IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 15, 2019, accepted December 2, 2019, date of publication December 10, 2019,
date of current version December 23, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2958816

Practical Lattice-Based Multisignature Schemes for Blockchains

CHANGSHE MA AND MEI JIANG
School of Computer Science, South China Normal University, Guangzhou 510631, China
Corresponding author: Mei Jiang (mmjiang07@gmail.com)

This work was supported by the National Natural Science Foundation of China under Grant 61672243.

ABSTRACT Compact multisignature is vital for shrinking the signature size of decentralized blockchain. All practical compact multisignature schemes have been constructed from the discrete logarithm problem which is potentially vulnerable to quantum computing attacks. Lattice-based multisignature schemes are potential candidates for resisting quantum attacks. However, the existing lattice-based multisignature schemes suffer either loose signatures or large public key and signature sizes after compressing, which makes them unsuitable for blockchains. In this paper, we first present a practical lattice-based multisignature scheme with much smaller signature sizes than previous lattice-based multisignature schemes. Then, we extend our scheme to support public key aggregation with almost the same performance. Both of our multisignature schemes are provably secure in the random oracle model under the ring version of the short integer solution (Ring-SIS) assumption. They outperform the recent lattice-based multisignature scheme proposed by Bansarkhani and Sturm (BS) in terms of both signature size and communication overhead.

INDEX TERMS Lattice, multisignature scheme, public key aggregation, random oracle model.

I. INTRODUCTION

In a multisignature scheme [1], a group of signers (denoted by U) can jointly produce a compact signature (which can be verified very similarly to the ordinary signature scheme) on a common message m. Let the potential signers be denoted by $1, \cdots, K$. The signer i has its public key pk_i and the corresponding private key sk_i. Obviously, there is a trivial way to generate a multisignature σ on a common message m by setting $\sigma = (\sigma_1, \cdots, \sigma_i, \cdots, \sigma_\ell)$, where $i \in U \subseteq \{1, \cdots, K\}$ and σ_i is signer i's signature on m [2]. However, the size of the multisignature and the verification cost in that case are linear to the number of signers (i.e. $|U|$). To be available and scalable, the length of a multisignature and the computational cost of every signer and the computational cost of multisignature verification should be close to those of a single ordinary signature [2]. Multisignature has many potential usages, for example, contract signing, trust routing, distribution of a certificate authority, and blockchain [3].

Applications to Blockchain: Since it is introduced by Itakura and Nakamura [1], multisignature has been studied extensively, which includes security models [4]–[6] and elegant signing protocols [7], [8]. Recent applications of the blockchain [3] have renewed the research interests of multisignature schemes from both cryptographic community and industry. Blockchain could be regarded as a public ledger and all committed transactions are stored in a list of blocks [9]. A typical modern blockchain for cryptocurrency applications consists of two parts: a proof-of-work protocol for delegating the creation of new blocks and a signature scheme for transaction verification [10]. Multisignature schemes can be used to reduce the amount of data written in blockchains efficiently. Concretely, standard transactions in the bitcoin blockchain are called ''single-signature transactions'', which only transfer and store one signature between transactors. With the development of the bitcoin network, the bitcoin blockchain supports more complicated M-of-N transactions that transfer M valid signatures from M of the N transactors and write all these signatures in the blockchain. In this case, compressing multiple signatures into one multisignature has great importance in the storage of blockchains.

On the other hand, public key aggregation [11] is also crucial for blockchains since multiple public keys can be replaced by one short aggregated public key, which may reduce the storage further and is possible to lower the verification cost. Therefore, the data of an M-of-N transaction written in each block contain a message, an aggregated public key and a short multisignature. It saves more memories of

The associate editor coordinating the review of this manuscript and approving it for publication was M. Saif Islam.

each block in blockchains than before. Although very recent multisignature schemes [11]–[13] support public key aggregation, they are all constructed from the discrete logarithm problem which will be potentially vulnerable to quantum computing attacks. Lattice-based cryptographic schemes are potential candidates for post-quantum cryptographic primitives. However, the existing most efficient lattice-based multisignature scheme proposed by Bansarkhani and Sturm [14] is unsuitable for blockchains due to its large public key and signature sizes.

The BS multisignature scheme is based on the practical digital signature scheme proposed by Güneysu, Lyubashevsky, and Pöppelmann (GLP) [15]. The GLP scheme has to repeat about 7 times to generate a signature with acceptable size and computational cost [15], due to its inherent rejection-sampling method. Repetition is not a problem for single-signature schemes, but a crucial problem for multisignature schemes since it will cause a communication bottleneck. To address this problem, the BS scheme uses larger parameters than the GLP scheme to reduce the probability of rejection. As a result, the signature size of the BS scheme is about 10 thousands of bits larger than the GLP scheme. Moreover, the modulo in the parameters of the BS scheme is out of the scope of the integer type, which makes it unacceptable for fast implementations. Although the BS scheme can use the same parameters as the GLP scheme, it is also unacceptable due to the exponential repetitions of its signing protocol. This state of affairs raises the question:

Is it possible to reduce the repetitions of the signing protocol of lattice-based multisignatures to an acceptable level while preserving a reasonable level of performance?

