entropy

MDPI

Article

DiLizium: A Two-Party Lattice-Based Signature Scheme

Jelizaveta Vakarjuk [1,2,*], Nikita Snetkov [1,2] and Jan Willemson [1]

[1] Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia; nsnetkov@cyber.ee (N.S.); janwil@cyber.ee (J.W.)
[2] STACC OÜ, Narva mnt 20, 51009 Tartu, Estonia
* Correspondence: jelizaveta.vakarjuk@cyber.ee

Abstract: In this paper, we propose DiLizium: a new lattice-based two-party signature scheme. Our scheme is constructed from a variant of the Crystals-Dilithium post-quantum signature scheme. This allows for more efficient two-party implementation compared with the original but still derives its post-quantum security directly from the Module Learning With Errors and Module Short Integer Solution problems. We discuss our design rationale, describe the protocol in full detail, and provide performance estimates and a comparison with previous schemes. We also provide a security proof for the two-party signature computation protocol against a classical adversary. Extending this proof to a quantum adversary is subject to future studies. However, our scheme is secure against a quantum attacker who has access to just the public key and not the two-party signature creation protocol.

Keywords: digital signatures; distributed signing; threshold signatures; lattice-based cryptography; Fiat–Shamir with aborts; post-quantum cryptography

Citation: Vakarjuk, J.; Snetkov, N.; Willemson, J. DiLizium: A Two-Party Lattice-Based Signature Scheme. *Entropy* **2021**, *23*, 989. https://doi.org/10.3390/e23080989

Academic Editors: Amin Sakzad and Khoa Nguyen

Received: 11 June 2021
Accepted: 28 July 2021
Published: 30 July 2021

1. Introduction

Ever since Peter Shor proposed an algorithm that was able to efficiently break most of the classical asymmetric cryptographic primitives such as RSA or ECDSA in the 1990s [1,2], research has been conducted to find quantum-resistant replacements. This work has recently been coordinated by the U.S. National Institute of Standards and Technology (NIST). In 2016, NIST announced an effort to standardise some of the proposed public key encryption algorithms, key-establishment algorithms, and signature schemes [3].

The primary focus of NIST is to obtain drop-in replacements for the current standardised primitives in order to ease the transition. However, not all of the current application areas are covered by the standardisation process.

One important class of examples is threshold signatures. In a (t, n)-threshold scheme, the secret key is shared between n users/devices. To create a valid signature, a subset of t users/devices should collaborate and use their secret key shares. Over the years, threshold versions of a number of major cryptographic algorithms including RSA and (EC)DSA have been studied [4–8]. Recent interest in threshold versions of ECDSA has been influenced by applications in blockchains. However, our motivation stems more from server-assisted RSA signatures, as proposed in the scheme by Buldas et al. [9]. This scheme has been implemented in a Smart-ID mobile application and was recognised as a qualified signature creation device (QSCD) in November 2018 [10]. In 2021, the number of Smart-ID users in the Baltic countries was estimated at 2.9 million [11].

In 2019, D. Cozzo and N. Smart analysed a number of NIST post-quantum standardisation candidates and concluded that none of them provide a threshold implementation that would be comparable in efficiency to threshold RSA or even ECDSA [12].

The goal of this research is to find a suitable version of one of the NIST signature candidates (Crystals-Dilithium) that would allow for a more efficient two-party implementation but would still provide post-quantum security. From this, we propose a concrete/specific two-party signature scheme and prove its security in this paper.

To be able to compute the private key or a signature forgery having access only to the public key, the attacker would need to solve Module Learning With Errors, Rejected Module Learning With Errors, and Module Short Integer Solution problems. These are considered hard even for quantum computers. We present security proof for the two-party signing protocol itself; however, it only provides security against a classical attacker. Extending the proof to cover quantum attackers is a task for future work.

Contributions

In this paper, we construct DiLizium, a lattice-based two-party signature scheme that follows the Fiat–Shamir with Aborts (FSwA) paradigm, and prove the security of the proposed scheme in the classical random oracle model. The security of the proposed signature scheme relies on the hardness of solving the Module-LWE and Module-SIS problems. Our two-party signature protocol is based on the scheme described in the paper by Kiltz et al., Appendix B [13]. Initially, we attempted to construct a two-party version of the Crystals-Dilithium digital signature submitted to the NIST PQC completion. However, we concluded that there are no straightforward approaches for modification thereof to the distributed version. Solutions require using a two-party computation protocol, which increases not only the signing time but also the communication complexity. The simplified version of the Crystals-Dilithium scheme from [13] is easier to work with because there are no additional bit decomposition algorithms used. Having a bit decomposition protocol would require an additional secure two-party computation protocol that would allow client and server to jointly compute high-order bits without revealing their private intermediate values. That would lead to an increased number of communication rounds and usage of additional security assumptions.

Additionally, DiLizium scheme does not require sampling from a discrete Gaussian distribution. We decided to use the scheme with uniform distribution because the Gaussian distribution is known to be hard to implement securely [14,15]. Our work follows the approach from [16], but instead of using homomorphic commitments, we decided to use the homomorphic hash function. We wanted to find an alternative to the commitment scheme introduced in [16] to increase the computational efficiency of the signature scheme. Due to the way our scheme is constructed, in the security proof, we rely on rejected Module-LWE, which is a non-standard security assumption, and it is not used in this work [16].

2. Related Work

In 2020, NIST announced fifteen third-round candidates for the PQC competition, of which seven were selected as finalists and eight were selected as alternate candidates [17]. The finalists in the digital signature category were Crystals-Dilithium [18], Falcon [19], and Rainbow [20]. Crystals-Dilithium and Falcon are both lattice-based signature schemes. Falcon has better performance, signature, and key sizes; however, its implementation is more complex, as it requires sampling from the Gaussian distribution and it utilises floating-point numbers to implement an optimised polynomial multiplication [19]. The performance of a Crystals-Dilithium signature scheme is slightly slower, and the signature and key sizes are larger than the ones in Falcon; however, the signature scheme itself has a simpler structure [18]. Rainbow is a multivariate signature scheme with fast signing and verifying processes. The size of the Rainbow signature is the shortest among the finalists; however, the public key size is the largest [20].

Due to the interest aroused by the PQC competition, several works were proposed that introduce lattice-based threshold signatures and lattice-based multisignatures. The works [21–26] focused on creating multisignatures that followed the FSwA paradigm. These schemes use rejection sampling, due to which the signing process is repeated until a valid signature is created. Additionally, intermediate values produced during the signature generation process need to be kept secret until the rejection sampling has been completed. There are currently no known techniques to prove the security of FSwA signatures if intermediate values are published before the rejection sampling is performed [16]. In

multisignatures [21–25], intermediate values are published before the rejection sampling is completed, which leads to incomplete security proofs in these works [16]. The work by M. Fukumitsu and S. Hasegawa [26] solves the problem with aborted executions of the protocol by introducing a non-standard hardness assumption (rejected Module-LWE).

In 2019, D. Cozzo and N. Smart analysed the second round NIST PQC competition signature schemes to determine whether it is possible to create threshold versions of these signature schemes [12]. The authors proposed a possible threshold version for each of the schemes using only generic Multiparty Computation (MPC) techniques, such as linear secret sharing and garbled circuits. As a result, the authors proposed that the most suitable signature scheme is Rainbow, which belongs to the multivariate family. The authors described a threshold version of Crystals-Dilithium, which is estimated to take around 12 s to produce a single signature. The authors explained that the problems with performance arise from the fact that the signature scheme consists of both linear and nonlinear operations and that it is inefficient to switch between these representations using generic MPC techniques. However, the goal of the current work is to focus on the two-party scenario. This means that some of the difficulties in D. Cozzo and N. Smart paper can be avoided.

R. Bendlin, S. Krehbiel, and C. Peikert proposed threshold protocols for generating a hard lattice with trapdoor and sampling from the discrete Gaussian distribution using the trapdoor [27]. These two protocols are the main building blocks for the Gentry–Peikert–Vaikuntanathan (GPV) signature scheme (based on hash-and-sign paradigm), where generating a hard lattice is needed for the key generation and Gaussian sampling is needed for the signing process. M. Kansal and R. Dutta proposed a lattice-based multisignature scheme with a single round signature generation that has key aggregation and signature compression in [28]. The underlying signature scheme follows neither the hash-and-sign nor FSwA paradigms, which are the main techniques used to construct lattice-based signature schemes.

In 2020, I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi proposed a lattice-based multisignature and distributed signing protocols that are based on the Dilithium-G signature scheme [16]. Dilithium-G is a version of the Crystals-Dilithium signature that requires sampling from a discrete Gaussian distribution [29]. The work contains complete classical security proofs for the proposed schemes. The work solves the problem with the aborted executions by using commitments such that, in the case of an abort, only commitment is published, the intermediate value itself stays secret. The proposed distributed signature scheme could potentially fit the Smart-ID framework; however, some questions need to be addressed. More precisely, the scheme is based on a modified version of Crystals-Dilithium from the NIST PQC competition project and uses Gaussian sampling. It is known that generating samples from the Gaussian distribution is nontrivial, which means that the insecure implementation may lead to side-channel attacks [30]. The open question is whether it is possible to use a version of the scheme more similar to the one being submitted to the NIST PQC competition.

3. Preliminaries

3.1. Notation

- Let $\mathbb{Z}$ be a ring of all integers. $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ denotes a ring of residue classes modulo q. $\mathbb{Z}[x]$ denotes a ring of polynomials in the variable x with integer coefficients.
- R denotes a quotient ring $\mathbb{Z}[x]/(x^n + 1)$, where $n \in \mathbb{N}$ and R_q denotes a quotient ring $\mathbb{Z}_q[x]/(x^n + 1)$, where $n \in \mathbb{N}$.
- Polynomials are denoted in italic lowercase p. $p \in R_q$ is a polynomial of degree bound by n: $p = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1}$. It can also be expressed in a vector notation through its coefficients $(p_0, p_1, \ldots, p_{n-1})$.
- Vectors are denoted in bold lowercase $\mathbf{v}$. $\mathbf{v} \in R_q^n$ is a vector of dimension n: $\mathbf{v} = (v_0, \ldots, v_{n-1})$, where each element v_i is a polynomial in R_q.

- Matrices are denoted in bold uppercase $\mathbf{A}$. $\mathbf{A} \in R_q^{n \times m}$ is a $n \times m$ matrix with elements in R_q.
- For an even positive integer α and for every $x \in \mathbb{Z}$, define $x' = x \mod^{\pm} \alpha$, as x' in the range $-\frac{\alpha}{2} < x' \leq \frac{\alpha}{2}$ such that $x' \equiv x \pmod{\alpha}$. For an odd positive integer α and for every $x \in \mathbb{Z}$, define $x' = x \mod^{\pm} \alpha$, as x' in the range $-\frac{\alpha-1}{2} \leq x' \leq \frac{\alpha-1}{2}$ such that $x' \equiv x \pmod{\alpha}$. For any positive integer α, define $x' = x \mod \alpha$, as x' in the range $0 \leq x' < \alpha$ such that $x' \equiv x \pmod{\alpha}$.
- For an element $p = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1} \in R_q$, its l_2 norm is defined as $||p||_2 = (\sum_i |p_i|^2)^{\frac{1}{2}}$.
- For an element $x \in \mathbb{Z}_q$, its infinity norm is defined as $||x||_\infty = |x \mod^{\pm} q|$, where $|x|$ denotes the absolute value of the element. For an element $p = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1} \in R_q$, $||p||_\infty = \max_i ||p_i||_\infty$. Similarly for an element $\mathbf{v} = (p_0, \ldots, p_n) \in R_q^n$, $||\mathbf{v}||_\infty = \max_i ||p_i||_\infty$.
- S_η denotes a set of all elements $p \in R$ such that $||p||_\infty \leq \eta$.
- $a \leftarrow A$ denotes sampling an element uniformly at random from the set A.
- $a \leftarrow \chi(A)$ denotes sampling an element from the distribution χ defined over the set A.
- $\lceil x \rceil$ denotes mapping x to the least integer greater than or equal to x (e.g., $\lceil 5.2 \rceil = 6$).
- The symbol $\perp$ is used to indicate a failure or rejection.

3.2. Definitions of Lattice Problems

Definition 1 (**Decisional Module-LWE**(q, n, m, η, χ))**.** *Let χ be an error distribution, given a pair $(\mathbf{A}, \mathbf{t}) \in (R_q^{n \times m} \times R_q^n)$ decide whether it was generated uniformly at random from $R_q^{n \times m} \times R_q^n$ or it was generated as $\mathbf{A} \leftarrow R_q^{n \times m}$, $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n)$ and $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.*

The advantage of adversary$\mathcal{A}$ in breaking decisional Module-LWE for the set of parameters (q, n, m, η, χ) can be defined as follows:

$$\mathrm{Adv}^{\text{Dec-MLWE}}_{(q,n,m,\eta,\chi)}(\mathcal{A}) := |\Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n \times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), \mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2, b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{t})] - \Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n \times m}, \mathbf{t} \leftarrow R_q^n, b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{t})]|.$$

Definition 2 (**Computational Module-LWE**(q, n, m, η, χ))**.** *Let χ be an error distribution, given a pair $(\mathbf{A}, \mathbf{t}) \in (R_q^{n \times m} \times R_q^n)$, where $\mathbf{A} \leftarrow R_q^{n \times m}$, $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n)$, and $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$ when finding a vector $\mathbf{s}_1$.*

The advantage of adversary $\mathcal{A}$ in breaking computational Module-LWE for the set of parameters (q, n, m, η, χ) can be defined as follows:

$$\mathrm{Adv}^{\text{Com-MLWE}}_{(q,n,m,\eta,\chi)}(\mathcal{A}) := \Pr[\mathbf{s}_1 = \mathbf{s}_1' : \mathbf{A} \leftarrow R_q^{n \times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), \mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2, \mathbf{s}_1' \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{t})].$$

Definition 3 (**Module-SIS**(q, n, m, η))**.** *Given a uniformly random matrix $\mathbf{A} \leftarrow R_q^{n \times m}$, find a vector $\mathbf{x} \leftarrow R_q^{n+m}$ such that $\left[\mathbf{A}|\mathbf{I}\right] \cdot \mathbf{x} = \mathbf{0}$ and $0 < ||\mathbf{x}||_\infty \leq \eta$.*

The advantage of adversary $\mathcal{A}$ in breaking Module-SIS for the set of parameters (q, n, m, η) can be defined as follows:

$$\mathrm{Adv}^{\text{MSIS}}_{(q,n,m,\eta)}(\mathcal{A}) := \Pr[\left[\mathbf{A}|\mathbf{I}\right] \cdot \mathbf{x} = \mathbf{0} \text{ and } 0 < ||\mathbf{x}||_\infty \leq \eta : \mathbf{A} \leftarrow R_q^{n \times m}, \mathbf{x} \leftarrow \mathcal{A}(\mathbf{A})].$$

Additionally, we define the rejected Module-LWE assumption adapted from [26].

