Efficient Lattice-Based Threshold Signatures with Functional Interchangeability

Guofeng Tang[1], Bo Pang[2], Long Chen[3], and Zhenfeng Zhang[3⋆]

[1] AntGroup `tangguofeng.gf@antgroup.com`
[2] Institute of Information Engineering Chinese Academy of Sciences `pangbo215@gmail.com`
[3] Institute of Software Chinese Academy of Sciences, Beijing, China `chenlong,zhenfeng@iscas.ac.cn`

Abstract. A threshold signature scheme distributes the ability to generate signatures through distributed key generation and signing protocols. A threshold signature scheme should be functionally interchangeable, meaning that a signature produced by a threshold scheme should be verifiable by the same algorithm used for non-threshold signatures. To resist future attacks from quantum adversaries, lattice-based threshold signatures are desirable. However, the performance of existing lattice-based threshold signing protocols is still far from practical.

This paper presents the first lattice-based t-out-of-n threshold signature scheme with functional interchangeability that has been implemented. To build an t-out-of-n access structure for arbitrary $t \leq n$, we first present a novel t-out-of-n version of the SPDZ MPC protocol. We avoid using the MPC protocol to evaluate hash operations for high concrete efficiency. Moreover, we design an efficient distributed rejection sampling protocol. Consequently, the online phase of our distributed signing protocol takes only 0.5 seconds in the two-party setting and 7.3 seconds in the 12-party setting according to our implementation. As a byproduct, our scheme also presents a periodic key refreshment mechanism and offers proactive security.

Keywords: Threshold signatures, Lattice-based signatures, Rejection sampling

1 Introduction

Threshold signatures are a cryptographic technique that allows a group of parties to generate and use a single digital signature, requiring a predefined number of participants (the threshold) to sign a message. A t-out-of-n threshold signature splits the signing key across n parties, such that: any subset of t honest

⋆ This work was supported by the National Key R&D Program of China 2021YFB3100100. Long Chen was also supported in part by Alibaba Innovative Research (AIR). Guofeng Tang and Bo Pang are co-first authors. The corresponding authors are Long Chen and Zhenfeng Zhang.

parties can produce a valid signature, without reconstructing the key; any subset of fewer than t parties cannot produce a signature, nor find anything about the key [1, 2]. It enables the distribution of trust placed on human operators, and offers a plan to prevent several single-points of failure in conventional digital signature implementations [3]. Additionally, threshold signatures should be *functionally interchangeable* [3]. This implies that a signature generated by a threshold scheme must be verifiable utilizing the same algorithm as conventional signatures. Functional interchangeability not only ensures the efficiency of the scheme in terms of signature, public key sizes, or verification time but also guarantees that if a client is capable of operating on the outputs of the non-threshold signature scheme, they can also handle the final output produced by the threshold implementation.

Threshold signatures have been found to have applications in multiple scenarios, including:

- **Secure cryptocurrency wallets**: Threshold signatures can be used to create secure cryptocurrency wallets [4–6], where multiple participants are required to sign transactions before they are executed. This can help prevent unauthorized access and theft of funds, as an attacker would need to compromise multiple parties to control the wallet.
- **Distributed key management**: In organizations, threshold signatures can be used to distribute the responsibility of managing critical cryptographic keys[7]. This can help reduce the risk of a single point of failure, as multiple parties need to collaborate to use the private key.
- **Decentralized identity systems**: Threshold signatures can be used in decentralized identity systems [8], where a user's identity is managed by multiple parties. This can provide users with more control over their personal information and reduce their reliance on centralized authorities.
- **Byzantine fault tolerance (BFT) consensus algorithms**: In decentralized systems, such as blockchains, threshold signatures can be used to achieve efficient BFT consensus [9, 10]. By requiring a certain number of participants to sign a block, malicious actors are prevented from controlling the network unless they control a significant portion of the signing parties.

Indeed, threshold signatures have been receiving increasing attention from researchers [11–16] due to the growing demand for their applications in various scenarios. Recently, NIST has published a roadmap [3] to present a structured approach for exploring the space of threshold schemes.

Various works have investigated the threshold version of classic signatures such as ECDSA [11–16]. However, it is now well-known that these schemes will become insecure in a "post-quantum" world where classical hard problems like the discrete logarithm can be solved by Shor's algorithm [17] using large-scale quantum computers. Recently, lattices have been recognized as a reliable foundation for post-quantum cryptography, and in 2016, NIST launched a competition to standardize post-quantum signature schemes [18]. Two of the three selected signature algorithms are lattice-based, namely Dilithium [19] and Falcon [20]. To ensure the security of popular cryptography systems such as secure wallets or BFT consensus in the post-quantum era, it is crucial to study post-quantum threshold signatures, particularly lattice-based schemes.

However, currently, no practical lattice-based threshold signature scheme with functional interchangeability exists. To illustrate this point, this paper revisits existing approaches to construct a lattice-based threshold signature.

TFHE Method. Boneh et al. [21] recently introduced a new primitive called the universal thresholdizer, which is built upon the threshold fully homomorphic encryption (TFHE) scheme. By using the universal thresholdizer, it is possible to obtain a threshold signature from any signature scheme with ease. Agrawal et al. [22] have optimized this method by reducing the signature size and improving adaptive security. However, these works are primarily theoretical and do not provide concrete efficiency analyses. In reality, this generic construction requires every participant to evaluate the signing circuit homomorphically via TFHE. Since a lattice-based signing circuit typically involves complex operations like Gaussian sampling and hash operations, such homomorphic evaluation is unlikely to be practical.

Generic MPC Method. The another approach of constructing a threshold signature is to evaluate the signing algorithm with the generic MPC scheme. For example, Bendlin et al. [23] leveraged generic multiparty computation (MPC) to evaluate the GPV signature [24]. Very recently, Cozzo et al. [25] studied the potential efficiency to compute signing circuits for all signature schemes in the NIST PQC competition via the generic MPC. Unfortunately, the result is pessimistic: the generic MPC approach seems far from practical for most of the lattice-based schemes. The reason is they leveraged different MPC methods to evaluate different operations while shifting between different MPC schemes is costly. To distributively evaluate the signing algorithm of the lattice-based signature scheme like Dilithium, they applied *Garbled Circuits* (GC) based MPC [26, 27] for non-linear operations (e.g., hash and rejection sampling) and *Secret Sharing* (SS) based MPC [28, 29] for linear operations (e.g., addition and scalar addition).

Schnorr-like Method. One may suggest to thresholdize lattice based signatures by mimicking the distributed Schnorr protocols [30, 31]. One attempt is made by Damgård et al. [32] to present a n-out-of-n threshold scheme of lattice-based Fiat-Shamir (FS) signatures. However, their scheme is *not functionally interchangeable*, i.e., the verification operation is different from that of the non-threshold scheme. Their parameters depend on the number n of parties, which results in the sizes of signature and public key grow with n, scaling as $O(\log^2 n)$. Due to the rejection sampling in the lattice-based FS signing algorithm, the expected number of repetitions to generate a threshold signature is M^n, where M is the expected number of repetitions in the non-threshold scheme. Hence the number of communication rounds also grows with n. Overall the client in [32] suffers from not only the interoperablity loss, but also the barrier of efficiency.

In a nutshell, an efficient lattice based threshold signature with functional interchangeability is still missing nowadays.

1.1 Our Results

In this paper, we construct the first lattice-based t-out-of-n threshold signature protocol, that is reported to be implemented. Compared with the previous works, our scheme has the following desirable properties:

– **Functional Interchangeability.** Our scheme meets the requirement of the NIST's threshold cryptography standardization guideline [3], i.e., the signature can be verified by the same algorithm as used for non-threshold signatures. Moreover, our sizes of the public key and signature are the same as those of the non-threshold scheme, independent of the number of parties participating in the distributed signing protocol.
– **High Efficiency.** According to our implementation, the online phase of our threshold signing protocol costs 0.5 seconds in a two-party setting and 7.3 seconds in the 12-party setting. The only known running time result of a lattice-based threshold signing protocol is approximately 12 seconds in the two-party setting from [25]. Compared with it, our work is much more efficient.
– **Proactive Security.** Our scheme can additionally provide the proactive security [33, 34]. When the key shares are refreshed periodically, our protocol remains unforgeable as long as at most $t-1$ parties are compromised during a period that starts at the beginning of one refreshment of secret key shares and ends at the end of the next refreshment of secret key shares.

1.2 Technical Overview

In theory, it is always possible to construct a function-interchangeable lattice-based threshold signature scheme if the signing key is shared among multiple parties and the signing algorithm is evaluated by a dishonest majority maliciously secure MPC. Note that the security of the MPC protocol guarantees that the malicious parties will not learn the information about the signing key shares from honest parties. However, this method requires careful consideration of the MPC protocol's efficiency and security. In particular, a generic MPC like SPDZ [28, 29] may not be efficient enough to handle the non-linear procedures involved in the signing algorithm, such as hash operations and rejection sampling. Additionally, since the secret is shared among n parties, the MPC protocol should guarantee that the computation can be carried out even if only t parties are invoked.

***t*-out-of-*n* SPDZ** We adapt the SPDZ protocol to use Shamir's Secret Sharing, enabling t-out-of-n shares of a secret value. When the signing key is shared with Shamir's Secret Sharing, any t parties can jointly compute the signature by evaluating the signing circuit.

The online computation in our SPDZ protocol based on Shamir's Secret Sharing scheme is straightforward, due to the linearity of Shamir's Secret Sharing.

The method for resisting malicious or incorrect shares still relies on MAC checking, as in SPDZ. So the new MPC protocol will not compromise the efficiency of the original SPDZ in the online phase.

Generating Beaver's triples under the form of t-out-of-n shares presents a primary challenge. To address this, we propose that each party P_i broadcasts the homomorphic encryption of their randomness values a_i and b_i. Using the homomorphic property, we can publicly compute $\mathsf{Enc}(a) = \sum_{i=1}^{n} \mathsf{Enc}(a_i)$, $\mathsf{Enc}(b) = \sum_{i=1}^{n} \mathsf{Enc}(b_i)$, and $\mathsf{Enc}(c) = \mathsf{Enc}(a) \cdot \mathsf{Enc}(b)$. We then use a novel method to ensure that each party obtains a t-out-of-n share of a, b, and c. Specifically, to reshare the secret value a with an input ciphertext of $\mathsf{Enc}(a)$, the parties first jointly choose a random mask for a, denoted as f. They then jointly decrypt $\mathsf{Enc}(a + f)$ to obtain the public $g = a + f$, which can be publicly split into t-out-of-n shares $\{g_i\}_{i=1}^{n}$. Since each party holds f_i and g_i, which is a t-out-of-n share of f and $a+f$, respectively, then the difference $f_i - g_i$ is a t-out-of-n share of a.

Avoiding Distributed Hash In a FS-type signature, the hash computation is from generating the challenge value $c \leftarrow \mathsf{H}(\mathbf{w}, \mu)$ where H is a hash function, $\mathbf{w}$ is the "commit" message and μ is a message to be signed. Evaluating hash circuit of H (e.g., SHAKE-256, SHA-3) with MPC is complex, we hope to avoid it although there have been many stepped-up results [35, 36]. If we make $\mathbf{w}$ be public, i.e., each party publicizes its own share of $\mathbf{w}$, then the computation of $\mathsf{H}(\mathbf{w}, \mu)$ can be run by each party locally without using MPC. However, this approach may cause a security issue. A lattice-based FS signature is composed of the challenge c and response $\mathbf{z}$, where $\mathbf{z}$ must satisfy $\|\mathbf{z}\|_\infty < B$. Before outputting it, checking that the infinity norm of vector $\mathbf{z}$ is smaller than B is necessary, and if $\|\mathbf{z}\|_\infty \geq B$, $\mathbf{z}$ is rejected and signing is aborted. This procedure is called rejection sampling. In a threshold signing protocol, $\mathbf{w}$ is revealed no matter whether $\mathbf{z}$ is rejected or accepted. Thus in its security proof, $\mathbf{w}$ has to be simulated in any case. But how to simulate rejected transcripts $(\mathbf{w}, c, \perp)$ is not clear, based on the standard lattice-based hardness assumptions.

Following [32], we avoid this security issue by revealing a commitment com of $\mathbf{w}$, instead of $\mathbf{w}$. Now the challenge c is derived via $c \leftarrow \mathsf{H}(com, \mu)$. Thus as long as com is revealed, the hash computation can be run by each party locally without using MPC. More importantly, simulating $(com, c, \mathbf{z})$ or $(com, c, \perp)$ is easy no matter whether rejection sampling is successful or not, based on the hiding property of commitments. The detailed description about this modified FS signature scheme is shown in Section 4.

Efficient Distributed Rejection Sampling In a threshold signing protocol, each party P_i can generate a signature share $\mathbf{z}_i$, as its own share of $\mathbf{z}$. Without rejection sampling, each party cannot publicize $\mathbf{z}_i$ from the perspective of security. The rejection sampling involves a logic statement: if $\|\mathbf{z}\|_\infty < B$, publicize it; otherwise restart the signing protocol. Since each party has only one share $\mathbf{z}_i$, t parties who participate in the signing protocol need to run a distributed rejection sampling $\mathsf{RejectS}$ protocol, jointly checking whether $\|\mathbf{z}\|_\infty < B$ or not, where $\mathbf{z} = (z^{(1)}, \cdots, z^{(K)})$ is a K-dimension vector. Following SPDZ, we use $\langle \mathbf{z} \rangle$ to denote each party P_i holds a share $\mathbf{z}_i$ such that $\sum_{i=1}^{t} \mathbf{z}_i = \mathbf{z} \mod q$.

In order to construct $\mathsf{RejectS}$ protocol, what we can directly call is a comparison protocol LTC from [37]. It takes $\langle x \rangle$ and public R as the input, outputs $\langle b \rangle$, such that $b = 1$ if $x < R$ and $b = 0$ otherwise. When applying it to $\mathsf{RejectS}$ protocol, the comparison result for $\mathbf{z}$'s each component can be easily obtained, that is $\langle b^{(j)} \rangle \leftarrow \mathsf{LTC}(\langle z^{(j)} \rangle, B)$ where $b^{(j)} = 1$ if $z^{(j)} < B$. The final bit result of rejection sampling should be $\prod_{j=1}^{K} b^{(j)}$, but it cannot be calculated locally since each bit is a shared secret. If we let parties trivially compute it via SPDZ multiplication, it will require $O(\log_2(K))$ communication rounds. As shown in the left of Fig. 1, we take $K = 8$ as an example, but K could not be so small in reality. Actually, the parameter K represents lattice dimension, and should be set more than 1000 even under a very low security strength, which results in more than 50 communication rounds.

