Practical Lattice-Based Distributed Signatures for a Small Number of Signers

Nabil Alkeilani Alkadri✉, Nico Döttling, and Sihang Pu

CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
{nabil.alkadri,doettling,sihang.pu}@cispa.de

Abstract. n-out-of-n distributed signatures are a special type of threshold t-out-of-n signatures. They are created by a group of n signers, each holding a share of the secret key, in a collaborative way. This kind of signatures has been studied intensively in recent years, motivated by different applications such as reducing the risk of compromising secret keys in cryptocurrencies. Towards maintaining security in the presence of quantum adversaries, Damgård et al. (J Cryptol 35(2), 2022) proposed lattice-based constructions of n-out-of-n distributed signatures and multi-signatures following the Fiat-Shamir with aborts paradigm (ASIACRYPT 2009). Due to the inherent issue of aborts, the protocols either require to increase their parameters by a factor of n, or they suffer from a large number of restarts that grows with n. This has a significant impact on their efficiency, even if n is small. Moreover, the protocols use trapdoor homomorphic commitments as a further cryptographic building block, making their deployment in practice not as easy as standard lattice-based Fiat-Shamir signatures. In this work, we present a new construction of n-out-of-n distributed signatures. It is designed specifically for applications with small number of signers. Our construction follows the Fiat-Shamir with aborts paradigm, but solves the problem of large number of restarts without increasing the parameters by a factor of n and utilizing any further cryptographic primitive. To demonstrate the practicality of our protocol, we provide a software implementation and concrete parameters aiming at 128 bits of security. Furthermore, we select concrete parameters for the construction by Damgård et al. and for the most recent lattice-based multi-signature scheme by Chen (CRYPTO 2023), and show that our approach provides a significant improvement in terms of all efficiency metrics. Our results also show that the multi-signature schemes by Damgård et al. and Chen as well as a multi-signature variant of our protocol produce signatures that are not smaller than a naive multi-signature derived from the concatenation of multiple standard signatures.

Keywords: n-out-of-n distributed signatures · threshold n-out-of-n signatures · Fiat-Shamir with aborts · lattice-based cryptography

1 Introduction

An n-out-of-n distributed signature is a signature on a single message that is jointly generated by a group of n signers. Before signing this message, the signers invoke a key generation protocol to create a pair of public and secret key, where each signer learns the public key and a share of the secret key only. The signature can be verified by the public key. An n-out-of-n distributed signature is a special type of threshold t-out-of-n signatures [18], hence also called a threshold n-out-of-n signature. The required security property of n-out-of-n distributed signatures is that it should be infeasible to generate a valid signature even if at most $n-1$ signers are corrupted. Distributed signature protocols constitute a fundamental cryptographic primitive, most notably, in the blockchain domain when it comes to authorize transactions in the presence of multiple signers. This minimizes the risk of compromising the secret key. Distributed signatures can provide institutional and personal account key management by multiple people.

Currently, real-life applications employing distributed signatures rely on constructions whose security is based on the hardness of number-theoretic assumptions. However, when taking into account recent developments of quantum computers, it is meanwhile known that these assumptions cannot be used long-term. In an effort to develop new constructions that are conjectured to be secure in the presence of quantum computers, few works based on lattice assumptions considered threshold t-out-of-n signatures (see Appendix A). Recently, Damgård et al. [17] proposed a lattice-based construction of n-out-of-n distributed signatures following the Fiat-Shamir with aborts paradigm [25]. Hence, it relies on the so-called *rejection sampling* when generating signatures. In the context of Fiat-Shamir signatures based on lattices, rejection sampling is a crucial tool that is used as a security check. Using a so-called masking term, it allows to verify that a secret (or secret-related) term is concealed and distributed independently from a public term that is computed using both the masking and secret term. In particular, it makes sure that signatures are distributed independently from the secret key. If the check fails, i.e., if rejection sampling rejects, the signing protocol is restarted in order to sample a fresh masking term. This is because all computations carried out up to rejection sampling are related to a certain masking term. In interactive protocols with multiple rejection sampling procedures such as distributed signatures, this has a significant negative impact on the efficiency. To see this, suppose that n signers would like to generate a signature, and each one has to restart the protocol M times on average, where M is determined in accordance to other parameters of the protocol (the smaller M is, the larger the signature size). Then, the total average number of restarts in such a protocol is given by M^n, which is large even for a small n. One way to make this number reasonably small is to increase the parameters by a factor of n [17]. However, this induces larger sizes of keys and signatures, even if n is small. In addition to the large number of restarts, the construction by Damgård et al. [17] uses a trapdoor homomorphic commitment scheme as a further cryptographic building block, which affects its efficiency in a significant way.

Due to the above mentioned drawbacks, we conclude that the protocol of [17] is not suitable for deployment in practice, even when it comes to applications with small number of signers, which is our main focus in this work and is apparent in real-world applications as we argue in the following. Consider the well-known problem of fraud management (CEO fraud), where a fraudster pretends to be a senior manager – often the CEO – in order to persuade a staff member to make an urgent payment to a supplier or business partner. This kind of social engineering can be easily prevented by employing distributed signatures. By means of a defined policy, contracts are only authorized by various decision makers, typically 2 or 3. In cryptocurrencies, distributed signatures add a great level of security by requiring few devices to authenticate transactions. A so-called *distributed wallet* requires all signers involved in the generation of such a wallet to agree before any transaction can be created. For instance, wallets developed by *Armory* allow at most 7 signers to authorize a transaction, while those developed by *BitGo* and *Coinbase* provide up to 3 signers[1].

1.1 Contribution

We present a new construction of n-out-of-n distributed signatures based on lattices over modules. It is designed to support applications with a small number of signers only. Similar to the protocol by Damgård et al. [17], our construction follows the Fiat-Shamir with aborts approach, but solves its drawbacks that we mentioned above. More precisely, it solves the problem of the large number of restarts without increasing the parameters by a factor of n, and it does not rely on any additional primitives like trapdoor homomorphic commitments. It also supports the *offline-online* paradigm. This feature is desirable in practice, since it allows expensive operations of our signing protocol to be pre-processed. Using the *rewinding* technique [7], we prove the security of our construction in the random oracle model [8] assuming the hardness of the Module Learning With Errors (MLWE) problem and Module Small Integer Solution (MSIS) problem. We provide a proof-of-concept implementation of our protocol demonstrating its practicality, and propose concrete parameters targeting 128 bits of security. In order to give a fair comparison, we also select concrete parameters for the distributed signature and multi-signature protocols by Damgård et al. [17] and for the most recent multi-signature protocol by Chen [13]. The reason of considering multi-signatures is because a distributed signature protocol can be derived from [13], and conversely we can derive a multi-signature scheme from our construction (see Section 1.3 for more details). The comparison is summarized in Table 1, which shows that our approach provides a significant improvement regarding all efficiency metrics.

1.2 Technical Overview

Similar to [17], our construction follows the Fiat-Shamir with aborts approach and can be seen as a distributed variant of the standard signature scheme $\mathsf{Dilithium\text{-}G}$ [21]. Basically, the key generation protocol generates an instance of the MLWE problem in a distributed way. Therefore, we give in this overview a high-level explanation of our signing protocol only. For ease of exposition, we

[1] `https://coinsutra.com/best-multi-signature-bitcoin-wallets/`

Table 1. Comparison between our construction, the constructions of distributed signatures and multi-signatures introduced by Damgård et al. [17], and the multi-signature scheme by Chen [13]. The corresponding parameters are given in Table 3 and consider 7 signers. Performance measures are only provided for our protocol, since no implementations are given in [17,13]. Parameter selection is described in Section 3.3. All numbers are rounded to the nearest integer. Sizes and communication costs are provided in kilobytes, while performance measures in milliseconds.

Protocol	Sizes			Communication per signer		Performance per signer		
	Public key	Distributed signature	Multi-signature	Key generation	Signature generation	Key generation	Signature generation	Signature verification
This work	7	12	12	7	25	2	41	2
[17]	21	2538	2594	21	2536	-	-	-
[13]	8	-	25	-	53	-	-	-

consider a group of just two signers. A generalization to $n \geq 2$ signers can be derived in a straightforward manner and is considered in Section 3. Our protocol operates over the rings $R = \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_q = \mathbb{Z}_q[X]/\langle X^N + 1\rangle$. Let $\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}]$ be a joint public matrix, where $\mathbf{I}_k$ is the identity matrix of dimension k and $\mathbf{A} \in R_q^{k\times\ell}$ is uniformly random. The public and secret key share of each signer are given by $(pk, sk_j) = ((\bar{\mathbf{A}}, \mathbf{b}), \mathbf{s}_j)$, where $\mathbf{b} = \bar{\mathbf{A}} \cdot \mathbf{s} \pmod q$, $\mathbf{s} = \mathbf{s}_1 + \mathbf{s}_2 \in R^{k+\ell}$, and $j \in \{1, 2\}$. Each share $\mathbf{s}_j$ is chosen uniformly random over a small subset of R. In Fig. 1, we present an informal overview of our signing protocol. We only present the behavior of signer S_1, as each signer plays the same role and performs the same steps. Note that all operations up to computing g_1 can be pre-processed without knowledge of the message. The remaining steps are carried out online. This reflects the support of offline-online paradigm. In the following we highlight the major techniques and differences to [17].

Removing Homomorphic Commitments. In lattice-based Fiat-Shamir signatures like Dilithium-G [21], a signer computes a *commitment* $\mathbf{v}$ to some randomness $\mathbf{y}$. In our case, $\mathbf{y}$ follows the Gaussian distribution $D^{k+\ell}_{\mathbb{Z}^N,\sigma}$ with standard deviation σ. Together with the message m, $\mathbf{v}$ is used to generate a *challenge* c via a cryptographic hash function Hash. Then, a *response* $\mathbf{z}$ is computed as $\mathbf{z} = \mathbf{y} + \mathbf{s}c$. After that, rejection sampling is carried out on $\mathbf{z}$ in order to make sure that $\mathbf{z}$ is distributed independently from $\mathbf{s}c$. The signing process is restarted if rejection sampling does not accept. Otherwise, the signature is given by $(\mathbf{z}, c)$. In n-out-of-n distributed signatures, the challenge is created via the sum of the commitments of all signers, and the final response is given by aggregating all individual responses. This means that each signer reveals its own commitment whether rejection sampling accepts or not. As indicated in [17], security cannot be proven if commitments of aborted executions are revealed. To circumvent this issue, [17] employ a lattice-based homomorphic commitment scheme whose statistical hiding property ensures that no information can be leaked from a commitment to $\mathbf{v}$, which is only revealed if rejection sampling is successful. Our approach is to use a specific regularity property [27]. It ensures that aborted

```
$\mathsf{S}_1(st, pk = (\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}], \mathbf{b}), sk_1 = \mathbf{s}_1, \text{message } m)$

$st = (1, \langle L \rangle)$, where $\langle L \rangle$ is a unique encoding of $L = (\mathbf{b}_1, \mathbf{b}_2)$ and
$\mathbf{b}_1 = \bar{\mathbf{A}} \cdot \mathbf{s}_1 \pmod q$, $\mathbf{b}_2 = \bar{\mathbf{A}} \cdot \mathbf{s}_2 \pmod q$

sample $\mathbf{y}_0, \mathbf{y}_1, \mathbf{y}_2$ from the Gaussian distribution $D^{k+\ell}_{\mathbb{Z}^N,\sigma}$
compute $\mathbf{v}_t = \bar{\mathbf{A}} \cdot \mathbf{y}_t \pmod q$ for all $t \in \{0,1,2\}$
set $\mathbf{v} = (\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2)$ and compute $g_1 = \mathsf{Hash}(\mathbf{v}, \mathbf{b}_1)$
                                                        $\xrightarrow{(1,g_1)}$
                                                        $\xleftarrow{(2,g_2)}$
                                                        $\xrightarrow{\mathbf{v}}$
                                                        $\xleftarrow{\mathbf{u}}$
check $\mathsf{Hash}(\mathbf{u}, \mathbf{b}_2) = g_2$, where $\mathbf{u} = (\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2)$
compute $\mathbf{w}_0 = \mathbf{v}_0 + \mathbf{u}_0$, $\mathbf{w}_1 = \mathbf{v}_0 + \mathbf{u}_1$, $\mathbf{w}_2 = \mathbf{v}_0 + \mathbf{u}_2$, $\ldots$, $\mathbf{w}_8 = \mathbf{v}_2 + \mathbf{u}_2$
compute $root$: the root of hash tree whose leaves are hash values of $\mathbf{w}_0, \ldots, \mathbf{w}_8$
compute $c = \mathsf{Hash}(root, m, \mathbf{b})$
find $i \in \{0,1,2\}$ such that rejection sampling on $\mathbf{z}_1 = \mathbf{y}_i + \mathbf{s}_1 c$ accepts
set $\mathbf{z}_1 = \perp$ if rejection sampling rejects for all $i$
                                                        $\xrightarrow{\mathbf{z}_1}$
                                                        $\xleftarrow{\mathbf{z}_2}$
restart if $\mathbf{z}_1 = \perp \ \vee \ \mathbf{z}_2 = \perp$
check $\|\mathbf{z}_2\|$ is small and $\bar{\mathbf{A}} \cdot \mathbf{z}_2 - \mathbf{b}_2 c = \mathbf{u}_j$ for some $j \in \{0,1,2\}$
compute $\mathbf{z} = \mathbf{z}_1 + \mathbf{z}_2$
let $i = 1$ and $j = 2$
compute $auth$: the authentication path of $\mathbf{w}_5 = \mathbf{v}_1 + \mathbf{u}_2$ in the hash tree
output $sig = (c, \mathbf{z}, auth)$ as a distributed signature on $m$
```

Fig. 1. Overview of our n-out-of-n distributed signature protocol. For simplicity, we consider in this overview a group of two signers only, where each signer computes and sends commitments to three Gaussian distributed masking vectors. The choice of three Gaussian masking vectors is only for presentation purposes.

executions do not leak any information so that security can be proven without utilizing any additional primitive. More concretely, increasing the standard deviation σ of $\mathbf{y}$ according to the regularity property makes the distribution of $\mathbf{v}$ statistically close to uniform over R_q^k. As demonstrated in Table 1, this improves the performance and complexity of the protocol, and produces much shorter signatures. We remark that the same regularity property must be satisfied by the commitment scheme used in [17]. In other words, we directly endow $\mathbf{v}$ with the regularity property, without having to compute an additional homomorphic commitment to $\mathbf{v}$ that is statistically hiding by the regularity property.

