DiLizium 2.0: Revisiting 2-out-of-2 threshold Dilithium*

Peeter Laud[1][0000−0002−9030−8142], Nikita Snetkov[1,2][0000−0002−1414−2080], and Jelizaveta Vakarjuk[1][0000−0001−6398−3663]

[1] Cybernetica AS, Mäealuse 2/1, 12618 Tallinn, Estonia
[2] Tallinn University of Technology, Akadeemia tee 15a, 12618 Tallinn, Estonia
{peeter.laud, nikita.snetkov, jelizaveta.vakarjuk}@cyber.ee

Abstract. In previous years there has been an increased interest in designing threshold signature schemes. Most of the recent works focus on constructing threshold versions of ECDSA or Schnorr signature schemes due to their appealing usage in blockchain technologies. Additionally, a lot of research is being done on cryptographic schemes that are resistant to quantum computer attacks.
In this work, we propose a new version of the two-party Dilithium signature scheme. The security of our scheme is based on the hardness of Module-LWE and Module-SIS problems. In our construction, we follow a similar logic as Damgård et al. (PKC 2021) and use an additively homomorphic commitment scheme. However, compared to them, our protocol uses signature compression techniques from the original Dilithium signature scheme which makes it closer to the version submitted to the NIST PQC competition. We focus on two-party signature schemes in the context of user authentication.

Keywords: public-key cryptography · threshold signatures · threshold cryptography · lattice-based cryptography · Fiat–Shamir with aborts · CRYSTALS-DILITHIUM.

1 Introduction

Threshold cryptography, particularly threshold signature schemes, has become a compelling research topic in recent years. Using these protocols, a distinct number of parties may transfer their joint right to generate a signature to any subset among themselves equal to or larger than a specific threshold. There are threshold variants of RSA [22,8], Schnorr [16,18], and ECDSA [19,10,6,1], which could be used in the blockchain or as an authentication solution. In this work, we focus on the latter use case where the private key is shared between the user's mobile device and a server. To create a signature for authentication, the mobile device and server should cooperate to create a single signature. The advantage of using threshold cryptography for authentication is that it helps to

*This research has been supported by European Regional Development Fund through the Estonian Centre of Excellence in ICT Research (EXCITE)

avoid reliance on trusted hardware to store the user's private key. If the private key is split between the mobile device and the server, an adversary who gets access to the mobile device will be unable to create valid signatures on behalf of the user without communicating with the server. This technology has been already successfully deployed in Baltic countries (having more than 3 million users), where the underlying signature scheme is a two-party RSA signature scheme [5].

In 1994, Peter Shor [21] described a quantum algorithm that allows solving factoring and discrete logarithm problems. When a powerful enough quantum computer will be constructed, the security of RSA, Schnorr, and ECDSA signature schemes and their threshold variants will be broken. To prevent information security systems from total breakdown, the National Institute of Standards and Technology (NIST) announced a Public-Key Post-Quantum Cryptographic (PQC) Algorithms competition in 2016. In this paper, we aim to construct a two-party signature scheme based on the Dilithium [11] signature, which is the primary post-quantum signature scheme that will be standardised by NIST. Our focus is to design a two-party signature scheme that would fit in the authentication use case to replace the currently used RSA.

1.1 Contribution

In this work, we construct a three-round two-party signature protocol based on Dilithium and we use an additively homomorphic commitment scheme from [3]. We use signature compression techniques from the Dilithium scheme; this leads to the problem of accommodating bit carry. We thus introduce an additional component to the final signature that we call *hint*. It is needed to adjust the value computed in the verification process taking a bit carry into account. We prove the security of our scheme in the classical random oracle model and show that the security of our scheme follows from the hardness of Module-SIS and Module-LWE problems. Additionally, we present results of implementation of the proposed scheme. We leave security proof in QROM for future work.

1.2 Related work

In this work we focus only on lattice-based threshold signature schemes, multisignature schemes such as Fukumitsu et al. [15] are out of scope of this paper.

Cozzo et al. [7] studied signature schemes that participated in the 2nd Round of the NIST PQC competition to develop threshold variants of those schemes. Among those signatures, there were three lattice-based schemes: Crystals-Dilithium, qTesla [2], and Falcon. By the authors' estimations of proposed threshold variants, it takes about 12 seconds to produce the Crystals-Dilithium signature; 16 seconds for qTesla signature; 6 seconds for Falcon signature. The main reason for such performance is that the authors applied only generic multiparty computation techniques, such as linear secret sharing and garbled circuits to construct threshold signature protocols with security against an active adversary.

Damgard et al. [9] proposed two n-out-of-n threshold signature schemes based on a Dilithium-G [12]. The first (DS_2) is a two-round threshold signature scheme with a trapdoor additively homomorphic commitment scheme. The second (DS_3) is a three-round threshold signature scheme with an additively homomorphic commitment scheme. Both commitment schemes are variants of the scheme proposed by Baum et al. [3]. However, the signature sizes in DS_2 and DS_3 protocols are bigger compared to the Crystals-Dilithium submitted to the NIST PQC competition, since those protocols do not utilize compression mechanisms.

Vakarjuk et al. [23] proposed a three-round 2-out-of-2 lattice-based signature scheme – DiLizium. The scheme is similar to the one proposed in [9] but uses a homomorphic hash function [20] instead of a homomorphic commitment scheme. This protocol is based on the scheme described in the paper by Kiltz et al., Appendix B [17]. One of the disadvantages of DiLizium is that its security proof relies on a non-standard rejected Module-LWE assumption introduced in [15]. The second, more serious problem, is that SWIFFT hash function is not additively homomorphic for all inputs.

Fu et al. [14] proposed a four-round two-party Crystals-Dilithium signature scheme using Fan et al. [13] fully homomorphic encryption scheme (FHE). Due to the high number of rounds and usage of FHE, the proposed protocol is less efficient compared to Damgard et al. [9] and Vakarjuk et al. [23]

1.3 Notation

Let R and R_q denote the rings $Z[x]/(x^n+1)$ and $Z_q[x]/(x^n+1)$, where $n \in \mathbb{N}$. We denote elements in R and R_q in italic lowercase letters p. We denote vectors with elements in R and R_q by bold lowercase $\mathbf{v}$ and matrices with elements in R and R_q by bold uppercase $\mathbf{A}$.

We follow the notation from [11] and use centered modular reduction $\mod^{\pm}\alpha$. For a positive integer α and for every $x \in \mathbb{Z}$, define $x' = x \mod^{\pm}\alpha$, as x' in the range $-\frac{\alpha}{2} < x' \leq \frac{\alpha}{2}$ when α is even and x' in the range $-\frac{\alpha-1}{2} \leq x' \leq \frac{\alpha-1}{2}$ when α is odd such that $x' \equiv x \pmod{\alpha}$.

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 $p \in R_q$, its infinity norm is defined as $\|p\|_\infty = \max_i \|p_i\|_\infty$ and its l_2 norm is defined as $\|p\|_2 = \sqrt{(\sum_i \|p_i\|_\infty^2)}$.

S_η denotes a set of all elements $p \in R_q$ 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. The symbol $\perp$ is used to indicate a failure or rejection.

1.4 Definitions

Definition 1 (Decisional Module-LWE). *Let χ be a probability distribution and $n, m \in \mathbb{Z}$. We define the advantage of adversary A in breaking decisional Module-LWE for the set of parameters (q, n, m, η, χ) as $\mathsf{Adv}^{\mathsf{D-MLWE}}_{(q,n,m,\eta,\chi)}(\mathsf{A}) := |P_0^{\mathsf{D-MLWE}} - P_1^{\mathsf{D-MLWE}}|$, where:*

$P_0^{\mathsf{D-MLWE}} = Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n \times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), \mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2, b \leftarrow \mathsf{A}(\mathbf{A}, \mathbf{t})]$
$P_1^{\mathsf{D-MLWE}} = Pr[b = 1 : \mathbf{A} \leftarrow R_q^{n \times m}, \mathbf{t} \leftarrow R_q^n, b \leftarrow \mathsf{A}(\mathbf{A}, \mathbf{t})]$.