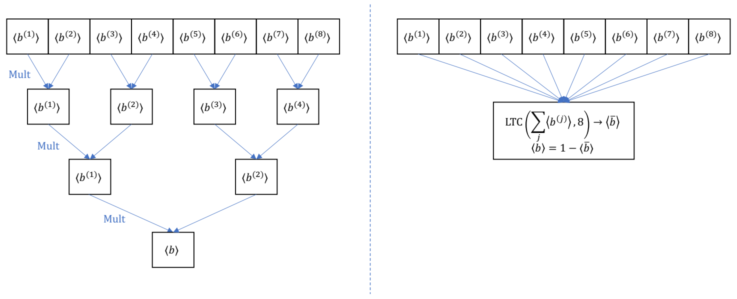

⟨b(1)⟩ ⟨b(2)⟩ ⟨b(3)⟩ ⟨b(4)⟩ ⟨b(5)⟩ ⟨b(6)⟩ ⟨b(7)⟩ ⟨b(8)⟩
Mult
⟨b(1)⟩ ⟨b(2)⟩ ⟨b(3)⟩ ⟨b(4)⟩
Mult
⟨b(1)⟩ ⟨b(2)⟩
Mult
⟨b⟩
⟨b(1)⟩ ⟨b(2)⟩ ⟨b(3)⟩ ⟨b(4)⟩ ⟨b(5)⟩ ⟨b(6)⟩ ⟨b(7)⟩ ⟨b(8)⟩
LTC(∑j ⟨b(j)⟩, 8) → ⟨b̄⟩
⟨b⟩ = 1 − ⟨b̄⟩

Fig. 1: Trivial method vs. Our method

In our construction of $\mathsf{RejectS}$ protocol, we use a novel method to avoid jointly computing $\prod_{j=1}^{K} b^{(j)}$. Firstly we observe that if and only if $b^{(j)} = 1$ for each $j \in [K]$, we have $\sum_{j=1}^{K} b^{(j)} = K$; otherwise $\sum_{j=1}^{K} b^{(j)} < K$. From this observation, we replace a series of invocations of Mult protocol by one invocation of LTC protocol. Concretely, the parties compute $\langle \sum_{j=1}^{K} b^{(j)} \rangle$ without communication, then run $\langle \bar{b} \rangle \leftarrow \mathsf{LTC}(\langle \sum_{j=1}^{K} b^{(j)} \rangle, K)$. From the above observation, $b \leftarrow 1 - \bar{b}$ represents the result of rejection sampling. As shown in the right of Fig. 1, our method makes the communication round be independent of K. More concretely, the LTC protocol from [37] requires 5 rounds with $q < 2^{32}$, then our $\mathsf{RejectS}$ protocol requires 10 rounds in total.

Achieving Proactive Security If the attacker is mobile, it can compromise all parties *one by one* in an adaptive way over time, then the secret key sk is stolen. To alleviate this attack, the parties need to refresh the shares of sk periodically, but make sk itself stay unchanged. Following proactive threshold

ECDSA [15], the shares can be refreshed by adding fresh shares of zeros, and using verifiable secret sharing (VSS) to check malicious behaviors. But obviously, the existing efficient VSS [38–40] are either based on easy problems in a "post-quantum" world or insecure in the dishonest majority setting. The lattice-based VSS constructions are mainly theoretical and not as efficient as we hope [41–43]. In this work, we invoke SPDZ's Mult protocol to generate shares of zeros. Then the malicious behaviors can be prevented by an efficient checking of MACs that is inherent in SPDZ-type shares.

1.3 Related Works

Similar to the threshold signature, a multi-signature is a primitive that allows a group of signers holding individual key pairs $(sk_1, pk_1), \cdots, (sk_n, pk_n)$ to jointly produce an aggregated signature on a message μ of their choice. It seems that a multi-signature can serve as a replacement for n-out-of-n threshold signature. But a multi-signature is probably not functional interchangeable, since it may require a completely new verification algorithm, with the input of n independent public keys or one aggregated public key. Recently, several lattice-based multi-signatures have been proposed [44, 45]. Following the work of [32] as described above, the scheme of [44] also requires $O(\log^2 n)$ public key and signature sizes, as well as M^n repetitions. In the scheme of [45], the verification algorithm takes n independent parts generated from n public keys as input, which results in their verification time being linear in n. In conclusion, the previous lattice-based multi-signatures cannot serve as the replacement for n-out-of-n threshold signatures, from the perspective of functional interchangeability.

2 Preliminaries

2.1 Notations

For positive integers a and b such that $a < b$ the integer notation $[a, b]$ is used to denote $a, a+1, \cdots, b$, and (a, b) is used to denote $a+1, \cdots, b-1$. If $a = 1$, $[a, b]$ is always simplified as $[b]$. If S is a set we use $s \overset{\$}{\leftarrow} S$ to indicate sampling s from the uniform distribution defined over S and use $|S|$ to denote the size of S. If D is a probability distribution we write $s \overset{\$}{\leftarrow} D$ to indicate sampling s from the distribution D. If A is an algorithm we write $s \leftarrow A$ to indicate assigning an output from A to s. We let $\mathbb{R}$ and $\mathbb{R}_q$ respectively denote the rings $\mathbb{Z}[X]/(X^N+1)$ and $\mathbb{Z}_q[X]/(X^N+1)$, for q an integer. Regular font letters denote elements in $\mathbb{R}$ or $\mathbb{R}_q$ (which includes elements in $\mathbb{Z}$ and $\mathbb{Z}_q$) and bold lower-case letters represent column vectors with coefficients in $\mathbb{R}$ or $\mathbb{R}_q$. By default, all vectors will be column vectors. Bold upper-case letters are matrices. Specifically $\mathbf{I}_k$ denotes a $k \times k$ identity matrix.

For an even (resp. odd) positive integer a, we define $r' = r \bmod^{\pm} a$ to be the unique element r' in the range $-\frac{a}{2} < r' \leq \frac{a}{2}$ (resp. $-\frac{a-1}{2} \leq r' \leq \frac{a-1}{2}$) such that

$r' = r \mod a$. For an element $w \in \mathbb{Z}_q$, we write $\|w\|_\infty$ to mean $|w \bmod^{\pm} q|$. We define the l_∞ and l_2 norms for $w = w_0 + w_1X + \cdots + w_{N-1}X^{N-1} \in \mathbb{R}$:

$$\|w\|_\infty = \max_i \|w_i\|_\infty,\ \ \|w\| = \sqrt{\|w_0\|_\infty^2 + \cdots + \|w_{N-1}\|_\infty^2}$$

Similarly, for a vector $\mathbf{w} = (w_1, \cdots, w_k) \in \mathbb{R}^k$, we define

$$\|\mathbf{w}\|_\infty = \max_i \|w_i\|_\infty,\ \ \|\mathbf{w}\| = \sqrt{\|w_1\|^2 + \cdots + \|w_k\|^2}$$

We rely on the following key set $S_\eta \subseteq \mathbb{R}$ parameterized by $\eta \geq 0$ consisting of small polynomials

$$S_\eta = \{x \bmod^{\pm} 2\eta : x \in \mathbb{R}\}$$

Moreover the challenge set $C \subseteq \mathbb{R}$ parameterized by $\tau \geq 0$ consists of small and sparse polynomials that have τ coefficients that are either -1 or 1 and the rest are 0. C will be used as the image of random oracle H.

2.2 Lattice Problems

We present the definitions of two standard lattice problems over rings: module short integer solution (MSIS) and learning with errors (MLWE) [46]. We also call them MSIS/MLWE assumption if for any probabilistic polynomial-time (PPT) adversaries the probability that they can solve a given problem is negligible.

Definition 1 ($\mathsf{MSIS}_{q,k,l,\gamma}$ **Problem).** *Given a random matrix* $\mathbf{A} \xleftarrow{\$} \mathbb{R}_q^{k\times l}$, *find a vector* $\mathbf{x} \in \mathbb{R}^{l+k}$ *such that* $[\mathbf{A}|\mathbf{I}_k] \cdot \mathbf{x} = \mathbf{0} \mod q$ *and* $0 < \|\mathbf{x}\|_\infty \leq \gamma$.

Definition 2 ($\mathsf{MLWE}_{q,k,l,\eta}$ **Problem).** *Given a pair* $(\mathbf{A}, \mathbf{t}) \in \mathbb{R}_q^{k\times l} \times \mathbb{R}_q^k$ *decide whether it was generated uniformly at random from* $\mathbb{R}_q^{k\times l} \times \mathbb{R}_q^k$, *or it was generated in a way that* $\mathbf{A} \xleftarrow{\$} \mathbb{R}_q^{k\times l}, \mathbf{s} \xleftarrow{\$} D^l, \mathbf{e} \xleftarrow{\$} D^k$ *and* $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e} \mod q$ *where* $D : \mathbb{R}_q \to [0, 1]$ *is a probability distribution. In particular, we consider* D *to be the uniform distribution over* S_η.

3 t-out-of-n version of SPDZ

In this section, we propose a modified version of the SPDZ protocol that supports t-out-of-n threshold access structures for arbitrary $t \leq n$, building upon previous work on SPDZ [28, 29] in the preprocessing model. The key difference in our approach is the use of Shamir's Secret Sharing [47] instead of additive secret sharing. For each evaluation, t parties are required to complete the computation instead of all n parties. If at least one honest party is present among the t parties involved, the output must be either an abort or the correct evaluation result.

Intuitively, the online phase of the protocol involves computing a function represented as a circuit, where privacy is achieved through Shamir's Secret Sharing of the inputs and outputs of each gate, and correctness is ensured by adding

Shamir's Secret Sharings of MACs on the inputs and outputs of each gate. Specifically, each player P_i holds a Shamir's secret share $\alpha_i \in F_p$ of a secret value α, which is considered a fixed MAC key that is secretly generated during the preprocessing phase (see Protocol 3). A data item $x \in F_p$ is said to be secretly shared if P_i holds a tuple $(x_i, \gamma(x)_i)$, where x_i is a Shamir's secret share of x, and $\gamma(x)_i$ is a Shamir's secret share of $\gamma(x) := \alpha \cdot x$. Here, $\gamma(x) = \alpha \cdot x$ serves as an information-theoretical MAC that authenticates x under the global key α, thereby guaranteeing the integrity of the value x even in the presence of malicious parties.

In the following, the tuple of shares is denoted as $\langle x \rangle \leftarrow ((x_1, \cdots, x_n), (\gamma(x)_1, \cdots, \gamma(x)_n))$. Here, for any subset $S \subseteq [n]$ with $|S| \geq t$, it holds that $\sum_{i \in S} \lambda_{i,S} \cdot x_i \mod q = x$ and $\sum_{i \in S} \lambda_{i,S} \cdot \gamma(x)_i \mod q = \gamma(x)$, where the Lagrangian coefficients are defined as $\lambda_{i,S} = \prod_{j \in S, j \neq i} \frac{j}{j-i}$. This notion can be extended to a vector. Specifically, if the secret $\mathbf{x} = (x^{(1)}, \cdots, x^{(K)})$ is a K-dimensional vector, then $\langle \mathbf{x} \rangle$ is composed of $\langle x^{(1)} \rangle, \cdots, \langle x^{(K)} \rangle$. Thus, the protocol can be described in detail as follows.

Online Phase The online operation can be easily derived from the linearity of Shamir's secret sharing. Let shares of secret values x, y are represented as follows

$$\langle x \rangle \leftarrow ((x_1, \cdots, x_n), (\gamma(x)_1, \cdots, \gamma(x)_n))$$
$$\langle y \rangle \leftarrow ((y_1, \cdots, y_n), (\gamma(y)_1, \cdots, \gamma(y)_n))$$

The linear operations (addition and scalar multiplication) can be performed on the $\langle \cdot \rangle$-sharings locally. The addition $\langle x + y \rangle \leftarrow \langle x \rangle + \langle y \rangle$ is easily generated by each party P_i computing $x_i + y_i \mod q$ and $\gamma(x)_i + \gamma(y)_i \mod q$. The scalar multiplication for a public constant $\bar{e}$ in $\mathbb{Z}_q$ with $\langle x \rangle$ can be performed by letting each party P_i multiplies its own share $x_i, \gamma(x)_i$ by $\bar{e}$. The addition for a public constant $\bar{e}$ in $\mathbb{Z}_q$ with $\langle x \rangle$ is performed by generating a t-out-of-n share $\bar{e}_i$ of $\bar{e}$ for each party and compute

$$\begin{aligned} \langle x + \bar{e} \rangle \leftarrow ((&x_1 + \bar{e}_1, x_2 + \bar{e}_2, \cdots, x_n + \bar{e}_n), \\ &(\gamma(x)_1 + \alpha_1 \cdot \bar{e}, \cdots, \gamma(x)_n + \alpha_n \cdot \bar{e})). \end{aligned}$$

Thus $\langle x + \bar{e} \rangle \leftarrow \langle x \rangle + \bar{e}$ represents that each party P_i adds $\bar{e}_i$ to its share x_i and adds $\alpha_i \cdot \bar{e}$ to its share $\gamma(x)_i$ for $i \in [n]$.

Computing multiplication requires one-round interaction. With pre-processed Beaver's triples $(\langle \bar{x} \rangle, \langle \bar{y} \rangle, \langle \bar{z} \rangle)$ such that $\bar{x} \cdot \bar{y} = \bar{z}$, two $\langle \cdot \rangle$-sharings $\langle x \rangle, \langle y \rangle$ can be multiplied as described in Protocol 1.

Particularly, by utilizing shares of random bits generated in the preprocessing phase, the parties can jointly sample a random value and obtain its shares. Suppose the parties have shares $\langle b^{(0)} \rangle, \cdots, \langle b^{(n_\gamma - 1)} \rangle$ of n_γ random bits, where each $b^{(j)} \xleftarrow{\$} \{0, 1\}$. In this case, they can generate shares $\langle r \rangle$ as follows:

$$\langle r \rangle \leftarrow \sum_{j=0}^{n_\gamma - 1} 2^j \cdot \langle b^{(j)} \rangle \text{ such that } r \xleftarrow{\$} [0, 2^{n_\gamma} - 1].$$

Protocol 1 $\mathsf{Mult}(\langle x\rangle, \langle y\rangle) \to \langle x \cdot y\rangle$

The parties takes shares $(\langle x\rangle, \langle y\rangle)$ as the input, do the following:

1. Take one pre-processed triple $(\langle \bar{x}\rangle, \langle \bar{y}\rangle, \langle \bar{z}\rangle)$ and open $\langle x\rangle - \langle \bar{x}\rangle$, $\langle y\rangle - \langle \bar{y}\rangle$ to get ϵ, ρ respectively.
2. The parties compute shares $\langle x \cdot y\rangle \leftarrow \langle \bar{z}\rangle + \epsilon \cdot \langle \bar{y}\rangle + \rho \cdot \langle \bar{x}\rangle + \epsilon \cdot \rho$

Protocol 2 Our sub-protocol $\mathsf{Reshare}(\mathsf{Enc}(\mathbf{a}))$ with t-out-of-n Shamir's secret sharing

The parties take a public ciphertext $\mathsf{Enc}(\mathbf{a})$ as the input. Output is a t-out-of-n share $\mathbf{a}_i$ of $\mathbf{a} \in \mathbb{Z}_q^K$ to each party P_i.

1. Each party P_i samples an uniform $\mathbf{f}_i \in \mathbb{Z}_q^K$. Broadcast the ciphertext $\mathsf{Enc}(\mathbf{f}_i)$. Define $\mathbf{f} = \sum_{i=1}^n \mathbf{f}_i \mod q$.
2. Each party P_i performs Shamir's secret sharing to re-share $\mathbf{f}_i$: generate t-out-of-n shares $\{\mathbf{f}_{i,j}\}_{j\in[n]}$ of $\mathbf{f}_i$, send $\mathbf{f}_{i,j}$ to P_j.
3. Each party P_i generates a zero-knowledge proof to prove the validity of $\mathsf{Enc}(\mathbf{f}_i)$. The protocol aborts if any proof fails.
4. The parties compute $\mathsf{Enc}(\mathbf{f}) = \mathsf{Enc}(\mathbf{f}_1) \oplus \cdots \oplus \mathsf{Enc}(\mathbf{f}_n)$, and $\mathsf{Enc}(\mathbf{a} + \mathbf{f}) = \mathsf{Enc}(\mathbf{a}) \oplus \mathsf{Enc}(\mathbf{f})$.
5. The parties jointly decrypt $\mathsf{Enc}(\mathbf{a} + \mathbf{f})$ and thereby obtain $\mathbf{a} + \mathbf{f}$. Let the party P_1 performs Shamir's secret sharing to re-share $\mathbf{r} = \mathbf{a} + \mathbf{f}$: generate t-out-of-n shares $\{\mathbf{r}_j\}_{j\in[n]}$ of $\mathbf{r}$, send $\mathbf{r}_j$ to P_j.
6. Each party P_i locally computes $\mathbf{a}_i = \mathbf{r}_i - \sum_{j=1}^n \mathbf{f}_{j,i} \mod q$, satisfying that $\sum_{i\in S} \lambda_{i,S} \cdot \mathbf{a}_i = \mathbf{a} \mod q$ with $|S| \geq t$.