In this paper, we focus on reducing the interactions of repetition-inherent multisignature schemes with acceptable parameters in the setting of a small signer group. Repetition is an inherent property of lattice-based Fiat-Shamir like (FS-like) [16] digital signature schemes [15], [17]. Lattice-based multisignature schemes can be constructed directly from the FS-like lattice-based signatures through the technique of [6]. However, it will result in either exponential repetitions or large parameters, which prohibits the availability of multisignature schemes even in the setting of small signer group. A lot of applications, such as bitcoin, usually use multisignature with small signer group. In this setting, reducing the probability of repetition to be significantly small with acceptable performance and supporting public key aggregation will lead to practical lattice-based multisignature schemes.

A. OUR CONTRIBUTIONS

In this paper, we present a practical lattice-based multisignature scheme (PLMS) and extend it to support public key aggregation in the setting of a small signer group. Our schemes are based on the GLP scheme and follow the FS-like digital signature structure which includes the committing phase, opening phase, and aggregating phase, but use a different approach rather than increasing parameters to avoid the restart of the protocol. Consequently, the parameters of our schemes are smaller than the BS scheme at the cost of increasing communication overhead a little bit. Moreover, our schemes have been proven secure in the random oracle model under the Ring-SIS assumption. The improved efficiency of our schemes as compared with the BS scheme may be attributed to the adoption of a new method to reduce both interactions and communications. Our method uses a basis to generate an exponential number of commitments and exchanges only the basis among signers during the committing phase of the signing protocol. As a result, our schemes can proceed exponential number of signing tries and hence can produce a multisignature with probability close to 1 for each execution of the signing protocol. In the opening phase, our schemes require an extra round of communication to exchange the success positions among signers. Thus, each execution of our signing protocol needs 4 rounds of interactions, which is one more round than the BS scheme. Nevertheless, our multisignature schemes beat the BS scheme in terms of both signature sizes and communications. A detailed comparison between our basic scheme PLMS and the BS scheme is described in Table 1.

Efficiency: As depicted by Table 1, our PLMS scheme requires only one repetition of the multisigning protocol, which means that the PLMS scheme can produce a multisignature without restarting the signing protocol, while maintaining the same public key size and security level as the BS scheme. Using the same parameters, the BS scheme restarts the MSign protocol nearly 12 times, which is far from practicality. Besides, the communication overhead consumed to generate a multisignatue in the BS scheme is nearly twice as much as our PLMS scheme. Furthermore, the signature bit size of the BS scheme is also longer than the PLMS scheme. In conclusion, our PLMS scheme beats the BS scheme in terms of signature sizes and communications.

TABLE 1. Comparisons betwwen BS scheme and PLMS scheme.

Scheme	Expected number of repetitions	Total Comm. (KB)	Sizes (KB)		Root Hermite factor	Parameter Set
			$\|pk\|$	$\|\sigma\|$		
BS	12.188	67.03	1.69	4.10	1.00395	SetI
PLMS	1.001	36.02	1.69	2.51	1.00395	
BS	12.194	70.02	1.75	4.41	1.00499	SetII
PLMS	1.001	37.32	1.75	2.86	1.00499	
BS	12.163	140.05	3.49	8.57	1.00215	SetIII
PLMS	1.001	74.50	3.49	5.18	1.00215	
BS	12.194	146.33	3.62	9.21	1.00265	SetIV
PLMS	1.001	77.39	3.62	5.78	1.00265	

Remark 1: We make comparisons between the BS scheme and our PLMS scheme with different sets of parameters which are listed in Table 2 in terms of the expected number of repetitions, total communications, public key sizes, signature sizes, and root Hermite factor. The detailed explanations of choosing parameters and calculating these attributes are located in section VI-A.