Removing Restarts. [17] suggest to increase σ by a factor of n in order to reduce the large number of protocol restarts inherent in carrying out rejection sampling. We use the tree of commitments technique by Alkadri et al. [3], where each signer generates and sends a specified number of commitments to the remaining signers. For example, in Fig. 1 we let signer S_1 generate three commitments $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$. Then, each one is added to $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2$ individually, where

$\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2$ are the commitments of signer S_2. After that, each signer computes a binary hash tree whose leaves are the hash values of the aggregated commitments $\mathbf{w}_0, \ldots, \mathbf{w}_8$. The root of this hash tree is used to generate the challenge c. Note that the input of the signing protocol includes a state information $st = (1, \langle L \rangle)$, where $\langle L \rangle$ is a unique encoding (e.g., lexicographical ordering) of the list of public key shares $L = (\mathbf{b}_1, \mathbf{b}_2)$. This state is obtained during executing the key generation protocol. Here, we assume, w.l.o.g., that $\mathbf{b}_1$ is the first entry of $\langle L \rangle$. The unique encoding of signers is crucial in our construction, since otherwise each signer may compute a different hash tree, and hence a different challenge. In order to obtain the same local ordering by each signer and allow to map them to the index of their public key shares in $\langle L \rangle$, each signer sends the entry of its public key share in the first round of the protocol. The correctness of the ordering is verified when checking the validity of the hash values (see Fig. 1). The commitments generated by each signer allow to carry out rejection sampling up to three times without the need to restart the signing protocol. The response for which rejection sampling accepts is sent out, and the signature is given by $(c, \mathbf{z}, \mathit{auth})$, where $\mathbf{z}$ is the aggregated response, and *auth* is the authentication path of the aggregated commitment corresponding to the used masking vectors. Using this technique not only reduces the number of restarts, or removes it at all, but also reduces the signature size. This is because an aggregated (homomorphic) commitment together with its opening, which are part of the signature in [17], are replaced with the root of the tree *root* and an authentication path *auth*, where the pair $(\mathit{root}, \mathit{auth})$ is just a short sequence of hash values. However, the number of commitment additions is given by ω^n, where ω is the number of commitments created by each of the n signers (in Fig. 1 we have $\omega = 3$ and $n = 2$). In other words, the computational complexity is $O(\omega^n)$. This is why our construction is only suitable for a limited number of signers, which is sufficient for several applications as demonstrated in the introduction. We note that in Fig. 1 we set $\omega = 3$ only for presentation purposes. In practice, ω is selected such that the protocol is restarted with a probability of choice. In our sample parameters, ω is selected such that each signer aborts with probability ≈ 0.

Round Complexity. Our signing protocol consists of three rounds (see Fig. 1). The first round, where the hash value of the commitments is sent, is a standard technique that is required to prove security. It allows the security reduction to simulate honest signer by extracting the commitments of corrupt signers from incoming queries to the hash function Hash (when modeled as random oracle). It also allows programming Hash accordingly before revealing the commitments. This standard first round is removed in [17] by adding a trapdoor feature to the homomorphic commitment scheme, while in [13] a straight-line simulation via a dual secret key is used. In order to reduce the complexity to two rounds, our construction can be instantiated with a trapdoor as in [12]. We choose to keep the first round instead in order to obtain an improved communication cost and smaller sizes of public keys and signatures.

1.3 Related Work

In Appendix A, we provide related work on lattice-based threshold signatures.

Multi-Signatures. A multi-signature scheme [23] resembles n-out-of-n distributed signature protocols. The differences are (1) each signer has its own key pair, i.e., it locally generates its public and secret key, where the public key is published before signing, (2) the group of signers is not required to be fixed, and each signer can initiate the signing protocol with a set of signers of its choice, and (3) unless a so-called *key aggregation* property is supported, verification does not use a single public key. Instead, it takes the set of public keys involved in signing the message. When it comes to the flexibility of choosing the group of signers, multi-signatures are more suitable than distributed signatures, but at the cost of more verification time and larger size of joint public key. The crucial property of a multi-signature is that it is *compact*, i.e., its size is not larger than the total size of signatures generated by each signer individually. If this property is not satisfied, then it is meaningful to use the *naive approach*, i.e., by simply concatenating the individual signatures created by each signer to produce a multi-signature. For instance, the signature scheme Dilithium [20] or Dilithium-G [21] can be used, which produces signatures of size less than 2.5 KB.

[17] observed that lattice-based multi-signature schemes prior their work have incomplete proof of security. In particular, their security proof does not consider simulating aborted executions of the signing protocol, which are inherent in the lattice setting due to carrying out rejection sampling. To solve this issue, [17] proposed a scheme that utilizes lattice-based trapdoor homomorphic commitments, while Boschini et al. [12] used trapdoors without homomorphic commitments. Chen [13] improved the schemes of [17,12] by introducing a so-called *dual signing simulation* technique, which allows to prove security without trapdoors.

Our construction can be easily turned into a multi-signature scheme. The difference is that each signer computes its own challenge and response using its own key pair. A multi-signature is given by the tuple $(root, \mathbf{z}, auth)$. In particular, the parameters are exactly the same, and the security proof is even simpler, since there is no dedicated key generation protocol. However, the compactness property is not satisfied for every small n. It seems that the regularity property of the commitments is the reason for this, even without using the tree of commitments technique. More concretely, commitments must be statistically hiding, while in standard signature schemes like Dilithium, they are only computationally hiding, since aborted executions are never revealed. Therefore, we choose not to present a multi-signature variant of our protocol. In fact, Table 1 shows that all current multi-signature schemes may be compact only for a large n.

Reducing Restarts. In order to solve the problem of the large number of restarts inherent in lattice-based interactive protocols following the Fiat-Shamir with aborts paradigm, Alkadri et al. [3] introduced a technique called *tree of commitments*. A tree of commitments is a binary hash tree whose leaves are constructed from many masking terms. This allows to reduce the number of restarts of lattice-based protocols by iteratively applying rejection sampling using different masking terms in one execution. By using a large enough number of masking terms, this even allows to completely eliminate restarts, i.e., with probability very close to one. However, this technique was used to construct efficient lattice-based blind signature schemes only [3,4], which involve just two parties (a signer and a user). The user blinds the signer's commitments via many random values, builds a hash tree from the blind commitments, and generates a blind signature from one of them without the need to abort and request a protocol restart from the signer. In this work, we use the technique in a multi-user setting, where each user generates multiple commitments and builds a hash tree to create a partial signature. Each leave of the tree corresponds to the sum of commitments, and each summand is created by one user. We show how to ensure that each signer computes the same hash tree, i.e., adding commitments in exactly the same way as the remaining signers. Otherwise, each signer would obtain a different root and signatures would not be verified.

2 Background

Notation. We denote by $\mathbb{N}, \mathbb{Z}$, and $\mathbb{R}$ the sets of natural numbers, integers, and real numbers, respectively. If $n \in \mathbb{N}$, we let $[n]$ denote the set $\{1, \ldots, n\}$. We denote the security parameter by $\lambda \in \mathbb{N}$, and abbreviate probabilistic polynomial-time by PPT and deterministic polynomial-time by DPT. We write $x \leftarrow\!\!\$\, D$ to denote that x is sampled randomly according to a distribution D. If S is a finite set, we also write $x \leftarrow\!\!\$\, S$ if x is chosen randomly from the uniform distribution over S. Let $q \in \mathbb{Z}_{>0}$. We write $\mathbb{Z}_q$ to denote the ring of integers modulo q with representatives in $[-\frac{q}{2}, \frac{q}{2}) \cap \mathbb{Z}$. Let N be a fixed power of two and consider the polynomial ring $\mathbb{Z}[X]$ in a variable X. We define the rings $R := \mathbb{Z}[X]/\langle X^N + 1\rangle$ and $R_q := \mathbb{Z}_q[X]/\langle X^N + 1\rangle$. Elements in R and R_q are denoted by regular font letters. Column vectors and matrices with coefficients in R or R_q are denoted by bold lower-case letters and bold upper-case letters, respectively. The identity matrix of dimension k is denoted by $\mathbf{I}_k$. The ℓ_p-norm of any $a \in R$ is defined by $\|a\|_p := (\sum_{i=0}^{N-1} |a_i|^p)^{1/p}$ if $p < \infty$ and by $\max\{|a_0|, \ldots, |a_{N-1}|\}$ if $p = \infty$. Similarly, the ℓ_p-norm of any $\mathbf{b} = (b_1, \ldots, b_k)^\top \in R^k$ is defined by $\|\mathbf{b}\|_p := (\sum_{i=1}^{k} \|b_i\|_p^p)^{1/p}$ if $p < \infty$ and by $\max\{\|b_1\|_p, \ldots, \|b_k\|_p\}$ if $p = \infty$. We write $\|\cdot\|$ instead of $\|\cdot\|_2$. We define the sets $S_\eta := \{f \in R_q \colon \|f\|_\infty \leq \eta\}$ and $\mathbb{T}_\kappa := \{f \in R_q \colon \|f\|_\infty = 1 \wedge \|f\|_1 = \kappa\}$. The discrete Gaussian distribution over $\mathbb{Z}^m$ with standard deviation $\sigma > 0$ and center $\mathbf{c} \in \mathbb{R}^m$ is the probability distribution $D_{\mathbb{Z}^m,\sigma,\mathbf{c}}$, which assigns to every $\mathbf{x} \in \mathbb{Z}^m$ the probability of occurrence given by $D_{\mathbb{Z}^m,\sigma,\mathbf{c}}(\mathbf{x}) := \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^m)$, where $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp(-\frac{\|\mathbf{x}-\mathbf{c}\|^2}{2\sigma^2})$ and $\rho_{\sigma,\mathbf{c}}(\mathbb{Z}^m) := \sum_{\mathbf{x}\in\mathbb{Z}^m} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. The subscript $\mathbf{c}$ is omitted when $\mathbf{c} = \mathbf{0}$. Additional background is provided in Appendix B.

Hardness Assumptions. We define the lattice problems: Module Learning With Errors (MLWE) and Module Small Integer Solution (MSIS).

Definition 1. *Let $pp = (N, k, \ell, q, \eta)$, where N, k, ℓ, q, η are positive integers. We say that* MLWE *holds w.r.t. pp if for every PPT algorithm* A *the advantage*

$\mathrm{Adv}_{\mathsf{A}}^{\mathsf{MLWE}}(pp)$ *is negligible in* λ*, where* $\mathrm{Adv}_{\mathsf{A}}^{\mathsf{MLWE}}(pp) :=$

$$\left| \Pr\left[b = 1 \colon \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\, R_q^{k\times\ell}; \mathbf{s} \leftarrow\!\!\$\, S_\eta^{k+\ell}; \\ \mathbf{t} := [\mathbf{I}_k|\mathbf{A}] \cdot \mathbf{s} \in R_q^k; \\ b \leftarrow\!\!\$\, \mathsf{A}(pp, \mathbf{A}, \mathbf{t}) \end{array} \right] - \Pr\left[b = 1 \colon \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\, R_q^{k\times\ell}; \\ \mathbf{t} \leftarrow\!\!\$\, R_q^k; \\ b \leftarrow\!\!\$\, \mathsf{A}(pp, \mathbf{A}, \mathbf{t}) \end{array} \right] \right|.$$

Definition 2. *Let* $pp = (N, k, \ell, q, \beta)$*, where* N, k, ℓ, q *are positive integers and* β *is a positive real. We say that* MSIS *holds w.r.t.* pp *if for every algorithm* A *the advantage* $\mathrm{Adv}_{\mathsf{A}}^{\mathsf{MSIS}}(pp)$ *is negligible in* λ*, where* $\mathrm{Adv}_{\mathsf{A}}^{\mathsf{MSIS}}(pp) :=$

$$\Pr[0 < \|\mathbf{x}\| \le \beta \wedge \mathbf{0} = [\mathbf{I}_k|\mathbf{A}] \cdot \mathbf{x} \pmod q \colon \mathbf{A} \leftarrow\!\!\$\, R_q^{k\times\ell}; \mathbf{x} \in R^{k+\ell} \leftarrow\!\!\$\, \mathsf{A}(pp, \mathbf{A})].$$

Estimating the hardness of MLWE and MSIS is described in Appendix C.

Distributed Signatures. We follow [17] to recall the syntax and security of n-out-of-n distributed signatures. We assume that many sessions of the signing protocol can be invoked concurrently, while key generation can be executed only once. All signers participating in both key and signature generation play the same role. Hence, we only present nth signer's behavior, who is the first one sending out a message in each round of interaction. Consequently, we assume that the adversary is *rushing*, i.e., based on the honest nth signer's message, the adversary is allowed to choose messages of the remaining $n-1$ corrupted signers.