Definition 2 (Computational Module-LWE). *Let χ be a probability distribution and $n, m \in \mathbb{Z}$. We define the advantage of adversary A in breaking computational Module-LWE for the set of parameters (q, n, m, η, χ) as follows:*
$\mathsf{Adv}^{\mathsf{C-MLWE}}_{(q,n,m,\eta,\chi)}(\mathsf{A}) := Pr[\mathbf{s}_1 = \mathbf{s}_1' : \mathbf{A} \leftarrow R_q^{n \times m}, (\mathbf{s}_1, \mathbf{s}_2) \leftarrow \chi(S_\eta^m \times S_\eta^n), \mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2, \mathbf{s}_1' \leftarrow \mathsf{A}(\mathbf{A}, \mathbf{t})]$.

Definition 3 (Module-SIS). *Let χ be a probability distribution and $n, m \in \mathbb{Z}$. We define the advantage of adversary A in breaking Module-SIS for the set of parameters (q, n, m, η) as follows:*
$\mathsf{Adv}^{\mathsf{MSIS}}_{(q,n,m,\eta)}(\mathsf{A}) := Pr[\big[\mathbf{A}|\mathbf{I}\big] \cdot \mathbf{x} = \mathbf{0}$ *and* $0 < \|\mathbf{x}\|_\infty \leq \eta : \mathbf{A} \leftarrow R_q^{n \times m}, \mathbf{x} \leftarrow \mathsf{A}(\mathbf{A})]$.

1.5 Commitment scheme

In this section, we present a description of an additively homomorphic commitment scheme that is used in our protocol. Definitions in this section are adapted from [9].

Definition 4 (Commitment scheme). *A commitment scheme consists of the following algorithms:*

- $\mathsf{ComSetup}(1^\lambda)$ *is an algorithm that takes as input security parameter λ and outputs a public set of parameters par that define set of commitment keys $\mathcal{K}$, message set $\mathcal{M}$, set of random elements $\mathcal{R}$ and set of commitments $\mathcal{C}$.*
- $\mathsf{ComKeyGen}(par)$ *is a key generation algorithm that takes as input set of parameters par and outputs a commitment key $ck \in \mathcal{K}$.*
- $\mathsf{Commit}_{ck}(m, r)$ *is an algorithm that takes as input a message $m \in \mathcal{M}$ and a randomness $r \in \mathcal{R}$ and outputs a commitment $c \in \mathcal{C}$.*
- $\mathsf{Open}_{ck}(m, r, c)$ *is an algorithm that outputs 1 if the input contains a valid commitment on a message m and outputs 0 otherwise.*

Definition 5 (Hiding). *We define the advantage of a probabilistic polynomial time adversary A in breaking the hiding property of the commitment scheme as $\mathsf{Adv}^{\mathsf{Hiding}}(\mathsf{A}) := |P_0^{\mathsf{Hiding}} - P_1^{\mathsf{Hiding}}|$, where:*
$P_0^{\mathsf{Hiding}} = Pr[b = 1 : par \leftarrow \mathsf{ComSetup}(1^\lambda), ck \leftarrow \mathsf{ComKeyGen}(par), m_0, m_1 \leftarrow \mathsf{A}(par, ck), c \leftarrow \mathsf{Commit}_{ck}(m_0), b \leftarrow \mathsf{A}(c)]$
$P_0^{\mathsf{Hiding}} = Pr[b = 1 : par \leftarrow \mathsf{ComSetup}(1^\lambda), ck \leftarrow \mathsf{ComKeyGen}(par), m_0, m_1 \leftarrow \mathsf{A}(par, ck), c \leftarrow \mathsf{Commit}_{ck}(m_1), b \leftarrow \mathsf{A}(c)]$

Definition 6 (Binding). *We define the advantage of a probabilistic polynomial time adversary A in breaking the binding property of the commitment scheme as follows:*
$\mathsf{Adv}^{\mathsf{Binding}}(\mathsf{A}) := Pr[m \neq m' \wedge \mathsf{Open}_{ck}(m, r, c) = 1 \wedge \mathsf{Open}_{ck}(m', r', c) = 1 : par \leftarrow \mathsf{ComSetup}(1^\lambda), ck \leftarrow \mathsf{ComKeyGen}(par), (m, r, c, m', r') \leftarrow \mathsf{A}(par, ck)]$.

Definition 7 (Uniform key). *A commitment scheme is called uniform if the output of the key generation algorithm* $\mathsf{ComKeyGen}(par)$ *is distributed uniformly over the set of commitment keys* $\mathcal{K}$*.*

Definition 8 (Min-entropy). *A commitment scheme is said to have at least* ξ*-bits of min-entropy if for all* $ck \in \mathcal{K}$ *and* $m \in \mathcal{M}$

$$\xi \leq -\log \max_{c \in \mathcal{C}} Pr[\mathsf{Commit}_{ck}(m, r) = c : r \leftarrow \mathcal{R}] \ .$$

Let $c \leftarrow \mathsf{Commit}_{ck}(m)$ (computed with r) and $c' \leftarrow \mathsf{Commit}_{ck}(m')$ (computed with r'). A commitment scheme is *additively homomorphic* if for any $m, m' \in \mathcal{M}$ it holds that $\mathsf{Open}_{ck}(c + c', m + m', r + r') = 1$.

Figure 1 presents an additively homomorphic commitment scheme that is used in our construction from Baum et al. [3]. At the beginning, the $\mathsf{ComSetup}(1^\lambda)$ algorithm is invoked that outputs set of parameters $par = (q, n, k, l, \beta, B)$.

$\mathsf{ComKeyGen}(par)$:

1. $\mathbf{A}'_1 \leftarrow R_q^{n \times (k-n)}$,
   $\mathbf{A}_1 = \begin{bmatrix} \mathbf{I}_n & \mathbf{A}'_1 \end{bmatrix} \in R_q^{n \times k}$
2. $\mathbf{A}'_2 \leftarrow R_q^{l \times (k-n-l)}$
   $\mathbf{A}_2 = \begin{bmatrix} \mathbf{0}^{l \times n} & \mathbf{I}_n & \mathbf{A}'_2 \end{bmatrix} \in R_q^{l \times k}$
3. **return** $ck := (\mathbf{A}_1, \mathbf{A}_2)$

$\mathsf{Commit}_{ck}(\mathbf{m} \in R_q^l, \mathbf{r} \leftarrow S_\beta^k)$:

1. $com := \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m} \end{bmatrix}$
2. **return** com

$\mathsf{Open}_{ck}(\mathbf{m}, com, \mathbf{r})$:

1. **if** $\begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0}^n \\ \mathbf{m} \end{bmatrix}$ and $\|\mathbf{r}\|_2 \leq B$ **return** 1
2. **else return** 0

Fig. 1. Commitment scheme from [3]

1.6 Dilithium

Dilithium is a lattice-based signature scheme that is constructed from identification protocol using Fiat-Shamir with aborts approach [11]. Dilithium uses supporting algorithms that extract high-order and low-order bits out of each coefficient of an element from the ring R_q. $\mathsf{Decompose}_q(\cdot)$ decomposes input r to $r = r_H \cdot \alpha + r_L$, such that $0 \leq r_H < \frac{(q-1)}{\alpha}$ and $\|r_L\|_\infty \leq \frac{\alpha}{2}$. To apply $\mathsf{Decompose}_q(\cdot)$ algorithm to an element (or vector of elements) from the ring R_q, one needs to apply $\mathsf{Decompose}_q(\cdot)$ on each coefficient separately.

Figure 2 presents a non-optimized version of Dilithium signature scheme [11] and supporting algorithms, on which the distributed signature protocol presented in this work is based. The challenge space $\mathcal{C} = \{c \in R_q : \|c\|_\infty = 1 \text{ and } \|c\|_2 = \sqrt{\tau}\}$ is parameterised by τ and consists of polynomials with small infinity norm. $\mathcal{C}$ is used as the image of the random oracle H_0.