B. TECHNIQUE OVERVIEW

The GLP scheme [15] is an FS-like practical lattice-based digital signature scheme, which is a ring version of [17]. It needs rejection-sampling to hide the private key, which means that its signing procedure has to repeat several times to generate a secure signature. In the GLP scheme, the expected number of repetitions is 7 according to the suggested parameters. The BS multisignature scheme [14] follows directly from the GLP scheme and requires 3 rounds of interactions between each signer to generate a signature. In the signing protocol of the BS scheme, each signer produces his signature by using a similar method described in the GLP scheme, and then all valid signatures can be aggregated to a multisignature. As a result, it has to repeat an exponential number of the signing protocol to output a secure signature, which results in an inefficient and impractical scheme. Concretely, we assume that ℓ signers are participating in the multisignature process. In the BS scheme, each signer follows the signing protocol honestly and generates a secure signature which passes the rejection sampling step with the probability $\frac{1}{7}$. Then the whole signing group produces a secure multisignature with probability $(\frac{1}{7})^{\ell}$, which implies that its signing protocol has to repeat $O\left(7^{\ell}\right)$ times. The exponential number of repetitions makes the BS scheme far from efficiency and practicality even there are only 2 signers.

There are two trivial ways to reduce the repetitions of the BS scheme. One is the approach of increasing parameters, which is adopted by the BS scheme and leads to large public key and signature sizes. The other is the approach of increasing communication overhead which is described below. Specifically, in the committing phase, each signer commits an exponential number of commitments (e.g. $O\left(7^{\ell}\right)$ commitments) instead of only one. In the opening phase, each signer opens all the commitments and the corresponding successful partial signature. In this way, every execution of the signing protocol will output a secure signature with probability at least $\frac{1}{2}$. However, it leads to another problem. The transmission of $O\left(7^{\ell}\right)$ ring elements may cause huge communications. For example, the length of each commitment is $n\left\lceil log_2^q \right\rceil \approx 3$KB (according to the suggested parameters in [15], $n = 1024$, $q = 16760833$), so the total length of communication costs in the first-round interaction is approximately 49.24 MB and 808.17 GB for 5 and 10 signers, respectively.

To address the above problems, our new approach uses a small number of short ring elements as a basis for each signer to generate an exponential number of commitments. Specifically, each signer chooses α different short ring elements $b_1, b_2, \ldots, b_\alpha$ at random and uses them as a basis to generate an exponential number of commitments. Each commitments $c_j = (b_1\ b_2\ \ldots\ b_\alpha) \cdot [j]_2^\alpha = j_1 b_1 + j_2 b_2 + \ldots + j_\alpha b_\alpha$. Therefore, these α ring elements can represent 2^α commitments. During the commitment phase, each signer just needs to transmit the basis $(b_1\ b_2\ \cdots\ b_\alpha)$ rather than 7^ℓ ring elements to other signers, which greatly reduces the communication costs. Considering there are 10 signers, the rough value of α is derived from the equation $2^\alpha = 7^\ell$, which means that $\alpha = \left\lceil log_2^{7^{10}} \right\rceil < 29$. Overall, we only need to transmit $O(\ell)$ ring elements to solve the huge communication costs presented in the above trivial method.

However, if we use the above representing method to the commitment phase of the BS multisignature scheme and keep its other phases unchanged, it will cause a leakage of the private key. In the BS scheme, the signature $\mathbf{z}_i$ of signer i is derived from $\mathbf{z}_i = \mathbf{S}_i\mathbf{c}_i + \mathbf{Y}_i$, where $\mathbf{S}_i$ is the private key of signer i, $\mathbf{c}_i$ is obtained by querying the random oracle H_1 and $\mathbf{Y}_i$ is a random number chosen by signer i. Without loss of generality, assume three signatures $\mathbf{z}_{i,1}, \mathbf{z}_{i,2}, \mathbf{z}_{i,3}$ are passing the rejection sampling step and the signer i broadcasts them to other signers in the third round of the signing protocol. These signatures can be denoted as follows:

$$\begin{aligned}
\mathbf{z}_{i,1} &= \mathbf{S}_i\mathbf{c}_{i,1} + \left(b_{i,1}\ b_{i,2}\ \ldots b_{i,\alpha}\right) \cdot [1]_2^\alpha \\
&= \mathbf{S}_i\mathbf{c}_{i,1} + b_{i,\alpha} \\
\mathbf{z}_{i,2} &= \mathbf{S}_i\mathbf{c}_{i,2} + \left(b_{i,1}\ b_{i,2}\ \ldots b_{i,\alpha}\right) \cdot [2]_2^\alpha \\
&= \mathbf{S}_i\mathbf{c}_{i,2} + b_{i,\alpha-1}, \\
\mathbf{z}_{i,3} &= \mathbf{S}_i\mathbf{c}_{i,3} + \left(b_{i,1}\ b_{i,2}\ \ldots b_{i,\alpha}\right) \cdot [3]_2^\alpha \\
&= \mathbf{S}_i\mathbf{c}_{i,3} + b_{i,\alpha} + b_{i,\alpha-1}
\end{aligned}$$

According to the above equations, we can extract the private key $\mathbf{S}_i$ easily by computing $\mathbf{S}_i = \frac{\mathbf{z}_{i,3}-\mathbf{z}_{i,2}-\mathbf{z}_{i,1}}{\mathbf{c}_{i,3}-\mathbf{c}_{i,2}-\mathbf{c}_{i,1}}$. To prevent such a leakage, our new method requires each signer i to broadcast those indices of successful signatures in advance to the opening phase. If all signers succeed in the same index, then each signer only opens the signature of this index to others.