To open $\langle x\rangle$, each party P_i broadcasts its own x_i, then locally sums up the received shares, and finally obtains the opened value x. To guarantee the correctness of x while avoiding opening the global key α, we observe that since x is public, the value $\gamma - \alpha x$ is a linear function of shared values γ, α. Therefore, players can compute shares of the value $\gamma(x) - \alpha x$ locally and then reconstruct $\gamma(x) - \alpha x$ and check if it is equal to 0, without revealing information on α.

Preprocessing Phase Our preprocessing phase is similar to that of SPDZ [28, 29, 48]. As described in the functionality $\mathcal{F}_{\mathsf{PREP}}$ of preprocessing phase (Protocol 3), the parties complete three tasks: (1) selecting a global MAC key, (2) generating multiple Beaver's triples, and (3) generating shares of random bits denoted as $\langle \mathbf{b}\rangle$.

For completeness, we present the concrete construction of preprocessing phase in Protocol 4. Overall, the entire procedure is the same as before, except for the replacement of the $\mathsf{Reshare}$ protocol component with a new one, which is illustrated in Protocol 2.

Protocol 3 Functionality $\mathcal{F}_{\mathsf{PREP}}$

1. It first specifies a t-out-of-n share α_i for each party, let the global MAC key be $\alpha \leftarrow \sum_{i\in S} \lambda_{i,S} \cdot \alpha_i \mod q$ for any subset $S \subseteq [n]$ with $|S| \geq t$.
2. It outputs a set of "multiplication triples": $\{\langle \bar{x}\rangle, \langle \bar{y}\rangle, \langle \bar{z}\rangle\}$ such that $\bar{x}, \bar{y} \xleftarrow{\$} \mathbb{Z}_q$ and $\bar{z} = \bar{x} \cdot \bar{y}$.
3. It outputs shares of bits: $\langle \mathbf{b}\rangle$ such that $\mathbf{b}$ is a binary vector.

4 A Signature Scheme from MLWE and MSIS

In this section, we propose a modification to the lattice-based signature framework to ensure that the security of a threshold signature scheme is based on standard assumptions. Our approach involves the addition of a commitment scheme, inspired by the ideas presented in [32].

The original lattice-based signature scheme for Fiat-Shamir transformation utilizes MLWE samples $\mathbf{A}$ and $\mathbf{t} = [\mathbf{A}|\mathbf{I}_k] \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix}$ as its public key. To sign a message μ, the scheme first samples a short random vector $\mathbf{y}$ and computes the "commit" message $\mathbf{w} \leftarrow [\mathbf{A}|\mathbf{I}_k]\mathbf{y}$, the challenge $c \leftarrow \mathsf{H}(\mathbf{w}, \mu)$, and the response $\mathbf{z} \leftarrow \mathbf{y} + c \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix}$. If $\|\mathbf{z}\|_\infty < B$, the output is $(c, \mathbf{z})$ as a signature. Otherwise, the signing process is aborted using rejection sampling theorem [49].

In a threshold version of the above signature scheme, the "commit" message $\mathbf{w}$ must be revealed since it is useful for deriving the challenge c for parties. Therefore, the security reduction needs to simulate transcripts $(\mathbf{w}, c, \mathbf{z})$ for the "non-aborting" case, as well as rejected $(\mathbf{w}, c, \perp)$ for the "aborted" case. Simulating $(\mathbf{w}, c, \mathbf{z})$ is easy based on the zero-knowledge (ZK) property of the underlying Σ-protocol [49]. However, simulating rejected $(\mathbf{w}, c, \perp)$ requires additional non-standard assumptions (e.g., rejected MLWE [50]).

Following [32], we avoid this issue by revealing a commitment of $\mathbf{w}$, instead of $\mathbf{w}$ itself. Concretely, the challenge c is derived via $c \leftarrow \mathsf{H}(com, \mu)$ instead of $c \leftarrow \mathsf{H}(\mathbf{w}, \mu)$ where com is a commitment of $\mathbf{w}$. Since com hides $\mathbf{w}$, simulating "aborted" $(com, c, \perp)$ becomes easy based on the hiding property of commitments. Only if rejection sampling is successful, $\mathbf{w}$ will be opened thus simulating "non-aborting" transcripts still relies on the ZK property. Concretely, we make use of a lattice-based commitment from [51] to construct the following signature scheme, which can be seen as a modified version of the Crystal-Dilithium scheme [19].

- $\mathsf{KeyGen}(1^\lambda) \rightarrow (pk, sk)$: On input of the security parameter λ, randomly choose $\mathbf{A} \xleftarrow{\$} \mathbb{R}_q^{k\times l}$, and $\mathbf{s} \xleftarrow{\$} S_\eta^l$, $\mathbf{e} \xleftarrow{\$} S_\eta^k$, compute $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e} \mod q$. Choose $\mathbf{A}' \xleftarrow{\$} \mathbb{R}_q^{k_1\times(k_2-k_1)}$ and construct $\mathbf{A}^{(1)} \leftarrow [\mathbf{I}_{k_1}|\mathbf{A}'] \in \mathbb{R}_q^{k_1\times k_2}$. Let $k_3 = k+l$, choose $\mathbf{A}'' \xleftarrow{\$} \mathbb{R}_q^{k_3\times(k_2-k_1)}$ and construct $\mathbf{A}^{(2)} \leftarrow [\mathbf{0}|\mathbf{A}''] \in \mathbb{R}_q^{k_3\times k_2}$. Return the public key $pk \leftarrow (\mathbf{A}, \mathbf{t}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)})$ and secret key $sk \leftarrow (\mathbf{s}, \mathbf{e})$.

– $\mathsf{Sign}(sk, \mu) \to \sigma$: Given the secret key $sk = (\mathbf{s}, \mathbf{e})$, and a message $\mu \in \{0,1\}^*$:
  1. Randomly choose $\mathbf{y} \xleftarrow{\$} S_{\gamma}^{l+k}$, compute $\mathbf{w} \leftarrow [\mathbf{A}|\mathbf{I}_k]\,\mathbf{y} \mod q$. Choose $\mathbf{r} \xleftarrow{\$} S_{\gamma_1}^{k_2}$ randomly, compute the commitment value of $\mathbf{w}$, $com \leftarrow \begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix}\mathbf{0}\\ \mathbf{w}\end{bmatrix} \mod q$.
  2. Compute $c \leftarrow \mathsf{H}(com, \mu)$ and $\mathbf{z} \leftarrow \mathbf{y} + c\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}$ If $\|\mathbf{z}\|_\infty \geq B$, restart the computation from Step 1, where $B = \gamma - \beta$ and β is a bound such that $\left\| c\begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}\right\|_\infty \leq \beta$ holds for all possible $c, \mathbf{s}, \mathbf{e}$. Otherwise, output the signature $\sigma \leftarrow (com, \mathbf{z}, \mathbf{r})$.

– $\mathsf{Verify}(pk, \mu, \sigma) \to 1/0$: Given the public key $pk = (\mathbf{A}, \mathbf{t}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)})$, a message μ and a signature $\sigma = (com, \mathbf{z}, \mathbf{r})$, compute $\mathbf{w} \leftarrow [\mathbf{A}|\mathbf{I}_k]\,\mathbf{z} - \mathsf{H}(com, \mu)\cdot \mathbf{t} \mod q$. Return 1 if $\|\mathbf{z}\|_\infty < B$, $\|\mathbf{r}\|_\infty < \gamma_1$, and $com = \begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix}\mathbf{0}\\ \mathbf{w}\end{bmatrix} \mod q$; otherwise return 0.

Theorem 1. *The signature scheme is unforgeable against chosen message attacks based on the* $\mathsf{MLWE}_{q,k,l,\eta}$, $\mathsf{MSIS}_{q,k,l+1,2(\gamma-\beta)}$ *assumptions on input* $[\mathbf{A}|\mathbf{t}]$, *as well as the binding property of the commitment com. Concretely, the binding property relies on the* $\mathsf{MSIS}_{q,k_1,k_2,2\gamma_1}$ *assumption on input* $\mathbf{A}^{(1)}$. *(The proof is implicitly included in the proof of Theorem 2.)*

5 A Lattice-Based Threshold Signature Scheme

In this section, we give a t-out-of-n threshold signature system, including three multi-party computation protocols ($\mathsf{DKeyGen}, \mathsf{DSign}, \mathsf{ShareRefresh}$).

1. $\mathsf{DKeyGen}(1^\lambda) \to (pk, \langle sk\rangle)$: The distributed key generation protocol (run by n parties) takes the security parameter λ as input, and outputs the public key and t-out-of-n shares of the corresponding secret key.
2. $\mathsf{ShareRefresh}(\langle sk\rangle) \to \langle sk\rangle$: The share refreshment protocol (run by n parties) takes secret key shares as input, outputs new shares of the same secret key sk.
3. $\mathsf{DSign}(\langle sk\rangle, pk, \mu) \to \sigma$: The distributed signing protocol (run by t parties) takes secret key shares, the public key and message as inputs, outputs a signature σ. We stress that σ can be verified by the conventional verification algorithm $\mathsf{Verify}(pk, \mu, \sigma) \to 1/0$ in Section 4.

The parameters for our threshold signature scheme are shown in Table 1.

5.1 Security Definition

In a t-out-of-n threshold signature scheme, the security definition can be described as follows. We consider a proactive model where the protocol's lifespan

Table 1: Parameters for our lattice-based threshold signature scheme

Parameter	Description
$n; t$	Number of parties; threshold value
m	The length of MAC keys $\boldsymbol{\alpha}$
N	A power of two defined the degree of $X^N + 1$
$q; n_q$	Prime modulus; $n_q = \log_2 q$
(k, l)	The height and width of random matrices $\mathbf{A}$
(k_1, k_2)	The height and width of random matrices $\mathbf{A}^{(1)}$
(k_3, k_2)	The height and width of random matrices $\mathbf{A}^{(2)}$
γ	A power of two defined the $\mathbf{y}$ coefficient range
γ_1	A power of two defined the $\mathbf{r}$ coefficient range
η	A power of two defined the $\mathbf{s}, \mathbf{e}$ coefficient range
τ	The number of $-1, 1$ in c
β	The bound such that $\left\lVert c \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix} \right\rVert_\infty \leq \beta$, $\beta = \tau \cdot \eta$
M	The expected number of rejection samplings
$k_3; K; K_2; B$	$k_3 = k + l; K = (k + l) \cdot N; K_2 = k_2 \cdot N; B = \gamma - \beta$
$n_\eta; n_\gamma; n_{\gamma_1}$	$n_\eta = \log_2(2\eta); n_\gamma = \log_2(2\gamma); n_{\gamma_1} = \log_2(2\gamma_1)$

is divided into separate time periods. We assume that at most $t - 1$ parties are corrupted during each period. A period is defined as starting at the beginning of one refreshment of secret key shares and ending at the end of the next refreshment of secret key shares, following the model proposed in [52].

Since all parties have the same role, we fix the index of honest party who has to send out the message first in each round of interaction. Assume that $\{P_1, P_2, \cdots, P_t\}$ is the set of parties participating in the signing protocol. For a mobile adversary $\mathcal{A}$, assume that it corrupts $\mathcal{P}_c = \{P_2, P_3, \cdots, P_t\}$ during the period tp (P_1 is honest), and may corrupt different parties $\mathcal{P}'_c = \{P_1, P_3, \cdots, P_t\}$ during the next period $\mathsf{tp} + 1$ (P_2 is honest). In the above case, we say that the $\mathcal{P}_c$-corrupted state is changed to the $\mathcal{P}'_c$-corrupted state. Between the two compromises, there is a refreshment phase which belongs to not only period tp, but also period $\mathsf{tp} + 1$. Thus during this refreshment phase, the parties P_1 and P_2 are both honest.

Recall how $\mathcal{A}$ works: it first participates in the key generation protocol to generate a public key pk for the threshold scheme. Then it queries signatures of several messages $\mu_1, \cdots, \mu_{q_s}$ by interacting with P_1 to run the signing protocol on those messages if it is under the $\mathcal{P}_c$-corrupted state. In the meanwhile, $\mathcal{A}$ may query the share-refreshment for changing to the $\mathcal{P}'_c$-corrupted state. In the end, $\mathcal{A}$ wins if it outputs a message $\mu^* \neq \mu_l$ for each $l \in [q_s]$ and a valid signature σ^* for it under the public key pk.

Formally, we define the security model using the following game $\mathsf{Expt}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}$ between an adversary $\mathcal{A}$ and the honest parties. Assume that $\mathcal{A}$ is under the $\mathcal{P}_c$-corrupted state from the beginning.

- At this stage, $\mathcal{A}$ can make the following queries:

- **DKG Query**: This query is allowed only once, and the honest parties run the distributed key generation protocol with $\mathcal{A}$ (controlling $\mathcal{P}_c$). Finally, output a public key pk to $\mathcal{A}$.
- **Share-Refreshment Query**: Upon receiving $\mathcal{P}'_c$, the honest parties and $\mathcal{A}$ (controlling $\mathcal{P}_c \cap \mathcal{P}'_c$) run the share-refreshment protocol. Finally, give the secret shares of $\mathcal{P}'_c \setminus (\mathcal{P}_c \cap \mathcal{P}'_c)$ to $\mathcal{A}$. For an instance with $\mathcal{P}_c = \{P_2, \cdots, P_t\}$ and $\mathcal{P}'_c = \{P_1, P_3, \cdots, P_t\}$, give P_1's secret share to $\mathcal{A}$. Then we enter into the next period under the $\mathcal{P}'_c$-corrupted state.
- **Signing Query**: Upon receiving a message μ, P_1 runs the signing protocol with $\mathcal{A}$ (controlling $\mathcal{P}_c$). Finally, output a signature σ to $\mathcal{A}$. Set $\mathsf{Mset} \leftarrow \mathsf{Mset} \cup \{\mu\}$.

• After the above queries, $\mathcal{A}$ outputs a message-signature pair (μ^*, σ^*). Output 1 if $\mathcal{A}$ wins, such that $\mu^* \notin \mathsf{Mset}$ and σ^* is a valid signature on μ^* under pk.

Definition 3. *A threshold signature scheme is said to be proactively secure if for any PPT adversary $\mathcal{A}$, its advantage of winning $\mathsf{Expt}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}$ is negligible in κ,*

$$\mathsf{Adv}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}(\mathcal{A}) = \Pr[\mathsf{Expt}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}(\mathcal{A}) \to 1] \leq \mathsf{negl}(\kappa)$$

5.2 Distributed Key Generation

We propose a distributed key generation protocol in which n parties interactively generate a public key.