$\mathsf{Decompose}_q(r, \alpha)$:
1. $r := r \mod q$
2. $r_L := r \mod^{\pm} \alpha$
3. **if** $r - r_L = q - 1$, **then** $r_H := 0, r_L := r_L - 1$
4. **else** $r_H := \frac{r - r_L}{\alpha}$
5. **return** (r_L, r_H)

$\mathsf{HighBits}_q(r, \alpha)$:
1. $(r_L, r_H) := \mathsf{Decompose}_q(r, \alpha)$
2. **return** r_H

$\mathsf{LowBits}_q(r, \alpha)$:
1. $(r_L, r_H) := \mathsf{Decompose}_q(r, \alpha)$
2. **return** r_L

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

$\mathsf{Sign}(sk, m)$:
1. $\mathbf{z} := \perp$
2. **while** $\mathbf{z} = \perp$ do:
   (a) $\mathbf{y} \leftarrow S_{\gamma-1}^l$
   (b) $\mathbf{w}_H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{y}, 2\gamma')$
   (c) $c := \mathsf{H}_0(m, \mathbf{w}_H) \in \mathcal{C}$
   (d) $\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$
   (e) **if** $\|\mathbf{z}\|_\infty \geq \gamma - \beta$ or $\|\mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma')\|_\infty \geq \gamma' - \beta$, then $\mathbf{z} := \perp$
3. **return** $\sigma = (\mathbf{z}, c)$

$\mathsf{Verify}(m, \sigma, pk)$:
1. $\mathbf{w}'_H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')$
2. **if** $\|\mathbf{z}\|_\infty < \gamma - \beta$ and $c = \mathsf{H}_0(m, \mathbf{w}'_H)$ **return** 1
3. **else return** 0

Fig. 2. Dilithium signature scheme

2 Our scheme

In this section, we introduce our two party signature scheme. Table 1 describes parameters for our signature scheme.

Parameter	Description
q	ring modulus
d	degree of polynomials
(k, l)	dimensions of public matrix $\mathbf{A}$
$(n_{com}, l_{com}, k_{com})$	dimensions of matrices in the commitment scheme
γ	size bound of the coefficients in the masking vector share; $\approx$ maximum coefficients of signature share
γ'	$2\gamma'\|(q-1)$; maximum coefficients of $\mathsf{LowBits}_q(\cdot)$ output
β	maximum coefficients of shares of $c\mathbf{s}_1$ and $c\mathbf{s}_2$
τ	number on nonzero elements in the output of hash function H_0
α	size bound of the coefficients in the random vector from the commitment scheme

Table 1. Description of parameters

Key Generation Before starting the key generation protocol, both parties invoke a $\mathsf{Setup}(1^\lambda)$ (with λ being a security parameter) algorithm that outputs a set of public parameters *par* that are defined in Table 1.

The first party's view of the key generation protocol is presented in Figure 3; the second party's actions are symmetric. The parties start with jointly generating $\mathbf{A}$. The parties sample their shares of the matrix and exchange hash commitments on their shares $\mathsf{H}_1(\mathbf{A}_i)$, this prevents a malicious party from choosing their share based on the share of the honest party. Upon receiving the commitment from the other party, they proceed by revealing matrix shares and verifying that the commitment was opened correctly. If the verification succeeds, parties derive a combined matrix $\mathbf{A}$. The next step consists of generating secret key shares $(\mathbf{s}_1^i, \mathbf{s}_2^i)$ and computing shares of a public vector $\mathbf{t}_i$. As in the previous step, parties exchange hash commitments $\mathsf{H}_2(\mathbf{t}_i)$, and only upon receiving a commitment from the other party, exchange vector shares $\mathbf{t}_i$. Parties proceed by verifying commitment opening and computing the second part of the public key $\mathbf{t}$.

Signing The formal definition of the signing protocol is presented in Figure 3. The first step of the signing protocol consists of generating a value that will be hashed to get a challenge $c \in \mathcal{C}$ in the underlying identification protocol. Parties cannot straightforwardly exchange their vectors $\mathbf{w}_i$ because of several reasons. Firstly, if $\mathbf{w}_i$ becomes revealed before the signature share gets rejected, $\mathbf{w}_i$ may leak some information about the secret key. Although there are no known attacks in the literature, the security proof of such a scheme would need to rely on a non-standard security assumption called Rejected Module-LWE, as was done in [15]. Secondly, if one of the parties knows $\mathbf{w}_i$ and $\mathbf{z}_i$ of the other party, they can extract $c\mathbf{s}_2^i$ from $\mathbf{z}_i$ and retrieve a part of the other party's secret key $\mathbf{s}_2^i$. Therefore, the parties exchange commitments $\mathbf{c}_i$, which are opened only if the signature shares pass the rejection sampling. Those commitments are aggregated using the homomorphic property of the commitment scheme, resulting value serves as input to the hash function H_0 to compute a challenge $c \in \mathcal{C}$. However, instead of exchanging commitments in the first communication rounds, the parties exchange a hash of their commitments $\mathsf{H}_3(\mathbf{c}_i)$. This step is analogous to the step in the key generation protocol, without it an adversary could adaptively choose a malicious $\mathbf{c}_i'$ after seeing the honest party's share. In the final communication round, the parties exchange their signature shares $\mathbf{z}_i$ together with the randomness $\mathbf{r}_i$ used to generate commitments. Finally, parties need to compute *hint* value $\mathbf{h}$ at Step 18 which helps to accommodate bit carry that can occur when adding high order bits of $\mathbf{w}_i$.

Verification The formal definition of the verification algorithm is presented in Figure 3. The verification algorithm in our scheme is different from the original Dilithium verification because we introduce additional components to the signature.

Hint correctness Let us denote $\alpha = 2\gamma'$. Given a number $x \in Z_q$, we can write it $x \equiv x^H \cdot \alpha + x^L \pmod{q}$, where

$$(x^H, x^L) \in \left\{0, \ldots, \frac{q-1}{\alpha} - 1\right\} \times \left\{-\frac{\alpha}{2} + 1, \ldots, \frac{\alpha}{2}\right\} \cup \left\{\left(0, -\frac{\alpha}{2}\right)\right\}$$

$\mathsf{KeyGen}_{P_1}(par)$:

1. $\mathbf{A}_1 \leftarrow R_q^{k \times l}$
2. $\longrightarrow P_2 : hk_1 := \mathsf{H}_1(\mathbf{A}_1)$
   $\longleftarrow P_2 : hk_2 := \mathsf{H}_1(\mathbf{A}_2)$
3. $\longrightarrow P_2 : \mathbf{A}_1$
   $\longleftarrow P_2 : \mathbf{A}_2$
4. **if** $\mathsf{H}_1(\mathbf{A}_2) \neq hk_2$,
   send out ABORT message
5. $\mathbf{A} := \mathbf{A}_1 + \mathbf{A}_2$
6. $(\mathbf{s}_1^1, \mathbf{s}_2^1) \leftarrow S_\eta^l \times S_\eta^k$
7. $\mathbf{t}_1 := \mathbf{A}\mathbf{s}_1^1 + \mathbf{s}_2^1$
8. $\longrightarrow P_2 : comk_1 := \mathsf{H}_2(\mathbf{t}_1)$
   $\longleftarrow P_2 : comk_2 := \mathsf{H}_2(\mathbf{t}_2)$
9. $\longrightarrow P_2 : \mathbf{t}_1$
   $\longleftarrow P_2 : \mathbf{t}_2$
10. **if** $\mathsf{H}_2(\mathbf{t}_2) \neq comk_2$,
    send out ABORT message
11. $\mathbf{t} := \mathbf{t}_1 + \mathbf{t}_2$
12. **output:** $sk_1 = (\mathbf{A}, \mathbf{t}_2, \mathbf{s}_1^1, \mathbf{s}_2^1)$,
    $pk = (\mathbf{A}, \mathbf{t})$

$\mathsf{Hint}(\mathbf{r}, \mathbf{r}', \alpha)$:

1. $\mathbf{h} := \mathbf{r} - \mathbf{r}'$
2. $\mathbf{h}_1 := \lfloor \frac{\mathbf{h}}{\alpha - 1} \rfloor$
3. $\mathbf{h}_2 := \mathbf{h} \mod^{\pm} \frac{\alpha}{2}$
4. **return** $(\mathbf{h}_1, \mathbf{h}_2)$

$\mathsf{UseHint}(\mathbf{r}, \mathbf{h}_1, \mathbf{h}_2, \alpha)$:

1. $\mathbf{h} := \mathbf{h}_1 \cdot \alpha + \mathbf{h}_2$
2. $\mathbf{r}' := \mathbf{r} - \mathbf{h}$
3. **return** $\mathbf{r}'$