C. ORGANIZATION

This paper is organized as follows. In Section 2, we recall the development history of multisignature schemes. In Section 3, we introduce some mathematical notations and recall some definitions. In Section 4, we introduce our multisignature schemes. In Section 5, we review the general forking lemma and give security theorems of our schemes. In Section 6, we analyze how to select parameters in our multisignature scheme PLMS and describe the experiments. In Section 7, we draw a conclusion on our multisignature schemes.

II. RELATED WORK

Multisignature schemes with provable security generally fall into one of two categories. In the first category are schemes constructed using the hash-then-sign approach [18], and in the second are FS-like schemes based on the Fiat-Shamir technique [16]. Although various multisignature schemes have been presented based on integer factoring [19], [20], discrete logarithms (DL) [6]–[8], and lattice hard problems [14] for decades, the research of multisignature is mainly focused on FS-like schemes.

Regarding the hash-then-sign setting, multisignatures were usually constructed sequentially and based on the RSA assumption. Itakura and Nakamura [1] proposed the first ordered multisignature scheme whose efficiency was improved by Okamoto [19] subsequently. In 1991, Ohta and Okamoto [21] proposed another sequential multisignature scheme and discussed its security. In 1997, Park *et al.* [20] improved the schemes of [19] and [22] by getting rid of the predefined signing order and providing better efficiency.

Related to the FS-like setting, resisting the rouge-key attack [6] is the main concern of multisignatures. Almost all the early proposed multisignature schemes (e.g. [23]–[25]) are vulnerable to the rouge-key attack. Until 1999, Ohta and Okamoto [26] constructed the first provably secure multisignature scheme in the random oracle model [27]. By 2001, Micali *et al.* [5] formalized and implemented a variant of multisignature scheme called *Accountable-Subgroup Multisignatures* (ASM), and provided the first security model for multisignature schemes including key generation without relying on trusted third parties. In 2006, Bellare and Neven [6] formalized a generalized forking lemma and proposed a three-round multisignature scheme based on DL problem in the plain public-key model which eliminated the key generation assumption presented in [5]. In 2008, Bagherzandi *et al.* [7] improved Bellare and Neven's scheme [6] by presenting a two-round multisignature scheme which is provably secure under the DL assumption in the key verification model. Through the same technique, Bagherzandi and Jarecki [28] also proposed two two-round multisignature schemes with security tightly related to the computational Diffie-Hellman (CDH) and decisional Diffie-Hellman (DDH) problems, respectively. In 2010, Ma *et al.* [8] proposed a two-round multisignature scheme based on the DL assumption in the plain public key model which improved schemes of [6] and [7].

Recently, FS-like multisignature schemes have received renewed interests since they can support public key aggregation which is a crucial requirement for distributed trust applications. In 2016, Syta *et al.* [29] proposed the CoSi multisignature scheme, a highly scalable multisignature scheme based on DL. In 2018, Maxwell *et al.* [11] presented the first multisignature scheme with key aggregation based on DL assumption which is provably secure in the plain public key model. Meanwhile, Boneh *et al.* [13] proposed pairing-based multisignature schemes supporting both public key aggregation and batch verification. Unfortunately, Drijvers *et al.* [12] pointed out subtle flaws in the security proofs of two-round multisignature schemes [7], [8], [11] and proposed a new variant of [7] that is provably secure and supports public key aggregation.

All the above-mentioned multisignature scheme are based on the RSA or DL related assumptions which are potentially vulnerable to quantum attacks. Lattice-based cryptographic schemes are amongst those resisting quantum attacks. In 2013, following the hash-then-sign approach, Kong *et al.* [30] presented two lattice-based multisignature schemes: one was a trivial broadcasting multisignature scheme whose length varied linearly with the number of the signers, the other was a sequential multisignature scheme whose length was constant and independent of the number of the signers. In 2016, Choi and Kim [31] and Bansarkhani and Sturm [14] presented FS-like lattice-based multisignature schemes, respectively. However, all the proposed lattice-based multisignature schemes suffer from either linear signature size or high interactions.

III. PRELIMINARIES

In the following section, we will present some mathematical notations and hard problems. We will also introduce the syntax and the security definition of multisignature schemes.