Definition 3. *An n-out-of-n distributed signature protocol is a tuple of algorithms* $\mathsf{DSig} = (\mathsf{PGen}, \mathsf{KGen}_j, \mathsf{Sign}_j, \mathsf{Verify})$*, where:*

PGen *is a PPT parameter generation algorithm that, on input* 1^λ*, returns public parameters* pp*, which implicitly contains* 1^λ*. We assume that* pp *is given as an implicit input to all algorithms.*

KGen_j, *for all* $j \in [n]$*, is a PPT interactive key generation algorithm that is run by each signer* S_j*. At the end of the protocol,* S_j *returns a state* st *and a pair* (pk, sk_j)*, where* pk *is a public key and* sk_j *is a secret key share.*

Sign_j, *for all* $j \in [n]$*, is a PPT interactive signing algorithm that is run by each signer* S_j*. Each* S_j *runs* Sign_j *on input a session identifier* sid*, a state information* st*, a public key* pk*, a secret key share* sk_j*, and a message* m*. At the end of the protocol,* S_j *returns a signature* sig*.*

Verify *is a DPT verification algorithm that, on input a public key* pk*, a message* m*, and a signature* sig*, returns 1 if* sig *is valid and 0 otherwise.*

Definition 4. *We say that* DSig *is* UF-CMA *secure (distributed signature unforgeability against chosen message attacks) w.r.t.* $pp \in \mathsf{PGen}(1^\lambda)$ *if for every adversary* A *that makes* q_{Sign} *signing queries to an oracle* $\mathsf{O}_n^{\mathsf{DSig}}$*, the following advantage is negligible in* λ*:*

$$\mathrm{Adv}_{\mathsf{DSig},\mathsf{A}}^{\text{UF-CMA}}(pp) = \Pr[\mathrm{Exp}_{\mathsf{DSig},\mathsf{A}}^{\text{UF-CMA}}(pp) = 1],$$

where the oracle $\mathsf{O}_n^{\mathsf{DSig}}$ *and the experiment* $\mathrm{Exp}_{\mathsf{DSig},\mathsf{A}}^{\text{UF-CMA}}$ *are defined in Fig. 2.*

```
$\mathrm{Exp}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}(pp)$
1: $\mathrm{L}_m \leftarrow \varnothing$
2: $(m^*, sig^*) \leftarrow\!\!\$\ \mathsf{A}^{\mathsf{O}^{\mathsf{DSig}}_n}(pp)$
3: if $m^* \in \mathrm{L}_m$ then
4:     return $0$
5: return $\mathsf{Verify}(pk, m^*, sig^*)$
```

```
$\mathsf{O}^{\mathsf{DSig}}_n(sid, m)$
1: $\mathit{flag} \leftarrow$ false
2: if $sid = 0$ then
3:     if $\mathit{flag} =$ true then
4:         return $\perp$
5:     $(st, (pk, sk_n)) \leftarrow\!\!\$\ \mathsf{KGen}_n()$
6:     $\mathit{flag} \leftarrow$ true
7: if $\mathit{flag} =$ false then
8:     return $\perp$
9: $\mathrm{L}_m \leftarrow \mathrm{L}_m \cup \{m\}$
10: $sig \leftarrow\!\!\$\ \mathsf{Sign}_n(sid, st, pk, sk_n, m)$
11: return $sig$
```

Fig. 2. Experiment $\mathrm{Exp}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}$. We define by L_m the set of all messages m such that (sid, m) was queried by A to its oracle as the first query with identifier $sid \neq 0$.

3 Distributed Signature Protocol

3.1 Protocol Description

Let $\mathsf{G}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{G}}$, $\mathsf{F}\colon \{0,1\}^* \to \{0,1\}^{\ell_\mathsf{F}}$, and $\mathsf{H}\colon \{0,1\}^* \to \mathbb{T}_\kappa$ be cryptographic hash functions. Define by $\mathsf{Expand}\colon \{0,1\}^* \to R_q^{k\times\ell}$ an extendable output function (XOF), e.g., SHAKE. Let $\omega \in \mathbb{N}_{>1}$. Following [3,4], we define the algorithms that build a tree of commitments:
<u>1. $\mathsf{HashTree}$</u> is a DPT algorithm whose input is ω commitments $\mathbf{v}_0, \ldots, \mathbf{v}_{\omega-1}$, where $\mathbf{A} \in R_q^{k\times\ell}$ and for all $j \in \{0\ldots,\omega-1\}$: $\mathbf{v}_j = [\mathbf{I}_k|\mathbf{A}]\cdot\mathbf{y}_j \pmod q$ and $\mathbf{y}_j \in R^{k+\ell}$. It returns a pair $(\mathit{root}, \mathit{tree})$, where root is the root of a binary hash tree of height $h = \lceil\log(\omega)\rceil$ whose leaves are the hash values $\mathsf{F}(\mathbf{v}_j)$, and tree is the sequence that consists of all leaves and inner nodes of the tree.
<u>2. $\mathsf{BuildAuth}$</u> is a DPT algorithm whose input is an index t, a sequence of nodes tree, and a height h. It returns $\mathit{auth} = (t, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$, where $\mathbf{a}_i \in \{0,1\}^{\ell_\mathsf{F}}$, $0 \le t < \omega$, and $0 \le i < h$. Let $\mathbf{z}'$ be some secret vector. The output auth represents the authentication path of a vector $\mathbf{z}_t = \mathbf{y}_t + \mathbf{z}'$, for which the rejection sampling procedure accepts, i.e., masking vector $\mathbf{y}_t$ ensures that $\mathbf{z}_t$ hides $\mathbf{z}'$.
<u>3. $\mathsf{RootCalc}$</u> is a DPT algorithm whose input is a commitment $\mathbf{v}$ and its authentication path $\mathit{auth} = (t, \mathbf{a}_0, \ldots, \mathbf{a}_{h-1})$. It returns the root of the hash tree that includes the leaf $\mathsf{F}(\mathbf{v})$ at index t and the inner nodes $\mathbf{a}_0, \ldots, \mathbf{a}_{h-1}$.
We define the following bijective mapping:
$\mathsf{IntIndex}_{\omega,n}\colon \{0,\ldots,\omega-1\}^n \to \{0,\ldots,\omega^n-1\}$; $(i^{(1)},\ldots,i^{(n)}) \mapsto \sum_{j=0}^{n-1} i^{(n-j)}\cdot\omega^j$.
$\mathsf{IntIndex}_{\omega,n}$ converts a tuple $(i^{(1)},\ldots,i^{(n)})$ into a unique positive integer. Let L be a finite set, we define by $\mathsf{Encode}(L)$ a unique encoding of L, e.g., lexicographical ordering. We also write $L[j]$ to denote the j^{th} entry of L. We let $\mathsf{Compress}$ and $\mathsf{Decompress}$ define algorithms for representing Gaussian elements via Huffman encoding. The first algorithm is used in the signing process to reduce the signature size, while the latter is used in the verification algorithm to reconstruct

```
$\mathsf{KGen}_n()$
$seed^{(n)} \leftarrow\!\!\$\ \{0,1\}^{\ell_{seed}}$
$\bar{g}^{(n)} \leftarrow \mathsf{G}(seed^{(n)}, n)$
**broadcast** $\bar{g}^{(n)}$
**receive** $\bar{g}^{(j)}$ for all $j \in [n-1]$
**broadcast** $seed^{(n)}$
**receive** $seed^{(j)}$ for all $j \in [n-1]$
**for** $j = 1$ **to** $n-1$ **do**
    **if** $\bar{g}^{(j)} \neq \mathsf{G}(seed^{(j)}, j)$ **then**
        **broadcast** *abort*
        **return** $(\perp, \perp, \perp)$
$seed \leftarrow \bigoplus_{j=1}^{n} seed^{(j)}$
$\mathbf{A} \leftarrow \mathsf{Expand}(seed)$, $\bar{\mathbf{A}} \leftarrow [\mathbf{I}_k|\mathbf{A}]$
$\mathbf{s}^{(n)} \leftarrow\!\!\$\ S_\eta^{k+\ell}$
$\mathbf{b}^{(n)} \leftarrow \bar{\mathbf{A}} \cdot \mathbf{s}^{(n)} \pmod q$
$\hat{g}^{(n)} \leftarrow \mathsf{G}(\mathbf{b}^{(n)}, n)$
**broadcast** $\hat{g}^{(n)}$
**receive** $\hat{g}^{(j)}$ for all $j \in [n-1]$
**broadcast** $\mathbf{b}^{(n)}$
**receive** $\mathbf{b}^{(j)}$ for all $j \in [n-1]$
**for** $j = 1$ **to** $n-1$ **do**
    **if** $\hat{g}^{(j)} \neq \mathsf{G}(\mathbf{b}^{(j)}, j)$ **then**
        **broadcast** *abort*
        **return** $(\perp, \perp, \perp)$
$\mathbf{b} \leftarrow \sum_{j=1}^{n} \mathbf{b}^{(j)} \pmod q$
$L \leftarrow \mathsf{Encode}(\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(n)})$
let $L[int] = \mathbf{b}^{(n)}$, $int \in [n]$
$st \leftarrow (int, L)$
$pk \leftarrow (seed, \mathbf{b})$
$sk^{(n)} \leftarrow \mathbf{s}^{(n)}$
**return** $(st, (pk, sk^{(n)}))$
```

Fig. 3. Key generation of our lattice-based n-out-of-n distributed signature protocol.

the Gaussian vector computed during signature generation. In the following we give a detailed description of our n-out-of-n distributed signature protocol. Its respective algorithms are given in Fig. 3,4. Since all signers play the same role, we only present n^{th} signer's behavior.

Parameter and key generation. PGen generates public parameters as given in Table 2. We assume that PGen is invoked by a trusted party. KGen_n first generates a uniformly random $\mathbf{A} \in R_q^{k\times\ell}$ in a distributed way. That is, it samples $seed^{(n)} \leftarrow\!\!\$\ \{0,1\}^{\ell_{seed}}$, computes $\bar{g}^{(n)} = \mathsf{G}(seed^{(n)}, n)$, and sends out $\bar{g}^{(n)}$. After receiving $\bar{g}^{(j)}$ for all $j \in [n-1]$, it sends out $seed^{(n)}$ and then receives $seed^{(j)}$. If $\bar{g}^{(j)} \neq \mathsf{G}(seed^{(j)}, j)$ for any $j \in [n-1]$, it aborts. Otherwise, it computes $seed = \bigoplus_{j=1}^{n} seed^{(j)}$, $\mathbf{A} = \mathsf{Expand}(seed)$, and sets $\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}]$. KGen_n proceeds by sampling $\mathbf{s}^{(n)} \leftarrow\!\!\$\ S_\eta^{k+\ell}$, computing a public key share $\mathbf{b}^{(n)} = \bar{\mathbf{A}} \cdot \mathbf{s}^{(n)} \pmod q$, and then sending out $\hat{g}^{(n)} = \mathsf{G}(\mathbf{b}^{(n)}, n)$. After receiving $\hat{g}^{(j)}$ for all $j \in [n-1]$, it sends out $\mathbf{b}^{(n)}$ and then receives $\mathbf{b}^{(j)}$. If $\hat{g}^{(j)} \neq \mathsf{G}(\mathbf{b}^{(j)}, j)$ for any $j \in [n-1]$, it aborts. Otherwise, it computes $\mathbf{b} = \sum_{j=1}^{n} \mathbf{b}^{(j)} \pmod q$, and encodes the list of public key shares $(\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(n)})$ via Encode to obtain an ordered list L. The state information is given by $st = (int, L)$, where int is the index of $\mathbf{b}^{(n)}$ in L, i.e., $L[int] = \mathbf{b}^{(n)}$. The public key and secret key share are given by $(pk, sk^{(n)}) = ((seed, \mathbf{b}), \mathbf{s}^{(n)})$. Note that based on the security of Expand and as long as at least one honest signer samples a seed correctly, the computed matrix $\mathbf{A}$ is guaranteed to be uniformly random. The same applies to the combined public key. As explained in [17], this prevents the so-called *rogue key attack* [28], i.e., to choose some malicious key share depending on the honest signer's share.

```
$\mathsf{Sign}_n(sid, st, pk, sk^{(n)}, m)$
1:  if $sid \in \mathrm{L}_{sid}$ then
2:      return $\perp$
3:  parse $st = (n, L = (L[1], \dots, L[n])$
                $= (\mathbf{b}^{(1)}, \dots, \mathbf{b}^{(n)}))$
4:  parse $(pk, sk^{(n)}) = ((seed, \mathbf{b}), \mathbf{s}^{(n)})$
5:  $\mathbf{A} \leftarrow \mathsf{Expand}(seed)$, $\bar{\mathbf{A}} \leftarrow [\mathbf{I}_k|\mathbf{A}]$
6:  for $i = 0$ to $\omega - 1$ do
7:      $\mathbf{y}_i \leftarrow\$ D^{k+\ell}_{\mathbb{Z}^N,\sigma}$
8:      $\mathbf{v}^{(n)}_i \leftarrow \bar{\mathbf{A}} \cdot \mathbf{y}_i \pmod q$
9:  $\mathbf{y} \leftarrow (\mathbf{y}_0, \dots, \mathbf{y}_{\omega-1})$
10: $\mathbf{v}^{(n)} \leftarrow (\mathbf{v}^{(n)}_0, \dots, \mathbf{v}^{(n)}_{\omega-1})$
11: $g^{(n)} \leftarrow \mathsf{G}(\mathbf{v}^{(n)}, \mathbf{b}^{(n)})$
12: broadcast $(n, g^{(n)})$
13: receive $(j, g^{(j)})$ for all $j \in [n-1]$
14: broadcast $\mathbf{v}^{(n)}$
15: receive $\mathbf{v}^{(j)} = (\mathbf{v}^{(j)}_0, \dots, \mathbf{v}^{(j)}_{\omega-1})$, $j \in [n-1]$

16: for $j = 1$ to $n - 1$ do
17:     if $g^{(j)} \neq \mathsf{G}(\mathbf{v}^{(j)}, L[j])$ then
18:         broadcast $abort$ and return $\perp$
19: for $t = 0$ to $\omega^n - 1$ do
20:     $(i^{(1)}, \dots, i^{(n)}) \leftarrow \mathsf{IntIndex}^{-1}_{\omega,n}(t)$
21:     $\mathbf{w}_t \leftarrow \sum_{j=1}^{n} \mathbf{v}^{(j)}_{i^{(j)}}$
22: $(root, tree) \leftarrow \mathsf{HashTree}(\mathbf{w}_0, \dots, \mathbf{w}_{\omega^n-1})$
23: $c \leftarrow \mathsf{H}(root, m, \mathbf{b})$
24: $\mathbf{z}' \leftarrow \mathbf{s}^{(n)}c$
25: $(\mathbf{z}^{(n)}, i^{(n)}) \leftarrow\$ \mathsf{IterateRej}(\mathbf{y}, \mathbf{z}')$
26: if $(\mathbf{z}^{(n)}, i^{(n)}) = (\perp, \perp)$ then
27:     broadcast $\perp$ and goto 6
28: broadcast $\mathbf{z}^{(n)}$
29: receive $\mathbf{z}^{(j)}$ for all $j \in [n-1]$
30: for $j = 1$ to $n - 1$ do
31:     if $\mathbf{z}^{(j)} = \perp$ then
32:         goto 6
33:     $(b, i^{(j)}) \leftarrow \mathsf{Vrf}(\bar{\mathbf{A}}, \mathbf{b}^{(j)}, \mathbf{v}^{(j)}, c, \mathbf{z}^{(j)})$
34:     if $(b, i^{(j)}) = (0, -1)$ then
35:         broadcast $abort$ and return $\perp$
36: $\mathbf{z} \leftarrow \sum_{j=1}^{n} \mathbf{z}^{(j)}$
37: $\mathbf{z} \leftarrow \mathsf{Compress}(\mathbf{z})$
38: $t \leftarrow \mathsf{IntIndex}_{\omega,n}(i^{(1)}, \dots, i^{(n)})$
39: $auth \leftarrow \mathsf{BuildAuth}(t, tree, h)$
40: return $sig = (c, \mathbf{z}, auth)$
```

```
$\mathsf{IterateRej}(\mathbf{y}, \mathbf{z}')$
1: parse $\mathbf{y} = (\mathbf{y}_0, \dots, \mathbf{y}_{\omega-1})$
2: $T \leftarrow \{0, \dots, \omega - 1\}$
3: while $T \neq \varnothing$ do
4:     $i \leftarrow\$ T$
5:     $T \leftarrow T \setminus \{i\}$
6:     $\mathbf{z} \leftarrow \mathbf{y}_i + \mathbf{z}'$
7:     if $\mathsf{RejSamp}(\mathbf{z}, \mathbf{z}') = 1$ then
8:         return $(\mathbf{z}, i)$
9: return $(\perp, \perp)$
```

```
$\mathsf{Vrf}(\bar{\mathbf{A}}, \mathbf{b}^{(j)}, \mathbf{v}^{(j)}, c, \mathbf{z}^{(j)})$
1: if $\|\mathbf{z}^{(j)}\| > B_z$ then
2:     return $(0, -1)$
3: parse $\mathbf{v}^{(j)} = (\mathbf{v}^{(j)}_0, \dots, \mathbf{v}^{(j)}_{\omega-1})$
4: $\mathbf{w}^{(j)} \leftarrow \bar{\mathbf{A}} \cdot \mathbf{z}^{(j)} - \mathbf{b}^{(j)}c \pmod q$
5: for $i = 0$ to $\omega - 1$ do
6:     if $\mathbf{v}^{(j)}_i = \mathbf{w}^{(j)}$ then
7:         return $(1, i)$
8: return $(0, -1)$
```

```
$\mathsf{Verify}(pk, m, sig)$
1:  parse $pk = (seed, \mathbf{b})$
2:  $\mathbf{A} \leftarrow \mathsf{Expand}(seed)$, $\bar{\mathbf{A}} \leftarrow [\mathbf{I}_k|\mathbf{A}]$
3:  parse $sig = (c, \mathbf{z}, auth)$
4:  $\mathbf{z} \leftarrow \mathsf{Decompress}(\mathbf{z})$
5:  if $\|\mathbf{z}\| > B$ then
6:      return 0
7:  $\mathbf{w} \leftarrow \bar{\mathbf{A}} \cdot \mathbf{z} - \mathbf{b}c \pmod q$
8:  $root \leftarrow \mathsf{RootCalc}(\mathbf{w}, auth)$
9:  if $c \neq \mathsf{H}(root, m, \mathbf{b})$ then
10:     return 0
11: return 1
```