$\mathsf{Sign}_{P_1}(sk_1, m)$

1. $ck \leftarrow \mathsf{H}_4(m, pk)$
2. $\mathbf{y}_1 \leftarrow S_{\gamma-1}^l, \quad \mathbf{w}_1 := \mathbf{A}\mathbf{y}_1$
3. $\mathbf{w}_1^H := \mathsf{HighBits}_q(\mathbf{w}_1, 2\gamma')$
4. $\mathbf{r}_1 \leftarrow S_\alpha^\kappa$
5. $\mathbf{c}_1 := \mathsf{Commit}_{ck}(\mathbf{w}_1^H, \mathbf{r}_1)$
6. $\longrightarrow P_2 : h_1 := \mathsf{H}_3(\mathbf{c}_1)$
   $\longleftarrow P_2 : h_2 := \mathsf{H}_3(\mathbf{c}_2)$
7. $\longrightarrow P_2 : \mathbf{c}_1$
   $\longleftarrow P_2 : \mathbf{c}_2$
8. **if** $\mathsf{H}_3(\mathbf{c}_2) \neq h_2$,
   send out ABORT message
9. $\mathbf{c} := \mathbf{c}_1 + \mathbf{c}_2, \quad c := \mathsf{H}_0(m, \mathbf{c}, pk)$
10. $\mathbf{z}_1 := \mathbf{y}_1 + c\mathbf{s}_1^1$
11. **if** $\|\mathbf{z}_1\|_\infty \geq \gamma - \beta$
    or $\|\mathsf{LowBits}_q(\mathbf{w}_1 - c\mathbf{s}_2^1, 2\gamma')\|_\infty \geq \gamma' - \beta$,
    send out RESTART message
12. $\longrightarrow P_2 : (\mathbf{z}_1, \mathbf{r}_1)$
    $\longleftarrow P_2 : (\mathbf{z}_2, \mathbf{r}_2)$
13. $\mathbf{w}_2^H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z}_2 - c\mathbf{t}_2, 2\gamma')$
14. **if** $\mathsf{Open}_{ck}(\mathbf{c}_2, \mathbf{w}_2^H, \mathbf{r}_2) \neq 1$,
    send out ABORT message
15. $\mathbf{z} := \mathbf{z}_1 + \mathbf{z}_2, \quad \mathbf{r} := \mathbf{r}_1 + \mathbf{r}_2$
16. $\widehat{\mathbf{w}^H} = \mathbf{w}_1^H + \mathbf{w}_2^H$
17. **if** $\|\mathsf{LowBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')\|_\infty \geq \gamma' - 2\beta$,
    send out RESTART message
18. $\mathbf{w}^H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')$,
19. $\mathbf{h} := (\mathbf{h}_1, \mathbf{h}_2) = \mathsf{Hint}(\widehat{\mathbf{w}^H}, \mathbf{w}^H, \frac{q-1}{2\gamma'})$
20. **output**: $\sigma = (\mathbf{z}, \mathbf{c}, \mathbf{r}, \mathbf{h})$

$\mathsf{Verify}(m, \sigma, pk)$:

1. Derive a commitment key $ck := \mathsf{H}_4(m, pk)$
2. Derive a challenge $c := \mathsf{H}_0(m, \mathbf{c}, pk)$
3. $\mathbf{w}^H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')$
4. $\widehat{\mathbf{w}^H} := \mathsf{UseHint}(\mathbf{w}^H, \mathbf{h}_1, \mathbf{h}_2, \frac{q-1}{2\gamma'})$
5. **if** $\mathsf{Open}_{ck}(\mathbf{c}, \widehat{\mathbf{w}^H}, \mathbf{r}) = 1$ and $\|\mathbf{z}\|_\infty < 2(\gamma - \beta)$ **return** 1
6. **else return** 0

Fig. 3. Specification of our two-party signature scheme

and this choice of (x^H, x^L) is unique. The values x^H and x^L are the *high bits* and *low bits* of x. Let $x_1, x_2 \in Z_q$. We want to find the possible values of $h := (x_1^H + x_2^H) - ((x_1 + x_2) \bmod q)^H$. We have

$$((x_1 + x_2) \bmod q)^H = (((x_1^H + x_2^H) \cdot \alpha + x_1^L + x_2^L) \bmod q)^H;$$

Consider the following cases. In the following we assume that $x_1, x_2 \in \{0, \ldots, q-1\}$.

- $x_1 \geq q - \frac{\alpha}{2}$, $x_2 \geq q - \frac{\alpha}{2}$. In this case, $x_1^H = x_2^H = 0$, and $(x_1 + x_2)^H \in \{\frac{q-1}{\alpha} - 1, 0\}$. Hence $h \in \{0, -\frac{q-1}{\alpha} + 1\}$.
- $x_1 \geq q - \frac{\alpha}{2}$, $x_2 < q - \frac{\alpha}{2}$. In this case, $x_1^H = 0$. The value $(x_1 + x_2)^H$ is either equal to x_2^H, or to $x_2^H - 1$; it cannot shift by more than that, when adding x_1 to x_2. Hence $h \in \{0, 1\}$.
- $x_1 < q - \frac{\alpha}{2}$, $x_2 \geq q - \frac{\alpha}{2}$. This is symmetric to previous case.
- $x_1 < q - \frac{\alpha}{2}$, $x_2 < q - \frac{\alpha}{2}$, $x_1 + x_2 < q - \frac{\alpha}{2}$. Then $h \in \{-1, 0, 1\}$, depending on the segment into which $x_1^L + x_2^L$ falls. If $x_1^L + x_2^L \in [\frac{\alpha}{2} + 1, \alpha]$, then $h = -1$. If $x_1^L + x_2^L \in [-\alpha + 2, -\frac{\alpha}{2}]$, then $h = 1$. If $x_1^L + x_2^L$ falls somewhere in the middle, then $h = 0$.
- $x_1 < q - \frac{\alpha}{2}$, $x_2 < q - \frac{\alpha}{2}$, $x_1 + x_2 \geq q - \frac{\alpha}{2}$. In this case, $(x_1 + x_2)^H$ "rolls over", the roll-over is by $\frac{q-1}{\alpha}$. We get $h \in \{\frac{q-1}{\alpha} - 1, \frac{q-1}{\alpha}, \frac{q-1}{\alpha} + 1\}$.

Hence there are **seven** possible values for h:

$$h \in \{-\frac{q-1}{\alpha} + 1, -1, 0, 1, \frac{q-1}{\alpha} - 1, \frac{q-1}{\alpha}, \frac{q-1}{\alpha} + 1\} \ .$$

When we calculate hint, for each integer coefficient h of $\mathbf{h} := \widehat{\mathbf{w}^H} - \mathbf{w}^H$ we produce two values:

1. $h_1 := \frac{h}{(\frac{q-1}{\alpha} - 1)} = \begin{cases} -1 & \text{for } h \in \{-\frac{q-1}{\alpha} + 1\} \\ 0 & \text{for } h \in \{-1, 0, 1\} \\ 1 & \text{for } h \in \{\frac{q-1}{\alpha} - 1, \frac{q-1}{\alpha}, \frac{q-1}{\alpha} + 1\} \end{cases}$. This component indicates whether the roll-out by $\frac{q-1}{\alpha}$ happened and whether it results in negative or positive number.
2. $h_2 := h \mod^{\pm} \frac{q-1}{2\alpha} = \begin{cases} -1 & \text{for } h \in \{\frac{q-1}{\alpha} - 1, 1\} \\ 0 & \text{for } h \in \{0, \frac{q-1}{\alpha}\} \\ 1 & \text{for } h \in \{-\frac{q-1}{\alpha} + 1, 1, \frac{q-1}{\alpha} + 1\} \end{cases}$. This component indicates whether the bit carry happened.

When we use the hint, we firstly scale component h_1 by $\frac{q-1}{\alpha}$ to perform roll-out for those coefficients where it is needed. And then we add component h_2 which accommodates bit carry. Therefore we get back all the seven possible values of h.