A. NOTATIONS

Throughout the paper, we assume that n is a positive integer which is a power of 2 and q is a prime convergent to 1 modulo $2n$. $\mathcal{R}^q$ represents the ring $\mathbb{Z}_q[x]/(x^n+1)$. The elements of $\mathcal{R}^q$ can be represented as polynomials of degree $n-1$ with coefficients in the range $[-(q-1)/2, (q-1)/2]$. $\mathcal{R}_d^q$ is a subset of $\mathcal{R}^q$ with coefficients in the range $[-d, d]$. We denote ring elements by boldface lower case letters e.g. $\mathbf{p}$. We assume that all vectors are column vectors, and v^T denotes the transpose of the vector v. The infinity norm of a vector v is denoted by $\|v\|_\infty$, and we usually avoid writing the subscript for the 2-norm. Let $[j]_2^\alpha$ denote the binary representation of number j with length α bits. If the length is shorter than α, then all of its higher-order bits will be set 0. For example, assume that $j=13$, $\alpha=7$, then $[13]_2^7 = 0001101$. For a set A, we write $a \xleftarrow{\$} A$ to show that a is chosen uniformly at random from A. Same to the paper [15], D_{32}^n denotes the set of polynomials of degree at most $n-1$ with 32 coefficients ± 1 and zero coefficients else. If b is a bit string, then $b[i]$ denotes the i-th bit of b. The logarithm is based on 2.

Our multisignature scheme is based on the Decisional Compact Knapsack (DCK) problem and the ring short integer solution (Ring-SIS) problem which are defined as below.

Definition 1 ($DCK_{q,n}$ Problem [32]): To solve the $DCK_{q,n}$ problem, one should distinguish between the uniform distribution over $\mathcal{R}^q \times \mathcal{R}^q$ and the distribution $(a, a\cdot s_1 + s_2)$ where a is uniformly random in $\mathcal{R}^q$ and $\mathbf{s}_1, \mathbf{s}_2$ are uniformly random in $\mathcal{R}_1^q$.

Definition 2 ($Ring-SIS_{m,q,\beta}$ Problem [33]): To solve the $Ring-SIS_{m,q,\beta}$ problem, one is given $a = (a_1, a_2, \ldots, a_m)^T \in \mathcal{R}_q^m$ a vector of m uniformly random polynomials, find a non-zero vector of small polynomials $x = (x_1, x_2, \ldots, x_m)^T \in \mathcal{R}_q^m$ such that $a^T x = \sum_{i=1}^m a_i \cdot x_i = 0 \ mod \ q$ and $0 < \|x\| < \beta$.

B. MULTISIGNATURE AND ITS SECURITY MODEL

1) SYNTAX OF MULTISIGNATURE SCHEME

Let $L = \{1, 2, \ldots, \ell\}$ denote the signing group in which there are ℓ signers. A multisignature scheme $\mathcal{MS} =$

(**MKeyGen**, **MSign**, **MVerify**) consists of three algorithms described below.

• **MKeyGen** $\left(1^{\theta}\right)$: the signer's public key and private key generation algorithm. On input 1^{θ} with the security parameter θ, the probabilistic algorithm outputs the public key pk_i and private key sk_i for signer i. Each signer runs it independently.

• **MSign** (pk, m): the multisignature generation algorithm. On input the multiset of public keys $pk = \{pk_1, pk_2, \ldots, pk_\ell\}$ and a message m, the probabilistic algorithm outputs a joint multisignature σ or a symbol $\perp$ to indicate failure.

• **MVerify** (σ, m, pk): the multisignature verification algorithm. On input a candidate signature σ, a message m and the set of public keys pk, the deterministic algorithm outputs **1** if the multisignature σ is valid for m and pk, otherwise outputs **0**.

The correctness of a multisignature scheme requires that if a group of signers interact with each other, share the same message m and follow the protocol honestly then they generate a joint signature σ such that **MVerify** outputs **1** on input σ, m, and pk.

2) SECURITY NOTION OF MULTISIGNATURE SCHEME

The security of a multisignature requires that it is infeasible to forge valid multisignatures with at least one honest signer. Following the previous work [14], we assume that there is only one honest signer in the whole signing process. The adversary pretends all other signers by choosing their public keys and private keys arbitrarily, and interacts with the honest signer before outputting its forgery. Now we turn to describe the security notion of multisignature schemes: *existential unforgeability under adaptively chosen message attacks* (EUF-CMA) [2]. A multisignature scheme is said to be EUF-CMA secure if the probability that any polynomial-time forger, after knowing some tuples of the message and its signature, produces a valid signature on a new message not signed before is negligible.