Definition 4 (**Rejected Module-LWE** $(q, n, m, \gamma, \chi, \beta)$)**.** *Let χ be an error distribution, and let C be a set of all challenges. Let $\mathbf{A} \leftarrow R_q^{n \times m}$, $\mathbf{s}_1, \mathbf{s}_2 \leftarrow \chi(S_\eta^m \times S_\eta^n)$, $\mathbf{y}_1, \mathbf{y}_2 \leftarrow \chi(S_{\gamma-1}^m \times S_{\gamma-1}^n)$, and $c \leftarrow C$. Assume that $\mathbf{y}_1 + c\mathbf{s}_1 \geq \gamma - \beta$ or $\mathbf{y}_2 + c\mathbf{s}_2 \geq \gamma - \beta$ hold. Given $(\mathbf{A}, \mathbf{w}, c)$, decide whether $\mathbf{w}$ was generated uniformly at random from R_q^n or it was generated as $\mathbf{w} = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$.*

The advantage of adversary$\mathcal{A}$ in breaking the rejected Module-LWE for the set of parameters $(q, n, m, \gamma, \chi, \beta)$ can be defined as follows:

$$\mathrm{Adv}^{\text{R-MLWE}}_{(q,n,m,\gamma,\chi,\beta)}(\mathcal{A}) := |\mathrm{Game}^{\text{R-MLWE}}_0 - \mathrm{Game}^{\text{R-MLWE}}_1|.$$

$\mathrm{Game}^{\text{R-MLWE}}_0 := \Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n\times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), (\mathbf{y}_1, \mathbf{y}_2) \leftarrow \chi(S_{\gamma-1}^m \times S_{\gamma-1}^n), c \leftarrow C, \mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2, b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{w}, c) \mid \mathbf{y}_1 + c\mathbf{s}_1 \geq \gamma - \beta \text{ or } \mathbf{y}_2 + c\mathbf{s}_2 \geq \gamma - \beta]$

$\mathrm{Game}^{\text{R-MLWE}}_1 := \Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n\times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), (\mathbf{y}_1, \mathbf{y}_2) \leftarrow \chi(S_{\gamma-1}^m \times S_{\gamma-1}^n), c \leftarrow C, \mathbf{w} \leftarrow R_q^n, b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{w}, c) \mid \mathbf{y}_1 + c\mathbf{s}_1 \geq \gamma - \beta \text{ or } \mathbf{y}_2 + c\mathbf{s}_2 \geq \gamma - \beta]$

3.3. Forking Lemma

The following forking lemma is adapted from [31]. x can be viewed as a public key of the signature scheme, and $h_1, \ldots, h_q$ can be viewed as replies to the random oracle queries.

Lemma 1 (General forking lemma). *Fix an integer $q \geq 1$ to be the number of queries. Fix set C of size $|C| \geq 2$. Let $\mathcal{B}$ be a randomised algorithm that takes as input $x, h_1, \ldots, h_q$, where $(h_1, \ldots, h_q) \in C$, and returns a pair with the first element being index i (integer in the range $\{0, \ldots, q\}$) and the second element being side output out. Let $\mathcal{IG}$ be a randomised input generation algorithm. Let the accepted probability of $\mathcal{B}$ be denoted as acc. This is the probability that $i \neq 0$ in the following experiment:*

- $x \leftarrow \mathcal{IG}$
- $h_1, \ldots, h_q \leftarrow C$
- $(i, out) \leftarrow \mathcal{B}(x, h_1, \ldots, h_q)$

The forking algorithm $\mathcal{F}_B$ connected with $\mathcal{B}$ is defined in Algorithm 1.

Algorithm 1 $\mathcal{F}_B(x)$

1: Pick random coins ρ for $\mathcal{B}$
2: $h_1, \ldots, h_q \leftarrow C$
3: $(i, out) \leftarrow \mathcal{B}(x, h_1, \ldots, h_q; \rho)$
4: If $i = 0$, then return $(0, \perp, \perp)$
5: Regenerate $h'_i, \ldots, h'_q \leftarrow C$
6: $(i', out') \leftarrow \mathcal{B}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_q; \rho)$
7: If $i = i'$ and $h_i \neq h'_i$, then return $(1, out, out')$
8: Otherwise, return $(0, \perp, \perp)$

Let us define the frk probability as

$$frk = \Pr[b = 1 : x \leftarrow \mathcal{IG}; (b, out, out') \leftarrow \mathcal{F}_B(x)].$$

Then,

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

Alternatively,

$$acc \leq \frac{q}{|C|} + \sqrt{q \cdot frk}.$$

3.4. Lattice-Based Signature Scheme

Lattice-based cryptography is a promising candidate for the post-quantum public key cryptography standards. Among all of the submissions to the NIST PQC competition, the majority of schemes belong to the lattice-based family [17]. Many lattice-based signatures are constructed from the identification schemes using the Fiat–Shamir (FS) transform. The

FS transform technique introduced in [32] allows for creating a digital signature scheme by combining an identification scheme with a hash function.

The following definition is adapted from [13].

Definition 5 (Identification scheme). *An identification scheme ID is defined as a tuple of algorithms* $ID := (IGen, P, C, V)$.

- *The key generation algorithm IGen takes as input system parameters par and returns the public key and secret key as output* (pk, sk). *Public key pk defines the set of challenges C, the set of commitments W, and the set of responses Z.*
- *The prover algorithm* $P = (P_1, P_2)$ *consists of two sub-algorithms.* P_1 *takes as input the secret key and returns a commitment* $w \in W$ *and a state st.* P_2 *takes as input the secret key, a commitment, a challenge, and a state and returns a response* $z \in Z \cup \{\perp\}$.
- *The verifier algorithm V takes as input the public key and the conversation transcript and outputs a decision bit* $b = 1$ *(accepted) or* $b = 0$ *(rejected).*

In the signature scheme that uses FS transform, the signing algorithm generates a transcript (w, c, z), where a challenge c is derived from a commitment w and the message to be signed m as follows $c := H(w||m)$. The signature $\sigma = (w, z)$ is valid if the transcript (w, c, z) passes the verification algorithm with $b = 1$. The publication [33] introduced a generalisation to this technique called Fiat–Shamir, with aborts transformation that takes into consideration aborting provers.

The following signature scheme (further referred to as the basic scheme) is a slightly modified version of the scheme [34]; the description below is based on a version described in [13] (Appendix B). The signature scheme makes use of a hash function, which produces a vector of size n with elements in $\{-1, 0, 1\}$ [18]. The hashing algorithm starts with applying a collision-resistant hash function (e.g., SHAKE-256) to the input to obtain a vector $\mathbf{s} \in \{0,1\}^\tau$ from the first τ bits of the hash function's output. Then, SampleInBall algorithm (Algorithm 2) is invoked to create a vector $\mathbf{c}$ in $\{-1, 0, 1\}^n$ with exactly τ nonzero elements. In each iteration of the for loop, the SampleInBall algorithm generates an element $j \in \{0, \ldots, i\}$ using the output of a collision-resistant hash function. Then, the algorithm performs shuffling of the elements in the vector $\mathbf{c}$ and takes an element from the vector $\mathbf{s}$ to generate -1 or 1. For an in-depth overview of the algorithm, refer to the original paper [18].

Algorithm 2 SampleInBall.

1: Initialise $\mathbf{c}$ as zero vector of length n
2: for $i := n - \tau$ to $n - 1$
   1. $j \leftarrow \{0, 1, \ldots, i\}$
   2. $s \leftarrow \{0, 1\}$
   3. $c_i := c_j$
   4. $c_j := (-1)^s$
3: return $\mathbf{c}$

All of the algebraic operations in the signature scheme are performed over the ring R_q. A formal definition of the key generation, signing, and verification is presented in Algorithms 3–5.

Algorithm 3 KeyGen(par).

1: $\mathbf{A} \leftarrow R_q^{k \times k}$
2: $\mathbf{s}_1, \mathbf{s}_2 \leftarrow S_\eta^k$
3: $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$
4: **return** $pk = (\mathbf{A}, \mathbf{t}), sk = (\mathbf{A}, \mathbf{t}, \mathbf{s}_1, \mathbf{s}_2)$

Algorithm 4 Sign(sk, m).

1: $(\mathbf{z}_1, \mathbf{z}_2) = (\perp, \perp)$
2: **while** $(\mathbf{z}_1, \mathbf{z}_2) = (\perp, \perp)$ do:
   1. $\mathbf{y}_1, \mathbf{y}_2 \leftarrow S^k_{\gamma_1 - 1}$
   2. $\mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$
   3. $c := H_0(m||\mathbf{w}) \in B_\tau$
   4. $\mathbf{z}_1 := \mathbf{y}_1 + c\mathbf{s}_1$ and $\mathbf{z}_2 := \mathbf{y}_2 + c\mathbf{s}_2$
   5. **if** $||\mathbf{z}_1||_\infty \geq \gamma_1 - \beta$ or $||\mathbf{z}_2||_\infty \geq \gamma_1 - \beta$, then $(\mathbf{z}_1, \mathbf{z}_2) := (\perp, \perp)$
3: **return** $\sigma = (\mathbf{z}_1, \mathbf{z}_2, c)$

Algorithm 5 Verify(pk, m, σ).

1: $\mathbf{w}' := \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}$
2: **if** $c = H_0(m||\mathbf{w}')$ and $||\mathbf{z}_1||_\infty < \gamma_1 - \beta$ and $||\mathbf{z}_2||_\infty < \gamma_1 - \beta$, **return** 1 (success).
3: **else**: **return** 0.

3.4.1. Correctness

Since $\mathbf{w} = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$, $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, $\mathbf{z}_1 = \mathbf{y}_1 + c\mathbf{s}_1$. and $\mathbf{z}_2 = \mathbf{y}_2 + c\mathbf{s}_2$, it holds that

$$\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}(\mathbf{y}_1 + c\mathbf{s}_1) + (\mathbf{y}_2 + c\mathbf{s}_2) - c(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2) = \mathbf{A}\mathbf{y}_1 + \mathbf{A}c\mathbf{s}_1 + \mathbf{y}_2 + c\mathbf{s}_2 - c\mathbf{A}\mathbf{s}_1 - c\mathbf{s}_2 = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2.$$

Therefore, if a signature was generated correctly, it will successfully pass the verification.

3.5. Homomorphic Hash Function

We decided to use a homomorphic hash function instead of a homomorphic commitment scheme as in [16].

Definition 6 (**Homomorphic hash function**). *Let $+$ be an operation defined over X, and let $\oplus$ be an operation defined over R. Let $x_1, x_2 \in X$ be any two inputs to the hash function. A hash function $f : X \to R$ is homomorphic if it holds that*

$$f(x_1 + x_2) = f(x_1) \oplus f(x_2).$$

Definition 7 (**Regular hash function**). *Let $\mathcal{F} = \{f_a\}_{a \in A}$, where $f_a : X \to R$ be a collection of functions indexed by a set A. A family of hash functions $\mathcal{F}$ is called ϵ-regular if the statistical distance between its output distribution $\{(a, f_a(x)) : a \leftarrow A, x \leftarrow X\}$ and the uniform distribution $\{(a, r) : a \leftarrow A, r \leftarrow R\}$ is at most ϵ.*

One of the homomorphic hash functions available is called SWIFFT; it is a special case of the function proposed in [35–37]. SWIFFT is a collection of compression functions that are provably one-way and collision-resistant [38]. Additionally, SWIFFT has several statistical properties that can be proven unconditionally: universal hashing, regularity, and randomness extraction. However, due to the linearity, SWIFFT functions are not pseudorandom. It follows that the function is not a suitable instantiation of a random oracle [38]. Therefore, in the security proofs of the two-party signature scheme, SWIFFT is not used as a random oracle. Security proof makes use of such provable properties as regularity and collision resistance.

4. Proposed Two-Party Signature Scheme (DiLizium)

In the following section, we define and give detailed description of our two-party signature scheme: DiLizium. We start by defining the distributed signature scheme; the following definition is adapted from [16].

Definition 8 (**Distributed signature protocol**). *Distributed signature protocol is a protocol between $P_1, \ldots, P_n$ parties that consists of the following algorithms:*

- *Generate public parameters par using security parameter λ as input: par $\leftarrow$ Setup(1^λ).*
- *Each party P_j generates a key pair consisting of secret key share and a public key using interactive algorithm and public parameters as input: $(sk_j, pk) \leftarrow KeyGen_j(par)$ for each $j \in \{1, \ldots, n\}$.*
- *To sign a message m, each party P_j runs an interactive signing algorithm using secret key share: $(\sigma) \leftarrow Sign_j(sk_j, m)$ for each $j \in \{1, \ldots, n\}$.*
- *To verify a signature, the verifier needs to check if Verify(pk, m, σ) = 1. If the signature was generated correctly, verification should always succeed.*

4.1. Specification and overview of DiLizium signature scheme

Table 1 describes the parameters used in the two-party signature scheme.

Table 1. Parameters for the two-party protocol.

Parameter	Description
n	degree bound of the polynomials in the ring
q	modulus
(k, k)	dimension of matrix and vectors used in the scheme
γ	size bound of the coefficients in the masking vector share
γ_2	size bound of coefficients in the composed masking vector
η	size bound of coefficients in the secret key share
τ	number on nonzero elements in the output of special hash function H_0
β	maximum possible coefficient of the client's and server's shares of $c\mathbf{s}_i$, where $i \in \{1, 2\}$
β_2	maximum possible coefficient of $c\mathbf{s}_i$, where $i \in \{1, 2\}$
(a, b, p)	parameters for the homomorphic hash function: $a \cdot b$ is the input length, b is the output length, and p is the modulus

4.1.1. Parameter setup

Let us assume that, before starting the key generation and signing protocols, the parties invoke a Setup(1^λ) function that, based on the security parameter λ, outputs a set of public parameters *par* that are described in Table 1.