To achieve this goal, the secret key $(\mathbf{s}, \mathbf{e})$ should be uniformly sampled from S_η jointly by the n parties. Recall that the parties can generate shares

$$\langle r \rangle \leftarrow \sum_{j=0}^{n_\eta - 1} 2^j \cdot \langle b^{(j)} \rangle \text{ s.t. } r \xleftarrow{\$} [0, 2^{n_\eta} - 1]$$

from pre-processed shares $\langle b^{(0)} \rangle, \cdots, \langle b^{(n_\eta - 1)} \rangle$ of bits. With $n_\eta = \log_2 2\eta$, each party locally computes its own share of a random integer $r \xleftarrow{\$} [0, 2\eta - 1]$. Then the parties can get shares $\langle \bar{r} \rangle$ of a coefficient $\bar{r} \xleftarrow{\$} [-\eta + 1, \eta]$ by computing $\langle \bar{r} \rangle \leftarrow \langle r \rangle - (\eta - 1)$. Since each coefficient of a polynomial in S_η belongs to the interval $[-\eta+1, \eta]$, sampling $(\mathbf{s}, \mathbf{e})$ is to sample K random integers from $[-\eta+1, \eta]$ with $K = (k + l)N$. To generate shares of K coefficients $\bar{r} \xleftarrow{\$} [-\eta + 1, \eta]$ in the above way, the parties can get shares $\langle \mathbf{s} \rangle, \langle \mathbf{e} \rangle$. To obtain the public key, parties compute $\langle \mathbf{t} \rangle \leftarrow \mathbf{A} \langle \mathbf{s} \rangle + \langle \mathbf{e} \rangle$ and then open $\langle \mathbf{t} \rangle$.

However, an active adversary may manipulate $\mathbf{t}$ by broadcasting a malicious share, which poses a security risk. To protect $\mathbf{t}$ from manipulation, parties need to check its MAC value. If the check passes, honest parties can trust the correctness of $\mathbf{t}$ and output it as the public key. If the check fails, honest parties must abort this protocol by outputting $\perp$.

It is important to note that a soundness error of a single MAC checking, as provided by the SPDZ protocol [28, 29], is not sufficiently small, especially against quantum adversaries. For a single MAC key in $\mathbb{F}_q$, the soundness error is $\frac{2}{q}$. However, when the security parameter $\lambda = 128$, a soundness error of less than 2^{-256} is required. Unfortunately, q is set to be less than 2^{30} in lattice-based FS signatures.

To achieve the desired soundness error, we replicate MAC keys by generating m global MAC keys $\boldsymbol{\alpha} = (\alpha^{(1)}, \cdots, \alpha^{(m)}) \in \mathbb{Z}_q^m$ during the preprocessing phase (see Protocol 3). The shares of a secret value $a \in \mathbb{Z}_q$ are represented as follows:

$$\begin{aligned}\langle a \rangle \leftarrow ((a_1, \cdots, a_n), (\gamma(a)_1^{(1)}, \cdots, \gamma(a)_1^{(m)}), \cdots,& \\ (\gamma(a)_n^{(1)}, \cdots, \gamma(a)_n^{(m)}))&\end{aligned}$$

where $\sum_{i \in S} \lambda_{i,S} \cdot a_i \mod q = a$ and $\sum_{i \in S} \lambda_{i,S} \cdot \gamma(a)_i^{(\xi)} = \alpha^{(\xi)} \cdot a \mod q$ for each $\xi \in [m]$ and subset $S \subseteq [n]$ with $|S| \geq t$. Party P_i owns $(a_i, \gamma(a)_i^{(1)}, \cdots, \gamma(a)_i^{(m)})$ for each $i \in [n]$. The checking procedure for replicated MACs is defined in Protocol 5, denoted as $\mathsf{MACCheck}_m$. The soundness of this protocol can be demonstrated by the following lemma:

Lemma 1. *The* $\mathsf{MACCheck}_m$ *protocol is sound, meaning that it outputs 0 except with a probability of* $\epsilon_s = (\frac{2}{q})^m$ *if at least one value or MAC is not computed correctly.*

Malicious parties may also attempt to use forged secret shares or violate the protocol's specifications during the signing procedure. To detect such behavior, the MAC shares of the secret key generated during the distributed key generation must be included as part of the shared secret key and used in the distributed signing protocol. The distributed signing protocol also uses $\mathsf{MACCheck}_m$, but with t-out-of-t shares as input instead of t-out-of-n shares used in the distributed key generation protocol. To modify Protocol 5 for the signing protocol, we only need to perform two steps: first, compute $\sigma_i^{(\xi)} \leftarrow \gamma_i^{(\xi)} - \alpha_i^{(\xi)} \cdot t_{\mathbf{r}}$ at Step 2; second, compute $\sigma^{(\xi)} \leftarrow \sum_{i=1}^{t} \sigma_i^{(\xi)} \mod q$ at Step 4.

By invoking $\mathsf{MACCheck}_m$ in Protocol 5, the distributed key generation is specified as Protocol 6.

5.3 Key Shares Refreshment

In this section, we present a refreshment protocol that enables the modification of each share of sk while preserving the value of sk itself. This protocol is invoked periodically to prevent mobile adversaries from compromising all parties in an adaptive manner.

A straightforward method is for each party to distribute fresh shares of zeros and then locally sum up the received shares. However, this method is vulnerable to attacks by malicious parties who may distribute malicious shares to disrupt the procedure. To address this vulnerability, one potential solution is to use homomorphic commitment-based verifiable secret sharing (VSS). This approach involves each party distributing fresh shares of zeros in the form of commitments and then providing zero-knowledge proofs to verify that the shared secret in the commitment is indeed zero. However, one can not directly derive a VSS scheme from the lattice-based homomorphic commitment [51][4]. The lattice-based VSS constructions [41–43] are not as practical as we need.

In this work, we use SPDZ's multiplication operations, $\mathsf{Mult}(\langle b\rangle, \langle 1-b\rangle) \rightarrow \langle x\rangle$, where b is a random bit and $x = b \cdot (1-b) = 0$. This allows us to generate shares of zero. Importantly, the correctness of the protocol can be ensured by efficiently checking MACs that are inherent in SPDZ-type shares $\langle x\rangle$. By generating shares $\langle \mathbf{0}\rangle$ of K zeros in parallel, each party P_i adds its own share of $\mathbf{0}$ to its secret key share $\begin{bmatrix} \mathbf{s}_i \\ \mathbf{e}_i \end{bmatrix}$. This changes $\begin{bmatrix} \mathbf{s}_i \\ \mathbf{e}_i \end{bmatrix}$ but leaves $(\mathbf{s}, \mathbf{e})$ unchanged.

Furthermore, we must update the global MAC key $\boldsymbol{\alpha}$. Our approach is making the parties generate shares of m random secret values $\langle \alpha^{(\xi)'}\rangle$ to form a new shared MAC key $\langle \boldsymbol{\alpha}'\rangle$. They then jointly compute the multiplication of $\langle \alpha^{(\xi)'}\rangle$ with the refreshed $\langle sk\rangle$ to obtain $\langle \alpha^{(\xi)'} \cdot sk\rangle$ for each $\xi \in [m]$. It is important to note that $\langle \alpha^{(\xi)'} \cdot sk\rangle$ for $\xi \in [m]$ form the new MAC shares of the secret key $\langle sk \cdot \boldsymbol{\alpha}'\rangle$ with respect to a new MAC key $\boldsymbol{\alpha}'$.

The detailed procedure is shown in Protocol 7.

5.4 Distributed Signing

In this subsection, we present a distributed signing protocol that enables any more than t parties to jointly generate a signature that can be verified by the Verify algorithm presented in Section 4. Since t parties are sufficient to generate the signature, we assume without loss of generality that only t parties, indexed by $[t] = \{1, 2, \ldots, t\}$, are involved in the signing protocol.

Recall that each party P_i for $i \in [t]$ possesses t-out-of-n shares in $\langle \mathbf{s}\rangle$ and $\langle \mathbf{e}\rangle$ after the distributed key generation. The first step in the protocol is to convert these t-out-of-n shares into t-out-of-t shares by multiplying $\lambda_{i,[t]}$ with $\mathbf{s}_i$, $\mathbf{e}_i$, and P_i's corresponding MAC shares. For each secret value $a \in \mathbb{Z}_q$, it is shared by t parties with shares $\langle a\rangle \leftarrow ((a_1, \ldots, a_t), (\gamma(a)_1, \ldots, \gamma(a)_t))$, where $\sum_{i\in[t]} a_i \bmod q = a$ and $\sum_{i\in[t]} \gamma(a)_i \bmod q = \gamma(a)$. These shares are referred to as t-out-of-t shares.

The second step in the protocol involves the joint sampling of short vectors $\mathbf{y} \in S_{\gamma}^{l+k}$ and $\mathbf{r} \in S_{\gamma_1}^{k_2}$ by the t parties. During the preprocessing phase, t-out-of-n shares $\langle \mathbf{b}\rangle$ of bits are generated. The t parties $\{P_i\}_{i\in[t]}$ convert their t-out-of-n shares into t-out-of-t shares by multiplying appropriate Lagrangian coefficients and use them to construct t-out-of-t shares $\langle \mathbf{y}\rangle$ and $\langle \mathbf{r}\rangle$. Next, the parties compute shares $\langle \mathbf{w}\rangle \leftarrow [\mathbf{A}|\mathbf{I}_k]\langle \mathbf{y}\rangle$ and commitment each shares $\begin{bmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \end{bmatrix} \langle \mathbf{r}\rangle + \begin{bmatrix} \mathbf{0} \\ \langle \mathbf{w}\rangle \end{bmatrix}$ as com_i.

Subsequently, each party P_i sends its own share com_i to the other parties and receives $com_{\bar{i}}$ for each $\bar{i} \neq i$. As a result, each party obtains the joined

[4] The limited homomorphic property of the lattice commitment [51] cannot support the homomorphic multiplication with a large constant factor without destroying the structure of the commitment.

commitment $com \leftarrow \sum_{i=1}^{t} com_i \bmod q$. The universal challenge $c \leftarrow \mathsf{H}(com, \mu)$ is computed locally using the commitment and a random value μ. With this universal challenge c, the parties can compute their signature shares as $\langle \mathbf{z} \rangle \leftarrow \langle \mathbf{y} \rangle + c \cdot \begin{bmatrix} \langle \mathbf{s} \rangle \\ \langle \mathbf{e} \rangle \end{bmatrix}$.

The signature is generated only if $\|\mathbf{z}\|_\infty < B$. The next step is to determine whether $\|\mathbf{z}\|_\infty < B$ in a distributed manner, where $B = \gamma - \beta$. This involves checking whether $-B < z^{(j)} \bmod {}^{\pm}q < B$ for each $j \in [K]$, where $\mathbf{z} = (z^{(1)}, \ldots, z^{(K)})$. Since $B < q/2$ always holds, the inequality $-B < z^{(j)} \bmod {}^{\pm}q < B$ is equivalent to $z^{(j)} + B \bmod q < 2B$.

To determine this condition in a distributed manner, we use the LTC protocol from [37]. This protocol takes a shared secret value $\langle x \rangle$ and a public constant R as input, and outputs $\langle b \rangle$ such that $b = 1$ if $x < R$ and $b = 0$ otherwise. When using the LTC protocol in our scheme, it should be invoked as $\mathsf{LTC}(\langle z^{(j)} \rangle + B, 2B) \rightarrow \langle b^{(j)} \rangle$ for each $j \in [K]$. To avoid $O(\log_2(K))$ communication rounds, parties calculate $\sum_j(\langle b^{(j)} \rangle)$ and call the comparison protocol again as $\mathsf{LTC}(\sum_j \langle b^{(j)} \rangle, K) \rightarrow \langle \bar{b} \rangle$. $\sum_j b^{(j)} < K$ if and only if at least one $b^{(j)} = 0$. Let $\langle b \rangle \leftarrow 1 - \langle \bar{b} \rangle$. If $\langle b \rangle$ is opened to be 0, the signing protocol needs to be restarted. The specification of the distributed rejection sampling $\mathsf{RejectS}$ is presented in Protocol 8. Further details on the LTC protocol can be found in [37, 53].

Here we show a complete description of the distributed signing protocol in Protocol 9.

6 Security Analysis

It has been proven that the concrete construction π_{PREP} realizes the functionality $\mathcal{F}_{\mathsf{PREP}}$ in many previous works, thus we omit the security analysis of the preprocessing phase, and refer the readers to [29]. The security of our threshold signatures is proven in the $\mathcal{F}_{\mathsf{PREP}}$ model, so the simulator $\mathcal{S}$ simulates the functionality for all adversaries. Concretely, $\mathcal{S}$ obtains the MAC key $\boldsymbol{\alpha}$ and secret shares of generated bits, triples which belong to corrupted parties, from simulating $\mathcal{F}_{\mathsf{PREP}}$. Besides, we also rely on the random oracle model. Therefore, the simulator $\mathcal{S}$ simulates the key generation, share-refreshment, and signing oracles, as well as the random oracle H and functionality $\mathcal{F}_{\mathsf{PREP}}$ for $\mathcal{A}$.

The security of our scheme can be demonstrated by the following theorem.

Theorem 2. *The threshold signature scheme described in Section 5 is proactively secure for any PPT adversary $\mathcal{A}$,*

$$\begin{aligned} \mathsf{Adv}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}(\mathcal{A}) \leq\ & \mathsf{Adv}_{\mathsf{MLWE}_{q,k,l,\eta}} + \mathsf{Adv}_{\mathsf{MLWE}_{q,\bar{k},k_2,\gamma_1}} + \\ & \sqrt{(Q_h + Q_s + 1) \cdot (\mathsf{Adv}_{\mathsf{MSIS}_{q,k_1,k_2,2\gamma_1}} + \mathsf{Adv}_{\mathsf{MSIS}_{q,k,l+1,2B}})} \\ & + \frac{Q_s \cdot (Q_h + Q_s) + Q_h + Q_s + 1}{|C|} + \epsilon_s \end{aligned} \tag{1}$$

with $\epsilon_s < 2^{-256}$ and $\bar{k} = k_1 + k + l$.

Proof. We present a simulator $\mathcal{S}$ that can solve the MLWE or MSIS problem with the assistance of $\mathcal{A}$, assuming that $\mathcal{A}$ wins in the game $\mathsf{Expt}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}$. Based on the hardness of the MLWE and MSIS problems, we can infer that the advantage of $\mathcal{A}$ is negligible.

Let $\Pr[\mathbf{G}_i]$ denote the probability that $\mathcal{A}$ wins in game $\mathbf{G}_i$. We provide security proof using the following series of games.

$\mathbf{G}_1$ In this game, $\mathcal{S}$ and $\mathcal{A}$ run the game $\mathsf{Expt}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}$. In the meanwhile, $\mathcal{S}$ keeps a table $\mathcal{T}_{\mathsf{H}}$ for random oracle H. When $\mathcal{A}$ queries H with the input of m, $\mathcal{S}$ first checks whether there exists an entry $(m, \mathsf{H}(m))$ or not in $\mathcal{T}_{\mathsf{H}}$. If yes, $\mathcal{S}$ sends $\mathsf{H}(m)$ to $\mathcal{A}$; otherwise $\mathcal{S}$ samples $\mathsf{H}(m) \overset{\$}{\leftarrow} C$, sends $\mathsf{H}(m)$ to $\mathcal{A}$ and adds $(m, \mathsf{H}(m))$ into table $\mathcal{T}_{\mathsf{H}}$. If $\mathcal{A}$ wins in this game, then it breaks the proactive security of our threshold signature scheme. Thus we have $\Pr[\mathbf{G}_1] = \mathsf{Adv}^{\mathsf{DSP}}_{\mathsf{EUF-CMA}}(\mathcal{A})$.