Definition 3 (EUF-CMA, [2]): A multisignature scheme $\mathcal{MS}$ = (MKeyGen, MSign, MVerify) is existentially unforgeable under adaptively chosen message attacks if for each polynomial-time forger $\mathcal{F}$, the probability that after seeing the public key pk and $\left\{(m_1, \sigma_1), (m_2, \sigma_2), \ldots \left(m_q, \sigma_q\right)\right\}$ for any q messages m_i chosen by the forger $\mathcal{F}$ at wish, $\mathcal{F}$ can produce a forged signature σ^* on message m^* such that Verify $(\sigma^*, m^*, pk) = 1$ and $m^* \notin \left\{m_1, m_2, \ldots m_q\right\}$, is negligible over the security parameter.

IV. OUR SCHEMES

In this section, we give two lattice-based multisignature schemes. At first, we present the basic scheme PLMS which improves the BS scheme [14] by avoiding the restart of the signing protocol with small parameters. Then, we extend the PLMS scheme to obtain the ELMS scheme to allow public key aggregation with almost the same performance.

A. THE PLMS SCHEME

To prove the security of the PLMS scheme, we need two cryptographic hash functions $H_1 : \mathcal{R}^q \longrightarrow \mathcal{R}^q$ and $H_2 : \{0,1\}^* \longrightarrow D_{32}^n$. A detailed description of our practical lattice-based multisignature (PLMS) scheme is as follows.

MKeyGen. All signers agree on choosing a random polynomial $\mathbf{a}$ from $\mathcal{R}^q$ as a shared part for the signing group. Each signer i selects two polynomials $\mathbf{s}_i, \mathbf{v}_i$ from $\mathcal{R}_1^q$ and sets his private key $\mathbf{sk}_i = (\mathbf{s}_i, \mathbf{v}_i)$. Each signer i computes his public key $\mathbf{pk}_i = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix} \mathbf{s}_i \\ \mathbf{v}_i \end{pmatrix} = \mathbf{a} \cdot \mathbf{s}_i + \mathbf{v}_i \ mod\ q$ and outputs $\left(\mathbf{pk}_i, \mathbf{sk}_i\right)$.

MSign. Let $L = \{1, 2, \cdots, \ell\}$ be the set of ℓ signers who participate in the multisignature generation. Let $\mathbf{pk} = \left\{\mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$ be the set of public keys corresponding to signers in set L. Also let $\alpha \geqslant \ell \left\lceil log^{\frac{1}{\left(1-\frac{64}{2\theta d+1}\right)^{2n}}} \right\rceil$. The **MSign** protocol proceeds in 4 rounds, where in each round each signer receives a message from every other signer, performs some local computation and sends a message to every other signer. Specifically, each signer i on input $\left(\mathbf{pk}, \mathbf{pk}_i, \mathbf{sk}_i, m\right)$ proceeds as follows.

Round 1:

- Local input: $\mathbf{sk}_i$, $\mathbf{pk} = \left\{\mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$, m.
- Computation: Choose $\mathbf{y}_{i,j} = \left(\mathbf{g}_{i,j}, \mathbf{h}_{i,j}\right) \xleftarrow{\$} \mathcal{R}_d^q \times \mathcal{R}_d^q$ and compute $\mathbf{r}_{i,j} = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix} \mathbf{g}_{i,j} \\ \mathbf{h}_{i,j} \end{pmatrix} = \mathbf{a} \cdot \mathbf{g}_{i,j} + \mathbf{h}_{i,j}\ mod\ q$ and query $\mathbf{t}_{i,j} = H_1\left(\mathbf{r}_{i,j}\right)$ for $1 \leqslant j \leqslant \alpha$.
- Send to signer k $(k \neq i)$: $\mathbf{t}_{i,1}, \mathbf{t}_{i,2}, \ldots, \mathbf{t}_{i,\alpha}$.

Round 2:

- Receive from signer k $(k \neq i)$: $\mathbf{t}_{k,1}, \mathbf{t}_{k,2}, \ldots, \mathbf{t}_{k,\alpha}$.
- Send to signer k $(k \neq i)$: $\mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,\alpha}$.