4.1.2. Key generation

H_1 and H_2 are some collision-resistant hash functions. The key generation protocol is parametrised by the set of public parameters *par*. The client begins the key generation process by sampling a share of matrix $\mathbf{A}_c$ and by sending out the commitment to this share $hk_c = H_1(\mathbf{A}_c)$. The server generates its matrix share $\mathbf{A}_s$ and sends commitment hk_s to the client. Upon receiving commitments, the client and server exchange matrix shares and check if the openings for the commitments were correct. If openings are successfully verified, the client and server locally compute composed matrix $\mathbf{A} = \mathbf{A}_c + \mathbf{A}_s$.

The client proceeds by generating two secret vectors $(\mathbf{s}_1^c, \mathbf{s}_2^c)$ and by computing its share of the public key $\mathbf{t}_c = \mathbf{A}\mathbf{s}_1^c + \mathbf{s}_2^c$. The client sends out a commitment to the public key share $comk_c = H_2(\mathbf{t}_c)$. The server samples its secret vectors $(\mathbf{s}_1^s, \mathbf{s}_2^s)$ and uses them to compute its public key share $\mathbf{t}_s$. Next, the server sends commitment to the public key share $comk_s$ to the client.

Once the client and server have received commitments from each other, the client and server exchange public key shares. Next, the client and server both locally check if the commitments were opened correctly. If these checks succeed, the client and server locally compute the composed public key $\mathbf{t} = \mathbf{t}_c + \mathbf{t}_s$. The final public key consists of composed matrix $\mathbf{A}$ and vector $\mathbf{t}$.

It is necessary to include the server's public key share $\mathbf{t}_s$ to the client's secret key sk_c and vice versa. During the signing process, the client needs to use the server's public key share to verify the correctness of a commitment.

Protocol 1 describes two-party key generation between the parties in a more formal way. Instructions of the protocol are the same for the client and server. Therefore, Protocol 1 presents the behavior of the nth party, $n \in \{c, s\}$.

4.1.3. Signing

$HomH$ is a homomorphic hash function from the SWIFFT family. H_0 is a hash function that outputs a vector of length n with exactly τ coefficients being either -1 or 1 and the rest being 0 as described in Algorithm 2. H_3 is a collision-resistant hash function.

The client starts the signing process by generating its shares of masking vectors $(\mathbf{y}_1^c, \mathbf{y}_2^c)$ and by computing a share of $\mathbf{w}$. Next, the client uses a homomorphic hash function to compute $com_c = HomH(\mathbf{w}_c)$ and hashes it using some collision-resistant hash function $h_c = H_3(com_c)$. The composed output of the homomorphic hash function $com = com_c + com_s$ is later used to derive a challenge. Therefore, it is crucial to ensure that com_c, com_s have not been chosen maliciously. Thus, before publishing these shares, the client and server should exchange commitments to the shares h_c, h_s.

The server, in turn, generates its shares of masking vectors $(\mathbf{y}_1^s, \mathbf{y}_2^s)$, computes its share of $\mathbf{w}$, and sends commitment to $com_s = HomH(\mathbf{w}_s)$. After receiving commitments h_c, h_s from each other, the client and server open the commitments by sending out shares com_s, com_c.

The client proceeds by checking if the server opened its commitment correctly. If the check succeeds, the client computes $com = com_c + com_s$ and derives challenge $c = H_0(m||com)$. Next, the client computes potential signature shares $(\mathbf{z}_1^c, \mathbf{z}_2^c)$ and performs rejection sampling. If all of the conditions in rejection sampling are satisfied, the client sends its signature share to the server.

The server checks if the client opened its commitment correctly. If the check succeeds, the server computes composed com and derives challenge c. Next, the server computes its potential signature shares $(\mathbf{z}_1^s, \mathbf{z}_2^s)$ and performs rejection sampling. If all of the conditions in rejection sampling are satisfied, the server sends its signature share to the client.

Finally, the client performs verification if com_s indeed contains $\mathbf{w}_s$. The client reconstructs $\mathbf{w}_s$ as $\mathbf{A}\mathbf{z}_1^s + \mathbf{z}_2^s - c\mathbf{t}_s$ and checks if it is a valid opening for com_s. If the check succeeds, the client computes the final signature $(\mathbf{z}_1, \mathbf{z}_2)$. The server performs the same verification that com_c indeed contains $\mathbf{w}_c$ using $(\mathbf{z}_1^c, \mathbf{z}_2^c)$ and $\mathbf{t}_c$. If the check succeeds, the server computes and outputs the final signature.

Protocol 2 describes the two-party signing process in the more formal way.

4.1.4. Verification

Verification is almost the same as in the original scheme except the verifier needs to apply homomorphic hash function on the reconstructed $\mathbf{w}'$ in order to check the correctness of challenge. Algorithm 6 describes verification in the more formal way.

4.1.5. Correctness

Since $\mathbf{w} = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$, $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, $\mathbf{z}_1 = \mathbf{y}_1 + c\mathbf{s}_1$ and $\mathbf{z}_2 = \mathbf{y}_2 + c\mathbf{s}_2$ it holds that:

$\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}(\mathbf{y}_1 + c\mathbf{s}_1) + (\mathbf{y}_2 + c\mathbf{s}_2) - c(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2) = \mathbf{A}\mathbf{y}_1 + \mathbf{A}c\mathbf{s}_1 + \mathbf{y}_2 + c\mathbf{s}_2 - c\mathbf{A}\mathbf{s}_1 - c\mathbf{s}_2 = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$.

Furthermore, by triangle inequality, it holds that if $||\mathbf{z}_1^s||_\infty < \gamma - \beta$ and $||\mathbf{z}_1^c||_\infty < \gamma - \beta$, then $||\mathbf{z}_1||_\infty = ||\mathbf{z}_1^s + \mathbf{z}_1^c||_\infty < ||\mathbf{z}_1^s||_\infty + ||\mathbf{z}_1^c||_\infty = 2\gamma - 2\beta$. The same holds for the second signature component $\mathbf{z}_2$. This means that γ_2 can be defined as $\gamma_2 = 2\gamma$ and $\beta_2 = 2\beta$. Therefore, if a signature was generated correctly, verification always succeed.

Protocol 1: KeyGen$_n(par)$;

1. **First message**:
   (a) $\mathbf{A}_n \leftarrow R_q^{k\times k}$, send out $hk_n := H_1(\mathbf{A}_n)$.
2. **Second message**:
   (a) Upon receiving hk_i, send out $\mathbf{A}_n$.
3. **Third message**
   (a) Upon receiving $\mathbf{A}_i$, verify if $H_1(\mathbf{A}_i) = hk_i$. Send out ABORT message if equality does not hold.
   (b) $\mathbf{A} := \mathbf{A}_n + \mathbf{A}_i$.
   (c) $\mathbf{s}_1^n, \mathbf{s}_2^n \leftarrow S_\eta^k$.
   (d) $\mathbf{t}_n := \mathbf{A}\mathbf{s}_1^n + \mathbf{s}_2^n$, send out $comk_n := H_2(\mathbf{t}_n)$.
4. **Fourth message**
   (a) Upon receiving $comk_i$, send out $\mathbf{t}_n$.
5. **Verification**
   (a) Upon receiving $\mathbf{t}_i$, verify if $H_2(\mathbf{t}_i) = comk_i$. Send out ABORT message if equality does not hold.
   (b) $\mathbf{t} := \mathbf{t}_n + \mathbf{t}_i$.
6. **Output**
   (a) Client's share of secret key $sk_c = (\mathbf{A}, \mathbf{t}_s, \mathbf{s}_1^c, \mathbf{s}_2^c)$.
   (b) Server's share of secret key $sk_s = (\mathbf{A}, \mathbf{t}_c, \mathbf{s}_1^s, \mathbf{s}_2^s)$.
   (c) Final public key $pk = (\mathbf{A}, \mathbf{t})$.

Protocol 2: Sign$_n(sk_n, m)$;

1. **First message**
   (a) $\mathbf{y}_1^n, \mathbf{y}_2^n \leftarrow S_{\gamma-1}^k$.
   (b) $\mathbf{w}_n := \mathbf{A}\mathbf{y}_1^n + \mathbf{y}_2^n$.
   (c) $com_n := HomH(\mathbf{w}_n)$, send out $h_n := H_3(com_n)$.
2. **Second message**
   (a) Upon receiving h_i, send out com_n.
3. **Third message**
   (a) Upon receiving com_i, verify if $H_3(com_i) = h_i$. Send out ABORT message if equality does not hold.
   (b) $com := com_n + com_i$, $c \in B_\tau := H_0(m||com)$.
   (c) $\mathbf{z}_1^n := \mathbf{y}_1^n + c\mathbf{s}_1^n$, $\mathbf{z}_2^n := \mathbf{y}_2^n + c\mathbf{s}_2^n$.
   (d) Perform rejection sampling:
   if $||\mathbf{z}_1^n||_\infty \geq \gamma - \beta$ or $||\mathbf{z}_2^n||_\infty \geq \gamma - \beta$, then send out RESTART message.
   else:
   (e) Send out $(\mathbf{z}_1^n, \mathbf{z}_2^n)$.
4. **Verification**
   (a) Upon receiving $(\mathbf{z}_1^i, \mathbf{z}_2^i)$, reconstruct $\mathbf{w}_i := \mathbf{A}\mathbf{z}_1^i + \mathbf{z}_2^i - c\mathbf{t}_i$.
   (b) Check if $HomH(\mathbf{w}_i) = com_i$ and send out ABORT message if check fails.
   (c) Compute final signature on message m as $\mathbf{z}_1 := \mathbf{z}_1^n + \mathbf{z}_1^i$ and $\mathbf{z}_2 := \mathbf{z}_2^n + \mathbf{z}_2^i$, $\sigma = (\mathbf{z}_1, \mathbf{z}_2, c)$.
5. **Upon receiving RESTART message**
   (a) Client and server start signing process again from the beginning.

Algorithm 6 Verify(pk, σ, m)

1: Compute $\mathbf{w}' := \mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}$.
2: **if** $c = \mathrm{H}(m||HomH(\mathbf{w}'))$ and $||\mathbf{z}_1||_\infty < \gamma_2 - \beta_2$ and $||\mathbf{z}_2||_\infty < \gamma_2 - \beta_2$: **return** 1 (success).
3: **else**: **return** 0.

Security

Definition 9 (**Existential Unforgeability under Chosen Message Attack).** *The distributed signature protocol is Existentially Unforgeable under Chosen Message Attack (DS-UF-CMA) if, for any probabilistic polynomial time adversary $\mathcal{A}$, its advantage of creating successful signature forgery is negligible. The advantage of adversary is defined as the probability of winning in the experiment $Exp^{DS\text{-}UF\text{-}CMA}$:*

$$Adv^{DS\text{-}UF\text{-}CMA}(\mathcal{A}) := Pr[Exp^{DS\text{-}UF\text{-}CMA}(\mathcal{A}) \to 1].$$

Experiment 1: $\mathbf{Exp^{DS\text{-}UF\text{-}CMA}}(\mathcal{A})$;

1. $\mathcal{M} \leftarrow \emptyset$
2. $kgen \leftarrow false$
3. $par \leftarrow Setup(1^\lambda)$
4. $(m^*, \sigma^*) \leftarrow \mathcal{A}^{KeyGen\mathcal{O}(.,.),\, Sign\mathcal{O}(.,.)}(par)$
5. $b \leftarrow Verify(m^*, \sigma^*, pk)$
6. *if* $b = 1$ *and* $m^* \notin \mathcal{M}$*: return* 1
7. *otherwise, return* 0

Oracle 1: KeyGen$\mathcal{O}$(par, sid, msg);

The oracle is initialised with the set of public parameters *par* generated by the Setup(1^λ) algorithm.

1. Upon receiving $(0, msg)$ if the flag $kgen = true$, then return $\perp$.
2. Upon receiving query with $sid = 0$ for the first time,
   (a) Initialise a machine $\mathcal{M}_0$. $\mathcal{M}_0$ uses the instructions of the party P_n in the key generation protocol KeyGen(*par*).
   (b) If P_n sends the first message according to the key generation protocol, then oracle returns this message.
3. If machine $\mathcal{M}_0$ has been already initialised,
   (a) Oracle gives the next incoming message *msg* to the $\mathcal{M}_0$.
   (b) Oracle returns reply that was received from $\mathcal{M}_0$.
   (c) If $\mathcal{M}_0$ finished the protocol with a local output (sk_n, pk), then oracle sets the flag $kgen = true$.

Oracle 2: Sign$\mathcal{O}$(sid, msg);

1. Upon receiving (sid, msg), if the flag $kgen = false$ and $sid \neq 0$, then return $\perp$.
2. Upon receiving query with sid for the first time,
   (a) Parse incoming message msg as message to be signed m.
   (b) Initialise a machine $\mathcal{M}_{sid}$. $\mathcal{M}_{sid}$ uses the instructions of the party P_n in the signing protocol Sign(par, sk_n, pk, m).
   (c) The message m is included in the set of all queried messages $\mathcal{M}$.
   (d) If P_n sends the first message according to the signing protocol, then oracle returns this message.
3. If machine $\mathcal{M}_{sid}$ has been already initialised,
   (a) Oracle gives the next incoming message msg to the $\mathcal{M}_{sid}$.
   (b) Oracle returns the reply that was received from $\mathcal{M}_{sid}$.
   (c) If $\mathcal{M}_{sid}$ finished the protocol with a local output σ, then the oracle returns this output.

Theorem 1. *Assume a homomorphic hash function $HomH : \{0,1\}^{a \cdot b} \rightarrow \mathbb{Z}_p^b$ is provably collision-resistant and ϵ-regular then for any probabilistic polynomial time adversary $\mathcal{A}$ that makes a single query to the key generation oracle; q_s queries to the signing oracle; and q_h queries to the random oracles H_0, H_1, H_2, and H_3, the distributed signature protocol is DS-UF-CMA secure in the random oracle model under Module-LWE, rejected Module-LWE, and Module-SIS assumptions.*

This section presents the main idea for the security proof of the proposed scheme; the full proof is given in Appendix A. The proof considers only the classical adversary and relies on the forking lemma. The idea of the proof is, given an adversary $\mathcal{A}$ that succeeds in creating forgeries for the distributed signature protocol, to construct an algorithm around it that can be used to solve Module-SIS problem or to break the collision resistance of the homomorphic hash function. The idea of our proof relies on the proofs from [7,16,31,39].

The proof consists of two major steps. In the first step, we construct a simulator $\mathcal{B}$. Algorithm $\mathcal{B}$ is constructed such that it fits all of the assumptions of the forking lemma. $\mathcal{B}$ simulates the behavior of a single honest party P_n without using its actual secret key share. In the second step, the forking algorithm is invoked to obtain two forgeries with distinct challenges and the same commitments.