$\mathbf{G}_2$ This game is identical to $\mathbf{G}_1$, except for the answering of signing queries. Specifically, we modify the simulation strategy of the signing oracle as follows.

The simulator $\mathcal{S}$ chooses $\mathbf{y} \overset{\$}{\leftarrow} S_{\gamma}^{l+k}$ $\mathbf{r} \overset{\$}{\leftarrow} S_{\gamma_1}^{k_2}$. Since $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{PREP}}$ (Protocol 3) for $\mathcal{A}$, it knows $\{\mathbf{b}_i\}_{i\in[2,t]}$ in $\langle \mathbf{b} \rangle$. Since $\langle \mathbf{y} \rangle, \langle \mathbf{r} \rangle$ are computed from $\langle \mathbf{b} \rangle$, $\mathcal{S}$ also knows $\{\mathbf{y}_i, \mathbf{r}_i\}_{i\in[2,t]}$ and has the ability to compute

$$\mathbf{y}_1 \leftarrow \mathbf{y} - \sum_{i\in[2,t]} \mathbf{y}_i \mod q, \mathbf{r}_1 \leftarrow \mathbf{r} - \sum_{i\in[2,t]} \mathbf{r}_i \mod q \tag{2}$$

It uses $\mathbf{y}_1, \mathbf{r}_1$ to run the following steps of $\mathsf{DSign}(\cdot)$ honestly.

Analysis: After simulating $\mathcal{F}_{\mathsf{PREP}}$, $\mathcal{S}$ knows corrupted parties' shares $\{\mathbf{y}_i, \mathbf{r}_i\}_{i\in[2,t]}$. Thus the distribution of $\mathbf{y}_1, \mathbf{r}_1$ computed as eq. (2) is the same with that of $\mathbf{G}_1$. Thus the minor change has no effect on the advantage $\Pr[\mathbf{G}_2] = \Pr[\mathbf{G}_1]$.

$\mathbf{G}_3$ In this game, we continue to modify the simulation strategy of the signing oracle as follows.

At Step 6, $\mathcal{S}$ runs the $\mathsf{RejectS}$ protocol with $\mathcal{A}$ using dummy input $\mathbf{0}$ instead of using honestly generated $\mathbf{z}_1$. Note that the simulator can compute

$$\mathbf{z}_i \leftarrow \mathbf{y}_i + c \cdot \begin{bmatrix} \mathbf{s}_i \\ \mathbf{e}_i \end{bmatrix} \mod q \tag{3}$$

for each $i \in [2,t]$ from simulating $\mathcal{F}_{\mathsf{PREP}}$. Thus $\mathcal{S}$ knows the dummy result b', such that $b' = 1$ if $\| \sum_{i\in[2,t]} \mathbf{z}_i \mod q\|_\infty < B$ and $b' = 0$ otherwise. After running $\mathsf{RejectS}$ protocol, $\mathcal{S}$ gets P_1's share in $\langle b' \rangle$ as the output.

At Step 8, it has to simulate P_1's share such that it is consistent with the honest result b, such that $b = 1$ if $\|\mathbf{z}\|_\infty < B$ and $b = 0$ otherwise where $\mathbf{z} \leftarrow \mathbf{y} + c \cdot \begin{bmatrix} \mathbf{s} \\ \mathbf{e} \end{bmatrix}$ is computed by the simulator. The simulation strategy is that $\mathcal{S}$ adds $b - b'$ to P_1's share of b' and $\boldsymbol{\alpha}(b - b')$ to P_1's share of corresponding MACs.

Analysis: In $\mathbf{G}_3$, $\mathcal{S}$ uses a dummy input to run $\mathsf{RejectS}$ protocol while it uses honestly generated $\mathbf{z}_1$ in $\mathbf{G}_2$. When we need to open shares $\langle a \rangle$ of any secret value a during running $\mathsf{RejectS}$ protocol, fresh shares of a random value are added to shares $\langle a \rangle$. That is all secrets of P_1 are perfectly hiding for the adversary $\mathcal{A}$. Thus $\mathcal{A}$ cannot distinguish whether P_1's input is honest $\mathbf{z}_1$ or not at Step 6. Moreover Step 7 guarantees that $\mathcal{S}$ can compute correct P_1's share of the dummy result b' since malicious behaviors will be checked at Step 7 except with probability ϵ_s. Finally the simulator adds $b - b'$ to P_1's share of b' and $\boldsymbol{\alpha}(b - b')$ to P_1's share of corresponding MAC, making the honest result be b at Step 8. Overall the advantage difference between $\mathbf{G}_2$ and $\mathbf{G}_3$ is soundness error in Step 7, $|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq \epsilon_s$.

$\mathbf{G}_4$ In this game, we continue to modify the simulation strategy of the signing oracle as follows.

(a) With the probability of $1/M$, the simulator $\mathcal{S}$ does as follows: it chooses $c \xleftarrow{\$} C, \mathbf{z} \xleftarrow{\$} S_B^{l+k}$ and computes $\mathbf{w} \leftarrow [\mathbf{A}|\mathbf{I}_k]\,\mathbf{z} - c \cdot \mathbf{t}$, as well as its commitment value

$$com \leftarrow \begin{bmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \end{bmatrix} \cdot \mathbf{r} + \begin{bmatrix} \mathbf{0} \\ \mathbf{w} \end{bmatrix} \tag{4}$$

It then programs $\mathsf{H}(com, \mu) \to c$. Note that $\mathcal{S}$ can compute corrupted parties' shares $\{com_i\}_{i\in[2,t]}$ of com since it knows $\{\mathbf{r}_i, \mathbf{y}_i\}_{i\in[2,t]}$ from simulating $\mathcal{F}_{\mathsf{PREP}}$. At Step 5, the simulator broadcasts $com_1 \leftarrow com - \sum_{i\in[2,t]} com_i \mod q$. At Step 8, the simulator broadcasts

$$\mathbf{z}_1 \leftarrow \mathbf{z} - \sum_{i\in[2,t]} \mathbf{z}_i \mod q. \tag{5}$$

where $\{\mathbf{z}_i\}_{i\in[2,t]}$ are computed as eq. (3).

(b) With probability $1 - 1/M$, the simulator $\mathcal{S}$ does as follows: it samples a random vector $\mathbf{w} \xleftarrow{\$} \mathbb{R}_q^{l+k}$ and computes its commitment as eq. (4). Similarly, $\mathcal{S}$ broadcasts $com_1 \leftarrow com - \sum_{i\in[2,t]} com_i \mod q$ at Step 5. Then it follows $\mathbf{G}_3$ until restarts in Step 8 or aborts in Step 7.

Analysis: With the probability of $1/M$, the simulator simulates $(c, \mathbf{z})$ by directly sampling them from uniform distributions, and computes $\mathbf{w}, com$ which are distributed identically to those in $\mathbf{G}_3$. One concern is com may be opened maliciously at Step 5, becoming com' and generating a different challenge c'. However these malicious behaviors will be checked at Step 7, thus $\mathbf{z}_1$ computed by $\mathcal{S}$ as eq. (5) has the same distribution with that in $\mathbf{G}_3$. Finally the adversary $\mathcal{A}$ has the same views in the two games as long as no collision occurs in reprogramming H. The probability is at most $\frac{Q_s\cdot(Q_h+Q_s)}{|C|}$. With the probability of $1 - 1/M$, the simulator commits to an uniform $\mathbf{w}$, instead of honestly generated $[\mathbf{A}|\mathbf{I}_k]\,\mathbf{y}$. Note that the adversary cannot distinguish this simulated com from the real one due to the hiding property of commitment. Following [51], its based on the $\mathsf{MLWE}_{q,\bar{k},k_2,\gamma_1}$ assumption on input $\begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix}$ with $\bar{k} = k_1 + k + l$. Overall, we have

$$|\Pr[\mathbf{G}_4] - \Pr[\mathbf{G}_3]| \leq \mathsf{Adv}_{\mathsf{MLWE}_{q,\bar{k},k_2,\gamma_1}} + \frac{Q_s \cdot (Q_h + Q_s)}{|C|}$$

$\mathbf{G}_5$ This game is identical to $\mathbf{G}_4$ except in answering share-refreshment queries. That is to say we change the simulation strategy of the share-refreshment oracle as follows.

Upon receiving a share-refreshment query on input $\mathcal{P}'_c = \{P_1, P_3, \cdots, P_t\}$, it aims to change the $\mathcal{P}_c$-corrupted state into the $\mathcal{P}'_c$-corrupted state. The simulator plays the roles of honest parties $\{P_1, P_2, P_{t+1}, \cdots, P_n\}$ to simulate the $\mathsf{ShareRefresh}(\cdot)$ protocol using dummy input $\mathbf{0}$ instead of using honest secret key shares. $\mathcal{S}$ chooses random values $(\mathbf{s}_1, \mathbf{e}_1, \gamma(\mathbf{s})_1, \gamma(\mathbf{e})_1)$ as P_1's new secret key shares, and then send them to $\mathcal{A}$.

Analysis: In this game, $\mathcal{S}$ uses a dummy input to run $\mathsf{ShareRefresh}$ protocol while it uses honest $\langle sk \rangle$ in the last game. When we need to open shares of any secret value during each execution of Mult protocol, fresh shares of a random value are added to the secret value. Thus the adversary $\mathcal{A}$ cannot distinguish whether $\mathcal{S}$ contributed correct input. (Note that $\mathcal{S}$ can pass the $\mathsf{MACCheck}_m$ process since it knows the complete $\boldsymbol{\alpha}$ from simulating $\mathcal{F}_{\mathsf{PREP}}$.) After a real execution of $\mathsf{ShareRefresh}$, P_1's shares $(\mathbf{s}_1, \mathbf{e}_1, \gamma(\mathbf{s})_1, \gamma(\mathbf{e})_1)$ are added to random values, which are uniformly distributed. Thus the adversary $\mathcal{A}$'s views in two games are identical. We have $\Pr[\mathbf{G}_5] = \Pr[\mathbf{G}_4]$.

$\mathbf{G}_6$ This game is identical to $\mathbf{G}_5$ except in answering the only key generation query. That is to say we change the simulation strategy of the key generation oracle as follows.

The simulator picks random matrices $\mathbf{A} \xleftarrow{\$} \mathbb{R}_q^{k \times l}$, $\mathbf{A}' \xleftarrow{\$} \mathbb{R}_q^{k_1 \times (k_2 - k_1)}$, $\mathbf{A}'' \xleftarrow{\$} \mathbb{R}_q^{k_3 \times (k_2 - k_1)}$ and vector $\mathbf{t} \xleftarrow{\$} \mathbb{R}_q^{k}$. At Step 1 of $\mathsf{DKeyGen}$ protocol, it samples $g_1 \xleftarrow{\$} C$ and broadcasts g_1. Upon receiving $\{g_i\}_{i \in [2,t]}$, the simulator searches H's table $\mathcal{T}_{\mathsf{H}}$ to find the corresponding pre-images $\{\mathbf{A}_i, \mathbf{A}'_i, \mathbf{A}''_i\}_{i \in [2,t]}$. It computes $\mathbf{A}_1 \leftarrow \mathbf{A} - \sum_{i \in [2,t]} \mathbf{A}_i \mod q$, $\mathbf{A}'_1 \leftarrow \mathbf{A}' - \sum_{i \in [2,t]} \mathbf{A}'_i \mod q$, $\mathbf{A}''_1 \leftarrow \mathbf{A}'' - \sum_{i \in [2,t]} \mathbf{A}''_i \mod q$ and broadcasts $\mathbf{A}_1, \mathbf{A}'_1, \mathbf{A}''_1$ at Step 1. After extracting $\{\mathbf{s}_i, \mathbf{e}_i\}_{i \in [2,t]}$ from simulating $\mathcal{F}_{\mathsf{PREP}}$, the simulator computes $\mathbf{t}_1 \leftarrow \mathbf{t} - \sum_{i \in [2,t]} (\mathbf{A}\mathbf{s}_i + \mathbf{e}_i) \mod q$ and broadcasts it at Step 3.

Analysis: Based on $\mathsf{MLWE}_{q,k,l,\eta}$ assumption, the adversary cannot distinguish uniform $\mathbf{t} \xleftarrow{\$} \mathbb{R}_q^k$ and MLWE samples $\mathbf{t} \leftarrow \mathbf{A}\mathbf{s} + \mathbf{e}$. Thus the distributions of $\mathbf{A}_1, \mathbf{t}_1$ are computationally close to those of $\mathbf{G}_5$. Overall we have

$$|\Pr[\mathbf{G}_6] - \Pr[\mathbf{G}_5]| \leq \mathsf{Adv}_{\mathsf{MLWE}_{q,k,l,\eta}}$$

Forking Lemma. As in $\mathbf{G}_6$ the combined public key $(\mathbf{A}, \mathbf{t})$ is uniformly distributed in $\mathbb{R}_q^{k \times l} \times \mathbb{R}_q^k$, it is also an instance of $\mathsf{MSIS}_{q,k,l+1,\gamma'}$ problem on input $[\mathbf{A}|\mathbf{t}]$. Due to uniform $\mathbf{A}', \mathbf{A}''$, $\begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix}$ also follows the uniform distribution. Now we prove the theorem by constructing $\mathcal{S}$ that (1) either breaks binding property of commitment, finding a solution to $\mathsf{MSIS}_{q,k_1,k_2,2\gamma_1}$ on input $\mathbf{A}^{(1)}$ following [51] or (2) finds a solution to $\mathsf{MSIS}_{q,k,l+1,2B}$ on input $[\mathbf{A}|\mathbf{t}]$.

If the adversary $\mathcal{A}$ wins in $\mathbf{G}_6$, i.e., it outputs a valid signature forgery $(\mu^*, (com^*, \mathbf{z}^*, \mathbf{r}^*))$, it must have queried $c^* \leftarrow \mathsf{H}(com^*, \mu^*)$. A standard forking lemma argument [54] shows that with probability ϵ_{frk} we immediately get two forgeries $(com^*, c^*, \mathbf{z}^*, \mathbf{r}^*, \mu^*)$ and $(com', c', \mathbf{z}', \mathbf{r}', \mu')$ by rewinding $\mathcal{A}$, where the probability ϵ_{frk} and $\Pr[\mathbf{G}_6]$ satisfy $\Pr[\mathbf{G}_6] \leq \frac{Q_h+Q_s+1}{|C|} + \sqrt{(Q_h+Q_s+1)\cdot \epsilon_{\mathsf{frk}}}$. The two forgeries satisfy $com^* = com'$, $\mu^* = \mu'$ and $c^* \neq c'$. If $\mathbf{w}^* \neq \mathbf{w}'$, we have

$$\begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix} \cdot \mathbf{r}^* + \begin{bmatrix}\mathbf{0}\\ \mathbf{w}^*\end{bmatrix} = \begin{bmatrix}\mathbf{A}^{(1)}\\ \mathbf{A}^{(2)}\end{bmatrix} \cdot \mathbf{r}' + \begin{bmatrix}\mathbf{0}\\ \mathbf{w}'\end{bmatrix}.$$

Then we have $\mathbf{r}^* \neq \mathbf{r}'$, and $\mathbf{A}^{(1)}(\mathbf{r}^* - \mathbf{r}') = \mathbf{0}$. Thus $\mathcal{S}$ can solve $\mathsf{MSIS}_{q,k_1,k_2,2\gamma_1}$ problem on input $\mathbf{A}^{(1)}$. If $\mathbf{w}^* = \mathbf{w}'$, we have $[\mathbf{A}|\mathbf{I}_k]\,\mathbf{z}^* - c^* \cdot \mathbf{t} = [\mathbf{A}|\mathbf{I}_k]\,\mathbf{z}' - c' \cdot \mathbf{t}$. That can be rewritten as

$$[\mathbf{A}|\mathbf{t}|\mathbf{I}_k] \begin{bmatrix}\mathbf{z}^{*(1)} - \mathbf{z}'^{(1)}\\ c' - c^*\\ \mathbf{z}^{*(2)} - \mathbf{z}'^{(2)}\end{bmatrix} = \mathbf{0}$$

That is $\mathcal{S}$ solves $\mathsf{MSIS}_{q,k,l+1,2B}$ problem on input $[\mathbf{A}|\mathbf{t}]$. Thus we have $\epsilon_{\mathsf{frk}} \leq \mathsf{Adv}_{\mathsf{MSIS}_{q,k_1,k_2,2\gamma_1}} + \mathsf{Adv}_{\mathsf{MSIS}_{q,k,l+1,2B}}$. Finally $\mathcal{A}$'s advantage is bounded as eq. (1).