Round 3:

- Receive from signer k $(k \neq i)$: $\mathbf{r}_{k,1}, \mathbf{r}_{k,2}, \ldots, \mathbf{r}_{k,\alpha}$.
- Computation: Check that $\mathbf{t}_{k,j} = H_1\left(\mathbf{r}_{k,j}\right)$ for $1 \leqslant k \leqslant \ell$, $k \neq i$, $1 \leqslant j \leqslant \alpha$. Abort the protocol with local output $\perp$ if this is not the case. For each $1 \leqslant f \leqslant 2^\alpha$, θ denotes the occurrence number of 1 in $[f]_2^\alpha$, compute $\mathbf{r}_{k,f} = \left(\mathbf{r}_{k,1}\ \mathbf{r}_{k,2}\ \ldots\ \mathbf{r}_{k,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant k \leqslant \ell$ and $\mathbf{r}_f = \sum_{k=1}^{\ell} \mathbf{r}_{k,f}\ mod\ q$ and query $\mathbf{c}_{i,f} = H_2\left(\mathbf{pk}_i, \mathbf{r}_f, \mathbf{pk}, m\right)$ and then compute $\mathbf{z}_{i,f} = \left(\mathbf{w}_{i,f}, \mathbf{x}_{i,f}\right) = \mathbf{sk}_i \cdot \mathbf{c}_{i,f} + \left(\mathbf{y}_{i,1}\ \mathbf{y}_{i,2}\ \ldots\ \mathbf{y}_{i,\alpha}\right) \cdot [f]_2^\alpha$ [1]. If $\mathbf{z}_{i,f} \in \mathcal{R}_{\theta d-32}^q \times \mathcal{R}_{\theta d-32}^q$, put the index f into the set $S_i = \left\{f \mid \mathbf{z}_{i,f} \in \mathcal{R}_{\theta d-32}^q \times \mathcal{R}_{\theta d-32}^q\right\}$.
- Send to signer k $(k \neq i)$: S_i.

Round 4:

- Receive from signer k $(k \neq i)$: S_k.
- Computation: Compute the intersection of all sets $S_1, S_2, \ldots, S_\ell$. **If the intersection is empty, restart**

[1] Please pay attention here. We do not reduct modulo q because all involved polynomials in this step have small parameters.

the signing protocol again. Let $\bar{f}$ be the smallest index in this intersection.

- Send to signer k ($k \neq i$): $\mathbf{z}_{i,\bar{f}}, \mathbf{c}_{i,\bar{f}}$.

At last, the designed combiner collects all $\mathbf{z}_{k,\bar{f}}$ and $\mathbf{c}_{k,\bar{f}}$ for $1 \leqslant k \leqslant \ell$, computes $\mathbf{z}_{\bar{f}} = \sum_{k=1}^{\ell} \mathbf{z}_{k,\bar{f}}$ and outputs the multisignature $\sigma = \left(\mathbf{z}_{\bar{f}}, \mathbf{c}_{1,\bar{f}}, \cdots, \mathbf{c}_{\ell,\bar{f}}\right)$. This designed combiner can be one of the signers in the set L.

MVerify. Given the public key set **pk** and a candidate multisignature $\sigma = \left(\mathbf{z}, \mathbf{c}_{1,\bar{f}} \cdots \mathbf{c}_{\ell,\bar{f}}\right)$ on the message m, the verifier computes $\mathbf{r} = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i$ and accepts the candidate multisignature if $\mathbf{c}_i = H_2\left(\mathbf{pk}_i, \mathbf{r}, \mathbf{pk}, m\right)$ for $1 \leq i \leq \ell$ and $\mathbf{z} \in \mathcal{R}^q_{\ell\cdot(\alpha d-32)} \times \mathcal{R}^q_{\ell\cdot(\alpha d-32)}$, and rejects otherwise.

Correctness. The intuition behind our PLMS scheme is as follows. If each signer executes the protocol honestly, then the multisignature $\sigma = \left(\mathbf{z}, \mathbf{c}_{1,\bar{f}}, \cdots, \mathbf{c}_{\ell,\bar{f}}\right)$ is derived from $\mathbf{z} = \sum_{i=1}^{\ell} \mathbf{z}_i$, $\mathbf{c}_i = H_2\left(\mathbf{pk}_i, \mathbf{r}, \mathbf{pk}, m\right)$ for $1 \leq i \leq \ell$ and $\mathbf{r} = \sum_{i=1}^{\ell} \mathbf{r}_i \mod q$. Therefore,

$$
\begin{aligned}
\mathbf{r} &= \left(\mathbf{a} \quad \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i \\
&= \left(\mathbf{a} \quad \mathbf{1}\right) \cdot \begin{pmatrix} \sum_{i=1}^{\ell} \mathbf{s}_i \cdot \mathbf{c}_i + \left(\mathbf{g}_{i,1}\ \mathbf{g}_{i,2}\ \cdots\ \mathbf{g}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^{\alpha} \\ \sum_{i=1}^{\ell} \mathbf{v}_i \cdot \mathbf{c}_i + \left(\mathbf{h}_{i,1}\ \mathbf{h}_{i,2}\ \cdots\ \mathbf{h}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^{\alpha} \end{pmatrix} \\
&\quad - \sum_{i=1}^{\ell} \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{s}_i\\ \mathbf{v}_i\end{pmatrix} \cdot \mathbf{c}_i \\
&= \sum_{i=1}^{\ell} \left(\mathbf{a}\ \mathbf{1}\right) \left(\mathbf{y}_{i,1}\ \mathbf{y}_{i,2}\ \cdots\ \mathbf{y}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^{\alpha} \\
&= \sum_{i=1}^{\ell} \left(\mathbf{r}_{i,1}\ \mathbf{r}_{i,2}\ \cdots\ \mathbf{r}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^{\alpha} \\
&= \mathbf{r}_{\bar{f}}
\end{aligned}
$$