Thus, it holds that

$$\mathsf{UseHint}(\mathbf{w}^H, \mathsf{Hint}(\mathbf{w}^H, \widehat{\mathbf{w}^H}, \frac{q-1}{\alpha}), \frac{q-1}{\alpha}) = \widehat{\mathbf{w}^H} = \mathbf{w}_1^H + \mathbf{w}_2^H. \tag{1}$$

Signature scheme correctness Let us examine two verification conditions separately. The first check is $\mathsf{Open}_{ck}(\mathbf{c}, \widehat{\mathbf{w}^H}, \mathbf{r}) = 1$, where

$$\mathbf{c} = \mathbf{c}_1 + \mathbf{c}_2 = \mathsf{Commit}_{ck}(\mathbf{w}_1^H, \mathbf{r}_1) + \mathsf{Commit}_{ck}(\mathbf{w}_2^H, \mathbf{r}_2) = \mathsf{Commit}_{ck}(\mathbf{w}_1^H + \mathbf{w}_2^H, \mathbf{r}_1 + \mathbf{r}_2)$$

$$\widehat{\mathbf{w}^H} = \mathsf{UseHint}(\mathbf{w}^H, \mathbf{h}_1, \mathbf{h}_2, \frac{q-1}{2\gamma'}) = \mathsf{UseHint}(\mathbf{w}^H, \mathsf{Hint}(\mathbf{w}^H, \widehat{\mathbf{w}^H}, \frac{q-1}{2\gamma'}), \frac{q-1}{2\gamma'})$$
$$\overset{\text{equation 1}}{=} \mathbf{w}_1^H + \mathbf{w}_2^H$$

Therefore, we have $\mathsf{Open}_{ck}(\mathsf{Commit}_{ck}(\mathbf{w}_1^H + \mathbf{w}_2^H, \mathbf{r}_1 + \mathbf{r}_2), \mathbf{w}_1^H + \mathbf{w}_2^H, \mathbf{r}_1 + \mathbf{r}_2) = 1$.

Second verification condition is $\|\mathbf{z}\|_\infty < 2(\gamma - \beta)$. As all the rejection sampling steps have been successfully passed, it holds that $\|\mathbf{z}_i\|_\infty < \gamma - \beta$. It follows that $\|\mathbf{z}\|_\infty = \|\mathbf{z}_1 + \mathbf{z}_2\|_\infty \leq \|\mathbf{z}_1\|_\infty + \|\mathbf{z}_2\|_\infty < \gamma - \beta + \gamma - \beta = 2(\gamma - \beta)$.

Finally, note that $\mathbf{w}_i^H = \mathsf{HighBits}_q(\mathbf{A}\mathbf{z}_i - c\mathbf{t}_i, 2\gamma') = \mathsf{HighBits}_q(\mathbf{w}_i - c\mathbf{s}i_2, 2\gamma') \overset{\text{lemma 1}}{=} \mathsf{HighBits}_q(\mathbf{w}_i, 2\gamma')$.

3 Security

Definition 9 (Existential Unforgeability under Chosen Message Attack). *Distributed signature protocol is Existentially Unforgeable under Chosen Message Attack (DS-UF-CMA) if for any probabilistic polynomial time adversary* A*, its advantage of creating a successful signature forgery is negligible. The advantage of* A *is defined as a probability of winning in the experiment* $\mathsf{Exp}^{\mathsf{DS-UF-CMA}}$ *given in Fig. 4:*

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

$\mathsf{Exp}^{\mathsf{DS-UF-CMA}}(\mathsf{A})$:
1. $\mathcal{M} \leftarrow \emptyset$
2. $kgen := false$
3. $par \leftarrow \mathsf{Setup}(1^\lambda)$
4. $(m^*, \sigma^*) \leftarrow \mathsf{A}^{\mathsf{DS}_n(\cdot)}(par)$
5. $b \leftarrow \mathsf{Verify}(m^*, \sigma^*, pk)$
6. **if** $b = 1$ and $m^* \notin \mathcal{M}$: **return** 1
7. **else return** 0

Fig. 4. Experiment $\mathsf{Exp}^{\mathsf{DS-UF-CMA}}(\mathsf{A})$

The main idea of our security proof relies on a similar proof from [9], we show that given an adversary that succeeds in creating a valid forgery with non-negligible probability one can break the computational binding of the commitment scheme or Module-SIS assumption.

Theorem 1. *Assume a commitment scheme is computationally binding, computationally hiding, uniform, additively homomorphic, and has* ξ*-bit min-entropy. Then for any probabilistic polynomial time adversary* 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* $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{H}_4$*, the distributed signature protocol is DS-UF-CMA secure in the random oracle model under decisional Module-LWE assumption for parameters* (q, k, l, η, U)*, and Module-SIS assumptions for parameters* $(q, k, l+1, 4(\gamma - \beta))$*.*

Proof. Let us have an adversary A against distributed signature protocol. We construct an algorithm S around A that simulates the behaviour of an honest party in the protocol and does not use an actual secret key of the honest party. S will use instructions from $\mathsf{SimKeyGen}(\cdot)$ and $\mathsf{SimSign}(\cdot)$ oracles defined in Figure 6 and Figure 7 respectively. We construct $\mathsf{SimKeyGen}(\cdot)$ and $\mathsf{SimSign}(\cdot)$ through the sequence of intermediate games and for each game, we evaluate its difference from the previous one starting from $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Sign}(\cdot)$ defined in Figure 3. Each intermediate game is detailed in the full version.

- **Game 0**: S interacts with A using instructions of $\mathsf{KeyGen}(\cdot)$ and $\mathsf{Sign}(\cdot)$.
- **Game 1**: we only change the signing process of S with respect to the previous game. The challenge c is sampled uniformly at random from the set C. S calculates signature share $\mathbf{z}_n$ without communicating with A. Upon receiving h_i, S runs $\mathsf{SearchHash}(\mathsf{HT}_3, h_i)$ to find a preimage $\mathbf{c}_i$ and calculates $\mathbf{c} = \mathbf{c}_n + \mathbf{c}_i$. Finally, S programs random oracle H_0 to respond the query $(m, \mathbf{c}, pk)$ with the value c.
- **Game 2**: we make the following changes to the signing process. If $\|\mathbf{z}_1\|_\infty \geq \gamma - \beta$, then sample $\mathbf{w}_n \leftarrow R_q^k$. Else define $\mathbf{w}_n := \mathbf{A}\mathbf{y}_n$ with $\mathbf{y}_n \leftarrow S_{\gamma-1}^l$. S proceeds as before by committing to high order bits of $\mathbf{w}_n$ and sending hash of corresponding commitment $\mathsf{H}_3(\mathbf{c}_n)$ to A, where $\mathbf{c}_n := \mathsf{Commit}_{ck}(\mathbf{w}_n^H, \mathbf{r}_n)$ and $\mathbf{r}_n \leftarrow S_\alpha^\kappa$.
- **Game 3**: S does not generate signature share $\mathbf{z}_n$ and does not perform first rejection sampling check (whether $\|\mathbf{z}_1\|_\infty < \gamma - \beta$). Instead S simulates rejection sampling. With probability $1 - \frac{|S_{\gamma-\beta-1}^k|}{|S_{\gamma-1}^l|}$, sample $\mathbf{w}_n \leftarrow R_q^k$. With probability $\frac{|S_{\gamma-\beta-1}^k|}{|S_{\gamma-1}^l|}$, sample $\mathbf{z}_n \leftarrow S_{\gamma-\beta-1}^l$ and define $\mathbf{w}_n := \mathbf{A}\mathbf{z}_n - c(\mathbf{t}_n - \mathbf{s}_2^n)$.
- **Game 4**: we completely remove the usage of the secret key from the signing process. Therefore, in case of no rejection (with probability $\frac{|S_{\gamma-\beta-1}^k|}{|S_{\gamma-1}^l|}$), $\mathbf{w}'_n := \mathbf{A}\mathbf{z}_n - c\mathbf{t}_n$.
- **Game 5**: we start changing the key generation process. S is given random $\mathbf{A}$, S samples $hk_n \leftarrow \{0,1\}^{l_1}$ and sends it to A. Upon receiving hk_i, S runs $\mathsf{SearchHash}(\mathsf{HT}_1, hk_i)$ to find a preimage $\mathbf{A}_i$ and calculates $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$. Finally, S programs random oracle H_1 to respond the query $\mathbf{A}_n$ with hk_n.
- **Game 6**: we continue changing the key generation process. S samples $\mathbf{t}_n \leftarrow R_q^k$, instead of computing $\mathbf{t}_n := \mathbf{A}\mathbf{s}_1^n + \mathbf{s}_2^n$.
- **Game 7**: S gets as an input a random final public key $\mathbf{t} \in R_q^k$, S samples $comk_n \leftarrow \{0,1\}^{l_2}$ and sends it to A. Upon receiving $comk_i$ from A, S runs $\mathsf{SearchHash}(\mathsf{HT}_2, comk_i)$ to find a preimage $\mathbf{t}_i$ and calculates $\mathbf{t}_n := \mathbf{t} - \mathbf{t}_i$. Finally, S programs random oracle H_2 to respond the query $\mathbf{t}_n$ with $comk_n$.