7 Performance

7.1 Efficiency Analysis

Table 2 and Table 3 present the theoretical costs of our distributed key generation (DKG), the share-refreshment and the distributed signing protocols, as well as the concert costs under the parameter settings specified in Table 4. Note that the parameters are set to achieve the security strength of 123 classical bits and 112 quantum bits. These protocols require a specific number of shared bits and multiplication triples generated during the preprocessing phase. Additionally, the online communication rounds and complexity are displayed, which represents the amount of data sent by each party. Here, $K = (k+l)\cdot N$, where k and l are the height and width of the random matrix $\mathbf{A}$ in the public key, N is the dimension of the polynomial corresponding to the ring, $n_q = \log_2 q$, and other parameters are described in Table 1.

The distributed rejection sampling sub-protocol, $\mathsf{RejectS}$, is the most expensive component of our signing protocol, DSign. In $\mathsf{RejectS}$, we utilize the comparison protocol, LTC, from [37]. This protocol leverages recent advancements in the generation and deployment of doubly authenticated shared bits (daBits [55]) and extended doubly authenticated bits (edaBits [53]). These correspond to shared integers in the arithmetic domain, whose bit decomposition is shared in the binary domain. One daBit can be generated from one shared bit, while one edaBit in $\mathbb{Z}_q$ can be generated from n_q shared bits. The generation of these bits occurs during the preprocessing phase.

We count the number of shared bits required in LTC, $\mathsf{RejectS}$, and DSign protocols, as shown in Table 3. The theoretical costs of the comparison protocol, LTC, from [37], are presented first. Since our $\mathsf{RejectS}$ invokes LTC $K+1$ times, it requires $K+1$ times as many bits and triples as LTC. The first K executions of LTC are called in parallel, followed by one additional call, resulting in a total of $2\log_2 n_q$ rounds. The sum of the theoretical costs of $\mathsf{RejectS}$ and the remaining parts is the total cost of DSign, shown in the last row of Table 3.

Note that in our t-out-of-n threshold signature scheme, firstly we ask all n parties to run the preprocessing protocol once, to generate a large amount of t-out-of-n authenticated shared bits and Beaver triples for the following distributed key generation phase (which is run once) and signing phase. For instance, the preprocessing runs once to generate one million triples, which can support the execution of signing protocol for 5 times. If the shared triples or bits are used up, the parties participating in the signing protocol currently can rerun the preprocessing protocol to generate t-out-of-t' shared triples or bits, where t' is the number of currently-online parties. As above, we have analyzed the number of bits and triples needed in each protocol in a theoretical way.

Table 2: The costs of the DKG and refreshment protocols

Protocols	Shared Bits	Shared Triples	Rounds	Comm.
DKeyGen	$n_\eta K$	0	5	$4n_qK$
ShareRefresh	$K + mn_q$	$(m+1)K$	4	$(m+1)n_qK$
DKeyGen	4096	0	5	23 KB
ShareRefresh	2324	26624	4	43 KB

Table 3: The costs of our distributed signing protocol and its main components

Protocols	Shared Bits	Shared Triples	Rounds	Comm.
LTC	n_q+1	$n_q\log_2 n_q$	$\log_2 n_q$	$2\lambda n_q$
RejectS	$(K+1)(n_q+1)$	$(K+1)n_q\log_2 n_q$	$2\log_2 n_q$	$2(K+1)\lambda n_q$
DSign - in total	$(K+1)(n_q+1)+K_2n_\gamma$	$(K+1)n_q\log_2 n_q$	$2\log_2 n_q+5$	$(2(K+1)\lambda+3K)n_q$
LTC	24	115	5	0.7 KB
RejectS	49152	235520	10	1.4 MB
DSign - in total	59392	235520	15	1.417 MB

Table 4: The setting of parameters

Parameters	N	q	m	τ	γ	γ_1
Values	256	8380417	12	39	2^8	2^8
Parameters	(k,l)	(k_1,k_2)	η	β	M	
Values	$(4,4)$	$(3,5)$	2	78	3.38	

7.2 Implementation

Our implementation is built on the top of MP-SPDZ [48][5]. More concretely, we modify the Reshare process of their codes to implement our preprocessing phase to generate t-out-of-n shares of shared bits and multiplication triples in $\mathbb{Z}_q$. Note that in this subsection, we will present the amortized preprocessing runtime for each protocol. Assume that the preprocessing can generate the number of triples required for 5 executions of signing at once, then the amortized time is $t_{\mathsf{all}}/5$ for DSign-Preprocessing, where t_{all} is the runtime it takes for the preprocessing to run once.

All the following experiments are performed on a set of Alibaba Cloud ecs.c7.2xlarge instances with a 2.70GHz processor and 16 gigabytes of RAM. All our programs are implemented in C++ for 16 threads and run in two network settings, namely, LAN and WAN settings. For benchmarks in the LAN setting, we created a set of 12 nodes, among which the bandwidth was generally between 300 and 400 MBps, and the round-trip latency was approximately 0.3 ms. For WAN setting, we chose 3 nodes located in 3 different cities inside one country, among which the bandwidth was 44 MBps, and the round-trip latency was approximately 40 ms.

LAN Benchmarks The experiment results for preprocessing and online phases of DKeyGen and ShareRefresh protocols can be seen in a line chart in Fig. 2. The distributed key generation runs only once, but the shares-refreshment runs periodically (once a month or every six months), thus we are more concerned about the efficiency of refreshment. In a two-party setting, its online phase only takes 0.24 seconds and in 12-party setting, it takes 6.39 seconds.

The experiment results for the DSign protocol are shown in a bar chart in Fig. 3. It shows the running times of the preprocessing and online phases, respectively. As the most time-consuming part in DSign online phase, the green bars represent the running times of RejectS protocol, which accounts for 80% - 90% of the total signing time. Due to its efficiency being mainly dependent on the comparison protocol, if more efficient LTC protocol emerges, the efficiency of our RejectS protocol will be improved significantly.

To compare our scheme with existing works that satisfy functional interchangeability, we consider two methods, namely, TFHE and generic MPC (GMPC),

[5] `https://github.com/data61/MP-SPDZ`

Fig. 2: The running times (seconds) of the distributed key generation and refreshment protocols over LAN

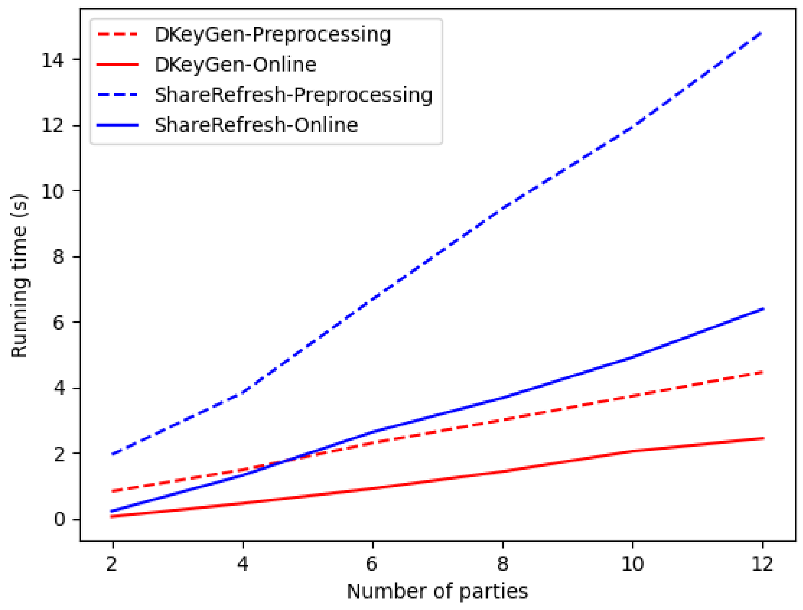
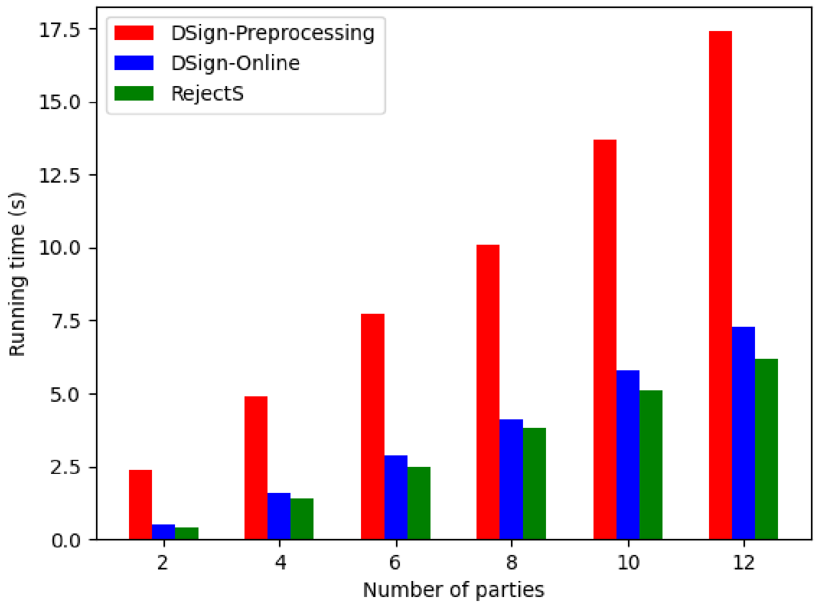


Fig. 3: The running times (seconds) of the distributed signing protocol over LAN



as analyzed in Section **??**. For TFHE-based works [21, 22], they are too theoretical to give a realistic efficiency analysis. For GMPC-based work [25], they tried to use garbled circuit (GC) to implement distributed rejection sampling process. Table 5 first compares our RejectS protocol with GC-based construction, we can perform most of the complex computations in preprocessing phase, resulting in a significant improvement in efficiency during the online phase, up to 30 times faster for a two-party setting. Moreover, Table 5 also shows that our DSign protocol is approximately 20 times faster than their distributed signing protocol. Even adding up the costs spent on both offline and online phases, our protocols costs 2.9 seconds for a two-party setting, that is 4 times faster than theirs. The last column "Threshold" shows our scheme supports arbitrary threshold access structure with $t \leq n$, instead of only full-threshold $t = n$.

Table 5: Comparison with GMPC-based scheme over LAN

Schemes	Runtime (s)	Comm.	Threshold
GC-based RejectS [25]	12	65 MB	$t = n$
Our RejectS	0.4	1.4 MB	$t \leq n$
DSign [25]	12.4	71 MB	$t = n$
Our DSign	0.5	1.417 MB	$t \leq n$

WAN Benchmarks In Table 6, we demonstrate the runtimes of three protocols under two kinds of threshold settings over WAN. Under $(t, n) = (2, 3)$ threshold setting, the DKeyGen, ShareRefresh protocols involve three participants, thus require more runtime compared to $(t, n) = (2, 2)$ setting with two participants. However, the runtime of the DSign protocol remains almost the same, as it involves two participants in both 2-out-of-2 and 2-out-of-3 threshold settings.

Table 6: The runtimes in seconds of our protocols over WAN

Threshold setting (t, n)	DKeyGen	ShareRefresh	DSign
$(2, 2)$	0.36	0.27	1.5
$(2, 3)$	0.59	0.53	1.65

In real-world deployments, optimizing for communication rounds is a common goal since latency may be the most time-consuming resource. Our scheme is practical even when latency is taken into account, thanks to our novel ideas of avoiding distributed hashing and reducing communication rounds for the RejectS protocol. Specifically, the distributed evaluation of hashing requires at least 10 rounds as shown in [36], while our protocol locally evaluates hashing without communication. Additionally, if generic MPC is used in a trivial way to run

RejectS, the number of communication rounds will be more than 50, whereas our approach only requires 10 rounds.

Table 7: Performance comparison with generic MPC for distributed signing protocol in the 2-out-of-3 threshold setting. Runtimes are in seconds.

Protocols	Runtime		Rounds	Comm.
	LAN	WAN		
GMPC-DSign	12.4	15.7	60	71 MB
Our DSign	0.5	1.65	15	1.417 MB

Table 7 shows the improvement of our distributed signing DSign protocol over GMPC method in both LAN and WAN settings. In particularly, the costs for GMPC-DSign are roughly estimated by combining the overhead of SHA-3 evaluation [36] and rejection sampling in GMPC [25]. In summary, our savings in terms of rounds and communication costs have enabled our protocol to remain 10 times faster in the WAN network setting.