Fig. 4. Signing protocol of our lattice-based n-out-of-n distributed signature protocol.

For the sake of domain separation [6], we also follow [17] by setting the index of the signer as part of the value to be hashed via G. This prevents a rushing adversary from forwarding a hash value sent by the honest signer and claiming knowledge of its preimage after receiving it from the honest signer. To save space, $\mathbf{s}^{(n)}$ can also be generated by expanding a random seed via an XOF.

Signing. S_n first checks that *sid* has not been used before, i.e., S_n is not executed if $sid \in \mathrm{L}_{sid}$, where L_{sid} is the list of already used session identifiers. We assume, w.l.o.g., that after ordering the list of public key shares in KGen_n, the index of $\mathbf{b}^{(n)}$ in the encoded list L is given by $int = n$, i.e., $L[n] = \mathbf{b}^{(n)}$. Then, S_n reconstructs $\mathbf{A}$ using Expand and sets $\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}]$. It proceeds by sampling ω vectors $\mathbf{y}_i$ from the Gaussian distribution $D^{k+\ell}_{\mathbb{Z}^N,\sigma}$ and computing commitments $\mathbf{v}_i^{(n)} = \bar{\mathbf{A}} \cdot \mathbf{y}_i \pmod q$, where $i \in \{0, \ldots, \omega-1\}$. Note that σ is chosen according to Lemma 2 so that each vector $\mathbf{v}_i^{(n)}$ is distributed statistically close to uniform over R_q^k. This prevents learning any information from the commitments in case of aborts, and hence maintains the security of the secret key share. Afterwards, signer S_n sets $\mathbf{v}^{(n)} = (\mathbf{v}_0^{(n)}, \ldots, \mathbf{v}_{\omega-1}^{(n)})$, computes $g^{(n)} = \mathsf{G}(\mathbf{v}^{(n)}, \mathbf{b}^{(n)})$, and sends out $(n, g^{(n)})$ to the remaining signers $\mathsf{S}_1, \ldots, \mathsf{S}_{n-1}$ in order to receive $(j, g^{(j)})$ for all $j \in [n-1]$. Sending the index $j \in [n]$ together with $g^{(j)}$ allows to map each signer to the index of its public key share in L. This way, all signers use the same local ordering of $\mathsf{S}_1, \ldots, \mathsf{S}_n$, which corresponds to the indices of the public key shares in L. Then, S_n sends out $\mathbf{v}^{(n)}$ and receives a similar vector $\mathbf{v}^{(j)}$ from each S_j. After that, S_n verifies that $g^{(j)} = \mathsf{G}(\mathbf{v}^{(j)}, \mathbf{b}^{(j)})$ for all $j \in [n-1]$, and aborts if this is not the case. S_n proceeds by computing ω^n vectors $\mathbf{w}_t$, where $t \in \{0, \ldots, \omega^n - 1\}$. These vectors correspond to all possible sums of n different commitments $\mathbf{v}_{i(j)}^{(j)}$, i.e., each sum includes one commitment from each signer. The commitments $\mathbf{w}_t$ are then used to generate a tree of commitments of height $h = \lceil \log(\omega^n) \rceil$ via algorithm $\mathsf{HashTree}$, which outputs $(root, tree)$. Note that due to the unique encoding of L, the indices of the signers are the same by all signers. Thus, all signers compute the same hash tree. Then, H is called on input $(root, m, \mathbf{b})$ to obtain c. After that, S_n runs $\mathsf{IterateRej}$ on input $(\mathbf{y}, \mathbf{z}')$, where $\mathbf{y} = (\mathbf{y}_0, \ldots, \mathbf{y}_{\omega-1})$ and $\mathbf{z}' = \mathbf{s}^{(n)}c$. This algorithm repeatedly keeps applying the rejection sampling algorithm $\mathsf{RejSamp}$ on input $(\mathbf{z}, \mathbf{z}')$, where $\mathbf{z} = \mathbf{y}_i + \mathbf{z}'$, until it accepts for some randomly chosen masking vector $\mathbf{y}_i$, where $i \in \{0, \ldots, \omega-1\}$. $\mathsf{IterateRej}$ outputs $(\mathbf{z}^{(n)}, i)$, where i corresponds to the masking vector for which $\mathsf{RejSamp}(\mathbf{z}^{(n)}, \mathbf{z}') = 1$. If $\mathsf{RejSamp}$ does not accept for all $\mathbf{y}_i$, then $\mathsf{IterateRej}$ returns $(\perp, \perp)$ and the protocol has to be restarted. In this case S_n broadcasts $\perp$ and restarts by generating ω fresh masking vectors $\mathbf{y}_i$, i.e., from line 6 of Sign_n. Otherwise, S_n broadcasts $\mathbf{z}^{(n)}$ and receives $\mathbf{z}^{(j)}$ from each S_j, where $j \in [n-1]$. If $\mathbf{z}^{(j)} = \perp$ for any $j \in [n-1]$, then S_n restarts from line 6. Otherwise, S_n verifies the correctness of each cosigner's signature by running Vrf on input $(\bar{\mathbf{A}}, \mathbf{b}^{(j)}, \mathbf{v}^{(j)}, c, \mathbf{z}^{(j)})$ for all $j \in [n-1]$. Vrf outputs a pair (b, i), where $b \in \{0, 1\}$ indicates accept or reject and i is the index of commitment $\mathbf{v}_i$ that corresponds to response $\mathbf{z}_i$. If one cosigner's signature is not valid, then Vrf returns $(0, -1)$, and S_n aborts. Otherwise, S_n proceeds by computing $\mathbf{z} = \sum_{j=1}^{n} \mathbf{z}^{(j)}$, compressing

the sum $\mathbf{z}$ via $\mathsf{Compress}$, and running $\mathsf{BuildAuth}$ to generate the authentication path *auth* associated to the index $t = \mathsf{IntIndex}_{\omega,n}(i^{(1)}, \ldots, i^{(n)})$ of commitment $\mathbf{w}_t$, where $\mathbf{w}_t$ is the sum of signer's commitments $\mathbf{v}^{(j)}_{i^{(j)}}$ that correspond to $\mathbf{z}$. Finally, S_n returns the signature $sig = (c, \mathbf{z}, auth)$.
Verification. Verify first computes $\mathbf{A}$ via Expand, sets $\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}]$, and checks that $\|\mathbf{z}\| \leq B$ after reconstructing $\mathbf{z}$ using algorithm $\mathsf{Decompress}$. Then, it computes $\mathbf{w} = \bar{\mathbf{A}} \cdot \mathbf{z} - \mathbf{b}c \pmod q$, and runs $\mathsf{RootCalc}$ on input $(\mathbf{w}, auth)$ to compute the root $root'$ of the tree of commitments that includes the leaf $\mathsf{F}(\mathbf{w})$ and its authentication path *auth*. The signature is accepted if and only if $c = \mathsf{H}(root, m, \mathbf{b})$.

3.2 Security Analysis

In this section we prove the security of our distributed signature protocol.

Theorem 1. *Let* DSig *be the* n*-out-of-*n *distributed signature protocol depicted in Fig. 3,4. For any PPT adversary that initiates a single key generation protocol of* DSig *by querying* $\mathsf{O}^{\mathsf{DSig}}_n$ *with* $sid = 0$*, initiates* q_{Sign} *signature generation protocols of* DSig *by querying* $\mathsf{O}^{\mathsf{DSig}}_n$ *with* $sid \neq 0$*, and makes* $q_E, q_{\mathsf{G}}, q_{\mathsf{F}}, q_{\mathsf{H}}$ *queries to the random oracle* $\mathsf{Expand}, \mathsf{G}, \mathsf{F}, \mathsf{H}$*, respectively,* DSig *is* UF-CMA *secure w.r.t.* $pp \in \mathsf{PGen}(1^\lambda)$ *if* MLWE *is hard w.r.t.* $pp' = (N, k, \ell, q, \eta)$ *and* MSIS *is hard w.r.t.* $pp'' = (N, k, \ell + 1, q, 2\sqrt{B^2 + \kappa})$*, where*

$$\begin{aligned}\mathrm{Adv}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}(pp) \leq & \frac{q_{\mathsf{F}}^2 + q_{\mathsf{F}}}{2^{\ell_{\mathsf{F}}}} + \frac{(q_{\mathsf{G}} + nq_{\mathsf{Sign}} + 1)^2}{2^{\ell_{\mathsf{G}}+1}} + \frac{q_E}{q^{k\ell N}} + \\ & q_{\mathsf{Sign}} \cdot \Big(\frac{q_{\mathsf{G}} + nq_{\mathsf{Sign}}}{|\mathbb{T}_\kappa|} + \frac{q_{\mathsf{H}} + q_{\mathsf{Sign}}}{|\mathbb{T}_\kappa|} + \frac{n}{2^{\ell_{\mathsf{G}}}} + \frac{2^{-\Omega(N)-100+1}}{M}\Big) + \\ & 2 \cdot \Big(\frac{(q_{\mathsf{G}} + 1)^2}{2^{\ell_{\mathsf{G}}+1}} + \frac{n}{2^{\ell_{\mathsf{G}}}}\Big) + \frac{q_{\mathsf{G}}}{2^{\ell_{seed}}} + \frac{q_{\mathsf{G}}}{q^{kN}} + \mathrm{Adv}^{\mathsf{MLWE}}_{\mathsf{D}}(pp') + \\ & \frac{(q_{\mathsf{H}} + q_{\mathsf{Sign}}) \cdot \omega^n}{|\mathbb{T}_\kappa|} + \sqrt{(q_{\mathsf{H}} + q_{\mathsf{Sign}}) \cdot \omega^n \cdot \mathrm{Adv}^{\mathsf{MSIS}}_{\mathsf{A}}(pp'')}.\end{aligned}$$