4.1.6. Simulation

In the key-generation process, we need to simulate the way the matrix share $\mathbf{A}_n$ and the public vector share $\mathbf{t}_n$ are constructed. Due to the use of random oracle commitments, once the simulator obtains the adversary's commitment hk_i, it can extract the matrix share $\mathbf{A}_i$. Next, the simulator computes its matrix share $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$ using a resulting random matrix $\mathbf{A} \in R_q^{k \times k}$ and programs random oracle $H_1(\mathbf{A}_n) := hk_n$.

Due to the Module-LWE assumption, the public vector share of the honest party $\mathbf{t}_n$ is indistinguishable from the uniformly random vector sampled from the ring R_q^k. Using the same strategy as that for the matrix share, the simulator sets it public vector share $\mathbf{t}_n := \mathbf{t} - \mathbf{t}_i$ after seeing the adversary's commitment and programs the random oracle $H_2(\mathbf{t}_n) := comk_n$.

The signature share generation starts with choosing a random challenge $c \in C$ from the set of all possible challenges. Then, the simulator proceeds with randomly sampling two signature shares from the set of all possible signature shares $\mathbf{z}_1^n, \mathbf{z}_2^n \leftarrow S_{\gamma-\beta-1}^k$. The share of vector $\mathbf{w}$ is computed from the signature shares, public vector share, and challenge as $\mathbf{w}_n = \mathbf{A}\mathbf{z}_1^n + \mathbf{z}_2^n - c\mathbf{t}_n$. Then, the simulator extracts value com_i from the adversary's commitment h_i, computes the composed value com, and programs random oracle $H_0(com||m) := c$.

4.1.7. Forking lemma

The combined public key consists of matrix $\mathbf{A}$ uniformly distributed in $R_q^{k\times k}$ and vector $\mathbf{t}$ uniformly distributed in R_q^k. We want to replace it with the Module-SIS instance $[\mathbf{A}'|\mathbf{I}]$, where $\mathbf{A}' \in R_q^{k\times(k+1)}$. The view of adversary does not change if we set $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$.

In order to conclude the proof, we need to invoke the forking lemma to receive two valid forgeries from the adversary that are constructed using the same commitment $com = com'$ but different challenges $c \neq c'$. Using these forgeries, it is possible to find a solution to the Module-SIS problem on input $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$ or to break the collision resistance of the homomorphic hash function.

As both forgeries $out = (com, c, \mathbf{z}_1, \mathbf{z}_2, m)$ and $out' = (com', c', \mathbf{z}_1', \mathbf{z}_2', m')$ are valid, it holds that

$$HomH(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}) = com = com' = HomH(\mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - c'\mathbf{t})$$

If $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} \neq \mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - c'\mathbf{t}$, then we found a collision for the homomorphic hash function. If $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - c'\mathbf{t}$, then it can be rearranged as $\mathbf{A}\mathbf{z}_1 - \mathbf{A}\mathbf{z}_1' + \mathbf{z}_2 - \mathbf{z}_2 - c\mathbf{t} + c'\mathbf{t} = \mathbf{0}$ and this in turn leads to

$$[\mathbf{A}|\mathbf{I}|\mathbf{t}] \begin{bmatrix} \mathbf{z}_1 - \mathbf{z}_1' \\ \mathbf{z}_2 - \mathbf{z}_2' \\ c' - c \end{bmatrix} = \mathbf{0}$$

Considering that $[\mathbf{A}|\mathbf{I}|\mathbf{t}]$ is an instance of Module-SIS problem, we found a solution for Module-SIS with parameters $(q, k, k+1, \xi)$, where $\xi \leq 2(\gamma_2 - \beta_2)$.

5. Performance

In this section, we analyse the performance of our scheme according to the following metrics:

- Number of communication rounds in key generation and signing protocols,
- Keys and signature sizes, and
- Number of rejection sampling rounds.

It should be noted that this section does not present the exact parameter choice for the scheme and does not argue the bit security of the scheme for these parameters. The parameter choice presented in this section is illustrative and is given to provide performance estimations of the proposed scheme. Choosing correct parameters for post-quantum schemes is a nontrivial multidimensional optimisation task as parameters should be chosen such that the scheme has the small signature and key sizes while having enough bits of security and an optimal number of communication rounds. Additionally, the security of the proposed scheme relies on rejected Module-LWE, which is not a well-studied assumption, and therefore, it is difficult to estimate the bit security of the proposed scheme. The parameters presented in Table 2 are chosen based on parameters proposed in Crystals-Dilithium [18] so that the expected number of repetitions of the signing process is practical.

5.1. Number of rejection sampling rounds

To estimate the number of rejection sampling rounds in the signing process, it is necessary to compute the probability that the following holds for both parties: $||\mathbf{z}_1^n||_\infty < \gamma - \beta$ and $||\mathbf{z}_2^n||_\infty < \gamma - \beta$. Let σ be a coefficient of $c\mathbf{s}_i^n$. If coefficients of $\mathbf{y}_i^n$ are in the range $\{-\gamma + \beta + 1 - \sigma, \ldots, \gamma - \beta - 1 - \sigma\}$, then the corresponding coefficients of $\mathbf{z}_i^n$ are in the range $\{-\gamma + \beta + 1, \ldots, \gamma - \beta - 1\}$. Therefore, the size of the correct coefficient range for $\mathbf{y}_i^n$ is $2(\gamma - \beta) - 1$ and the coefficients of $\mathbf{y}_i^n$ have $2\gamma - 1$ possibilities. Then, the probability that every coefficient of $\mathbf{y}_i^n$ in the correct range is as follows:

$$\left(\frac{2(\gamma - \beta) - 1}{2\gamma - 1}\right)^{n\cdot k}$$

As the client and server sample vectors $\mathbf{y}_i^n$ independently in the beginning of the signing protocol, the probability that the check succeeds for both signature components on the client and server side is the following:

$$\Pr[\text{success}] = \left(\frac{2(\gamma-\beta)-1}{2\gamma-1}\right)^{n\cdot k\cdot 4}$$

The expected number of repetitions can be estimated as $E = 1/\Pr[\text{success}]$.

5.2. Signature and key sizes

The **public key** consists of two components: matrix $\mathbf{A} \in R_q^{k\times k}$ and vector $\mathbf{t} \in R_q^k$. The matrix $\mathbf{A}$ can be generated out of 256-bit seed using an extendable output function, as proposed in the Crystals-Dilithium signature scheme [18]. While using this approach, only the seed used to generate the matrix needs to be stored. As both parties need to generate their matrix share, two seeds should be stored to represent matrix $\mathbf{A}$. Each seed is converted to the matrix form using an extendable output function; after that, two matrix shares can be added together. The size of the public key in bytes is as follows:

$$\frac{2\cdot 256 + n\cdot k\cdot \lceil \log(q)\rceil}{8}.$$

The **secret key** of the party P_n consists of two vectors $\mathbf{s}_1^n$, $\mathbf{s}_2^n \in S_\eta^k$, matrix $\mathbf{A}$, and vector $\mathbf{t}_i \in R_q^k$. It should be noted that vectors $\mathbf{s}_1^n$, $\mathbf{s}_2^n$ may contain negative values as well, so one bit should be reserved for each coefficient to indicate the sign. Therefore, the size of the secret key in bytes can be computed as follows:

$$\frac{2\cdot n\cdot k\cdot (\lceil \log(\eta)\rceil + 1) + 2\cdot 256 + n\cdot k\cdot \lceil \log(q)\rceil}{8} = \frac{n\cdot k\cdot (2\cdot (\lceil \log(\eta)\rceil + 1) + \lceil \log(q)\rceil) + 2\cdot 256}{8}.$$

Finally, a **signature** consists of three components: $\mathbf{z}_1, \mathbf{z}_2 \in S^k_{\gamma_2-\beta_2-1}$, and $c \in \{0,1\}^n$ with exactly τ coefficients being either -1 or 1 and the rest being 0. All of the components may contain negative values, so for each coefficient of $\mathbf{z}_1, \mathbf{z}_2$, c one bit should be reserved to indicate the sign. With regard to storing c, it is possible to store only the positions of -1 and 1 in c. Therefore, the size of the signature in bytes can be computed as

$$\frac{2\cdot n\cdot k\cdot (\lceil \log(\gamma_2-\beta_2-1)\rceil + 1) + \tau\cdot (\lceil \log(n)\rceil + 1)}{8}.$$

In order to better understand key and signature sizes, let us assume the choice of parameters defined in Table 2. The key and signature sizes corresponding to this choice of parameters are listed in Table 3.

Table 2. Illustrative parameters.

Parameters	Sizes
n	256
q	8,380,417
(k,k)	(5,5)
γ	2^{19}
γ_2	2^{20}
η	2
τ	60
β	120
β_2	240
(a,b,p)	$(64,16,257)$

Table 3. Key and signature sizes and expected number of repetitions of the signing protocol.

Values	Sizes
Public key pk	3744 bytes
Secret key share sk_i	4384 bytes
Signature σ	6788 bytes
Expected number of repetitions	3.23

5.3. Communication between client and server

In order to generate a key pair, four rounds of communication between the client and server are needed. Table 4 shows the sizes of messages that are exchanged between the client and server during the key generation process using illustrative parameters from Table 2. The first message is output of a hash function (commitment to matrix $\mathbf{A}_i$), which consists of 256 bits. The second message contains not the matrix share itself but the seed of 256 bits that was use to generate it. The third message is output of a hash function (commitment to vector $\mathbf{t}_i$), which consists of 256 bits. The fourth message is the share of public key $\mathbf{t}_i$, the size of which is $\frac{n \cdot k \cdot \lceil \log(q) \rceil}{8}$ bytes.

The number of communication rounds during the signing process depends on the number of rejections E. If there are no rejections, the signature generation process requires three rounds of communication between the client and server. For E rejections, the number of communication rounds equals to $2E + 1$. Table 5 shows the sizes of messages exchanged between the client and server during the signing process using illustrative parameters from Table 2. The first message is output of a hash function (commitment to com_i), which consists of 256 bits. The size of the second message in the signing process com_i is caused by the structure of SWIFFT hash function. To calculate $HomH(\mathbf{w}_i)$, the vector $\mathbf{w}_i$ that consists of nk elements is divided into 15 input blocks of 256 bytes each. The output produced by the homomorphic hash function consists of 15 blocks of 128 bytes each. The third message consists of the signature shares $(\mathbf{z}_1^i, \mathbf{z}_2^i)$, the size of which is $\frac{2 \cdot n \cdot k \cdot (\lceil \log(\gamma_2 - \beta_2 - 1) \rceil + 1)}{8}$ bytes.

Table 4. Message sizes in the key generation process.

Messages	Sizes
First message hk_i	256 bits
Second message $\mathbf{A}_i$ (as seed)	256 bits
Third message $comk_i$	256 bits
Fourth message $\mathbf{t}_i$	3680 bytes

Table 5. Message sizes in the signing process.

Messages	Sizes
First message (h_i)	256 bits
Second message com_i	1920 bytes
Third message $(\mathbf{z}_1^i, \mathbf{z}_2^i)$	6080 bytes

6. Comparison to Prior Work

Table 6 presents the comparison of our scheme DiLizium with other lattice-based threshold signature schemes [16,27,40]. Column "Rounds" shows the number of communication rounds in signing protocol; for the schemes with rejection sampling, it is assumed that the rejection sample passes from the first attempt. We also provide a more detailed comparison with [16], due to the fact both works are based on variants of Crystals-Dilithium and have a similar structure. We leave out the comparison with publications [21–26] as these discuss multisignatures instead of threshold.

Table 6. Comparison with prior work.

	Functionality	Paradigm	Rounds	Security
Bendlin et al. [27]	t-out-of-n	Hash-and-Sign	1	Gentry et al. [41]
Boneh et al. [40]	t-out-of-n	Any (Universal Thresholdizer)	1	LWE
Dåmgard et al. [16] DS_2	n-out-of-n	FSwA	2	MLWE
Dåmgard et al. [16] DS_3	n-out-of-n	FSwA	3	MLWE, MSIS
Our protocol (DiLizium)	2-out-of-2	FSwA	3	MLWE, MSIS, R-MLWE

The threshold signature schemes from [16] are based on Dilithium-G, which is a version of Crystals-Dilithium that uses sampling from a discrete Gaussian distribution for the generation of secret vectors. The usage of Gaussian distribution helps to decrease the number of rejections in signature schemes that follow the FSwA paradigm [16]. However, the implementation of sampling from a discrete Gaussian distribution in a manner secure against side-channel attacks is considered difficult. Therefore, in our scheme, we decided to use sampling from a uniform distribution.

Due to the structure of our scheme, we use a non-standard security assumption that was introduced in [26]. In future work, we aim to modify security proof such that it will no longer be needed to rely on rejected Module-LWE. The security of threshold signature scheme from [16] relies only on standard problem: Module-LWE and Module-SIS.

Additionally, we compare the message sizes of Dåmgard et al. [16] DS_3 with our scheme. Since this paper [16] does not provide an instantiation of parameters, we use the recommended parameters for Dilithium-G from [29] for the signature scheme. We only changed the modulus q, since by Theorem 4 [16], q should satisfy $q = 5 \mod 8$. We selected parameters for the homomorphic commitment scheme based on the third parameter set from Baum et al. [42] (Table 2) such that the conditions from Lemma 5 and Lemma 7 are satisfied. Tables 7 and 8 present parameters that are needed to compute message sizes. We only provide a comparison of messages sent during the signing process because messages exchanged during the key generation process are similar in both schemes (Table 9).

The first message is the output of a hash function (commitment to com_n), which consists of 256 bits. The second message is a commitment com_n. The homomorphic commitment scheme defined in Dåmgard et al. [16] (Figure 7) describes a commitment to a single ring element $w \in R_q$. In order to commit to a vector $\mathbf{w} \in R_q^k$, it is proposed to commit to each vector element separately. Therefore, the byte size of the second message can be computed as $\frac{k_{sig} \cdot N \cdot (m + k_{com}) \cdot \lceil \log(q_{com}) \rceil}{8}$. The third message consists of a signature share $\mathbf{z}_n$ and an opening for the commitment $\mathbf{r}_n$. We know that, for a valid signature share, it holds that $||\mathbf{z}_n||_2 \leq B_{sig}$ and we know that $||x||_\infty \leq ||x||_2$. The signature share may contain negative values, so for each coefficient of $\mathbf{z}_n$, one bit should be reserved to indicate the sign. Therefore, the approximate byte size of the signature share is $\frac{n \cdot (l + k_{sig}) \cdot (\lceil \log(B_{sig}) \rceil + 1)}{8}$. For a valid opening, it holds that $||\mathbf{r}_n||_2 \leq B_{com}$. The value $\mathbf{r}_n$ also may contain negative values, so for each coefficient of $\mathbf{r}_n$, one bit should be reserved to indicate the sign. The approximate byte size of the commitment opening is $\frac{k_{sig} \cdot N \cdot m' \cdot (\lceil \log(B_{com}) \rceil + 1)}{8}$.