Protocol 4 The preprocessing protocol π_{PREP}

1. It specifies the global MAC key α, and distributes a t-out-of-n share α_i to each party P_i, where $\alpha \leftarrow \sum_{i\in S} \lambda_{i,S} \cdot \alpha_i \mod q$ for any subset $S \subseteq [n]$ with $|S| \geq t$.
   (a) Each party P_i samples an uniform $\alpha'_i \in \mathbb{Z}_q$. Broadcast the ciphertext $e_{\alpha'_i} \leftarrow \mathsf{Enc}(\alpha'_i)$. Define $\alpha = \sum_{i=1}^{n} \alpha'_i$.
   (b) Each party P_i performs Shamir's secret sharing to re-share α'_i: generate t-out-of-n shares $\{\alpha_{i,j}\}_{j\in[n]}$, send $\alpha_{i,j}$ to P_j.
   (c) Each party P_i generates a zero-knowledge proof to prove the validity of $e_{\alpha'_i}$. The protocol aborts if any proof fails.
   (d) All parties set $e_\alpha \leftarrow \sum_{i=1}^{n} e_{\alpha'_i}$, and each party P_i locally computes $\alpha_i \leftarrow \sum_{j=1}^{n} \alpha_{j,i}$.
2. Output a set of "multiplication triples": $\{\langle\bar{x}\rangle, \langle\bar{y}\rangle, \langle\bar{z}\rangle\}$ such that $\bar{x}, \bar{y} \xleftarrow{\$} \mathbb{Z}_q$ and $\bar{z} = \bar{x} \cdot \bar{y}$.
   (a) Each party P_i samples two uniform $\bar{\mathbf{x}}'_i, \bar{\mathbf{y}}'_i \in \mathbb{Z}_q^K$. Broadcast two ciphertexts $e_{\bar{\mathbf{x}}'_i} \leftarrow \mathsf{Enc}(\bar{\mathbf{x}}'_i), e_{\bar{\mathbf{y}}'_i} \leftarrow \mathsf{Enc}(\bar{\mathbf{y}}'_i)$. Define: $\bar{\mathbf{x}} = \sum_{i=1}^{n} \bar{\mathbf{x}}'_i, \bar{\mathbf{y}} = \sum_{i=1}^{n} \bar{\mathbf{y}}'_i$.
   (b) Each party P_i performs Shamir's secret sharing to re-share $\bar{\mathbf{x}}'_i, \bar{\mathbf{y}}'_i$: generate t-out-of-n shares $\{\bar{\mathbf{x}}_{i,j}\}_{j=1}^{n}, \{\bar{\mathbf{y}}_{i,j}\}_{j=1}^{n}$, send $\bar{\mathbf{x}}_{i,j}, \bar{\mathbf{y}}_{i,j}$ to P_j.
   (c) Each party P_i generates a zero-knowledge proof to prove the validity of $e_{\bar{\mathbf{x}}'_i}$ and $e_{\bar{\mathbf{y}}'_i}$. The protocol aborts if any proof fails.
   (d) All parties set $e_{\bar{\mathbf{x}}} \leftarrow \sum_{i=1}^{n} e_{\bar{\mathbf{x}}'_i}, e_{\bar{\mathbf{y}}} \leftarrow \sum_{i=1}^{n} e_{\bar{\mathbf{y}}'_i}$. And each party P_i locally computes $\bar{\mathbf{x}}_i \leftarrow \sum_{j=1}^{n} \bar{\mathbf{x}}_{j,i}, \bar{\mathbf{y}}_i \leftarrow \sum_{j=1}^{n} \bar{\mathbf{y}}_{j,i}$. Obviously, $\{\bar{\mathbf{x}}_i\}$ and $\{\bar{\mathbf{y}}_i\}$ are t-out-of-n shares of $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ repectively.
   (e) Let $\bar{\mathbf{z}} = \bar{\mathbf{x}} \cdot \bar{\mathbf{y}}$, all parties set $e_{\bar{\mathbf{z}}} \leftarrow e_{\bar{\mathbf{x}}} \cdot e_{\bar{\mathbf{y}}}$. The parties jointly run the Reshare protocol (Protocol 2): $\{\bar{\mathbf{z}}_i\} \leftarrow \mathsf{Reshare}(e_{\bar{\mathbf{z}}})$.
   (f) All parties set $e_{\gamma(\bar{\mathbf{x}})} \leftarrow e_{\bar{\mathbf{x}}} \cdot e_\alpha, e_{\gamma(\bar{\mathbf{y}})} \leftarrow e_{\bar{\mathbf{y}}} \cdot e_\alpha$, and $e_{\gamma(\bar{\mathbf{z}})} \leftarrow e_{\bar{\mathbf{z}}} \cdot e_\alpha$.
   (g) The parties jointly run the Reshare protocol: $\{\gamma(\bar{\mathbf{x}})_i\}_{i\in[n]} \leftarrow \mathsf{Reshare}(e_{\gamma(\bar{\mathbf{x}})})$, $\{\gamma(\bar{\mathbf{y}})_i\}_{i\in[n]} \leftarrow \mathsf{Reshare}(e_{\gamma(\bar{\mathbf{y}})})$, and $\{\gamma(\bar{\mathbf{z}})_i\}_{i\in[n]} \leftarrow \mathsf{Reshare}(e_{\gamma(\bar{\mathbf{z}})})$.
   (h) Ouputs t-out-of-n authenticated multiplication triples: $\langle\bar{\mathbf{x}}\rangle = (\{\bar{\mathbf{x}}_i\}, \{\gamma(\bar{\mathbf{x}})_i\})$, $\langle\bar{\mathbf{y}}\rangle = (\{\bar{\mathbf{y}}_i\}, \{\gamma(\bar{\mathbf{y}})_i\})$, and $\langle\bar{\mathbf{z}}\rangle = (\{\bar{\mathbf{z}}_i\}, \{\gamma(\bar{\mathbf{z}})_i\})$.
3. Output a set of shared bits: $\langle\mathbf{b}\rangle$ such that $\mathbf{b}$ is a vector of bits.
   (a) Each party P_i samples an uniform $\mathbf{c}_i \in \mathbb{Z}_q^K$. Broadcast the ciphertext $e_{\mathbf{c}_i} \leftarrow \mathsf{Enc}(\mathbf{c}_i)$. Define: $\mathbf{c} = \sum_{i=1}^{n} \mathbf{c}_i$.
   (b) Each party P_i generates a zero-knowledge proof to prove the validity of $e_{\mathbf{c}_i}$. The protocol aborts if any proof fails.
   (c) All parties set $e_{\mathbf{c}} \leftarrow \sum_{i=1}^{n} e_{\mathbf{c}_i}$, compute $e_{\mathbf{c}^2} \leftarrow e_{\mathbf{c}} \cdot e_{\mathbf{c}}$.
   (d) The parties jointly decrypt $e_{\mathbf{c}^2}$ and thereby obtain $\mathbf{d} = \mathbf{c}^2$.
   (e) If any slot position in $\mathbf{d}$ equals zero, then set it to one.
   (f) A fixed square root $\mathbf{t}$ of $\mathbf{d}$ is taken, say the one for which each slot position is odd when represented in $[1, \cdots, q-1]$.
   (g) Compute $e_{\mathbf{v}} \leftarrow \mathbf{t}^{-1} \cdot e_{\mathbf{c}}$. This is encryption of $\mathbf{v} = \mathbf{t}^{-1} \cdot \mathbf{c}$, that is a message for which each slot position contains $\{-1, 1\}$, bar the one which we replaced in step e).
   (h) All parties set $e_{\gamma(\mathbf{v})} \leftarrow e_{\mathbf{v}} \cdot e_\alpha$.
   (i) The parties jointly run the Reshare protocol: $\{\mathbf{v}_i\} \leftarrow \mathsf{Reshare}(e_{\mathbf{v}})$, and $\{\gamma(\mathbf{v})_i\} \leftarrow \mathsf{Reshare}(e_{\gamma(\mathbf{v})})$.
   (j) Outputs $\langle\mathbf{b}\rangle \leftarrow (1/2) \cdot (\langle\mathbf{v}\rangle + 1)$.

Protocol 5 $\mathsf{MACCheck}_m(\langle \mathbf{t} \rangle, \mathbf{t}) \rightarrow 1/0$

For each $i \in [n]$, each party P_i takes opened $\mathbf{t} = (t^{(1)}, \cdots, t^{(K)})$, $\mathbf{t}$'s MAC value share $\gamma(\mathbf{t})_i^{(1)}, \cdots, \gamma(\mathbf{t})_i^{(m)}$ in $\langle \mathbf{t} \rangle$ as well as pre-processed MAC key share $\boldsymbol{\alpha}_i = (\alpha_i^{(1)}, \cdots, \alpha_i^{(m)})$ as the input, does the following:

1. The parties jointly sample a random vector $(r^{(1)}, \cdots, r^{(K)})$, compute $t_{\mathbf{r}} \leftarrow \sum_{j=1}^{K} r^{(j)} \cdot t^{(j)} \mod q$.
2. For each $\xi \in [m]$, each party P_i splits $\gamma(\mathbf{t})_i^{(\xi)}$ into a vector $(\gamma(t^{(1)})_i^{(\xi)}, \cdots, \gamma(t^{(K)})_i^{(\xi)})$ and computes $\gamma_i^{(\xi)} \leftarrow \sum_{j=1}^{K} r^{(j)} \cdot \gamma(t^{(j)})_i^{(\xi)} \mod q$, $\sigma_i^{(\xi)} \leftarrow \lambda_{i,[n]} \cdot (\gamma_i^{(\xi)} - \alpha_i^{(\xi)} \cdot t_{\mathbf{r}}) \mod q$ where $\lambda_{i,[n]} = \prod_{j \in [n], j \neq i} \frac{j}{j-i}$.
3. Each party P_i computes $c_i \leftarrow \mathsf{H}(\{\sigma_i^{(\xi)}\}_{\xi=1}^{m})$, then broadcasts c_i.
4. Each party broadcasts $\{\sigma_i^{(\xi)}\}_{\xi=1}^{m}$, and checks if $\mathsf{H}(\{\sigma_{\bar{i}}^{(\xi)}\}_{\xi=1}^{m}) = c_{\bar{i}}$ for all $\bar{i} \neq i$. If not, output 0 and abort. From this, all parties obtain $\sigma^{(\xi)} \leftarrow \sum_{i=1}^{n} \sigma_i^{(\xi)} \mod q$ for $\xi \in [m]$.
5. If $\sigma^{(\xi)} = 0$ for each $\xi \in [m]$, the parties output 1; otherwise output 0.

Protocol 6 $\mathsf{DKeyGen}(1^\lambda) \rightarrow (pk, \langle sk \rangle)$

Initialize: Each party P_i calls $\mathcal{F}_{\mathsf{PREP}}$ to get share $\boldsymbol{\alpha}_i = (\alpha_i^{(1)}, \cdots, \alpha_i^{(m)})$ of m MAC key, and its own share in $\langle \mathbf{b} \rangle$ where $\mathbf{b}$ is a binary vector.
Generate: On input the security parameter λ, the parties do the following

1. Each party P_i samples $\mathbf{A}_i \xleftarrow{\$} \mathbb{R}_q^{k \times l}$, $\mathbf{A}'_i \xleftarrow{\$} \mathbb{R}_q^{k_1 \times (k_2 - k_1)}$, $\mathbf{A}''_i \xleftarrow{\$} \mathbb{R}_q^{k_3 \times (k_2 - k_1)}$ and broadcasts $g_i \leftarrow \mathsf{H}(\mathbf{A}_i, \mathbf{A}'_i, \mathbf{A}''_i)$. After receiving $g_{\bar{i}}$ for each $\bar{i} \neq i$, party P_i broadcasts $\mathbf{A}_i, \mathbf{A}'_i, \mathbf{A}''_i$. After receiving $\mathbf{A}_{\bar{i}}, \mathbf{A}'_{\bar{i}}, \mathbf{A}''_{\bar{i}}$ from others, party P_i checks whether $g_{\bar{i}} = \mathsf{H}(\mathbf{A}_{\bar{i}}, \mathbf{A}'_{\bar{i}}, \mathbf{A}''_{\bar{i}})$ for each $\bar{i} \neq i$ or not. If not, party P_i outputs $\perp$ and aborts. Otherwise all parties generate $\mathbf{A} \leftarrow \sum_i \mathbf{A}_i \mod q, \mathbf{A}' \leftarrow \sum_i \mathbf{A}'_i \mod q$ and $\mathbf{A}'' \leftarrow \sum_i \mathbf{A}''_i \mod q$. Then construct $\mathbf{A}^{(1)} \leftarrow [\mathbf{I}_{k_1} | \mathbf{A}']$ and $\mathbf{A}^{(2)} \leftarrow [\mathbf{0} | \mathbf{A}'']$.
2. For each $j \in [K]$, the parties choose shares of n_η bits from $\langle \mathbf{b} \rangle$, that is $(\langle b^{(0)} \rangle, \cdots, \langle b^{(n_\eta - 1)} \rangle)$. Then for each $j \in [K]$, they compute $\langle s^{(j)} \rangle \leftarrow \sum_{\zeta=0}^{n_\eta - 1} 2^\zeta \cdot \langle b^{(\zeta)} \rangle - (\eta - 1)$. The shares of K integers, $\{\langle s^{(j)} \rangle\}_{j \in [K]}$ can form $\langle \mathbf{s} \rangle, \langle \mathbf{e} \rangle$.
3. The parties compute $\langle \mathbf{t} \rangle \leftarrow \mathbf{A} \langle \mathbf{s} \rangle + \langle \mathbf{e} \rangle$. Open $\langle \mathbf{t} \rangle$: each party P_i broadcasts $\mathbf{t}_i$ and receives $\{\mathbf{t}_{\bar{i}}\}_{\bar{i} \neq i}$ from others, then computes $\mathbf{t} \leftarrow \sum_i \mathbf{t}_i \mod q$.
4. The parties call $\mathsf{MACCheck}_m$ (Protocol 5) on input $\langle \mathbf{t} \rangle, \mathbf{t}$. If the result is 0, they output $\perp$ and abort. Otherwise, they output the public key $pk \leftarrow (\mathbf{A}, \mathbf{t}, \mathbf{A}^{(1)}, \mathbf{A}^{(2)})$, and each party stores its own share in $\langle \mathbf{s} \rangle, \langle \mathbf{e} \rangle$ as well as its shares of the MAC key $\boldsymbol{\alpha}_i$.

Protocol 7 $\mathsf{ShareRefresh}(\langle \mathbf{s} \rangle, \langle \mathbf{e} \rangle) \to \langle sk \rangle$

Initialize: The parties call $\mathcal{F}_{\mathsf{PREP}}$ to get shares $\langle \mathbf{b} \rangle$ of bits and shares $\langle \bar{x} \rangle, \langle \bar{y} \rangle, \langle \bar{x} \cdot \bar{y} \rangle$ of multiplication triples.

Generate: Each party P_i takes its own share in $\langle \mathbf{s} \rangle, \langle \mathbf{e} \rangle$ as the input, generates and stores its new share:

1. For each $j \in [K]$, the parties choose shares $\langle b^{(j)} \rangle$ of one bit from $\langle \mathbf{b} \rangle$, and run $\mathsf{Mult}(\langle b^{(j)} \rangle, 1 - \langle b^{(j)} \rangle) \to \langle x^{(j)} \rangle$ such that $x^{(j)} = 0$. The shares of K zeros, $\{\langle x^{(j)} \rangle\}_{j \in [K]}$ can form $\langle \mathbf{0} \rangle$ such that $\mathbf{0}$ is a K-dimension zero vector.
2. Compute $\langle sk \rangle \leftarrow \begin{bmatrix} \langle \mathbf{s} \rangle \\ \langle \mathbf{e} \rangle \end{bmatrix} + \langle \mathbf{0} \rangle$.
3. For each $\xi \in [m]$, use shared bits to generate a random value $\langle \alpha^{(\xi)'} \rangle \leftarrow \sum_{\zeta=1}^{n_q} 2^{\zeta} \langle b^{(\zeta)} \rangle$. Then m shares $\langle \alpha^{(\xi)'} \rangle$ can form $\langle \boldsymbol{\alpha}' \rangle$ where $\boldsymbol{\alpha}' \in \mathbb{Z}_q^m$, and run $\mathsf{Mult}(\langle sk \rangle, \langle \boldsymbol{\alpha}' \rangle) \to \langle sk \cdot \boldsymbol{\alpha}' \rangle$. Call $\mathsf{MACCheck}_m$ (Protocol 5) on input all opened values in Mult protocol and their shares.
4. Discard the MAC shares of $\langle sk \rangle, \langle sk \cdot \boldsymbol{\alpha}' \rangle$, let value shares of $\langle sk \cdot \boldsymbol{\alpha}' \rangle$ be the refreshed MAC shares of $\langle sk \rangle$. After this, $\boldsymbol{\alpha}'$ be the new MAC key, and the refreshed $\langle sk \rangle$ represents that each party holds a refreshed value share sk_i and MAC share $[sk \cdot \boldsymbol{\alpha}']_i$

Protocol 8 $\mathsf{RejectS}(\langle \mathbf{z} \rangle, B) \to \langle b \rangle$

Each party takes its own share in $\langle \mathbf{z} \rangle$ and public B as the input, and gets its own share in $\langle b \rangle$ as the output where $b = 1$ if $\|\mathbf{z}\|_\infty < B$ and $b = 0$ otherwise. It does the following:

1. Split $\langle \mathbf{z} \rangle$ into shares of K integers $(\langle z^{(1)} \rangle, \cdots, \langle z^{(K)} \rangle)$. For each $j \in [K]$, the parties run
$$\mathsf{LTC}(\langle z^{(j)} \rangle + B, 2B) \to \langle b^{(j)} \rangle.$$
2. Parties run $\mathsf{LTC}(\sum_j \langle b^{(j)} \rangle, K) \to \langle \bar{b} \rangle$.
3. Output $\langle b \rangle \leftarrow 1 - \langle \bar{b} \rangle$.