Proof. Let A be an adversary that wins the experiment $\mathrm{Exp}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}(pp)$ given in Fig. 2 with advantage $\mathrm{Adv}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}(pp)$. We assume, w.l.o.g., that S_n is an honest signer. We construct a reduction R that uses A in a black-box manner and simulates the behavior of S_n without using honestly generated key pairs. Then, we use the forking algorithm (see Appendix B.1) to solve MSIS w.r.t. pp''. The simulation of key and signature generation is presented in Fig. 5 and 6, respectively. They are derived via the following intermediate hybrids:
Hybrid H_0:
Random oracle simulation. We assume that R is given $h_i \leftarrow_\$ \mathbb{T}_\kappa$ as input, for all $i \in [q_{\mathsf{Sign}} + q_{\mathsf{H}}]$. For each of oracles $\mathsf{O}_{\mathsf{Expand}}, \mathsf{O}_{\mathsf{G}}, \mathsf{O}_{\mathsf{F}}$, and O_{H}, reduction R maintains a list $\mathrm{L}_{\mathsf{Expand}}, \mathrm{L}_{\mathsf{G}}, \mathrm{L}_{\mathsf{F}}$, and L_{H}, respectively. These lists are initialized with the empty set, and store pairs consisting of queries to the respective oracle and their answers. Also, R maintains a counter *ctr* initialized by 0. If an oracle was previously queried on some input, then R looks up its entry in the respective

```
SimKGen_n(A, b)
1: $seed \leftarrow_\$ \{0,1\}^{\ell_{seed}}$
2: $\mathsf{O}_{\mathsf{Expand}}(seed) \leftarrow \mathbf{A}$
3: $\bar{g}^{(n)} \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{G}}}$
4: broadcast $\bar{g}^{(n)}$
5: receive $\bar{g}^{(j)}$ for all $j \in [n-1]$
6: $(bad'_1, alert, (seed^{(1)}, 1), \ldots, (seed^{(n-1)}, n-1)) \leftarrow \mathsf{Search}(\bar{g}^{(1)}, \ldots, \bar{g}^{(n-1)})$
7: if $bad'_1 = $ true then
8:     return $(\perp, \perp)$    {simulation fails}
9: if $alert = $ true then
10:     $seed^{(n)} \leftarrow_\$ \{0,1\}^{\ell_{seed}}$
11: $seed^{(n)} \leftarrow seed \bigoplus_{j=1}^{n-1} seed^{(j)}$
12: if $(seed^{(n)}, n)$ is set then
13:     $bad'_2 \leftarrow$ true
14:     return $(\perp, \perp)$    {simulation fails}
15: $\mathsf{O}_{\mathsf{G}}(seed^{(n)}, n) \leftarrow \bar{g}^{(n)}$
16: broadcast $seed^{(n)}$
17: receive $seed^{(j)}$ for all $j \in [n-1]$
18: for $j = 1$ to $n-1$ do
19:     if $\bar{g}^{(j)} \neq \mathsf{O}_{\mathsf{G}}(seed^{(j)}, j)$ then
20:         broadcast $abort$
21:         return $(\perp, \perp)$
22: if $alert = $ true then
23:     $bad'_3 \leftarrow$ true
24:     return $(\perp, \perp)$    {simulation fails}
25: $\hat{g}^{(n)} \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{G}}}$
26: broadcast $\hat{g}^{(n)}$
27: receive $\hat{g}^{(j)}$ for all $j \in [n-1]$
28: $(bad'_4, alert', (\mathbf{b}^{(1)}, 1), \ldots, (\mathbf{b}^{(n-1)}, n-1)) \leftarrow \mathsf{Search}(\hat{g}^{(1)}, \ldots, \hat{g}^{(n-1)})$
29: if $bad'_4 = $ true then
30:     return $(\perp, \perp)$    {simulation fails}
31: if $alert' = $ true then
32:     $\mathbf{b}^{(n)} \leftarrow_\$ R_q^k$
33: $\mathbf{b}^{(n)} \leftarrow \mathbf{b} - \sum_{j=1}^{n-1} \mathbf{b}^{(j)}$
34: if $(\mathbf{b}^{(n)}, n)$ is set then
35:     $bad'_5 \leftarrow$ true
36:     return $(\perp, \perp)$    {simulation fails}
37: $\mathsf{O}_{\mathsf{G}}(\mathbf{b}^{(n)}, n) \leftarrow \hat{g}^{(n)}$
38: broadcast $\mathbf{b}^{(n)}$
39: receive $\mathbf{b}^{(j)}$ for all $j \in [n-1]$
40: for $j = 1$ to $n-1$ do
41:     if $\hat{g}^{(j)} \neq \mathsf{O}_{\mathsf{G}}(\mathbf{b}^{(j)}, j)$ then
42:         broadcast $abort$
43:         return $(\perp, \perp)$
44: if $alert' = $ true then
45:     $bad'_6 \leftarrow$ true
46:     return $(\perp, \perp)$    {simulation fails}
47: $L \leftarrow \mathsf{Encode}(\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(n)})$
48: let $L[int] = \mathbf{b}^{(n)}$, $int \in [n]$
49: $st \leftarrow (int, L)$
50: $pk \leftarrow (seed, \mathbf{b})$
51: return $(st, pk)$
```

```
Search(g^(1), ..., g^(n-1))
1: $bad \leftarrow$ false, $alert \leftarrow$ false
2: for $j = 1$ to $n-1$ do
3:     if $g^{(j)}$ is set then
4:         let $(str, \mathsf{O}_{\mathsf{G}}(str)) \in \mathrm{L}_{\mathsf{G}} \colon \mathsf{O}_{\mathsf{G}}(str) = g^{(j)}$
5:         if $\exists (str', \mathsf{O}_{\mathsf{G}}(str')) \in \mathrm{L}_{\mathsf{G}} \colon (str \neq str') \wedge (\mathsf{O}_{\mathsf{G}}(str') = g^{(j)})$ then
6:             $bad \leftarrow$ true    {more than one preimage found}
7:         $str^{(j)} \leftarrow str$
8:     else
9:         $str^{(j)} \leftarrow \perp$, $alert \leftarrow$ true    {no preimage found}
10: return $(bad, alert, str^{(1)}, \ldots, str^{(n-1)})$
```

Fig. 5. Simulation of key generation.

```
$\mathsf{SimSign}_n(sid, st, pk, m)$
1: if $sid \in \mathrm{L}_{sid}$ then
2:     return $\perp$
3: parse $st = (n, L = (L[1], .., L[n]))$
4: parse $pk = (seed, \mathbf{b})$
5: $\mathbf{A} \leftarrow \mathsf{O}_{\mathsf{Expand}}(seed)$
6: $\bar{\mathbf{A}} \leftarrow [\mathbf{I}_k | \mathbf{A}]$
7: $c \leftarrow\!\!\$\ \mathbb{T}_\kappa$
8: With probability $\delta$ do
9:     $\mathbf{v}_0^{(n)}, \ldots, \mathbf{v}_{\omega-1}^{(n)} \leftarrow\!\!\$\ R_q^k$
10:    $\mathbf{z}^{(n)} \leftarrow \perp$
11: With probability $1 - \delta$ do
12:    $i^{(n)} \leftarrow\!\!\$\ \{0, \ldots, \omega - 1\}$
13:    $\mathbf{z}^{(n)} \leftarrow\!\!\$\ D_{\mathbb{Z}^N, \sigma}^{k+\ell}$
14:    $\mathbf{v}_{i^{(n)}}^{(n)} \leftarrow \bar{\mathbf{A}} \cdot \mathbf{z}^{(n)} - \mathbf{b}^{(n)} c \pmod q$
15:    for $i = 0$ to $\omega - 1$ do
16:        if $i = i^{(n)}$ then
17:            continue
18:        $\mathbf{y}_i \leftarrow\!\!\$\ D_{\mathbb{Z}^N, \sigma}^{k+\ell}$
19:        $\mathbf{v}_i^{(n)} \leftarrow \bar{\mathbf{A}} \cdot \mathbf{y}_i \pmod q$
20: $\mathbf{v}^{(n)} \leftarrow (\mathbf{v}_0^{(n)}, \ldots, \mathbf{v}_{\omega-1}^{(n)})$
21: $g^{(n)} \leftarrow \mathsf{O}_{\mathsf{G}}(\mathbf{v}^{(n)}, \mathbf{b}^{(n)})$
22: broadcast $(n, g^{(n)})$
23: receive $(j, g^{(j)})$ for all $j \in [n-1]$
24: $(bad_1, alert, (\mathbf{v}^{(1)}, \mathbf{b}^{(1)}), \ldots, (\mathbf{v}^{(n-1)},$
    $\mathbf{b}^{(n-1)})) \leftarrow \mathsf{Search}(g^{(1)}, \ldots, g^{(n-1)})$
25: if $bad_1 =$ true then
26:     return $\perp$            {simulation fails}
27: if $alert =$ true then
28:     broadcast $\mathbf{v}^{(n)}$
29: else
30:     parse $\mathbf{v}^{(j)} = (\mathbf{v}_0^{(j)}, \ldots, \mathbf{v}_{\omega-1}^{(j)})$
        for all $j \in [n-1]$
31:     for $t = 0$ to $\omega^n - 1$ do
32:         $(i^{(1)}, \ldots, i^{(n)}) \leftarrow \mathsf{IntIndex}_{\omega,n}^{-1}(t)$
33:         $\mathbf{w}_t \leftarrow \sum_{j=1}^{n} \mathbf{v}_{i^{(j)}}^{(j)}$
34:     $(root, tree) \leftarrow$
                $\leftarrow \mathsf{HashTree}(\mathbf{w}_0, \ldots, \mathbf{w}_{\omega^n - 1})$
35:     if $\mathsf{O}_{\mathsf{H}}(root, m, \mathbf{b})$ is set then
36:         $bad_2 \leftarrow$ true
37:         return $\perp$        {simulation fails}
38:     $\mathsf{O}_{\mathsf{H}}(root, m, \mathbf{b}) \leftarrow c$
39:     broadcast $\mathbf{v}^{(n)}$
40: receive $\mathbf{v}^{(j)} = (\mathbf{v}_0^{(j)}, \ldots, \mathbf{v}_{\omega-1}^{(j)})$
    for all $j \in [n-1]$
41: for $j = 1$ to $n - 1$ do
42:     if $g^{(j)} \neq \mathsf{O}_{\mathsf{G}}(\mathbf{v}^{(j)}, L[j])$ then
43:         broadcast $abort$ and return $\perp$
44: if $alert =$ true then
45:     $bad_3 \leftarrow$ true
46:     return $\perp$            {simulation fails}
47: if $\mathbf{z}^{(n)} = \perp$ then
48:     broadcast $\perp$ and goto 8
49: broadcast $\mathbf{z}^{(n)}$
50: receive $\mathbf{z}^{(j)}$ for all $j \in [n-1]$
51: for $j = 1$ to $n - 1$ do
52:     if $\mathbf{z}^{(j)} = \perp$ then goto 8
53:     $(b, i^{(j)}) \leftarrow \mathsf{Vrf}(\bar{\mathbf{A}}, \mathbf{b}^{(j)}, \mathbf{v}^{(j)}, c, \mathbf{z}^{(j)})$
54:     if $(b, i^{(j)}) = (0, -1)$ then
55:         broadcast $abort$ and return $\perp$
56: $\mathbf{z} \leftarrow \sum_{j=1}^{n} \mathbf{z}^{(j)}$, $\mathbf{z} \leftarrow \mathsf{Compress}(\mathbf{z})$
57: $t \leftarrow \mathsf{IntIndex}_{\omega,n}(i^{(1)}, \ldots, i^{(n)})$
58: $auth \leftarrow \mathsf{BuildAuth}(t, tree, h)$
59: return $sig = (c, \mathbf{z}, auth)$
```

Fig. 6. Simulation of signature generation. Simulator $\mathsf{SimSign}_n$ assumes that $\mathsf{SimKGen}_n$ has been previously invoked. Algorithm Search is given in Fig. 5, and Vrf in Fig. 4.

list and returns its answer. Otherwise, for queries to $\mathsf{O}_{\mathsf{Expand}}, \mathsf{O}_{\mathsf{G}}, \mathsf{O}_{\mathsf{F}}$, reduction R selects a uniformly random answer from the respective range and updates the respective list. However, for each query to O_{H}, the counter ctr is incremented by one. Then, the answer $h_{ctr} \in \mathbb{T}_\kappa$ is returned and the list L_{H} is updated.
Honest signer simulation. R invokes Sign_n exactly as given in Fig. 4.
Forgery. When A returns a forgery $(sig^* = (c^*, \mathbf{z}^*, auth^*), m^*)$, R proceeds as follows: It returns $(0, 0, \perp)$ if $m^* \in \mathrm{L}_m$ or $\mathsf{Verify}(pk, m^*, sig^*) \neq 1$. Otherwise, R finds an index $i^* \in [q_{\mathsf{Sign}} + q_{\mathsf{H}}]$ such that $c^* = h_{i^*}$, and returns (i^*, t^*, out^*), where $t^* \in \{0, \ldots, \omega^n - 1\}$ is included in $auth^*$ and $out^* = (root^*, c^*, \mathbf{z}^*, auth^*, m^*)$. $root^*$ is obtained by running Verify. We let $\Pr[H_i]$ denote the probability that R does not return $(0, 0, \perp)$ at hybrid H_i. Then we have $\Pr[H_0] = \mathrm{Adv}^{\text{UF-CMA}}_{\mathsf{DSig},\mathsf{A}}(pp)$.
<u>**Hybrid** H_1:</u> In this hybrid we modify R from H_0 as follows:
Random oracle simulation. Oracle O_{F} is simulated as follows:

1. If $\mathsf{O}_{\mathsf{F}}(str)$ is set, then return $\mathsf{O}_{\mathsf{F}}(str)$.
2. Select $\mathsf{O}_{\mathsf{F}}(str) \leftarrow_{\$} \{0,1\}^{\ell_{\mathsf{F}}}$.
3. If there exists a pair $(str', \mathsf{O}_{\mathsf{F}}(str')) \in \mathrm{L}_{\mathsf{F}}\colon (str \neq str') \wedge (\mathsf{O}_{\mathsf{F}}(str) = \mathsf{O}_{\mathsf{F}}(str'))$, then simulation fails.
4. If there exists $(str', \mathsf{O}_{\mathsf{F}}(str')) \in \mathrm{L}_{\mathsf{F}}\colon str' = \mathsf{O}_{\mathsf{F}}(str)$, then simulation fails.
5. Return $\mathsf{O}_{\mathsf{F}}(str)$ and update L_{F}.