Table 7. Illustrative parameters for Dåmgard et al. DS_3 [16].

Parameters for Signature Scheme	Description	Sizes
n	The degree bound of the polynomials in the ring	256
q_{sig}	Modulus	8,380,781
(k_{sig}, l)	Dimension of matrix and vectors	(4,4)
η	Size bound of coefficients in the secret key share	5
τ	Number on non-zero elements in the output of hash function H_0	60
σ_{sig}	Standard deviation of the Gaussian distribution	≈17,900
B_{sig}	The maximum l_2 norm of signature share $\mathbf{z}_j \in R_{q_{sig}}^{l+k_{sig}}$	≈990,000

Table 8. Illustrative parameters for Dåmgard et al. [16] statistically binding commitment scheme.

Parameters for Commitment Scheme	Description	Sizes
N	The degree bound of the polynomials in the ring	1024
q_{com}	Modulus	$\approx 2^{55}$
(m, m', k_{com})	Dimension of matrices and vectors	(6,9,1)
σ_{com}	Standard deviation of the Gaussian distribution	≈46,000
$B_{com} = 4\sigma_{com}\sqrt{m'N}$	The maximum l_2 of commitment opening $\mathbf{r}_j \in R_{q_{com}}^{m'}$	≈17,664,000

Table 9. Illustrative comparison of DiLizium and Dåmgard et al. DS_3 message sizes.

Messages	DiLizium	Dåmgard et al. DS_3
First message	256 bits	256 bits
Second message	1920 bytes	197,120 bytes
Third message	6080 bytes	125,184 bytes

From Table 9, we can see that the size of the second and the third messages in Dåmgard et al.'s DS_3 scheme is much larger than that in DiLizium. The reason is that scheme DS_3 Dåmgard et al. uses lattice-based homomorphic commitments. For the signature scheme to be secure, it was required to have statistical binding. Parameters that guarantee statistical binding are not very practical; however, there may exist optimal parameter choice.

Currently, it is not possible to provide a more detailed comparison of the efficiency of these schemes. The main reason is that neither of the works have a reference implementation yet. Therefore, we leave the implementation of the proposed scheme and a detailed comparison for future research.

7. Conclusions

Nowadays, threshold signature schemes have a variety of practical applications. There are several efficient threshold versions of the RSA and (EC)DSA signature schemes that are used in practice. However, threshold instantiations of post-quantum signature schemes are less researched. Previous researches have demonstrated that creating threshold post-quantum signatures is a highly non-trivial task. Some of the proposed schemes yield inefficient implementation, while others have incomplete security proofs.

In this work, we presented a new lattice-based two-party signature scheme: DiLizium. Our construction uses the SWIFFT homomorphic hash function to compute commitment in the signing process. We provide security proof for our scheme in the classical random oracle model under the Module-LWE, rejected Module-LWE, and Module-SIS assumptions. The proposed scheme can potentially substitute distributed RSA and ECDSA signature schemes in authentication applications such as Smart-ID [11]. This would allow for using these applications even in the quantum computing era.

Compared with the scheme proposed in [16], this work does not use sampling from the discrete Gaussian distribution and does not use a lattice-based homomorphic commitment

scheme. In the key generation and signing processes, our scheme uses uniform sampling, which facilitates secure implementations in the future.

The security proof of the proposed scheme is based on non-standard security assumption: rejected Module-LWE. Removing this assumption from the security proof is an important part of future work. Furthermore, the concept of homomorphic hash functions is new and has not been properly studied yet. We aim to research the properties and usage of homomorphic hash functions more deeply in future work. The implementation of the proposed scheme and the exact choice of parameters for this implementation is left for future research, which may also involve optimisation of the size of keys and signature, and security proof against the quantum adversary.

Author Contributions: Conceptualisation, J.V.; formal analysis and scheme design, J.V. and N.S.; security proof, J.V.; performance analysis, J.V. and N.S.; writing—original draft preparation, J.V. and N.S.; writing—review and editing, J.V., N.S. and J.W.; visualisation, J.V.; supervision, J.W.; project administration, J.W.; funding acquisition, J.W. All authors have read and agreed to the published version of the manuscript.

Funding: This paper has been supported by the Estonian Personal Research grant number 920 and European Regional Development Fund through the grant number EU48684.

Acknowledgments: The authors are grateful to Ahto Buldas and Alisa Pankova for their support throughout the process of research.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RSA	Rivest–Shamir–Adleman cryptosystem
DSA	Digital Signature Algorithm
ECDSA	Elliptic Curve Digital Signature Algorithm
NIST	National Institute of Standards and Technology
PQC	Post-Quantum Cryptography
QSCD	Qualified Electronic Signature Creation Device
LWE	Learning with Errors
SIS	Short Integer Solution
FSwA	Fiat–Shamir with Aborts
MPC	Multiparty Computation
GPV	Gentry–Peikert–Vaikuntanathan
FS	Fiat-Shamir
SHAKE	Secure Hash Algorithm and KECCAK
DS-UF-CMA	Distributed Signature Unforgeability Against Chosen Message Attacks
naHVZK	no-abort Honest-Verifier Zero-Knowledge

Appendix A. Full Security Proof

This section presents detailed security proof for the two-party signature scheme.

Definition A1 (no-abort Honest-Verifier Zero-Knowledge). *An identification scheme is said to be ϵ_{ZK}-naHVZK if there exists a probabilistic expected polynomial-time algorithm Sim that is given only the public key pk and that outputs (w, c, z) such that the following holds:*

- *The distribution of the simulated transcript produced by Sim ($(w, c, z) \leftarrow Sim(pk)$) has a statistical distance at most ϵ_{ZK} from the real transcript produced by the transcript algorithm $(w', c', z') \leftarrow Trans(sk)$.*
- *The distribution of c from the output $(w, c, z) \leftarrow Sim(pk)$ conditioned on $c \neq \perp$ is uniformly random over the set C.*

Theorem A1. *Assume a homomorphic hash function* $HomH : \{0,1\}^{a \cdot b} \rightarrow \mathbb{Z}_p^b$ *is provably collision-resistant and* ϵ*-regular; then for any probabilistic polynomial time adversary* $\mathcal{A}$ *that makes a single query to the key generation oracle,* q_s *queries to the signing oracle, and* q_h *queries to the random oracles* H_0, H_1, H_2, H_3*, the distributed signature protocol is DS-UF-CMA secure in the random oracle model under Module-LWE, rejected Module-LWE, and Module-SIS assumptions.*

Proof. Given an adversary $\mathcal{A}$ that succeeds in breaking the distributed signature protocol with advantage $\text{Adv}^{\text{DS-UF-CMA}}(\mathcal{A})$, a simulator $\mathcal{B}$ is constructed. $\mathcal{B}$ simulates the behaviour of the single honest party without using honestly generated secret keys for the computation. Algorithm $\mathcal{B}$ is constructed such that it fits all the assumptions of the forking lemma. By the definition of forking algorithm, it was required that $\mathcal{B}$ is given a public key and a random oracle query replies as input. $\mathcal{B}$ simulates the behaviour of the honest party P_n, and the party P_i is corrupted by the adversary. The algorithm $\mathcal{B}$ is defined in Algorithm A1.

Algorithm A1 $\mathcal{B}(pk, h_1, \ldots, h_{q_h+q_s+1})$.

1: Create empty hash tables HT_i for $i \in \{0, \ldots, 3\}$.
2: Create a set of queried messages $\mathcal{M} = \emptyset$.
3: Simulate the honest party oracle as follows:
- Upon receiving a query from $\mathcal{A}$ of the form (sid, msg), reply to the query as described in $Sim\mathcal{O}_{KeyGen}$ (Oracle A1) and $Sim\mathcal{O}_{Sign}$ (Oracle A2).
- If one of the oracles terminates with output of the form $(0, \perp)$, then $\mathcal{B}$ also terminates with the same output $(0, \perp)$.

4: Simulate random oracles as follows:
- Upon receiving a query from $\mathcal{A}$ to the random oracle, reply to the query as described in Algorithm A2.

5: Upon receiving a forgery $\sigma = (\mathbf{z}_1, \mathbf{z}_2, c)$ on message m' from $\mathcal{A}$:
- If $m' \in \mathcal{M}$, then $\mathcal{B}$ terminates with output $(0, \perp)$.
- Compute $com' := HomH(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t})$.
- Make query $c' \leftarrow H_0(m'||com')$.
- If $c \neq c'$ or $||\mathbf{z}_1||_\infty \geq \gamma_2 - \beta_2$ or $||\mathbf{z}_2||_\infty \geq \gamma_2 - \beta_2$, then $\mathcal{B}$ terminates with output $(0, \perp)$.
- Find index $i_f \in [q_h + q_s + 1]$ such that $c' = h_{i_f}$. $\mathcal{B}$ terminates with the output $(i_f, out = (com', c', \mathbf{z}_1, \mathbf{z}_2, m'))$

Oracle A1: $Sim\mathcal{O}_{KeyGen}(par, sid, msg)$**;**

The oracle is initialised with the set of public parameters *par* generated by Setup(1^λ) algorithm.

1. Upon receiving $(0, msg)$, if the flag $kgen = true$, then return $\perp$.
2. Upon receiving query with $sid = 0$ for the first time,
   (a) Initialise a machine $\mathcal{M}_0$. $\mathcal{M}_0$ uses the instructions of $Sim_{KeyGen}(par, \mathbf{A}, \mathbf{t})$ (Algorithm A10).
   (b) If P_n sends the first message according to the key generation protocol, then oracle returns this message.
3. If machine $\mathcal{M}_0$ has been already initialised,
   (a) Oracle gives the next incoming message msg to the $\mathcal{M}_0$.
   (b) Oracle returns reply that was received from $\mathcal{M}_0$.
   (c) If $\mathcal{M}_0$ finished the protocol with a local output $(\mathbf{t}_n, pk)$, then oracle sets the flag $kgen = true$.

Oracle A2: $Sim\mathcal{O}_{Sign}(sid, msg)$**;**

1. Upon receiving (sid, msg), if the flag $kgen = false$ and $sid \neq 0$, then return $\perp$.
2. Upon receiving query with sid for the first time,
   (a) Parse incoming message msg as message to be signed m.
   (b) Initialise a machine $\mathcal{M}_{sid}$. $\mathcal{M}_{sid}$ uses the instructions of $Sim_{Sign}(sid, \mathbf{t}_n, pk, m)$ (Algorithm A7).
   (c) The message to be signed m is included in the set of all queried messages $\mathcal{M}$.
   (d) If P_n sends the first message according to the signing protocol, then oracle returns this message.
3. If machine $\mathcal{M}_{sid}$ has been already initialised,
   (a) Oracle gives the next incoming message msg to the $\mathcal{M}_{sid}$.
   (b) Oracle returns reply that was received from $\mathcal{M}_{sid}$.
   (c) If $\mathcal{M}_{sid}$ finished the protocol with a local output σ, then the oracle returns this output.

Appendix A.1. Random oracle simulation

There are several random oracles that need to be simulated:

1. $H_0 : \{0,1\}^* \rightarrow C$
   [C is a set of all vectors in $\{-1, 0, 1\}^n$ with exactly τ nonzero elements]
2. $H_1 : \{0,1\}^* \rightarrow \{0,1\}^{l_1}$
3. $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{l_2}$
4. $H_3 : \{0,1\}^* \rightarrow \{0,1\}^{l_3}$

All of the random oracles are simulated as described in Algorithm A2. Additionally, there is a searchHash(HT, h) algorithm for searching entries from the hash table defined in Algorithm A3.

Algorithm A2 $H_i(x)$.

HT_i is a hash table that is initially empty.

1: On a query x, return element $HT_i[x]$ if it was previously defined.
2: Otherwise, sample output y uniformly at random from the range of H_i and return $HT_i[x] := y$

Algorithm A3 searchHash(HT, h)

1: For value h, find its preimage m in the hash table such that $HT[m] = h$.
2: If preimage of value h does not exist, set flag *alert* and set preimage $m = \perp$.
3: If for value h more than one preimage exists in hash table HT, set flag *bad*.
4: **Output:** $(m, alert, bad)$

Simulators for the key generation and signing processes were constructed using several intermediate games. The goal was to remove the usage of the actual secret key share of the party P_n from both processes. Let $\Pr[\mathbf{G}_i]$ denote the probability that $\mathcal{B}$ does not output $(0,\perp)$ in the game $\mathbf{G}_i$. This means that the adversary must have created a valid forgery (as defined in Algorithm A1). Then, $\Pr[\mathbf{G}_0] = \mathrm{Adv}^{\text{DS-UF-CMA}}(\mathcal{A})$. In Game 0, $\mathcal{B}$ simulates the honest party behaviour using the same instructions as in the original $\text{KeyGen}_n(par)$ and $\text{Sign}_n(sk_n, m)$ protocols.

Appendix A.2. Game 1

In Game 1, only signing process is changed with respect to the previous game. The simulator for the signing process in Game 1 is described in Algorithm A4. Challenge c is now sampled uniformly at random, and the signature shares are computed without communicating with the adversary. Changes with respect to the previous game are highlighted.

Algorithm A4 $Sim_{Sign}(sk_n, pk, m)$.

1: $c \leftarrow C$.
2: $\mathbf{y}_1^n, \mathbf{y}_2^n \leftarrow S_{\gamma-1}^k$.
3: $\mathbf{w}_n := \mathbf{A}\mathbf{y}_1^n + \mathbf{y}_2^n$.
4: $\mathbf{z}_1^n := \mathbf{y}_1^n + c\mathbf{s}_1^n$ and $\mathbf{z}_2^n := \mathbf{y}_2^n + c\mathbf{s}_2^n$.
5: $com_n \leftarrow HomH(\mathbf{w}_n)$, send out $h_n \leftarrow H_3(com_n)$.
6: Upon receiving h_i, search for $(com_i, alert, bad_7) \leftarrow \text{searchHash}(HT_3, h_i)$.
7: If the flag bad_7 is set, then simulation fails with output $(0, \perp)$.
If the flag *alert* is set, then send out com_n.
8: $com := com_n + com_i$.
9: Program random oracle H_0 to respond queries $(m||com)$ with c.
Set $HT_0[(m||com)] := c$. If $HT_0[(m||com)]$ has been already set, set flag bad_8 and the simulation fails with output $(0, \perp)$.
10: Send out com_n. Upon receiving com_i:
- if $H_3(com_i) \neq h_i$: send out ABORT.
- if the flag *alert* is set and $H_3(com_i) = h_i$: set the flag bad_9 and the simulation fails with output $(0, \perp)$.

