserving applications. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 484–502. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_26
4. Alkeilani Alkadri, N., El Bansarkhani, R., Buchmann, J.: On lattice-based interactive protocols: an approach with less or no aborts. In: Liu, J.K., Cui, H. (eds.) ACISP 2020. LNCS, vol. 12248, pp. 41–61. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55304-3_3
5. Alkeilani Alkadri, N., Harasser, P., Janson, C.: BlindOR: an efficient lattice-based blind signature scheme from or-proofs. Cryptology ePrint Archive, Report 2021/1385
6. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: ACM CCS 2006
7. Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: ACM CCS 93
8. Camenisch, J., Neven, G., Shelat: Simulatable adaptive oblivious transfer. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 573–590. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_33
9. Chaum, D.: Blind signatures for untraceable payments. In: Chaum, D., Rivest, R.L., Sherman, A.T. (eds.) Advances in Cryptology. LNCS, pp. 199–203. Springer, Boston, MA (1983). https://doi.org/10.1007/978-1-4757-0602-4_18
10. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-48658-5_19
11. Fischlin, M.: Round-optimal composable blind signatures in the common reference string model. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 60–77. Springer, Heidelberg (2006). https://doi.org/10.1007/11818175_4
12. Fischlin, M., Schröder, D.: Security of blind signatures under aborts. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 297–316. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00468-1_17

13. Garg, S., Rao, V., Sahai, A., Schröder, D., Unruh, D.: Round optimal blind signatures. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 630–648. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_36
14. Hauck, E., Kiltz, E., Loss, J.: A modular treatment of blind signatures from identification schemes. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019. LNCS, vol. 11478, pp. 345–375. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17659-4_12
15. Rückert, M.: Lattice-based blind signatures. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 413–430. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_24
16. Juels, A., Luby, M., Ostrovsky, R.: Security of blind digital signatures (extended abstract). In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 150–164. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052233
17. Lyubashevsky, V.: Fiat-shamir with aborts: applications to lattice and factoring-based signatures. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 598–616. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10366-7_35
18. Lyubashevsky, V.: Lattice signatures without trapdoors. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 738–755. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_43
19. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. J. Cryptology **13**(3), 361–396 (2000)
20. Rückert, M.: Lattice-based blind signatures. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 413–430. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17373-8_24
21. Schröder, D., Unruh, D.: Security of blind signatures revisited. J. Cryptology, **30**(2), 470–494 (2017)