Note that oracle O_{F} is simulated in a way that excludes collisions and chains. This ensures that each node output by algorithm $\mathsf{HashTree}$ has a unique preimage, and prevents spanning hash trees with cycles. This simulation is within statistical distance of at most $(q_{\mathsf{F}}^2 + q_{\mathsf{F}})/2^{\ell_{\mathsf{F}}}$ from an oracle that allows collisions and chains.
Honest signer simulation. R selects $c \leftarrow_{\$} \mathbb{T}_\kappa$ and computes signature part $\mathbf{z}^{(n)}$ without interacting with A. After that, R proceeds as in previous hybrid by sending out $(n, g^{(n)})$. Upon receiving $(j, g^{(j)})$ for all $j \in [n-1]$, R finds corresponding preimages $(\mathbf{v}^{(j)}, \mathbf{b}^{(j)})$. Then, R proceeds by computing $root$ and programming O_{H} such that $c := \mathsf{O}_{\mathsf{H}}(root, m, \mathbf{b})$. Simulation fails if for any $g^{(j)}$ more that one preimage were found or no corresponding preimage exists in L_{G}. Note that H_1 is identical to H_0 from A's point of view, except at simulating O_{F} and the events bad_1, bad_2, bad_3 appeared in Fig. 6, where bad_1 is the event that at least one collision is found during at most $q_{\mathsf{G}} + nq_{\mathsf{Sign}}$ queries to O_{G}, bad_3 is the event that A predicted one of the $n-1$ outputs of O_{G} without querying it, and bad_2 is the event that programming O_{H} fails at least once out of q_{Sign} queries to O_{H} due to one of the following cases:

1. O_{G} has been queried by A on $(\mathbf{v}^{(n)}, \mathbf{b}^{(n)})$ during at most $q_{\mathsf{G}} + nq_{\mathsf{Sign}}$ queries. This means that A knows $(root, m, \mathbf{b})$ and could intentionally query O_{H} on $(root, m, \mathbf{b})$.
2. $\mathsf{O}_{\mathsf{H}}(root, m, \mathbf{b})$ has been set during at most $q_{\mathsf{H}} + q_{\mathsf{Sign}}$ prior queries to O_{H}.

Therefore $\left|\Pr[H_1] - \Pr[H_0]\right| \leq \frac{q_{\mathsf{F}}^2 + q_{\mathsf{F}}}{2^{\ell_{\mathsf{F}}}} + \Pr[bad_1] + \Pr[bad_2] + \Pr[bad_3]$, where

$$\Pr[bad_1] \leq \frac{(q_{\mathsf{G}} + nq_{\mathsf{Sign}})(q_{\mathsf{G}} + nq_{\mathsf{Sign}} + 1)/2}{2^{\ell_{\mathsf{G}}}} \leq \frac{(q_{\mathsf{G}} + nq_{\mathsf{Sign}} + 1)^2}{2^{\ell_{\mathsf{G}}+1}},$$
$$\Pr[bad_2] \leq q_{\mathsf{Sign}}\Big(\frac{q_{\mathsf{G}} + nq_{\mathsf{Sign}}}{|\mathbb{T}_\kappa|} + \frac{q_{\mathsf{H}} + q_{\mathsf{Sign}}}{|\mathbb{T}_\kappa|}\Big), \text{ and } \Pr[bad_3] \leq \frac{nq_{\mathsf{Sign}}}{2^{\ell_{\mathsf{G}}}}.$$

<u>**Hybrid H_2**</u>: This hybrid is identical to H_1 except at the following points:

Honest signer simulation. R does not generate $\mathbf{z}^{(n)}$ honestly, and simulates rejection sampling as follows: With probability δ, sample $\mathbf{v}_0^{(n)}, \ldots, \mathbf{v}_{\omega-1}^{(n)} \leftarrow\!\!\$\, R_q^k$ and set $\mathbf{z}^{(n)} = \perp$. Otherwise, sample $i^{(n)} \leftarrow\!\!\$\, \{0, \ldots, \omega-1\}$ and $\mathbf{z}^{(n)} \leftarrow\!\!\$\, D_{\mathbb{Z}^N,\sigma}^{k+\ell}$. Then, compute $\mathbf{v}_{i^{(n)}}^{(n)} = \bar{\mathbf{A}} \cdot \mathbf{z}^{(n)} - \mathbf{b}^{(n)} c \pmod q$. The remaining $\mathbf{v}_i^{(n)}$, for all $i \in \{0, \ldots, \omega-1\} \backslash \{i^{(n)}\}$, are computed honestly (see Fig. 6). By Lemma 5 in Appendix D we obtain $\left|\Pr[H_2] - \Pr[H_1]\right| \leq q_{\mathsf{Sign}} \cdot \frac{2^{-\Omega(N)-100+1}}{M}$.

<u>**Hybrid H_3**</u>: At this point, simulation does not rely on the actual secret key share $\mathbf{s}^{(n)}$ (see $\mathsf{SimSign}_n$, Fig. 6). In this hybrid, R samples $seed \leftarrow\!\!\$\, \{0,1\}^{\ell_{seed}}$ and programs $\mathsf{O}_{\mathsf{Expand}}$ such that $\mathbf{A} := \mathsf{O}_{\mathsf{Expand}}(seed)$. Then, it computes $seed^{(n)}$ a posteriori, after extracting A's committed shares $seed^{(1)}, \ldots, seed^{(n-1)}$ via algorithm Search, i.e., by searching the recorded queries to O_{G} (see $\mathsf{SimKGen}_n$, Fig. 5). Note that H_3 is identical to H_2 from A's point of view, except at programming $\mathsf{O}_{\mathsf{Expand}}$ and the events bad'_1, bad'_2, bad'_3 appeared in $\mathsf{SimKGen}_n$. Therefore we have $\left|\Pr[H_3] - \Pr[H_2]\right| \leq \frac{q_E}{q^{k\ell N}} + \frac{(q_{\mathsf{G}}+1)^2}{2^{\ell_{\mathsf{G}}+1}} + \frac{q_{\mathsf{G}}}{2^{\ell_{seed}}} + \frac{n}{2^{\ell_{\mathsf{G}}}}$, where $\Pr[bad'_1] \leq ((q_{\mathsf{G}}+1)q_{\mathsf{G}}/2)/2^{\ell_{\mathsf{G}}}$ is the probability that at least one collision is found during at most q_{G} queries to O_{G}, $\Pr[bad'_2]$ is the probability that programming O_{G} fails, which occurs if O_{G} has been previously queried by A on $(seed^{(n)}, n)$ during at most q_{G} queries, and the probability that guessing a uniformly random $seed^{(n)}$ is at most $1/2^{\ell_{seed}}$ for each query, and $\Pr[bad'_3] \leq n/2^{\ell_{\mathsf{G}}}$ is the probability that A predicted one of the $n-1$ outputs of O_{G} without querying it.

<u>**Hybrid H_4**</u>: This hybrid is identical to H_3 except that R samples public key share $\mathbf{b}^{(n)} \leftarrow\!\!\$\, R_q^k$ instead of computing $\mathbf{b}^{(n)} = \bar{\mathbf{A}} \cdot \mathbf{s} \pmod q$, where $\mathbf{s}^{(n)} \leftarrow\!\!\$\, S_\eta^{k+\ell}$. If A can distinguish between H_3 and H_4, then A can be used to break the MLWE assumption w.r.t. pp'. Therefore we have $\left|\Pr[H_4] - \Pr[H_3]\right| \leq \mathrm{Adv}_{\mathsf{D}}^{\mathsf{MLWE}}(pp')$.

<u>**Hybrid H_5**</u>: In this hybrid, R computes its public key share $\mathbf{b}^{(n)}$ a posteriori, after extracting A's committed shares $\mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(n-1)}$ via Search, i.e., by searching the recorded queries to O_{G} (see $\mathsf{SimKGen}_n$, Fig. 5). Note that H_5 is identical to H_4 from A's point of view, except at the events bad'_4, bad'_5, bad'_6 appeared in $\mathsf{SimKGen}_n$. Therefore we have $\left|\Pr[H_5] - \Pr[H_4]\right| \leq \frac{(q_{\mathsf{G}}+1)^2}{2^{\ell_{\mathsf{G}}+1}} + \frac{q_{\mathsf{G}}}{q^{kN}} + \frac{n}{2^{\ell_{\mathsf{G}}}}$, where $\Pr[bad'_4], \Pr[bad'_5]$, and $\Pr[bad'_6]$ are calculated as in H_3.

<u>**Forking**</u>: Given $\mathbf{A}' \in R_q^{k \times (\ell+1)}$ as input, the goal is to solve MSIS w.r.t. pp''. To this end, R writes $\mathbf{A}' = [\mathbf{A}|\mathbf{b}] \in R_q^{k\times\ell} \times R_q^k$ and generates the remaining parameters of DSig to obtain pp and run A on input pp. This does not change the view of A at all. In order to use the forking lemma (Appendix B.1), we define its instance generator algorithm IGen such that it outputs $(\mathbf{A}, \mathbf{b})$. Then, R runs forking algorithm $\mathsf{Frk}_{\mathbb{T}_\kappa,\mathsf{A}}$ on input $(\mathbf{A}, \mathbf{b})$. With probability frk, we obtain two forgeries out, out', where $out = (root, c, \mathbf{z}, auth, m)$ and $out' = (root', c', \mathbf{z}', auth', m')$. Thus we obtain $\Pr[H_5] = acc \leq \frac{(q_{\mathsf{H}}+q_{\mathsf{Sign}})\cdot\omega^n}{|\mathbb{T}_\kappa|} + \sqrt{(q_{\mathsf{H}}+q_{\mathsf{Sign}})\cdot\omega^n \cdot frk}$. Simulating O_{F} as given in hybrid H_1 ensures that both $auth = (t, str_0, \ldots, str_{h-1})$ and $auth' = (t', str'_0, \ldots, str'_{h-1})$ include the same sequence of hash values, i.e., $str_i = str'_i$ for all $i \in \{0, \ldots, h-1\}$ and $h = \lceil \log(\omega^n) \rceil$. By the forking lemma

we have $root = root'$, $t = t'$, $m = m'$, and $c \neq c'$. Moreover, the view of A is identical in both executions until the forking index i^*. Since $auth = auth'$, we have $\mathbf{w} = \mathbf{w}'$, where $\mathbf{w} = \bar{\mathbf{A}} \cdot \mathbf{z} - \mathbf{b}c$ and $\mathbf{w}' = \bar{\mathbf{A}} \cdot \mathbf{z}' - \mathbf{b}c'$. Thus, we obtain

$$[\mathbf{I}_k|\mathbf{A}|\mathbf{b}] \cdot \begin{bmatrix} \mathbf{z} - \mathbf{z}' \\ c' - c \end{bmatrix} = \mathbf{0}.$$

Note that $0 < \|c' - c\| \leq 2\sqrt{\kappa}$, and since both forgeries are valid we have $\|\mathbf{z}\| \leq B$ and $\|\mathbf{z}'\| \leq B$. Therefore, $\|\mathbf{z} - \mathbf{z}'\| \leq 2B$ and the vector $[\mathbf{z} - \mathbf{z}'|c' - c]^\top$ constitutes a non-trivial solution to MSIS w.r.t. pp''. Hence, $frk \leq \mathrm{Adv}_{\mathsf{A}}^{\mathsf{MSIS}}(pp'')$.

3.3 Concrete Parameters

In this section we propose sample parameters for our distributed signature protocol and the protocols introduced by Damgård et al. [17] and Chen [13]. The parameters are presented in Table 3. The corresponding sizes of public keys and signatures as well as the communication cost of key and signature generation are given in Table 1. The hardness of the underlying instances of MLWE and MSIS are estimated as described in Appendix C. We provide a proof-of-concept implementation for our protocol in C++ [2] and evaluate it on a regular laptop (Macbook Air M1) with 3.2 GHz CPU and 8 GB RAM. The performance results are shown in Table 1. In the following we highlight some key points regarding the parameter selection.

The parameters of our protocol are chosen according to the constraints given in Table 2. In particular, the modulus q together with the standard deviation σ are selected such that each commitment generated by a signer is distributed statistically close to uniform over R_q^k, and the underlying instances of MLWE and MSIS are sufficiently hard. We note that for all schemes we set $q > \beta$, where β is the bound of a solution to MSIS. This prevents the existence of trivial solutions like $(q, 0, \ldots, 0)$. The number of commitments ω is chosen such that with probability very close to 1, signers compute a response without the need to restart the signing protocol, i.e., the number of restarts per signer is $S = 1/(1 - 2^{-25}) \approx 1$. Therefore, the whole signing protocol does not abort at all with very high probability, i.e., $\bar{M} \approx 1$. Note that this value of S is reasonable in practice, and there is no need to increase ω to obtain a value of S more closer to 1, e.g., $S = 1/(1 - 2^{-50})$. Increasing ω would reduce the performance of the signing protocol in a significant way, which is not desired in practice. For a reasonably small $\bar{M}$, [17] suggests to set $\alpha = 11n$ to obtain $\bar{M} = 3$, while $\alpha = 8.5n$ is suggested in [13] so that $\bar{M} \approx 5$. We set $\alpha = 11n = 77$ to obtain $\bar{M} = 3$ for both schemes. The trapdoor homomorphic commitment scheme used in [17] commits to a single element from R_q. Therefore, a commitment to each coefficient of a vector from R_q is computed separately[3]. Concrete parameters of the commitment scheme are given by the tuple $(N, q, s) = (1024, \approx 2^{45}, \approx 2^{25})$. The remaining

[2] Source code: `https://anonymous.4open.science/r/distSig-Lattice-2D48`

[3] As stated in [17], there is no efficiency gain from extending the construction to commit to vectors from R_q.

Table 2. Parameters of our distributed signature protocol.