11: Otherwise, run rejection sampling, if it did not pass: send out RESTART and go to the step 1.
12: Otherwise, send out $(\mathbf{z}_1^n, \mathbf{z}_2^n)$. Upon receiving RESTART, go to step 1.
13: Upon receiving $(\mathbf{z}_1^i, \mathbf{z}_2^i)$, reconstruct $\mathbf{w}_i := \mathbf{A}\mathbf{z}_1^i + \mathbf{z}_2^i - c\mathbf{t}_i$ and check that $HomH(\mathbf{w}_i) = com_i$, if not: send out ABORT.
14: Otherwise, set $\mathbf{z}_1 := \mathbf{z}_1^n + \mathbf{z}_1^i$, $\mathbf{z}_2 := \mathbf{z}_2^n + \mathbf{z}_2^i$ and output composed signature $\sigma := (\mathbf{z}_1, \mathbf{z}_2, c)$.

Appendix A.2.1. Game 0 $\rightarrow$ Game 1:

The difference between Game 0 and Game 1 can be expressed using the *bad* events that can happen with the following probabilities:

- Pr[bad_7] is the probability that at least one collision occurs during at most $q_h + 2q_s$ queries to the random oracle H_3 made by adversary or simulator. This means that two values $com_j \neq com_j'$ were found such that $HT_3[com_j] = HT_3[com_j']$. As all of the responses of H_3 are chosen uniformly at random from $\{0,1\}^{l_3}$ and there are at most $q_h + 2q_s$ queries to the random oracle H_3, the probability of at least one collision occurring can be expressed as $\frac{\left((q_h + 2q_s)(q_h + 2q_s + 1)\right)/2}{2^{l_3}} \leq \frac{(q_h + 2q_s + 1)^2}{2^{l_3+1}}$, where l_3 is the length of H_3 output.
- Pr[bad_8] is the probability that programming random oracle H_0 fails at least once during q_s queries. This event can happen in the following two cases: $H_3(com_n)$ was previously queried by the adversary or it was not queried by the adversary:
  - Case 1: $H_3(com_n)$ has been already asked by adversary during at most $q_h + 2q_s$ queries to H_3. This means that the adversary knows *com* and may have queried $H_0(m||com)$ before. This event corresponds to guessing the value of com_n.
    Let the uniform distribution over $\mathbb{Z}_p^b$ be denoted as X and the distribution of $HomH$ output be denoted as Y. As $HomH$ is ϵ-regular (for some negligibly small ϵ), it holds that $\text{SD}(X, Y) \leq \epsilon$. Then, for any subset T of $\mathbb{Z}_p^b$, by definition of statistical distance, it holds that $\Pr[X \in T] \leq \Pr[Y \in T] + \epsilon$. Therefore, for a uniform distribution X, the probability of guessing Y by T is bounded by $\frac{1}{|\mathbb{Z}_p^b|} + \epsilon$.
    Since com_n was produced by $\mathcal{B}$ in the beginning of the signing protocol completely independently from $\mathcal{A}$, the probability that $\mathcal{A}$ queried $H_3(com_n)$ is at most $\frac{1}{|\mathbb{Z}_p^b|} + \epsilon$ for each query.
  - Case 2: $HT_0[m||com]$ has been set by adversary or simulator by chance during at most $q_h + q_s$ prior queries to the H_0. Since $\mathcal{A}$ has not queried $H_3(com_n)$ before, adversary does not know com_n and the view of $\mathcal{A}$ is completely independent

from *com*. The probability that *com* occurred by chance in one of the previous queries to H_0 is at most $(q_h + q_s)\left(\frac{1}{|\mathbb{Z}_p^b|} + \epsilon\right)$.

- $\Pr[bad_9]$ is the probability that the adversary predicted at least one of two outputs of the random oracle H_3 without making a query to it. In this case, there is no record in the hash table HT_3 that corresponds to the preimage com_j. This can happen with probability at most $\frac{2}{2^{l_3}}$ for each signing query.

  Therefore, the difference between two games is

$$|\Pr[\mathbf{G}_1] - \Pr[\mathbf{G}_0]| \leq \Pr[bad_7] + \Pr[bad_8] + \Pr[bad_9] \leq \frac{(q_h + 2q_s + 1)^2}{2^{l_3+1}} + q_s\left((q_h + 2q_s)\left(\frac{1}{|\mathbb{Z}_p^b|} + \epsilon\right) + (q_h + q_s)\left(\frac{1}{|\mathbb{Z}_p^b|} + \epsilon\right) + \frac{2}{2^{l_3}}\right) = \frac{(q_h + 2q_s + 1)^2}{2^{l_3+1}} + q_s\left(\left(\frac{1}{|\mathbb{Z}_p^b|} + \epsilon\right) \cdot (2q_h + 3q_s) + \frac{2}{2^{l_3}}\right).$$

Appendix A.3. Game 2

In Game 2, when the signature share gets rejected, simulator commits to a uniformly random vector $\mathbf{w}_n$ from the ring R_q instead of committing to a vector computed as $\mathbf{A}\mathbf{y}_1^n + \mathbf{y}_2^n$. The simulator for the signing process in Game 2 is described in Algorithm A5.

Appendix A.3.1. Game 1 $\rightarrow$ Game 2:

The difference between Game 1 and Game 2 can be expressed with the probability that the adversary can distinguish simulated commitment with random $\mathbf{w}_n$ from the real one when the rejection sampling algorithm does not pass. If the signature shares are rejected, it means that $\mathbf{z}_1^n \geq \gamma - \beta$ or $\mathbf{z}_2^n \geq \gamma - \beta$.

Let us assume that there exists an adversary $\mathcal{D}$ who succeeds in distinguish simulated commitment with random $\mathbf{w}_n$ from the real one with nonnegligible probability:

$$\mathrm{Adv}(\mathcal{D}) = \Pr[b = b' : \mathbf{A} \leftarrow R_q^{k\times k}, \mathbf{s}_1, \mathbf{s}_2 \leftarrow S_{\eta-1}^k, c \leftarrow C, \mathbf{y}_1, \mathbf{y}_2 \leftarrow S_{\gamma-1}^k, \mathbf{w}_0 \leftarrow \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2, \mathbf{w}_1 \leftarrow R_q^k, b \leftarrow \{0,1\}, h_b \leftarrow HomH(\mathbf{w}_b), b' \leftarrow \mathcal{D}(\mathbf{A}, h_b, c) \mid \mathbf{y}_1 + c\mathbf{s}_1 \geq \gamma - \beta \text{ or } \mathbf{y}_2 + c\mathbf{s}_2 \geq \gamma - \beta].$$

Then, the adversary $\mathcal{D}$ can be used to construct an adversary $\mathcal{A}_{R-MLWE}$ who solves the rejected Module-LWE for parameters $(q, k, k, \gamma, U, \beta)$, where U is the uniform distribution. The adversary $\mathcal{A}_{R-MLWE}$ is defined in Algorithm A6.

Algorithm A5 $Sim_{Sign}(sk_n, pk, m)$.

1: $c \leftarrow C$.
2: $\mathbf{y}_1^n, \mathbf{y}_2^n \leftarrow S_{\gamma-1}^k$.
3: $\mathbf{z}_1^n := \mathbf{y}_1^n + c\mathbf{s}_1^n$ and $\mathbf{z}_2^n := \mathbf{y}_2^n + c\mathbf{s}_2^n$.
4: Run rejection sampling; if it does not pass, proceed as follows:
   1. $\mathbf{w}_n \leftarrow R_q^k$.
   2. $com_n \leftarrow HomH(\mathbf{w}_n)$, send out $h_n \leftarrow H_3(com_n)$.
   3. Upon receiving h_i, search for $(com_i, alert, bad_7) \leftarrow \text{searchHash}(HT_3, h_i)$.
   4. If the flag bad_7 is set, then simulation fails with output $(0, \perp)$. If the flag *alert* is set, then send out com_n.
   5. $com := com_n + com_i$.
   6. Program random oracle H_0 to respond queries $(m||com)$ with c. Set $HT_0[(m||com)] := c$. If $HT_0[(m||com)]$ has been already set, set flag bad_8 and the simulation fails with output $(0, \perp)$.
   7. Send out com_n. Upon receiving com_i:
      - if $H_3(com_i) \neq h_i$: send out ABORT.
      - if the flag *alert* is set and $H_3(com_i) = h_i$: set the flag bad_9 and the simulation fails with output $(0, \perp)$.
   8. Otherwise, send out RESTART and go to step 1.
5: If rejection sampling passes, proceed as follows:
   1. $\mathbf{w}_n := \mathbf{A}\mathbf{y}_1^n + \mathbf{y}_2^n$.
   2. $com_n \leftarrow HomH(\mathbf{w}_n)$, send out $h_n \leftarrow H_3(com_n)$.
   3. Upon receiving h_i, search for $(com_i, alert, bad_7) \leftarrow \text{searchHash}(HT_3, h_i)$.
   4. If the flag bad_7 is set, then simulation fails with output $(0, \perp)$. If the flag *alert* is set, then continue.
   5. $com := com_n + com_i$.
   6. Program random oracle H_0 to respond queries $(m||com)$ with c. Set $HT_0[(m||com)] := c$. If $HT_0[(m||com)]$ has been already set, set flag bad_8 and the simulation fails with output $(0, \perp)$.
   7. Send out com_n.Upon receiving com_i:
      - if $H_3(com_i) \neq h_i$: send out ABORT.
      - if the flag *alert* is set and $H_3(com_i) = h_i$: set the flag bad_9 and the simulation fails with output $(0, \perp)$.
   8. Otherwise, send out $(\mathbf{z}_1^n, \mathbf{z}_2^n)$. Upon receiving RESTART, go to step 1.
   9. Upon receiving $(\mathbf{z}_1^i, \mathbf{z}_2^i)$, reconstruct $\mathbf{w}_i := \mathbf{A}\mathbf{z}_1^i + \mathbf{z}_2^i - c\mathbf{t}_i$ and check that $HomH(\mathbf{w}_i) = com_i$, if not: send out ABORT.
   10. Otherwise, set $\mathbf{z}_1 := \mathbf{z}_1^n + \mathbf{z}_1^i$, $\mathbf{z}_2 := \mathbf{z}_2^n + \mathbf{z}_2^i$ and output composed signature $\sigma := (\mathbf{z}_1, \mathbf{z}_2, c)$.

Algorithm A6 $\mathcal{A}_{R-MLWE}(\mathbf{A}, \mathbf{w}_b, c)$.

1: $h_b \leftarrow HomH(\mathbf{w}_b)$
2: $b' \leftarrow \mathcal{D}(\mathbf{A}, h_b, c)$
3: **return** b'

As a consequence, the difference between the two games is bounded by the following:

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq q_s \cdot \text{Adv}^{\text{R-MLWE}}_{(q,k,k,\gamma,U,\beta)}$$

Appendix A.4. Game 3

In Game 3, the simulator does not generate the signature shares honestly and thus does not perform rejection sampling honestly. Rejection sampling is simulated as follows:

- **Rejection case**: with probability $1 - \left(1 - \frac{|S_{\gamma-\beta-1}^k|}{|S_{\gamma-1}^k|}\right)^2$ simulator generates commitment to the random $\mathbf{w}_n$ as in the previous game.
- **Otherwise**, sample signature shares from the set $S_{\gamma-\beta-1}$ and compute $\mathbf{w}_n$ out of it.

The simulator for the signing process in Game 3 is described in Algorithm A7.

Appendix A.4.1. Game 2 → Game 3:

The signature shares generated in Algorithm A7 are indistinguishable from the real ones because of the ϵ_{ZK}-naHVZK property of the underlying identification scheme from [13], appendix B. Therefore, the difference between Game 2 and Game 3 can be defined as follows:

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \epsilon_{ZK}$$

According to the proof from [13], $\epsilon_{ZK} = 0$ for the underlying identification scheme.

Algorithm A7 $Sim_{Sign}(\mathbf{t}_n, pk, m)$.

1: With probability $1 - \left(1 - \frac{|S^k_{\gamma-\beta-1}|}{|S^k_{\gamma-1}|}\right)^2$, proceed as follows:

1. $c \leftarrow C$.
2. $\mathbf{w}_n \leftarrow R^k_q$.
3. $com_n \leftarrow HomH(\mathbf{w}_n)$, send out $h_n \leftarrow H_3(com_n)$.
4. Upon receiving h_i, search for $(com_i, alert, bad_7) \leftarrow \text{searchHash}(HT_3, h_i)$.
5. If the flag bad_7 is set, then simulation fails with output $(0, \perp)$. If the flag $alert$ is set, then send out com_n.
6. $com := com_n + com_i$.
7. Program random oracle H_0 to respond queries $(m||com)$ with c. Set $HT_0[(m||com)] := c$. If $HT_0[(m||com)]$ has been already set, set flag bad_8 and the simulation fails with output $(0, \perp)$.
8. Send out com_n. Upon receiving com_i:
   - if $H_3(com_i) \neq h_i$: send out ABORT.
   - if the flag $alert$ is set and $H_3(com_i) = h_i$: set the flag bad_9 and the simulation fails with output $(0, \perp)$.
9. Otherwise, send out RESTART and go to step 1.