Our next step is to embed an instance of Module-SIS $[\mathbf{A}'|\mathbf{I}]$ and a challenge commitment key $ck^\star \leftarrow \mathsf{ComKeyGen}(par)$ to our proof. The final description of algorithm S is given in Figure 5. We define an input generation algorithm IG for the forking lemma (Lemma 3) such that it outputs a tuple $(\mathbf{A}, \mathbf{t}, ck^\star)$. We proceed by constructing an algorithm B around S that either breaks the binding

of the commitment scheme with respect to $ck^\star$ or finds a solution to Module-SIS on input $\mathbf{A}' = [\mathbf{A}|\mathbf{t}]$.

Random oracle simulation:

- $\mathsf{H}_0(x)$: Parse input as $(m, \mathbf{c}, pk)$; Make query $\mathsf{H}_4(m, pk)$; If $\mathsf{HT}_0[\mathbf{c}, m, pk]$ is not set, increment counter ctr and set $\mathsf{HT}_0[\mathbf{c}, m, pk] := h_{ctr}$; Return $\mathsf{HT}_0[\mathbf{c}, m, pk]$.
- $\mathsf{H}_1(x)$: If $\mathsf{HT}_1[x]$ is not set, set $\mathsf{HT}_1[x] \leftarrow \{0,1\}^{l_1}$; Return $\mathsf{HT}_1[x]$.
- $\mathsf{H}_2(x)$: If $\mathsf{HT}_2[x]$ is not set, set $\mathsf{HT}_2[x] \leftarrow \{0,1\}^{l_2}$; Return $\mathsf{HT}_2[x]$.
- $\mathsf{H}_3(x)$: If $\mathsf{HT}_3[x]$ is not set, set $\mathsf{HT}_3[x] \leftarrow \{0,1\}^{l_3}$; Return $\mathsf{HT}_3[x]$.
- $\mathsf{H}_4(x)$: Parse input as (m, pk); If $\mathsf{HT}_4[m, pk]$ is not set, with probability ω set $\mathsf{HT}_4[m, pk] \leftarrow \mathcal{K}$, with probability $(1-\omega)$, set $\mathsf{HT}_4[m, pk] := ck^\star$; Return $\mathsf{HT}_4[m, pk]$.

$\mathsf{SearchHash}(\mathsf{HT}, h)$:

1. For value h, find its preimage m in the hash table such that $\mathsf{HT}[m] = h$.
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:** $(m, alert, bad)$

Honest party oracle simulation: A makes a query to the honest party oracle. S interacts with A using instructions defined in the Figure **??**, but instead of invoking $\mathsf{KeyGen}_{P_n}(\cdot)$ and $\mathsf{Sign}_{P_n}(\cdot)$ protocols, S invokes $\mathsf{SimKeyGen}(\cdot)$ (Figure 6) and $\mathsf{SimSign}(\cdot)$ (Figure 7) respectively.

Forgery: A outputs a forgery $(m^*, \sigma^* = (\mathbf{z}^*, \mathbf{c}^*, \mathbf{r}^*, \mathbf{h}^*))$. Given a forgery, S proceeds follows:

1. If $m^* \notin \mathcal{M}$, return $(0, \perp)$.
2. Derive $ck^* \leftarrow \mathsf{H}_4(m^*, pk)$ and $c^* \leftarrow \mathsf{H}_0(m^*, \mathbf{c}^*, pk)$. Compute $\widehat{\mathbf{w}^H} = \mathsf{HighBits}_q(\mathbf{A}\mathbf{z}^* - c^*\mathbf{t}, \gamma') - \mathbf{h}^*$.
3. If $\mathsf{Open}_{ck}(\mathbf{c}^*, \widehat{\mathbf{w}^H}, \mathbf{r}^*) \neq 1$ or $\|\mathbf{z}^*\|_\infty \geq 2(\gamma - \beta)$, return $(0, \perp)$. If $\mathsf{HT}_4[m^*, pk] \neq ck^\star$, return $(0, \perp)$.
4. Find $i_f \in [Q_h + Q_s + 1]$ such that $c^* = h_{i_f}$ and return $(i_f, out = (\mathbf{z}^*, \mathbf{c}^*, \mathbf{r}^*, \mathbf{h}^*, c^*, m^*, ck^*))$

Fig. 5. Final description of the algorithm S

Algorithm B invokes the forking algorithm F (Figure 8) on input $(\mathbf{A}, \mathbf{t}, ck^\star)$. With probability frk we obtain two valid forgeries $out = (\mathbf{z}^*, \mathbf{c}^*, \mathbf{r}^*, \mathbf{h}^*, c^*, m^*, ck^*)$ and $out' = (\mathbf{z}', \mathbf{c}', \mathbf{r}', \mathbf{h}', c', m', ck')$.

By the construction of the forking algorithm, it holds that all the values generated before the fork are the same in both forgeries: $\mathbf{c}^* = \mathbf{c}'$, $m^* = m'$ and $ck^* = ck' = ck^\star$. We also know that $c^* \neq c'$ by the definition of the forking algorithm. Therefore, it holds that $\mathsf{Open}_{ck^*}(\mathbf{c}^*, \widehat{\mathbf{w}^{H^*}}, \mathbf{r}^*) = \mathsf{Open}_{ck^*}(\mathbf{c}^*, \widehat{\mathbf{w}^{H'}}, \mathbf{r}') = 1$, where

$$\widehat{\mathbf{w}^{H^*}} = \mathsf{UseHint}(\mathbf{w}^{H*}, \mathbf{h}_1^*, \mathbf{h}_2^*, \frac{q-1}{2\gamma'}) = \mathbf{w}_1^{H*} + \mathbf{w}_2^{H*}$$

$\mathsf{SimKeyGen}(par, \mathbf{A}, \mathbf{t})$:

1. Sample $hk_n \leftarrow \{0,1\}^{l_1}$, send out hk_n
2. Upon receiving hk_i:
   (a) $(alert, bad_4, \mathbf{A}_i) \leftarrow \mathsf{SearchHash}(\mathsf{HT}_1, hk_i)$.
   (b) if bad_4 is set, return $(0, \perp)$.
   (c) if $alert$ is set, sample $\mathbf{A}_n \leftarrow R_q^{k \times l}$. Otherwise, define $\mathbf{A}_n := \mathbf{A} - \mathbf{A}_i$.
   (d) Set $\mathsf{HT}_1[\mathbf{A}_n] := hk_n$. If $\mathsf{HT}_1[\mathbf{A}_n]$ has been already set, then set the flag bad_5 and return $(0, \perp)$.
   (e) Send out $\mathbf{A}_n$.
3. Upon receiving $\mathbf{A}_i$:
   (a) if $\mathsf{H}_1(\mathbf{A}_i) \neq hk_i$, send out ABORT.
   (b) if $alert$ is set and $\mathsf{H}_1(\mathbf{A}_i) = hk_i$, set the flag bad_6 and return $(0, \perp)$.
4. Sample $comk_n \leftarrow \{0,1\}^{l_2}$, send out $comk_n$
5. Upon receiving $comk_i$:
   (a) $(alert, bad_7, \mathbf{t}_i) \leftarrow \mathsf{SearchHash}(\mathsf{HT}_2, comk_i)$.
   (b) if bad_7 is set, return $(0, \perp)$.
   (c) if $alert$ is set, sample $\mathbf{t}_n \leftarrow R_q^k$. Otherwise, define $\mathbf{t}_n := \mathbf{t} - \mathbf{A}_i$.
   (d) Set $\mathsf{HT}_2[\mathbf{t}_n] := comk_n$. If $\mathsf{HT}_2[\mathbf{t}_n]$ has been already set, then set the flag bad_8 and return $(0, \perp)$.
   (e) Send out $\mathbf{A}_n$.
6. Upon receiving $\mathbf{t}_i$:
   (a) if $\mathsf{H}_2(\mathbf{t}_i) \neq comk_i$, send out ABORT.
   (b) if $alert$ is set and $\mathsf{H}_2(\mathbf{t}_i) = comk_i$, set the flag bad_9 and return $(0, \perp)$.