Parameter	Description	Bounds
n	No. signers	ω^n small
N	Defines the ring R	power of two
k, ℓ	Dimension of matrix $\mathbf{A}$	$k, \ell \in \mathbb{N}_{>0}$
q	Modulus	prime, $q = 1 \pmod{2N}$
η	Bound of $\lVert \mathbf{s}^{(n)} \rVert_\infty$	$\eta \in \mathbb{Z}_{>0}$
κ	Specifies the set $\mathbb{T}_\kappa$	$2^\kappa \binom{N}{\kappa} \geq 2^\lambda$
σ	Standard deviation of $\mathbf{z}^{(n)}$	$\sigma = \alpha \lVert \mathbf{s}c \rVert = \alpha\kappa\eta\sqrt{(k+\ell)N}$, $\alpha > 0$, $\sigma > \frac{2N \cdot q^{\frac{k}{k+\ell} + \frac{2}{N(k+\ell)}}}{\sqrt{2\pi}}$
ω	No. vectors $\mathbf{y}_i$	$\omega \in \mathbb{N}_{>1}$, $(1 - \frac{1-2^{-100}}{M})^\omega \leq \delta$, $M = \exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2})$, $\delta > 0$
h	Tree height	$h = \lceil \log(\omega^n) \rceil$
S	No. restarts of Sign_n	$S = 1/(1-\delta)$
$\bar{M}$	Total No. restarts	$\bar{M} = S^n$
B_z	Bound of $\lVert \mathbf{z}^{(n)} \rVert$	$B_z = \gamma\sigma\sqrt{(k+\ell)N}$, $\gamma > 0$
B	Bound of $\lVert \mathbf{z} \rVert$	$B = \sqrt{n} B_z$
ℓ_{seed}	Input length of Expand	$\ell_{seed} \geq \lambda$
$\ell_{\mathsf{G}}, \ell_{\mathsf{F}}$	Output length of G, F	$\ell_{\mathsf{G}}, \ell_{\mathsf{F}} \geq 2\lambda$

parameters can be easily derived form this tuple. They are selected to support homomorphic additions of 7 commitments, and such that all security properties are satisfied (see [17, Section 5.2] for details). The communication cost of each signing protocol is given by the total amount of data sent per signer, including the number of restarts, i.e., $\bar{M} \cdot (|R_1| + |R_2|) + |R_3|$, where for $i \in \{1, 2, 3\}$, the term $|R_i|$ denotes the length of the bit string sent by each signer in the i^{th} round. Note that $|R_1| = 0$ in [17,13]. The size of any Gaussian element is computed according to Lemma 1, i.e., the values of t and γ are selected such that the bounds in Lemma 1 hold with probability at most 2^{-80}. Finally, we would like to note that the scheme by Chen [13] is the most suitable one for applications requiring a large number of signers.

Conclusion

In this paper we have presented a new lattice-based construction of n-out-of-n distributed signatures. Our protocol follows the Fiat-Shamir with aborts paradigm and supports applications with a small number of signers only. We proposed sample parameters and provided a comparison with similar works showing the significant improvement and practicality of our approach. An interesting extension to our work is to provide a security proof in the quantum random oracle model [10]. The possibility of both rewinding and programming the random oracle in the quantum setting have already been shown, e.g., in [19,24,31].

Table 3. Concrete parameters for our distributed signature protocol and the protocols proposed in [17,13]. The parameters consider $n = 7$ signers and target 128 bits of security. We fix $N = 256$ and set $\kappa = 23$ for all schemes so that $|\mathbb{T}_\kappa| \geq 2^{128}$ and the challenge space $\mathbb{T}_\kappa$ provides at least 128 bits of entropy. In [17,13], the total number of restarts $\bar{M}$ is denoted by M_n. The output length of hash functions is set to 256 bits.

Parameter	Our protocol	Damgård et al. [17]	Chen [13]
k	5	5	6
ℓ	7	4	9
q	$\approx 2^{45}$	$\approx 2^{27}$	$\approx 2^{43}$
η	1	3	1
ω	2	-	-
α	72090	77	77
σ	91899568	255024	78293860
σ'	-	-	37127790
$\bar{M}$	1	3	3

Acknowledgements. This work was funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Agrawal, S., Stehlé, D., Yadav, A.: Round-optimal lattice-based threshold signatures, revisited. In: Bojanczyk, M., Merelli, E., Woodruff, D.P. (eds.) 49th International Colloquium on Automata, Languages, and Programming, ICALP 2022. LIPIcs, vol. 229, pp. 8:1–8:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
3. Alkeilani Alkadri, N., El Bansarkhani, R., Buchmann, J.: On lattice-based interactive protocols: An approach with less or no aborts. pp. 41–61 (2020). https://doi.org/10.1007/978-3-030-55304-3_3
4. Alkeilani Alkadri, N., Harasser, P., Janson, C.: BlindOR: an efficient lattice-based blind signature scheme from OR-proofs. pp. 95–115 (2021). https://doi.org/10.1007/978-3-030-92548-2_6
5. Becker, A., Ducas, L., Gama, N., Laarhoven, T.: New directions in nearest neighbor searching with applications to lattice sieving. pp. 10–24 (2016). https://doi.org/10.1137/1.9781611974331.ch2
6. Bellare, M., Davis, H., Günther, F.: Separate your domains: NIST PQC KEMs, oracle cloning and read-only indifferentiability. pp. 3–32 (2020). https://doi.org/10.1007/978-3-030-45724-2_1
7. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. pp. 390–399 (2006). https://doi.org/10.1145/1180405.1180453

8. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. pp. 62–73 (1993). https://doi.org/10.1145/168588.168596
9. Bendlin, R., Krehbiel, S., Peikert, C.: How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. pp. 218–236 (2013). https://doi.org/10.1007/978-3-642-38980-1_14
10. Boneh, D., Dagdelen, Ö., Fischlin, M., Lehmann, A., Schaffner, C., Zhandry, M.: Random oracles in a quantum world. pp. 41–69 (2011). https://doi.org/10.1007/978-3-642-25385-0_3
11. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. pp. 565–596 (2018). https://doi.org/10.1007/978-3-319-96884-1_19
12. Boschini, C., Takahashi, A., Tibouchi, M.: MuSig-L: Lattice-based multi-signature with single-round online phase. pp. 276–305 (2022). https://doi.org/10.1007/978-3-031-15979-4_10
13. Chen, Y.: DualMS: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In: Advances in Cryptology – CRYPTO 2023. pp. 716–747 (2023). https://doi.org/10.1007/978-3-031-38554-4_23
14. Chen, Y.: Réduction de réseau et sécurité concrete du chiffrement completement homomorphe. Ph.D. thesis, ENS-Lyon, France (2013)
15. Chen, Y., Nguyen, P.Q.: BKZ 2.0: Better lattice security estimates. pp. 1–20 (2011). https://doi.org/10.1007/978-3-642-25385-0_1
16. Cozzo, D., Smart, N.P.: Sharing the LUOV: Threshold post-quantum signatures. pp. 128–153 (2019). https://doi.org/10.1007/978-3-030-35199-1_7
17. Damgård, I., Orlandi, C., Takahashi, A., Tibouchi, M.: Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices **35**(2), 14 (Apr 2022). https://doi.org/10.1007/s00145-022-09425-3
18. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. pp. 307–315 (1990). https://doi.org/10.1007/0-387-34805-0_28
19. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Security of the Fiat-Shamir transformation in the quantum random-oracle model. pp. 356–383 (2019). https://doi.org/10.1007/978-3-030-26951-7_13
20. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
21. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehle, D.: Crystals – dilithium: Digital signatures from module lattices. Cryptology ePrint Archive, Paper 2017/633 (2017), https://eprint.iacr.org/archive/2017/633/20170627:201152
22. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. pp. 197–206 (2008). https://doi.org/10.1145/1374376.1374407
23. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Research & Development (71), 1–8 (1983)
24. Liu, Q., Zhandry, M.: Revisiting post-quantum Fiat-Shamir. pp. 326–355 (2019). https://doi.org/10.1007/978-3-030-26951-7_12
25. Lyubashevsky, V.: Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. pp. 598–616 (2009). https://doi.org/10.1007/978-3-642-10366-7_35
26. Lyubashevsky, V.: Lattice signatures without trapdoors. pp. 738–755 (2012). https://doi.org/10.1007/978-3-642-29011-4_43

27. Lyubashevsky, V., Peikert, C., Regev, O.: A toolkit for ring-LWE cryptography. pp. 35–54 (2013). `https://doi.org/10.1007/978-3-642-38348-9_3`
28. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. pp. 245–254 (2001). `https://doi.org/10.1145/501983.502017`
29. Micciancio, D., Regev, O.: Lattice-based Cryptography, pp. 147–191. Springer Berlin Heidelberg (2009)
30. Schnorr, C., Euchner, M.: Lattice basis reduction: Improved practical algorithms and solving subset sum problems. Math. Program. **66**, 181–199 (1994), `https://doi.org/10.1007/BF01581144`
31. Zhandry, M.: How to record quantum queries, and applications to quantum indifferentiability. pp. 239–268 (2019). `https://doi.org/10.1007/978-3-030-26951-7_9`

A More Related Work

Threshold Signatures. Few works proposed lattice-based constructions of t-out-of-n threshold signatures [9,16,11,1]. The first one by Bendlin et al. [9] gives a threshold variant of standard hash-and-sign signatures by Gentry et al. [22]. The main downside of this protocol is that only a priori bounded number of online non-interactive signing operations can be performed before an offline interactive protocol must be performed. This offline protocol includes a threshold Gaussian sampling phase, which is carried out using generic multiparty computation (MPC). Cozzo and Smart [16] show that the lattice-based signature schemes that have been submitted to the NIST post-quantum standardization process have significant issues when converting them into threshold ones using relatively generic MPC techniques. The main issue is the need to carry out the rejection sampling procedure, which requires to keep intermediate values secret until after performing rejection sampling and comparing them with given constants. Moreover, they require several rounds of communication and a mixture of linear and non-linear operations that incur costly transformations between both representations. Boneh et al. [11] propose a generic framework that requires several other cryptographic primitives as building blocks, including deterministic signatures, threshold fully homomorphic encryption, and a homomorphic signature scheme. Due to the involvement of heavy cryptographic primitives, it is not clear if their construction can be adapted in practical applications. Agrawal et al. [1] improve the construction by Boneh et al. [11] bringing it closer to practice.

B Additional Background

The next lemma is for the tail bound of Gaussian vectors.

Lemma 1 ([26, Lemma 4.4]). *Let $\sigma, t, \gamma \in \mathbb{R}_{>0}$ and $m \in \mathbb{N}_{>0}$. Then we have:*

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

We rely on the following lemma, which is a certain regularity theorem.

Lemma 2 ([27, Corollary 7.5]). *Let* $\mathbf{A} \leftarrow\!\!\$\; R_q^{k \times \ell}$ *and* $\bar{\mathbf{A}} = [\mathbf{I}_k | \mathbf{A}] \in R_q^{k \times (k+\ell)}$. *Let* $\sigma > \frac{2N \cdot q^{\frac{k}{k+\ell} + \frac{2}{N(k+\ell)}}}{\sqrt{2\pi}}$ *and* $\mathbf{x} \leftarrow\!\!\$\; D^{k+\ell}_{\mathbb{Z}^N, \sigma}$. *Then, the distribution of* $\bar{\mathbf{A}} \cdot \mathbf{x} \pmod q$ *is within statistical distance* $2^{-\Omega(N)}$ *of the uniform distribution over* R_q^k.

The next lemma is a variant of the rejection sampling lemma specified for $D_{\mathbb{Z}^m, \sigma}$.

Lemma 3 ([26, Theorem 4.6]). *Define* $V := \{\mathbf{v} \in \mathbb{Z}^m \colon \|\mathbf{v}\| \leq T\}$, *where* $T > 0$. *Let* $\sigma = \alpha T$ *for some* $\alpha > 0$, *and* $h \colon V \to \mathbb{R}$ *be a probability distribution. Then, there exists a constant* $M > 0$ *such that* $\exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2}) \leq M$, *and the following two algorithms are within statistical distance of at most* $2^{-100}/M$:

1. $\mathbf{v} \leftarrow\!\!\$\; h$; $\mathbf{z} \leftarrow\!\!\$\; D_{\mathbb{Z}^m, \sigma, \mathbf{v}}$; *output* $(\mathbf{z}, \mathbf{v})$ *with probability* $\frac{1 - 2^{-100}}{M}$.
2. $\mathbf{v} \leftarrow\!\!\$\; h$; $\mathbf{z} \leftarrow\!\!\$\; D_{\mathbb{Z}^m, \sigma}$; *output* $(\mathbf{z}, \mathbf{v})$ *with probability* $1/M$.

We let $\mathsf{RejSamp}$ denote an algorithm that carries out rejection sampling on $\mathbf{z}$, where $\mathbf{z} \leftarrow\!\!\$\; D_{\mathbb{Z}^m, \sigma, \mathbf{v}}$, $\|\mathbf{v}\| \leq T$, and $\sigma = \alpha T$. That is, on input $(\mathbf{z}, \mathbf{v})$, $\mathsf{RejSamp}$ returns 1 if $\mathbf{z}$ is accepted and 0 if rejected. By Lemma 3, the output 1 indicates that the distribution of $\mathbf{z}$ is within statistical distance of at most $2^{-100}/M$ from $D_{\mathbb{Z}^m, \sigma}$, where $\exp(\frac{12}{\alpha} + \frac{1}{2\alpha^2}) \leq M$. $\mathsf{RejSamp}$ returns 1 with probability $\approx 1/M$, and hence the expected number of restarts necessary to return 1 is given by M.