Protocol 9 $\mathsf{DSign}(\langle sk\rangle, pk, \mu) \to \sigma$

Initialize: Each party P_i calls $\mathcal{F}_{\mathsf{PREP}}$ to get its own share in $\langle \mathbf{b}\rangle$, where $\mathbf{b}$ is a binary vector.

Generate: The parties $\{P_i\}_{i\in[t]}$ take secret key shares $(\langle \mathbf{s}\rangle, \langle \mathbf{e}\rangle)$, public key pk and message μ as the input, they do the following:

1. Each party P_i first transfers their t-out-of-n shares in $\langle \mathbf{s}\rangle, \langle \mathbf{e}\rangle, \langle \mathbf{b}\rangle$ into t-out-of-t shares via multiplying $\lambda_{i,[t]}$.
2. For each $j \in [K]$, the parties choose shares of n_γ bits from $\langle \mathbf{b}\rangle$, that is $(\langle b^{(0)}\rangle, \cdots, \langle b^{(n_\gamma - 1)}\rangle)$. Then for each $j \in [K]$, they compute $\langle y^{(j)}\rangle \leftarrow \sum_{\zeta=0}^{n_\gamma-1} 2^\zeta \cdot \langle b^{(\zeta)}\rangle - (\gamma - 1)$. The shares of K integers, $\{\langle y^{(j)}\rangle\}_{j\in[K]}$ can form shares $\langle \mathbf{y}\rangle$.
3. For each $j \in [K_2]$, the parties choose shares of n_{γ_1} bits from $\langle \mathbf{b}\rangle$, that is $(\langle b^{(0)}\rangle, \cdots, \langle b^{(n_{\gamma_1} - 1)}\rangle)$. Then for each $j \in [K_2]$, they compute $\langle r^{(j)}\rangle \leftarrow \sum_{\zeta=0}^{n_{\gamma_1}-1} 2^\zeta \cdot \langle b^{(\zeta)}\rangle - (\gamma_1 - 1)$. The shares of K_2 integers, $\{\langle r^{(j)}\rangle\}_{j\in[K_2]}$ can form shares $\langle \mathbf{r}\rangle$.
4. The parties compute $\langle \mathbf{w}\rangle \leftarrow [\mathbf{A}|\mathbf{I}_k]\,\langle \mathbf{y}\rangle$ and the commitment $\langle com\rangle \leftarrow \begin{bmatrix} \mathbf{A}^{(1)} \\ \mathbf{A}^{(2)} \end{bmatrix} \cdot \langle \mathbf{r}\rangle + \begin{bmatrix} \mathbf{0} \\ \langle \mathbf{w}\rangle \end{bmatrix}$ with each share as com_i.
5. Open $\langle com\rangle$: Each party P_i broadcasts his commitment share com_i and receives $\{com_{\bar{i}}\}_{\bar{i}\neq i}$ from others, and thus computes $\mathbf{w}$'s commitment $com \leftarrow \sum_{i=1}^{n} com_i \mod q$. Then they locally compute the challenge $c \leftarrow \mathsf{H}(com, \mu)$.
6. The parties compute $\langle \mathbf{z}\rangle \leftarrow \langle \mathbf{y}\rangle + c \cdot \begin{bmatrix} \langle \mathbf{s}\rangle \\ \langle \mathbf{e}\rangle \end{bmatrix}$. Run $\mathsf{RejectS}(\langle \mathbf{z}\rangle, B) \to \langle b\rangle$ (Protocol 8).
7. The parties call $\mathsf{MACCheck}_m$ (Protocol 5) on input all opened values so far and their shares. If it outputs 0, they output $\perp$ and abort.
8. Open $\langle b\rangle$: each party P_i broadcasts b_i and receives $\{b_{\bar{i}}\}_{\bar{i}\neq i}$ from others, and computes $b \leftarrow \sum_{i=1}^{n} b_i \mod q$. Then call $\mathsf{MACCheck}_m$ on input $(\langle b\rangle, b)$. If it outputs 0, they output $\perp$ and abort. If $b = 0$, the parties go back to Step 2. Otherwise, they open $\langle \mathbf{z}\rangle$ and $\langle \mathbf{r}\rangle$ to get $\mathbf{z}, \mathbf{r}$. Output the signature $\sigma \leftarrow (com, \mathbf{z}, \mathbf{r})$ if $\mathsf{Verify}(pk, \mu, \sigma) \to 1$. Otherwise, output $\perp$ and abort.

Bibliography

[1] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Robust threshold DSS signatures," in *International Conference on the Theory and Application of Cryptographic Techniques - EUROCRYPT*, vol. 1070. Springer, 1996, pp. 354–371.

[2] V. Shoup, "Practical threshold signatures," in *International Conference on the Theory and Application of Cryptographic Techniques - EUROCRYPT*, ser. LNCS, vol. 1807. Springer, 2000, pp. 207–220.

[3] L. T. Brandão, M. Davidson, A. Vassilev *et al.*, "Nist roadmap toward criteria for threshold schemes for cryptographic primitives," 2020.

[4] S. Goldfeder, J. Bonneau, J. Kroll, and E. Felten, "Securing bitcoin wallets via threshold signatures," 2014.

[5] S. Goldfeder, R. Gennaro, H. Kalodner, J. Bonneau, J. A. Kroll, E. W. Felten, and A. Narayanan, "Securing bitcoin wallets via a new dsa/ecdsa threshold signature scheme," in *et al.*, 2015.

[6] R. Gennaro, S. Goldfeder, and A. Narayanan, "Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security," in *Applied Cryptography and Network Security: 14th International Conference, ACNS 2016, Guildford, UK, June 19-22, 2016. Proceedings 14*. Springer, 2016, pp. 156–174.

[7] Y. Harchol, I. Abraham, and B. Pinkas, "Distributed ssh key management with proactive rsa threshold signatures," in *Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings 16*. Springer, 2018, pp. 22–43.

[8] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller, "Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1348–1366.

[9] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[10] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1187–1201.

[11] Y. Lindell, "Fast secure two-party ECDSA signing," in *Annual International Cryptology Conference - CRYPTO*, vol. 10402. Springer, 2017, pp. 613–644.

[12] Y. Lindell and A. Nof, "Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody," in

Conference on Computer and Communications Security - CCS. ACM, 2018, pp. 1837–1854.

[13] R. Gennaro and S. Goldfeder, "Fast multiparty threshold ECDSA with fast trustless setup," in *Conference on Computer and Communications Security - CCS.* ACM, 2018, pp. 1179–1194.

[14] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Two-party ECDSA from hash proof systems and efficient instantiations," in *Annual International Cryptology Conference - CRYPTO*, vol. 11694. Springer, 2019, pp. 191–221.

[15] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled, "UC non-interactive, proactive, threshold ECDSA with identifiable aborts," in *Conference on Computer and Communications Security - CCS.* ACM, 2020, pp. 1769–1787.

[16] G. Castagnos, D. Catalano, F. Laguillaumie, F. Savasta, and I. Tucker, "Bandwidth-efficient threshold EC-DSA," in *International Conference on Practice and Theory of Public-Key Cryptography - PKC*, vol. 12111. Springer, 2020, pp. 266–296.

[17] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, pp. 1484–1509, 1997.

[18] NIST, "Call for proposals," https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization/Call-for-Proposals, 2016-12-21.

[19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *Trans. Cryptogr. Hardw. Embed. Syst.*, pp. 238–268, 2018.

[20] P.-A. Fouque, J. Hoffstein, P. Kirchner, V. Lyubashevsky, T. Pornin, T. Prest, T. Ricosset, G. Seiler, W. Whyte, and Z. Zhang, "Falcon: Fast-fourier lattice-based compact signatures over ntru," *Submission to the NIST's post-quantum cryptography standardization process*, vol. 36, 2018.

[21] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai, "Threshold cryptosystems from threshold fully homomorphic encryption," in *Annual International Cryptology Conference - CRYPTO*, vol. 10991. Springer, 2018, pp. 565–596.

[22] S. Agrawal, D. Stehlé, and A. Yadav, "Towards practical and round-optimal lattice-based threshold and blind signatures," *IACR Cryptol. ePrint Arch.*, p. 381, 2021.

[23] R. Bendlin, S. Krehbiel, and C. Peikert, "How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE," in *International Conference on Applied Cryptography and Network Security - ACNS*, vol. 7954. Springer, 2013, pp. 218–236.

[24] C. Gentry, C. Peikert, and V. Vaikuntanathan, "Trapdoors for hard lattices and new cryptographic constructions," in *Symposium on the Theory of Computing - STOC*, 2008, pp. 197–206.

[25] D. Cozzo and N. P. Smart, "Sharing the LUOV: threshold post-quantum signatures," in *IMA Conference on Cryptography and Coding - IMACC*, ser. LNCS, vol. 11929. Springer, 2019, pp. 128–153.

[26] X. Wang, S. Ranellucci, and J. Katz, "Global-scale secure multiparty computation," in *Conference on Computer and Communications Security - CCS*. ACM, 2017, pp. 39–56.

[27] C. Hazay, P. Scholl, and E. Soria-Vazquez, "Low cost constant round MPC combining BMR and oblivious transfer," *J. Cryptol.*, pp. 1732–1786, 2020.

[28] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias, "Multiparty computation from somewhat homomorphic encryption," in *Annual International Cryptology Conference - CRYPTO*, vol. 7417. Springer, 2012, pp. 643–662.

[29] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart, "Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits," in *ESORICS*, vol. 8134. Springer, 2013, pp. 1–18.

[30] M. Abe and S. Fehr, "Adaptively secure feldman VSS and applications to universally-composable threshold cryptography," in *Annual International Cryptology Conference - CRYPTO*, vol. 3152. Springer, 2004, pp. 317–334.

[31] C. Komlo and I. Goldberg, "FROST: flexible round-optimized schnorr threshold signatures," *IACR Cryptol. ePrint Arch*, p. 852, 2020.

[32] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi, "Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices," in *International Conference on Practice and Theory of Public-Key Cryptography - PKC*, vol. 12710. Springer, 2021, pp. 99–130.

[33] R. Ostrovsky and M. Yung, "How to withstand mobile virus attacks (extended abstract)," in *Symposium on Principles of Distributed Computing-PODC*. ACM, 1991, pp. 51–59.

[34] A. Nicolosi, M. N. Krohn, Y. Dodis, and D. Mazières, "Proactive two-party signatures for user authentication," in *NDSS 2003*. The Internet Society, 2003.

[35] M. Keller, P. Scholl, and N. P. Smart, "An architecture for practical actively secure MPC with dishonest majority," in *Conference on Computer and Communications Security - CCS*. ACM, 2013, pp. 549–560.

[36] M. Kraitsberg, Y. Lindell, V. Osheter, N. P. Smart, and Y. T. Alaoui, "Adding distributed decryption and key generation to a ring-lwe based CCA encryption scheme," in *ACISP*. Springer, 2019, pp. 192–210.

[37] E. Makri, D. Rotaru, F. Vercauteren, and S. Wagh, "Rabbit: Efficient comparison for secure multi-party computation," in *Financial Cryptography and Data Security - FC*, vol. 12674. Springer, 2021, pp. 249–270.

[38] P. Feldman, "A practical scheme for non-interactive verifiable secret sharing," in *Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1987, pp. 427–437.

[39] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual International Cryptology Conference - CRYPTO*, vol. 576. Springer, 1991, pp. 129–140.

[40] A. Chandramouli, A. Choudhury, and A. Patra, “A survey on perfectly secure verifiable secret-sharing,” *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 232:1–232:36, 2022.

[41] B. Rajabi and Z. Eslami, “A verifiable threshold secret sharing scheme based on lattices,” *Information Sciences*, vol. 501, pp. 655–661, 2019.

[42] H. Pilaram, T. Eghlidos, and R. Toluee, “An efficient lattice-based threshold signature scheme using multi-stage secret sharing,” *IET Information Security*, vol. 15, no. 1, pp. 98–106, 2021.

[43] R. E. Bansarkhani and M. Meziani, “An efficient lattice-based secret sharing construction,” in *Information Security Theory and Practice. Security, Privacy and Trust in Computing Systems and Ambient Intelligent Ecosystems - 6th IFIP WG 11.2 International Workshop, WISTP 2012, Egham, UK, June 20-22, 2012. Proceedings*, ser. Lecture Notes in Computer Science, I. G. Askoxylakis, H. C. Pöhls, and J. Posegga, Eds., vol. 7322. Springer, 2012, pp. 160–168.

[44] C. Boschini, A. Takahashi, and M. Tibouchi, “Musig-l: Lattice-based multi-signature with single-round online phase,” in *Annual International Cryptology Conference - CRYPTO*, vol. 13508. Springer, 2022, pp. 276–305.

[45] N. Fleischhacker, M. Simkin, and Z. Zhang, “Squirrel: Efficient synchronized multi-signatures from lattices,” in *Conference on Computer and Communications Security - CCS*. ACM, 2022, pp. 1109–1123.

[46] A. Langlois and D. Stehlé, “Worst-case to average-case reductions for module lattices,” *Des. Codes Cryptogr.*, vol. 75, no. 3, pp. 565–599, 2015.

[47] A. Shamir, “How to share a secret,” *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[48] M. Keller, V. Pastro, and D. Rotaru, “Overdrive: Making SPDZ great again,” in *International Conference on the Theory and Application of Cryptographic Techniques - EUROCRYPT*, vol. 10822. Springer, 2018, pp. 158–189.

[49] V. Lyubashevsky, “Lattice signatures without trapdoors,” in *International Conference on the Theory and Application of Cryptographic Techniques - EUROCRYPT*, vol. 7237. Springer, 2012, pp. 738–755.

[50] M. Fukumitsu and S. Hasegawa, “A lattice-based provably secure multisignature scheme in quantum random oracle model,” in *ProvSec*, vol. 12505. Springer, 2020, pp. 45–64.

[51] C. Baum, I. Damgård, V. Lyubashevsky, S. Oechsner, and C. Peikert, “More efficient commitments from structured lattice assumptions,” in *International Conference on Security and Cryptography for Networks - SCN*, vol. 11035. Springer, 2018, pp. 368–385.

[52] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, “Proactive secret sharing or: How to cope with perpetual leakage,” in *Annual International Cryptology Conference - CRYPTO*, vol. 963. Springer, 1995, pp. 339–352.

[53] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl, “Improved primitives for MPC over mixed arithmetic-binary circuits,” in *Annual International Cryptology Conference - CRYPTO*, vol. 12171. Springer, 2020, pp. 823–852.

[54] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Conference on Computer and Communications Security - CCS*. ACM, 2006, pp. 390–399.

[55] D. Rotaru and T. Wood, "Marbled circuits: Mixing arithmetic and boolean circuits with active security," in *International Conference on Cryptology in India - INDOCRYPT*, vol. 11898. Springer, 2019, pp. 227–249.