Fig. 6. Simulator for the key generation protocol

$$\widehat{\mathbf{w}^{H'}} = \mathsf{UseHint}(\mathbf{w}^{H'}, \mathbf{h}'_1, \mathbf{h}'_2, \frac{q-1}{2\gamma'}) = \mathbf{w}_1^{H'} + \mathbf{w}_2^{H'}$$

There are two cases – $\widehat{\mathbf{w}^{H^*}} \neq \widehat{\mathbf{w}^{H'}}$ (Case 1) and $\widehat{\mathbf{w}^{H^*}} = \widehat{\mathbf{w}^{H'}}$ (Case 2). From the first case it follows that A has found two valid openings for the commitment $\mathbf{c}^*$ under the same commitment key $ck^\star$. This means that A has broken the binding property of the commitment scheme.

In the second case, we have $\mathbf{w}_1^{H*} + \mathbf{w}_2^{H*} = \mathbf{w}_1^{H'} + \mathbf{w}_2^{H'}$. We can rewrite it as follows

$(\mathbf{A}\mathbf{z}_1^* - c^*\mathbf{t}_1 + \mathbf{x}_1^*) + (\mathbf{A}\mathbf{z}_2^* - c^*\mathbf{t}_2 + \mathbf{x}_2^*) = (\mathbf{A}\mathbf{z}'_1 - c'\mathbf{t}_1 + \mathbf{x}'_1) + (\mathbf{A}\mathbf{z}'_2 - c'\mathbf{t}_2 + \mathbf{x}'_2)$, where for each $\mathbf{x} \in \{\mathbf{x}_1^*, \mathbf{x}_2^*, \mathbf{x}'_1, \mathbf{x}'_2\}$ it holds that $\|\mathbf{x}\|_\infty < \gamma' - \beta$. After rearranging the equation above we get

$\mathbf{A}(\mathbf{z}_1^* + \mathbf{z}_2^*) - c^*(\mathbf{t}_1 + \mathbf{t}_2) + (\mathbf{x}_1^* + \mathbf{x}_2^*) = \mathbf{A}(\mathbf{z}'_1 + \mathbf{z}'_2) - c'(\mathbf{t}_1 + \mathbf{t}_2) + (\mathbf{x}'_1 + \mathbf{x}'_2)$

$\Rightarrow \mathbf{A}\mathbf{z}^* - c^*\mathbf{t} + \mathbf{x}^* = \mathbf{A}\mathbf{z}' - c'\mathbf{t} + \mathbf{x}' \Rightarrow \mathbf{A}(\mathbf{z}^* - \mathbf{z}') + \mathbf{t}(c' - c^*) + (\mathbf{x}^* - \mathbf{x}') = 0$.

Finally we obtain the following equation

$$[\mathbf{A}|\mathbf{t}|\mathbf{I}] \cdot \begin{bmatrix} \mathbf{z}^* - \mathbf{z}' \\ c' - c^* \\ \mathbf{x}^* - \mathbf{x}' \end{bmatrix} = 0, \tag{2}$$

where $[\mathbf{A}|\mathbf{t}|\mathbf{I}]$ is an instance of Module-SIS problem, $\|\mathbf{z}^* - \mathbf{z}'\|_\infty \leq 4(\gamma - \beta)$, $\|\mathbf{x}^* - \mathbf{x}'\|_\infty \leq 4(\gamma' - \beta)$ and $\|c' - c^*\|_\infty = 2$. It means that A has found a solution for Module-SIS problem with parameters $(q, k, l+1, \delta)$, where $\delta = \|\mathbf{z}^* - \mathbf{z}'\|_\infty$.

$\mathsf{SimSign}(\mathbf{t}_n, pk, m)$:

1. $c \leftarrow \mathcal{C}$
2. With probability $\frac{|S^k_{\gamma-\beta-1}|}{|S^l_{\gamma-1}|}$ sample $\mathbf{z}_n \leftarrow S^l_{\gamma-\beta-1}$ and set $\mathbf{w}_n := \mathbf{A}\mathbf{z}_n - c\mathbf{t}_n$. Otherwise set $\mathbf{w}_n \leftarrow R^k_q$
3. $\mathbf{w}^H_n := \mathsf{HighBits}_q(\mathbf{w}_n, 2\gamma')$
4. $\mathbf{r}_n \leftarrow S^\kappa_\alpha$ and $\mathbf{c}_n := \mathsf{Commit}_{ck}(\mathbf{w}^H_n, \mathbf{r}_n)$, <u>send out</u> $h_n := \mathsf{H}_3(\mathbf{c}_n)$
5. Upon receiving h_i:
   - $(\mathbf{c}_i, alert, bad_1) \leftarrow \mathsf{SearchHash}(\mathsf{HT}_3, h_i)$
   - if bad_1 is set, return $(0, \perp)$
   - if $alert$ is set, then <u>send out</u> $\mathbf{c}_n$
   - else $\mathbf{c} = \mathbf{c}_n + \mathbf{c}_i$. Set $\mathsf{HT}_0[m, \mathbf{c}, pk] := c$. If $\mathsf{HT}_0[m, \mathbf{c}, pk]$ has been already set, then set the flag bad_2 and return $(0, \perp)$. <u>Send out</u> $\mathbf{c}_n$.
6. Upon receiving $\mathbf{c}_i$:
   (a) if $\mathsf{H}_3(\mathbf{c}_i) \neq h_i$, send out ABORT
   (b) if the flag $alert$ is set and $\mathsf{H}_3(\mathbf{c}_i) = h_i$, set the flag bad_3 and return $(0, \perp)$.
   (c) otherwise, <u>send out</u> $(\mathbf{z}_n, \mathbf{r}_n)$. Upon receiving RESTART, go to step 1.
7. Upon receiving $(\mathbf{z}_i, \mathbf{r}_i)$:
   (a) $\mathbf{w}^H_2 := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z}_2 - c\mathbf{t}_2, 2\gamma')$
   (b) **if** $\mathsf{Open}_{ck}(\mathbf{c}_2, \mathbf{w}^H_2, \mathbf{r}_2) \neq 1$, send out ABORT message
   (c) $\mathbf{z} := \mathbf{z}_1 + \mathbf{z}_2, \quad \mathbf{r} := \mathbf{r}_1 + \mathbf{r}_2$
   (d) $\widehat{\mathbf{w}^H} = \mathbf{w}^H_1 + \mathbf{w}^H_2$
   (e) **if** $\|\mathsf{LowBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')\|_\infty \geq \gamma' - 2\beta$, send out RESTART message
   (f) $\mathbf{w}^H := \mathsf{HighBits}_q(\mathbf{A}\mathbf{z} - c\mathbf{t}, 2\gamma')$
   (g) $\mathbf{h} := (\mathbf{h}_1, \mathbf{h}_2) = \mathsf{Hint}(\widehat{\mathbf{w}^H}, \mathbf{w}^H, \frac{q-1}{2\gamma'})$

Fig. 7. Simulator for the signing protocol

4 Evaluation

We have implemented our two-party signature scheme in Java 17 with the Bouncy Castle library for SHAKE256 implementation using parameters presented in Table 2. The running time average for 1000 executions is 1.48 ms for key generation, 174.65 ms for signing, and 1.18 ms for verification running on a laptop with AMD Ryzen 5 PRO 3500U 2.1 GHz CPU and 16 RAM. Note that the running time for key generation and signing does not take into account network delay to transmit intermediate messages from the client to the server, which should also be scaled by the number of rejections in case of signing. The average number of rejections for 1000 executions is around 101.55. The size of the private key share is 8864 bytes, the size of the public key is 2976 and the size of the signature is 21120 bytes. Presented sizes and number of rejections can be potentially optimised through the right choice of parameters, which is left for future work.

..