B.1 Forking Lemma

Let $\mathcal{C}$ be some finite set and $\mathcal{R}$ be some randomness space. Let IGen be a PPT algorithm, and consider an algorithm A that, on input an instance $x \in \mathsf{IGen}$ and random values $h_1, \ldots, h_q \in \mathcal{C}$, returns a pair $(\mathit{idx}, \mathit{out})$, where $0 \leq \mathit{idx} \leq q$ and out is a side output related to h_{idx}. The index $\mathit{idx} = 0$ indicates that A has failed to compute a side output out related to any of the values $h_1, \ldots, h_q$. The general forking lemma [7] gives a lower bound on the probability of the forking experiment in which A, if run twice on the same instance x and randomness $r \in \mathcal{R}$, but partially different values from $\mathcal{C}$, will return the same index idx and two side outputs out and out', which are related to the values h_{idx} and h'_{idx}, respectively. The experiment fails if both runs of A return two different indices, or if $h_{\mathit{idx}} = h'_{\mathit{idx}}$. For the security proof of our n-out-of-n distributed signature protocol we need a minor version of the general forking lemma. This version was given in [4]. It considers an algorithm A that further returns a second index as part of the output, i.e., A returns a tuple $(\mathit{idx}_1, \mathit{idx}_2, \mathit{out})$, where idx_1 and out are as before, and $0 \leq \mathit{idx}_2 < \omega$ for $\omega \in \mathbb{N}_{>0}$. The forking experiment succeeds only if both runs of A return the same pair of indices $(\mathit{idx}_1, \mathit{idx}_2)$ and $h_{\mathit{idx}_1} \neq h'_{\mathit{idx}_1}$.

Lemma 4. *Let* $q, \omega \in \mathbb{N}_{>0}$, $\mathcal{C}$ *be a finite set of size* $|\mathcal{C}| \geq 2$, *and* $\mathcal{R}$ *be a randomness space. Let* IGen *be a PPT algorithm, and* A *be a PPT algorithm that, on input* $x \in \mathsf{IGen}$ *and* $h_1, \ldots, h_q \in \mathcal{C}$, *outputs a tuple* $(\mathit{idx}_1, \mathit{idx}_2, \mathit{out})$, *where* $0 \leq \mathit{idx}_1 \leq q$ *and* $0 \leq \mathit{idx}_2 < \omega$. *Define the accepting probability and the forking probability of* A *by*

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

$\text{Exp}^{\text{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$
```
1: $x \leftarrow\$ \mathsf{IGen}$
2: $h_1, \ldots, h_q \leftarrow\$ \mathcal{C}$
3: $(idx_1, idx_2, out) \leftarrow\$ \mathsf{A}(x, h_1, \ldots, h_q)$
4: if $1 \leq idx_1 \leq q$ then
5:     return $1$
6: return $0$
```

$\text{Exp}^{\text{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$
```
1: $x \leftarrow\$ \mathsf{IGen}$
2: $(b, out, out') \leftarrow\$ \mathsf{Frk}_{\mathcal{C},\mathsf{A}}(x)$
3: return $b$
```

$\mathsf{Frk}_{\mathcal{C},\mathsf{A}}(x)$
```
1: $r \leftarrow\$ \mathcal{R}$
2: $h_1, \ldots, h_q \leftarrow\$ \mathcal{C}$
3: $(idx_1, idx_2, out) \leftarrow \mathsf{A}(x, h_1, \ldots, h_q; r)$
4: if $idx_1 = 0$ then
5:     return $(0, \perp, \perp)$
6: $h'_{idx_1}, \ldots, h'_q \leftarrow\$ \mathcal{C}$
7: $(idx'_1, idx'_2, out') \leftarrow$
       $\leftarrow \mathsf{A}(x, h_1, \ldots, h_{idx_1 - 1}, h'_{idx_1}, \ldots, h'_q; r)$
8: if $(idx_1 = idx'_1) \wedge (idx_2 = idx'_2) \wedge$
       $(h_{idx_1} \neq h'_{idx_1})$ then
9:     return $(1, out, out')$
10: return $(0, \perp, \perp)$
```

Fig. 7. Definition of experiments $\text{Exp}^{\text{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$, $\text{Exp}^{\text{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$, and forking algorithm $\mathsf{Frk}_{\mathcal{C},\mathsf{A}}$.

where the experiments $\text{Exp}^{\text{Acc}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$ *and* $\text{Exp}^{\text{Frk}}_{\mathsf{IGen},\mathcal{C},\mathsf{A}}$ *are depicted in Fig. 7. Then, we have* $frk \geq acc \cdot \left(\frac{acc}{q \cdot \omega} - \frac{1}{|\mathcal{C}|}\right)$. *Alternatively,* $acc \leq \frac{q \cdot \omega}{|\mathcal{C}|} + \sqrt{q \cdot \omega \cdot frk}$.

C Hardness Estimation of MLWE and MSIS

In this section, we explain the methodology that we follow in this work to estimate the hardness of MLWE and MSIS. First, we remark that all known algorithms solving MLWE and MSIS do not exploit their algebraic structure.

Estimating the hardness of MLWE w.r.t. $pp = (N, k, \ell, q, \eta)$ is carried out by using the LWE-*Estimator*[4] presented by Albrecht et al. [2].

Given $pp = (N, k, \ell, q, \beta)$ and $\mathbf{A} = [a_{i,j}]_{1 \leq i \leq k, 1 \leq j \leq \ell} \in R_q^{k \times \ell}$, the hardness of MSIS w.r.t. pp is equivalent to solving the Shortest Vector Problem (SVP), i.e., finding a non-trivial vector, whose ℓ_2-norm is bounded by β, in the lattice $\{\mathbf{x} \in \mathbb{Z}^m : \mathbf{0} = [\mathbf{I}_d | \mathbf{A}'] \cdot \mathbf{x} \pmod q\}$, where $d = kN$, $m = (k + \ell)N$, and $\mathbf{A}'$ is the matrix obtained by computing the *rotation matrix* of each entry of $\mathbf{A}$, i.e.,

$$\mathbf{A}' = \begin{bmatrix} \mathsf{Rot}(a_{1,1}) & \ldots & \mathsf{Rot}(a_{1,\ell}) \\ \vdots & \ddots & \vdots \\ \mathsf{Rot}(a_{k,1}) & \ldots & \mathsf{Rot}(a_{k,\ell}) \end{bmatrix} \in \mathbb{Z}_q^{kN \times \ell N}.$$

We recall that the rotation matrix of any $a = \sum_{i=0}^{N-1} a_i X^i \in R$ is defined by

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

where $\mathbf{a} = (a_0, \ldots, a_{N-1})^\top$, $\mathsf{rot}(a) := (-a_{N-1}, a_0, \ldots, a_{N-2})^\top$, and for all other $k \in \{2, \ldots, N-1\}$: $\mathsf{rot}^k(a) := \mathsf{rot}(\mathsf{rot}^{k-1}(a))$.

[4] `https://github.com/malb/lattice-estimator`

The best known algorithm for finding short non-trivial vectors is due to Schnorr and Euchner [30]. It is called the Block-Korkine-Zolotarev algorithm (BKZ), and was improved in practice by Chen and Nguyen [15]. As a subroutine, BKZ uses an SVP solver in lattices of dimension b, where b is called the *block size*. The best known classical algorithm for SVP with no memory restrictions is due to Becker et al. [5], and it takes time $\approx 2^{0.292\,b}$. The time required by BKZ to run with block size b on an m-dimensional lattice $\mathcal{L}$ is given by (see, e.g. [5])

$$8m\,2^{0.292\,b+16.4}. \tag{1}$$

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. [15,14])

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

and $\det(\mathcal{L})$ is the determinant of $\mathcal{L}$. Micciancio and Regev [29] showed that it is better to run algorithm BKZ with a maximum of $m = \sqrt{d\log(q)/\log(\delta)}$ columns of the matrix $[\mathbf{I}_d|\mathbf{A}']$. The coefficients of the solution output by BKZ and correspond to the dropped columns are then set to zero. This allows to find a non-zero vector of length $\min(q, 2^{2\sqrt{d\log(q)\log(\delta)}})$. In other words, when considering $\delta^m \det(\mathcal{L})^{1/m}$ as a function of m, Micciancio and Regev [29] showed that the minimum of this function is given by the value $2^{2\sqrt{d\log(q)\log(\delta)}}$, and it is obtained when $m = \sqrt{d\log(q)/\log(\delta)}$. Therefore, in order to compute the time required by BKZ to solve MSIS w.r.t. pp, we first determine δ by setting $\beta = 2^{2\sqrt{d\log(q)\log(\delta)}}$, where $d = kN$ and $m = (k+\ell)N$. After that, we compute the minimum block size b required to achieve δ by using (2). The resulted b is put in (1) to obtain the desired time.

D Indistinguishability of Hybrids H_2 and H_1

The following lemma establishes the statistical distance between the hybrids H_2 and H_1 defined in the proof of Theorem 1.

Lemma 5. *Let σ be as in Lemma 2, M be as in Lemma 3, and $\delta > 0$ such that $(1 - \frac{1-2^{-100}}{M})^\omega \leq \delta$. Let $\mathbf{A} \leftarrow\!\!\$\; R_q^{k\times\ell}$, $\bar{\mathbf{A}} = [\mathbf{I}_k|\mathbf{A}] \in R_q^{k\times(k+\ell)}$, $\mathbf{s} \leftarrow\!\!\$\; S_\eta^{k+\ell}$, and $\mathbf{b} = \bar{\mathbf{A}}\cdot\mathbf{s} \pmod q$. Then, the output distributions of the algorithms A_0 and A_1 defined in Fig. 8 are within statistical distance of at most $2^{-\Omega(N)+1}\cdot 2^{-100}/M$.*

Proof. The proof is similar to the one of [12, Lemma B.8], which is performed via standard hybrid arguments. The only difference here is that in algorithm A_0 rejection sampling is carried out at most ω times, using Gaussian masking vectors $\mathbf{y}_0, \ldots, \mathbf{y}_{\omega-1}$. The goal is to make sure that the distribution of $\mathbf{z} = \mathbf{y}_i + \mathbf{s}c$ is independent of $\mathbf{s}c$. The random choice of $\rho \in [0,1)$ and doing the test in line 13 is a standard implementation of the rejection sampling procedure. By Lemma 3, rejection sampling accepts with probability $(1-2^{-100})/M$, and $\mathbf{z}$ is

```
$\mathsf{A}_0(\bar{\mathbf{A}}, \mathbf{b}, \mathbf{s})$
$c \leftarrow_\$ \mathbb{T}_\kappa$
$T \leftarrow \{0, \ldots, \omega - 1\}$
for $j = 0$ to $\omega - 1$ do
    $\mathbf{y}_j \leftarrow_\$ D^{k+\ell}_{\mathbb{Z}^N,\sigma}$
    $\mathbf{v}_j \leftarrow \bar{\mathbf{A}} \cdot \mathbf{y}_j \pmod q$
$\mathbf{v} \leftarrow (\mathbf{v}_0, \ldots, \mathbf{v}_{\omega-1})$
$\mathbf{z}' \leftarrow \mathbf{s}c$
while $T \neq \varnothing$ do
    $i \leftarrow_\$ T$
    $T \leftarrow T \setminus \{i\}$
    $\mathbf{z} \leftarrow \mathbf{y}_i + \mathbf{z}'$
    $\rho \leftarrow_\$ [0, 1)$
    if $\rho \leq \frac{1}{M} \cdot \exp\left(\frac{-2\langle \mathbf{z}, \mathbf{z}' \rangle + \|\mathbf{z}'\|^2}{2\sigma^2}\right)$ then
        return $(\mathbf{A}, \mathbf{b}, \mathbf{v}, c, \mathbf{z}, i)$
$(\mathbf{z}, i) \leftarrow (\perp, \perp)$
return $(\mathbf{A}, \mathbf{b}, \mathbf{v}, c, \mathbf{z}, i)$
```

```
$\mathsf{A}_1(\bar{\mathbf{A}}, \mathbf{b})$
$c \leftarrow_\$ \mathbb{T}_\kappa$
$T \leftarrow \{0, \ldots, \omega - 1\}$
$i \leftarrow_\$ T$
$\mathbf{z} \leftarrow_\$ D^{k+\ell}_{\mathbb{Z}^N,\sigma}$
$\mathbf{v}_i \leftarrow \bar{\mathbf{A}} \cdot \mathbf{z} - \mathbf{b}c \pmod q$
for $j = 0$ to $\omega - 1$ do
    if $j = i$ then
        continue
    $\mathbf{y}_j \leftarrow_\$ D^{k+\ell}_{\mathbb{Z}^N,\sigma}$
    $\mathbf{v}_j \leftarrow \bar{\mathbf{A}} \cdot \mathbf{y}_j \pmod q$
$\mathbf{v} \leftarrow (\mathbf{v}_0, \ldots, \mathbf{v}_{\omega-1})$
$\rho \leftarrow_\$ [0, 1)$
if $\rho \leq 1 - \delta$ then
    return $(\mathbf{A}, \mathbf{b}, \mathbf{v}, c, \mathbf{z}, i)$
$(\mathbf{z}, i) \leftarrow (\perp, \perp)$
return $(\mathbf{A}, \mathbf{b}, \mathbf{v}, c, \mathbf{z}, i)$
```

Fig. 8. The algorithms that show the Indistinguishability of hybrids H_2 and H_1 defined in the proof of Theorem 1.

within statistical distance of $2^{-100}/M$ from the Gaussian distribution $D^{k+\ell}_{\mathbb{Z}^N,\sigma}$. When using ω masking vectors $\mathbf{y}_0, \ldots, \mathbf{y}_{\omega-1}$, instead of only one, algorithm A_0 returns $(\mathbf{z}, i) \neq (\perp, \perp)$ with probability $1 - (1 - \frac{1-2^{-100}}{M})^\omega \leq 1 - \delta$. Lemma 2 is applied twice in order to obtain a statistical distance of $2^{-\Omega(N)}$ between a vector $\bar{\mathbf{A}} \cdot \mathbf{y} \in R_q^k$, for $\mathbf{y} \leftarrow_\$ D^{k+\ell}_{\mathbb{Z}^N,\sigma}$, and a uniformly random vector from R_q^k. We refer to [12, Lemma B.8] for more details.