2: Otherwise, proceed as follows:

1. $c \leftarrow C$.
2. $\mathbf{z}^n_1 \leftarrow S^k_{\gamma-\beta-1}$ and $\mathbf{z}^n_2 \leftarrow S^k_{\gamma-\beta-1}$.
3. $\mathbf{w}_n := \mathbf{A}\mathbf{z}^n_1 + \mathbf{z}^n_2 - c\mathbf{t}_n$.
4. $com_n \leftarrow HomH(\mathbf{w}_n)$, send out $h_n \leftarrow H_3(com_n)$.
5. Upon receiving h_i, search for $(com_i, alert, bad_7) \leftarrow \text{searchHash}(HT_3, h_i)$.
6. If the flag bad_7 is set, then simulation fails with output $(0, \perp)$. If the flag $alert$ is set, then continue.
7. $com := com_n + com_i$.
8. Program random oracle H_0 to respond queries $(m||com)$ with c. Set $HT_0[(m||com)] := c$. If $HT_0[(m||com)]$ has been already set, set flag bad_8 and the simulation fails with output $(0, \perp)$.
9. Send out com_n. Upon receiving com_i:
   - if $H_3(com_i) \neq h_i$: send out ABORT.
   - if the flag $alert$ is set and $H_3(com_i) = h_i$: set the flag bad_9 and the simulation fails with output $(0, \perp)$.
10. Otherwise, send out $(\mathbf{z}^n_1, \mathbf{z}^n_2)$. Upon receiving RESTART, go to step 1.
11. Upon receiving $(\mathbf{z}^i_1, \mathbf{z}^i_2)$, reconstruct $\mathbf{w}_i := \mathbf{A}\mathbf{z}^i_1 + \mathbf{z}^i_2 - c\mathbf{t}_i$ and check that $HomH(\mathbf{w}_i) = com_i$, if not: send out ABORT.
12. Otherwise, set $\mathbf{z}_1 := \mathbf{z}^n_1 + \mathbf{z}^i_1$, $\mathbf{z}_2 := \mathbf{z}^n_2 + \mathbf{z}^i_2$ and output composed signature $\sigma := (\mathbf{z}_1, \mathbf{z}_2, c)$.

Appendix A.5. Game 4

Now, the signing process does not rely on the actual secret key share of the honest party P_n. In the next games, the key generation process is changed so that it does not use secret keys as well. In this game, the simulator is given a predefined uniformly

random matrix $\mathbf{A} \leftarrow R_q^{k\times k}$, and the simulator defines its own matrix share out of it. By definition, the algorithm $\mathcal{B}$ (Algorithm A1) receives a pre-generated public key pk as the input. Therefore, the simulator in Game 4 is given a predefined matrix $\mathbf{A}$, and in the later games, the simulator is changed so that it receives the entire public key and uses it to compute its shares $\mathbf{A}_n, \mathbf{t}_n$. The simulator for the key generation process in Game 4 is described in Algorithm A8.

Algorithm A8 $Sim_{KeyGen}(par, \mathbf{A})$.

1: Send out $hk_n \leftarrow \{0,1\}^{l_1}$.
2: Upon receiving hk_i:
- search for $(\mathbf{A}_i, alert, bad_1) \leftarrow \text{searchHash}(HT_1, hk_i)$.
- if the flag bad_1 is set, then simulation fails with output $(0, \perp)$.
- if the flag $alert$ is set, then sample $\mathbf{A}_n \leftarrow R_q^{k\times k}$.
  Otherwise, define $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$.

3: Program random oracle H_1 to respond queries $\mathbf{A}_n$ with hk_n. Set $HT_1[\mathbf{A}_n] := hk_n$. If $HT_1[\mathbf{A}_n]$ has been already set, then set the flag bad_2 and the simulation fails with output $(0, \perp)$.
4: Send out $\mathbf{A}_n$. Upon receiving $\mathbf{A}_i$:
- if $H_1(\mathbf{A}_i) \neq hk_i$: send out ABORT.
- if the flag $alert$ is set and $H_1(\mathbf{A}_i) = hk_i$: set the flag bad_3 and the simulation fails with output $(0, \perp)$.

5: $(\mathbf{s}_1^n, \mathbf{s}_2^n) \leftarrow S_\eta^k \times S_\eta^k$.
6: $\mathbf{t}_n := \mathbf{A}\mathbf{s}_1^n + \mathbf{s}_2^n$, send out $comk_n := H_2(\mathbf{t}_n)$.
7: Upon receiving $comk_i$, send out $\mathbf{t}_n$.
8: Upon receiving $\mathbf{t}_i$, check that $H_2(\mathbf{t}_i) = comk_i$. If not: send out ABORT.
9: Otherwise, $\mathbf{t} := \mathbf{t}_n + \mathbf{t}_i$, $pk := (\mathbf{A}, \mathbf{t})$ and $sk := (\mathbf{A}, \mathbf{t}_i, \mathbf{s}_n, \mathbf{s}_n')$.

Appendix A.5.1. Game 3 $\rightarrow$ Game 4:

The distribution of public matrix $\mathbf{A}$ does not change between Game 3 and Game 4. The difference between Game 3 and Game 4 can be expressed using *bad* events that happen with the following probabilities:

- $\Pr[bad_1]$ is the probability that at least one collision occurs during at most q_h queries to the random oracle H_1 made by adversary or simulator. This can happen with probability at most $\frac{q_h(q_h+1)/2}{2^{l_1+1}}$, where l_1 is the length of H_1 output.
- $\Pr[bad_2]$ is the probability that programming random oracle H_1 fails, which happens if $H_1(\mathbf{A}_n)$ has been previously asked by adversary during at most q_h queries to the random oracle H_1. This event corresponds to guessing random $\mathbf{A}_n$, for each query the probability of this event is bounded by $\frac{1}{q^{n\cdot k\cdot k}}$.
- $\Pr[bad_3]$ is the probability that adversary predicted at least one of two outputs of the random oracle H_1 without making a query to it. This can happen with probability at most $\frac{2}{2^{l_1}}$.

  Therefore, the difference between the two games is

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \Pr[bad_1] + \Pr[bad_2] + \Pr[bad_3] \leq \frac{(q_h+1)q_h}{2^{l_1+1}} + \frac{q_h}{q^{n\cdot k\cdot k}} + \frac{2}{2^{l_1}}$$

Appendix A.6. Game 5

In Game 5, the simulator picks public key share $\mathbf{t}_n$ randomly from the ring instead of computing it using secret keys. The simulator for the key generation process in Game 5 is described in Algorithm A9.

Algorithm A9 $Sim_{KeyGen}(par, \mathbf{A})$.

1: Send out $hk_n \leftarrow \{0,1\}^{l_1}$.
2: Upon receiving hk_i:
- search for $(\mathbf{A}_i, alert, bad_1) \leftarrow \text{searchHash}(HT_1, hk_i)$.
- if the flag bad_1 is set, then simulation fails with output $(0, \perp)$.
- if the flag $alert$ is set, then sample $\mathbf{A}_n \leftarrow R_q^{k \times k}$. Otherwise, define $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$.

3: Program random oracle H_1 to respond queries $\mathbf{A}_n$ with hk_n. Set $HT_1[\mathbf{A}_n] := hk_n$. If $HT_1[\mathbf{A}_n]$ has been already set, then set the flag bad_2 and the simulation fails with output $(0, \perp)$.
4: Send out $\mathbf{A}_n$. Upon receiving $\mathbf{A}_i$:
- if $H_1(\mathbf{A}_i) \neq hk_i$: send out ABORT.
- if the flag $alert$ is set and $H_1(\mathbf{A}_i) = hk_i$: set the flag bad_3 and the simulation fails with output $(0, \perp)$.

5: $\mathbf{t}_n \leftarrow R_q^k$, send out $comk_n = H_2(\mathbf{t}_n)$.
6: Upon receiving $comk_i$, send out $\mathbf{t}_n$.
7: Upon receiving $\mathbf{t}_i$, check that $H_2(\mathbf{t}_i) = comk_i$. If not: send out ABORT.
8: Otherwise, $\mathbf{t} := \mathbf{t}_n + \mathbf{t}_i$, $pk := (\mathbf{A}, \mathbf{t})$.

Appendix A.6.1. Game 4 $\rightarrow$ Game 5:

In Game 5, public key share $\mathbf{t}_n$ is sampled uniformly at random from R_q^k instead of computing it as $\mathbf{A}\mathbf{s}_1^n + \mathbf{s}_2^n$, where $\mathbf{s}_1^n, \mathbf{s}_2^n$ are random elements from S_η^k. As matrix $\mathbf{A}$ follows the uniform distribution over $R_q^{k \times k}$, if adversary can distinguish between Game 3 and Game 4, this adversary can be used as a distinguisher that breaks the decisional Module-LWE problem for parameters (q, k, k, η, U), where U is the uniform distribution.

Therefore, the difference between two games is bounded by the advantage of adversary in breaking decisional Module-LWE:

$$|\Pr[\mathbf{G}_5] - \Pr[\mathbf{G}_4]| \leq \text{Adv}^{\text{Dec-MLWE}}_{(q,k,k,\eta,U)}$$

Appendix A.7. Game 6

In Game 6, the simulator uses as input a random resulting public key $\mathbf{t} \in R_q^k$ to compute its own share $\mathbf{t}_n$. The simulator for the key generation process in Game 6 is described in Algorithm A10.

Appendix A.7.1. Game 5 $\rightarrow$ Game 6:

The distributions of $\mathbf{t}, \mathbf{t}_n$ do not change with respect to Game 5. The difference between Game 5 and Game 6 can be expressed using *bad* events that happen with the following probabilities:

- $\Pr[bad_4]$ is the probability that at least one collision occurs during at most q_h queries to the random oracle H_2 made by adversary or simulator. This can happen with probability at most $\frac{q_h(q_h+1)/2}{2^{l_2+1}}$, where l_2 is the length of H_2 output.
- $\Pr[bad_5]$ is the probability that programming random oracle H_2 fails, which happens if $H_2(\mathbf{t}_n)$ was previously asked by adversary during at most q_h queries to the random oracle H_2. This event corresponds to guessing a uniformly random $\mathbf{t}_n \in R_q^k$, for each query the probability of this event is bounded by $\frac{1}{q^{n \cdot k}}$.
- $\Pr[bad_6]$ is the probability that adversary predicted at least one of two outputs of the random oracle H_2 without making a query to it. This can happen with probability at most $\frac{2}{2^{l_2}}$.

Therefore, the difference between the two games is

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \Pr[bad_4] + \Pr[bad_5] + \Pr[bad_6] \leq \frac{(q_h+1)q_h}{2^{l_2+1}} + \frac{q_h}{q^{n \cdot k}} + \frac{2}{2^{l_2}}$$

Algorithm A10 $Sim_{KeyGen}(par, \mathbf{A}, \mathbf{t})$.

1: Send out $hk_n \leftarrow \{0,1\}^{l_1}$.
2: Upon receiving hk_i:
- search for $(\mathbf{A}_i, alert, bad_1) \leftarrow \text{searchHash}(HT_1, hk_i)$.
- if the flag bad_1 is set, then simulation fails with output $(0, \perp)$.
- if the flag *alert* is set, then sample $\mathbf{A}_n \leftarrow R_q^{k \times k}$. Otherwise, define $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$.

3: Program random oracle H_1 to respond queries $\mathbf{A}_n$ with hk_n. Set $HT_1[\mathbf{A}_n] := hk_n$. If $HT_1[\mathbf{A}_n]$ has been already set, then set the flag bad_2 and the simulation fails with output $(0, \perp)$.
4: Send out $\mathbf{A}_n$. Upon receiving $\mathbf{A}_i$:
- if $H_1(\mathbf{A}_i) \neq hk_i$: send out ABORT.
- if the flag *alert* is set and $H_1(\mathbf{A}_i) = hk_i$: set the flag bad_3 and the simulation fails with output $(0, \perp)$.

5: Send out $comk_n \leftarrow \{0,1\}^{l_2}$.
6: Upon receiving $comk_i$, search for $(\mathbf{t}_i, alert, bad_4) \leftarrow \text{searchHash}(HT_2, comk_i)$.
7: If the flag bad_4 is set, then simulation fails with output $(0, \perp)$.
8: Compute public key share:
- If the flag *alert* is set, $\mathbf{t}_n \leftarrow R_q^k$.
- Otherwise, $\mathbf{t}_n := \mathbf{t} - \mathbf{t}_i$.

9: Program random oracle H_2 to respond queries $\mathbf{t}_n$ with $comk_n$. Set $HT_2[\mathbf{t}_n] := comk_n$. If $HT_2[\mathbf{t}_n]$ has been already set, set flag bad_5 and the simulation fails with output $(0, \perp)$.
10: Send out $\mathbf{t}_n$. Upon receiving $\mathbf{t}_i$:
- if $H_2(t_i) \neq comk_i$: send out ABORT.
- if the flag *alert* is set and $H_2(\mathbf{t}_i) = comk_i$: set the flag bad_6 and simulation fails with output $(0, \perp)$.

11: Otherwise, $\mathbf{t} := \mathbf{t}_n + \mathbf{t}_i$, $pk := (\mathbf{A}, \mathbf{t})$.

Appendix A.8. Forking Lemma

Now, both key generation and signing do not rely on the actual secret key share of the honest party P_n. In order to conclude the proof, it is needed to invoke forking lemma to receive two valid forgeries that are constructed using the same commitment $com = com'$ but different challenges $c \neq c'$.

Currently, the combined public key consists of matrix $\mathbf{A}$ uniformly distributed in $R_q^{k \times k}$ and vector $\mathbf{t}$ uniformly distributed in R_q^k. We want to replace it with Module-SIS instance $[\mathbf{A}'|\mathbf{I}]$, where $\mathbf{A}' \in R_q^{k \times (k+1)}$. The view of adversary will not be changed if we set $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$.

Let us define an input generation algorithm $\mathcal{IG}$ such that it produces the following input: $(\mathbf{A}, \mathbf{t})$ for the $\mathcal{F}_B$. Now, let us construct $\mathcal{B}'$ around the previously defined simulator $\mathcal{B}$. $\mathcal{B}'$ invokes the forking algorithm $\mathcal{F}_B$ on the input $(\mathbf{A}, \mathbf{t})$.

As a result, with probability frk two valid forgeries $out = (com, c, \mathbf{z}_1, \mathbf{z}_2, m)$ and $out' = (com', c', \mathbf{z}_1', \mathbf{z}_2', m')$ are obtained. Here, by the construction of $\mathcal{F}_B$, it holds that $c \neq c'$, $com = com'$, $m = m'$. The probability frk satisfiesollowing:

$$\Pr[\mathbf{G}_6] = acc \leq \frac{q_h + q_s + 1}{|C|} + \sqrt{(q_h + q_s + 1) \cdot frk}$$

Since both signatures are valid, it holds that

$$HomH(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}) = com = com' = HomH(\mathbf{A}\mathbf{z}_1' + \mathbf{z}_2' - c'\mathbf{t})$$

Let us examine the following cases:

Case 1: $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} \neq \mathbf{A}\mathbf{z}'_1 + \mathbf{z}'_2 - c'\mathbf{t}$, and $\mathcal{B}'$ is able to break the collision resistance of the hash function (that is hard under the worst-case difficulty of finding short vectors in cyclic/ideal lattices), as was proven in [35,36].