Parameter	Value
q (the same for signature and commitment)	8380417
d	256
(k, l)	$(4, 4)$
γ	2^{17}
γ'	$\frac{q-1}{88}$
β	78
τ	39
$(n_{com}, l_{com}, k_{com})$	$5, 4, 15$
α	256

Table 2. Parameters chosen for implementation

References

1. Abram, D., Nof, A., Orlandi, C., Scholl, P., Shlomovits, O.: Low-bandwidth threshold ECDSA via pseudorandom correlation generators. Cryptology ePrint Archive, Report 2021/1587 (2021), https://ia.cr/2021/1587
2. Alkim, E., Barreto, P.S., Bindel, N., Krämer, J., Longa, P., Ricardini, J.E.: The lattice-based digital signature scheme qTESLA. In: International Conference on Applied Cryptography and Network Security. pp. 441–460. Springer (2020)
3. Baum, C., Damgård, I., Lyubashevsky, V., Oechsner, S., Peikert, C.: More efficient commitments from structured lattice assumptions. Cryptology ePrint Archive, Report 2016/997 (2016), https://ia.cr/2016/997
4. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006. pp. 390–399. ACM (2006). https://doi.org/10.1145/1180405.1180453
5. Buldas, A., Kalu, A., Laud, P., Oruaas, M.: Server-supported RSA signatures for mobile devices. In: Foley, S.N., Gollmann, D., Snekkenes, E. (eds.) Computer Security – ESORICS 2017. pp. 315–333. Springer International Publishing, Cham (2017)
6. Castagnos, G., Catalano, D., Laguillaumie, F., Savasta, F., Tucker, I.: Bandwidth-efficient threshold EC-DSA. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. pp. 266–296. Springer International Publishing, Cham (2020)
7. Cozzo, D., Smart, N.P.: Sharing the LUOV: Threshold Post-quantum Signatures. In: Albrecht, M. (ed.) Cryptography and Coding – 17th IMA International Conference, IMACC 2019, Oxford, UK, December 16-18, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11929, pp. 128–153. Springer (2019). https://doi.org/10.1007/978-3-030-35199-1_7
8. Damgård, I., Mikkelsen, G.L., Skeltved, T.: On the security of distributed multiprime RSA. In: Lee, J., Kim, J. (eds.) Information Security and Cryptology - ICISC 2014. pp. 18–33. Springer International Publishing, Cham (2015)
9. Damgård, I., Orlandi, C., Takahashi, A., Tibouchi, M.: Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In: Garay, J.A. (ed.) Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on

Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I. Lecture Notes in Computer Science, vol. 12710, pp. 99–130. Springer (2021). https://doi.org/10.1007/978-3-030-75245-3_5
10. Doerner, J., Kondi, Y., Lee, E., Shelat, A.: Secure two-party threshold ECDSA from ECDSA assumptions. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 980–997 (2018). https://doi.org/10.1109/SP.2018.00036
11. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-dilithium: A lattice-based digital signature scheme. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268
12. Ducas, L., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals–dilithium: Digital signatures from module lattices (2018), https://repository.ubn.ru.nl/bitstream/handle/2066/191703/191703.pdf
13. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. IACR Cryptol. ePrint Arch. **2012**, 144 (2012)
14. Fu, Y., Zhao, X.: Secure two-party dilithium signing protocol. In: 2021 17th International Conference on Computational Intelligence and Security (CIS). pp. 444–448. IEEE (2021)
15. Fukumitsu, M., Hasegawa, S.: A lattice-based provably secure multisignature scheme in quantum random oracle model. In: Nguyen, K., Wu, W., Lam, K.Y., Wang, H. (eds.) Provable and Practical Security. pp. 45–64. Springer International Publishing, Cham (2020)
16. Garillot, F., Kondi, Y., Mohassel, P., Nikolaenko, V.: Threshold schnorr with stateless deterministic signing from standard assumptions. In: Malkin, T., Peikert, C. (eds.) Advances in Cryptology – CRYPTO 2021. pp. 127–156. Springer International Publishing, Cham (2021)
17. Kiltz, E., Lyubashevsky, V., Schaffner, C.: A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. In: Nielsen, J.B., Rijmen, V. (eds.) Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III. Lecture Notes in Computer Science, vol. 10822, pp. 552–586. Springer (2018). https://doi.org/10.1007/978-3-319-78372-7_18
18. Komlo, C., Goldberg, I.: FROST: Flexible round-optimized schnorr threshold signatures. In: Dunkelman, O., Jacobson, Jr., M.J., O'Flynn, C. (eds.) Selected Areas in Cryptography. pp. 34–65. Springer International Publishing, Cham (2021)
19. Lindell, Y.: Fast secure two-party ECDSA signing. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017. pp. 613–644. Springer International Publishing, Cham (2017)
20. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: A modest proposal for FFT hashing. In: Fast Software Encryption, 15th International Workshop, FSE 2008, Lausanne, Switzerland, February 10-13, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5086, pp. 54–72. Springer (2008)
21. Shor, P.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700
22. Shoup, V.: Practical threshold signatures. In: Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques. p. 207–220. EUROCRYPT'00, Springer-Verlag (2000)
23. Vakarjuk, J., Snetkov, N., Willemson, J.: Dilizium: A two-party lattice-based signature scheme. Entropy **23**(8) (2021). https://doi.org/10.3390/e23080989

A Supporting lemmas

The following two lemmas that are adapted form [11] and required for the Dilithium signature scheme correctness and security.

Lemma 1 ([11], Lemma 2). *If* $\|s\|_\infty \leq \beta$ *and* $\|\mathsf{LowBits}_q(r,\alpha)\|_\infty \leq \frac{\alpha}{2} - \beta$, *then* $\mathsf{HighBits}_q(r,\alpha) = \mathsf{HighBits}_q(r+s,\alpha)$.

Lemma 2 ([17], Lemma 4.3). *Let* $(\mathbf{s}_1, \mathbf{s}_2) \in S_\eta^l \times S_\eta^k$. *If* $\|c\mathbf{s}\|_\infty \leq \beta$ *then the following holds:*

For any $c \in \mathcal{C}$ *and* $\mathbf{z} \in S^k_{\gamma-\beta-1}$

$$\Pr_{\mathbf{y} \leftarrow S^l_{\gamma-1}}[\mathbf{z} = \mathbf{y} + c\mathbf{s}_1] = \Pr_{\mathbf{y} \leftarrow S^l_{\gamma-1}}[\mathbf{y} = \mathbf{z} - c\mathbf{s}_1] = \frac{1}{|S^l_{\gamma-1}|}, \tag{3}$$

$$\Pr_{\mathbf{y} \leftarrow S^l_{\gamma-1}}[\mathbf{y} + c\mathbf{s}_1 \in S^k_{\gamma-\beta-1}] = \frac{|S^k_{\gamma-\beta-1}|}{|S^l_{\gamma-1}|}. \tag{4}$$

Lemma 3 (General forking lemma [4]).

Fix an integer $Q \geq 1$ *to be the number of queries. Fix set* C *of size* $|C| \geq 2$. *Let* 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* (i, out) *where* i *is an index (integer in the range* $\{0, \ldots, Q\}$*) and* out *is a side output. Let* IG *be a randomised input generation algorithm. Let* F *be a forking algorithm connected with* B *that is defined in Figure 8.*

Let us define the following probabilities:

$acc := \Pr[i \neq 0 : x \leftarrow \mathsf{IG}, h_1, \ldots, h_Q \leftarrow C, (i, out) \leftarrow \mathsf{B}(x, h_1, \ldots, h_Q)]$

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

Then, $frk \geq acc \cdot \left(\frac{acc}{Q} - \frac{1}{|C|}\right)$. *Alternatively,*

$$acc \leq \frac{Q}{|C|} + \sqrt{Q \cdot frk} \tag{5}$$

$\mathsf{F}(x)$:
1. pick random coins ρ for B
2. $h_1, \ldots, h_Q \leftarrow C$
3. $(i, out) \leftarrow \mathsf{B}(x, h_1, \ldots, h_Q; \rho)$
4. **if** $i = 0$, **return** $(0, \perp, \perp)$
5. regenerate $h'_i, \ldots, h'_Q \leftarrow C$
6. $(i', out') \leftarrow \mathsf{B}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_Q; \rho)$
7. **if** $i = i'$ and $h_i \neq h'_i$, **return** $(1, out, out')$
8. **else return** $(0, \perp, \perp)$

Fig. 8. Forking algorithm