Case 2: $\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}\mathbf{z}'_1 + \mathbf{z}'_2 - c'\mathbf{t}$. It can be rearranged as $\mathbf{A}\mathbf{z}_1 - \mathbf{A}\mathbf{z}'_1 + \mathbf{z}_2 - \mathbf{z}_2 - c\mathbf{t} + c'\mathbf{t} = \mathbf{0}$, and this, in turn, leads to

$$\left[\mathbf{A}|\mathbf{I}|\mathbf{t}\right] \begin{bmatrix} \mathbf{z}_1 - \mathbf{z}'_1 \\ \mathbf{z}_2 - \mathbf{z}'_2 \\ c' - c \end{bmatrix} = \mathbf{0}$$

Now, recall that $\left[\mathbf{A}|\mathbf{I}|\mathbf{t}\right]$ is an instance of Module-SIS problem; this means that we found a solution for Module-SIS with parameters $(q, k, k+1, \xi)$, where $\xi \leq 2(\gamma_2 - \beta_2)$.

Therefore, the probability frk is the following:

$$frk \leq \mathrm{Adv}^{\mathrm{MSIS}}_{(q,k,k+1,\xi)} + \mathrm{Adv}^{\mathrm{CR}}$$

Finally, taking into account that the underlying identification scheme has perfect naHVZK (i.e., $\epsilon_{ZK} = 0$), the advantage of the adversary is bounded by the following:

$$\mathrm{Adv}^{\text{DS-UF-CMA}}(\mathcal{A}) \leq \frac{(q_h + 2q_s + 1)^2}{2^{l_3+1}} + q_s \cdot \left(\left(\frac{1}{|\mathbb{Z}_p^b|}\right) \cdot (2q_h + 3q_s) + \frac{2}{2^{l_3}}\right) +$$
$$q_s \cdot \mathrm{Adv}^{\text{R-MLWE}}_{(q,k,k,\gamma,U,\beta)} + \frac{(q_h+1)q_h}{2^{l_1+1}} + \frac{q_h}{q^{n\cdot k\cdot k}} + \frac{2}{2^{l_1}} + \mathrm{Adv}^{\text{Dec-MLWE}}_{(q,k,k,\eta,U)} +$$
$$\frac{(q_h+1)q_h}{2^{l_2+1}} + \frac{q_h}{q^{n\cdot k}} + \frac{2}{2^{l_2}} + \frac{q_h + q_s + 1}{|C|} + \sqrt{(q_h + q_s + 1) \cdot \left(\mathrm{Adv}^{\mathrm{MSIS}}_{(q,k,k+1,\xi)} + \mathrm{Adv}^{\mathrm{CR}}\right)}$$

□

References

1. Shor, P.W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM J. Comput.* 1997, *26*, 1484–1509. doi:10.1137/S0097539795293172.
2. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, 20–22 November 1994; pp. 124–134. doi:10.1109/SFCS.1994.365700.
3. Chen, L.; Jordan, S.; Liu, Y.K.; Moody, D.; Peralta, R.; Perlner, R.; Smith-Tone, D. *Report on Post-Quantum Cryptography*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2016. doi:10.6028/nist.ir.8105.
4. Shoup, V. Practical Threshold Signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2000; Volume 1807, pp. 207–220. doi:10.1007/3-540-45539-6_15.
5. Damgård, I.; Koprowski, M. Practical Threshold RSA Signatures without a Trusted Dealer. In *International Conference on the Theory and Applications of Cryptographic Techniques*; Springer: Berlin/Heidelberg, Germany, 2001; Volume 2045, pp. 152–165. doi:10.1007/3-540-44987-6_10.
6. MacKenzie, P.D.; Reiter, M.K. Two-party generation of DSA signatures. *Int. J. Inf. Sec.* 2004, *2*, 218–239. doi:10.1007/s10207-004-0041-0.
7. Lindell, Y. Fast Secure Two-Party ECDSA Signing. In *Annual International Cryptology Conference*; Katz, J., Shacham, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10402, pp. 613–644. doi:10.1007/978-3-319-63715-0_21.
8. Doerner, J.; Kondi, Y.; Lee, E.; Shelat, A. Secure Two-party Threshold ECDSA from ECDSA Assumptions. In Proceedings of the 2018 IEEE Symposium on Security and Privacy, SP San Francisco, CA, USA, 20–24 May 2018; pp. 980–997. doi:10.1109/SP.2018.00036.
9. Buldas, A.; Kalu, A.; Laud, P.; Oruaas, M. Server-Supported RSA Signatures for Mobile Devices. In Proceedings of the Computer Security– ESORICS 2017, Oslo, Norway, 11–15 September 2017; pp. 315–333.
10. SK ID Solutions. *eID scheme: SMART-ID*; Public version, 1.0.; 2019. Available online: https://www.ria.ee/sites/default/files/content-editors/EID/smart-id_skeemi_kirjeldus.pdf (accessed on 29 July 2021).
11. Solutions, S.I. Smart-ID Is a Smart Way to Identify Yourself, 2021. Available online: https://www.smart-id.com/ (accessed on 15 April 2021).
12. Cozzo, D.; Smart, N.P. Sharing the LUOV: Threshold Post-quantum Signatures. In Proceedings of the Cryptography and Coding—17th IMA International Conference, IMACC 2019, Oxford, UK, 16–18 December 2019; Volume 11929, pp. 128–153. doi:10.1007/978-3-030-35199-1_7.

13. Kiltz, E.; Lyubashevsky, V.; Schaffner, C. A Concrete Treatment of Fiat-Shamir Signatures in the Quantum Random-Oracle Model. Cryptology ePrint Archive, Report 2017/916. 2017. Available online: https://eprint.iacr.org/2017/916 (accessed on 29 July 2021).
14. Bruinderink, L.G.; Hülsing, A.; Lange, T.; Yarom, Y. Flush, Gauss, and Reload - A Cache Attack on the BLISS Lattice-Based Signature Scheme. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2016—18th International Conference, Santa Barbara, CA, USA, 17–19 August 2016; Volume 9813, pp. 323–345. doi:10.1007/978-3-662-53140-2_16.
15. Espitau, T.; Fouque, P.; Gérard, B.; Tibouchi, M. Side-Channel Attacks on BLISS Lattice-Based Signatures: Exploiting Branch Tracing against strongSwan and Electromagnetic Emanations in Microcontrollers. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–3 November 2017; pp. 1857–1874. doi:10.1145/3133956.3134028.
16. Damgård, I.; Orlandi, C.; Takahashi, A.; Tibouchi, M. Two-round n-out-of-n and Multi-Signatures and Trapdoor Commitment from Lattices. Cryptology ePrint Archive, Report 2020/1110. 2020. Available online: https://eprint.iacr.org/2020/1110 (accessed on 29 July 2021).
17. Moody, D.; Alagic, G.; Apon, D.C.; Cooper, D.A.; Dang, Q.H.; Kelsey, J.M.; Liu, Y.K.; Miller, C.A.; Peralta, R.C.; Perlner, R.A.; et al. *Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process*; Technical Report; National Institute of Standards and Technology: Gaithersburg, MD, USA 2020. doi:10.6028/nist.ir.8309.
18. Lyubashevsky, V.; Ducas, L.; Kiltz, E.; Lepoint, T.; Schwabe, P.; Seiler, G.; Stehle, D.; Bai, S. CRYSTALS-Dilithium. Algorithm Specifications and Supporting Documentation. 2020. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions (accessed on 29 July 2021).
19. Prest, T.; Fouque, P.A.; Hoffstein, J.; Kirchner, P.; Lyubashevsky, V.; Pornin, T.; Ricosset, T.; Seiler, G.; Whyte, W.; Zhang, Z. Falcon: Fast-Fourier Lattice-Based Compact Signatures over NTRU. 2020. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions (accessed on 29 July 2021).
20. Ding, J.; Chen, M.S.; Petzoldt, A.; Schmidt, D.; Yang, B.Y.; Kannwischer, M.; Patarin, J. Rainbow—Algorithm Specification and Documentation. The 3rd Round Proposal. 2020. Available online: https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions (accessed on 29 July 2021).
21. Bansarkhani, R.E.; Sturm, J. An Efficient Lattice-Based Multisignature Scheme with Applications to Bitcoins. In Proceedings of the Cryptology and Network Security—15th International Conference, CANS 2016, Milan, Italy, 14–16 November 2016; Volume 10052, pp. 140–155. doi:10.1007/978-3-319-48965-0_9.
22. Fukumitsu, M.; Hasegawa, S. A Tightly-Secure Lattice-Based Multisignature. In Proceedings of the 6th on ASIA Public-Key Cryptography Workshop, APKC@AsiaCCS 2019, Auckland, New Zealand, 8 July 2019; pp. 3–11. doi:10.1145/3327958.3329542.
23. Tso, R.; Liu, Z.; Tseng, Y. Identity-Based Blind Multisignature From Lattices. *IEEE Access* **2019**, *7*, 182916–182923. doi:10.1109/ACCESS.2019.2959943.
24. Ma, C.; Jiang, M. Practical Lattice-Based Multisignature Schemes for Blockchains. *IEEE Access* **2019**, *7*, 179765–179778. doi:10.1109/ACCESS.2019.2958816.
25. Toluee, R.; Eghlidos, T. An efficient and secure ID-based multi-proxy multi-signature scheme based on lattice. Cryptology ePrint Archive, Report 2019/1031. 2019. Available online: https://eprint.iacr.org/2019/1031 (accessed on 29 July 2021).
26. Fukumitsu, M.; Hasegawa, S. A Lattice-Based Provably Secure Multisignature Scheme in Quantum Random Oracle Model. In Proceedings of the Provable and Practical Security—14th International Conference, ProvSec 2020, Singapore, 29 November–1 December 2020; Volume 12505, pp. 45–64. doi:10.1007/978-3-030-62576-4_3.
27. Bendlin, R.; Krehbiel, S.; Peikert, C. How to Share a Lattice Trapdoor: Threshold Protocols for Signatures and (H)IBE. In Proceedings of the Applied Cryptography and Network Security—11th International Conference, ACNS 2013, Banff, AB, Canada, 25–28 June 2013; Volume 7954, pp. 218–236. doi:10.1007/978-3-642-38980-1_14.
28. Kansal, M.; Dutta, R. Round Optimal Secure Multisignature Schemes from Lattice with Public Key Aggregation and Signature Compression. In Proceedings of the Progress in Cryptology—AFRICACRYPT 2020—12th International Conference on Cryptology in Africa, Cairo, Egypt, 20–22 July 2020, Volume 12174, pp. 281–300. doi:10.1007/978-3-030-51938-4_14.
29. Ducas, L.; Lepoint, T.; Lyubashevsky, V.; Schwabe, P.; Seiler, G.; Stehle, D. CRYSTALS—Dilithium: Digital Signatures from Module Lattices. Cryptology ePrint Archive, Report 2017/633. 2017. Available online: https://eprint.iacr.org/2017/633 (accessed on 29 July 2021).
30. Pessl, P.; Bruinderink, L.G.; Yarom, Y. To BLISS-B or not to be: Attacking strongSwan's Implementation of Post-Quantum Signatures. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, 30 October–3 November 2017; pp. 1843–1855. doi:10.1145/3133956.3134023.
31. Bellare, M.; Neven, G. Multi-signatures in the plain public-key model and a general forking lemma. In Proceedings of the 13th ACM Conference on Computer and Communications Security—CCS'06, Alexandria, VA, USA, 30 October – 3 November 2006. doi:10.1145/1180405.1180453.
32. Fiat, A.; Shamir, A. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In Proceedings of the Advances in Cryptology—CRYPTO'86, Santa Barbara, CA, USA, 11–15 August 1986; Voloume 263, pp. 186–194. doi:10.1007/3-540-47721-7_12.
33. Lyubashevsky, V. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In Proceedings of the Advances in Cryptology—ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, 6–10 December 2009; Volume 5912, pp. 598–616.

34. Güneysu, T.; Lyubashevsky, V.; Pöppelmann, T. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2012—14th International Workshop, Leuven, Belgium, 9–12 September 2012; Volume 7428, pp. 530–547. doi:10.1007/978-3-642-33027-8_31.
35. Lyubashevsky, V.; Micciancio, D. Generalized Compact Knapsacks Are Collision Resistant. In Proceedings of the Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, 10–14 July 2006; Volume 4052, pp. 144–155. doi:10.1007/11787006_13.
36. Micciancio, D. Generalized Compact Knapsacks, Cyclic Lattices, and Efficient One-Way Functions from Worst-Case Complexity Assumptions. In Proceedings of the 43rd Symposium on Foundations of Computer Science (FOCS 2002), Vancouver, BC, Canada, 16–19 November 2002; pp. 356–365. doi:10.1109/SFCS.2002.1181960.
37. Peikert, C.; Rosen, A. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In Proceedings of the Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, 4–7 March 2006; Volume 3876, pp. 145–166. doi:10.1007/11681878_8.
38. Lyubashevsky, V.; Micciancio, D.; Peikert, C.; Rosen, A. SWIFFT: A Modest Proposal for FFT Hashing. In Proceedings of the Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, 10–13 February 2008, Volume 5086, pp. 54–72. doi:10.1007/978-3-540-71039-4_4.
39. Bellare, M.; Neven, G. New Multi-Signature Schemes and a General Forking Lemma. 2005. Available online: https://soc1024.ece.illinois.edu/teaching/ece498ac/fall2018/forkinglemma.pdf (accessed on 29 July 2021).
40. Boneh, D.; Gennaro, R.; Goldfeder, S.; Jain, A.; Kim, S.; Rasmussen, P.M.R.; Sahai, A. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In Proceedings of the Advances in Cryptology—CRYPTO 2018, Santa Barbara, CA, USA, 19–23 August 2018; pp. 565–596.
41. Gentry, C.; Peikert, C.; Vaikuntanathan, V. Trapdoors for Hard Lattices and New Cryptographic Constructions. Cryptology ePrint Archive, Report 2007/432. 2007. Available online: https://eprint.iacr.org/2007/432 (accessed on 29 July 2021).
42. Baum, C.; Damgård, I.; Lyubashevsky, V.; Oechsner, S.; Peikert, C. More Efficient Commitments from Structured Lattice Assumptions. In *Security and Cryptography for Networks*; Catalano, D., De Prisco, R., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 368–385.