Hence,

$$
\mathbf{c}_i = H_2\left(\mathbf{pk}_i, \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i, \mathbf{pk}, m\right)
$$

which indicates that the verification equation holds. It remains to prove the security of the scheme.

Communication Optimization: The communication overhead of our PLMS scheme may be an efficiency bottleneck, although they are much less than those of the trivial scheme mentioned in section I-B. However, the communication burden of our scheme mainly consists of the transmission of all S_i's in the 3rd round. Here, we introduce an optimal method to reduce necessary bits to present an element in S_i. During the **MSign** protocol, each signer i chooses a bit string bf_i of length 2^{α}, whose bits are all initialized to be zero. In the 3rd round, if $\mathbf{z}_{i,f} \in \mathcal{R}^q_{\theta d-32} \times \mathcal{R}^q_{\theta d-32}$, then the signer i sets the f-th bit of bf_i to be 1. At last she sends bf_i instead of S_i to the other signers. After receiving all bit strings $bf_1, bf_2, \ldots, bf_\ell$, the signer i computes $br = \bigoplus_{i=1}^{\ell} bf_i$, where $\bigoplus$ denotes the bitwise AND operation of two bit strings. If each bit of br is 0, then the signer i restarts the signing protocol. Otherwise, the signer i chooses the smallest index $\bar{f}$ such that $br[\bar{f}] = 1$ and broadcasts $\mathbf{z}_{i,\bar{f}}$ to other signers. In this way, the communication complexity of the 3rd round is reduced from $O(\|\sigma\|\, 2^{\alpha})$ to $O(2^{\alpha})$, which makes our PLMS scheme more practical for small signer group (e.g. less than 20 signers).

B. EXTENSION TO SUPPORT PUBLIC KEY AGGREGATION

Our PLMS scheme can extend to support public key aggregation through the technique of [11]. To prove the security of the scheme supporting public key aggregation, we need three cryptographic hash functions $H_0 : \{0,1\}^* \longrightarrow D_{32}^n$, $H_1 : \mathcal{R}^q \longrightarrow \mathcal{R}^q$ and $H_2 : \{0,1\}^* \longrightarrow D_{32}^n$. A detailed description of our extended multisignature (ELMS) scheme is as follows.

MKeyGen. All signers agree on choosing a random polynomial $\mathbf{a}$ from $\mathcal{R}^q$ as a shared part for the signing group. Each signer i selects two polynomials $\mathbf{s}_i, \mathbf{v}_i$ from $\mathcal{R}_1^q$ and sets his private key $\mathbf{sk}_i = (\mathbf{s}_i, \mathbf{v}_i)$. Each signer i computes his public key $\mathbf{pk}_i = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{s}_i\\ \mathbf{v}_i\end{pmatrix} = \mathbf{a} \cdot \mathbf{s}_i + \mathbf{v}_i \ mod\ q$ and outputs $\left(\mathbf{pk}_i, \mathbf{sk}_i\right)$.

MSign. Let $L = \{1, 2, \cdots, \ell\}$ be the set of ℓ signers who participate in the multisignature generation. Let $\mathbf{pk} = \left\{\mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$ be the set of public keys corresponding to signers in set L. Also let $\alpha \geqslant \ell \left\lceil log^{\frac{1}{\left(1-\frac{2048}{2\theta d+1}\right)^{2n}}} \right\rceil$. The **MSign** protocol proceeds in 4 rounds, where in each round each signer receives a message from every other signer, performs some local computation and sends a message to every other signer. Specifically, each signer i on input $\left(\mathbf{pk}, \mathbf{pk}_i, \mathbf{sk}_i, m\right)$ proceeds as follows.

Round 1:

- Local input: $\mathbf{sk}_i$, $\mathbf{pk} = \left\{\mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$, m.
- Computation: Query $\mathbf{u}_i = H_0\left(\mathbf{pk}, \mathbf{pk}_i\right)$ for $1 \leqslant i \leqslant \ell$ and compute $\mathbf{apk} = \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i \ mod\ q$, choose $\mathbf{y}_{i,j} = \left(\mathbf{g}_{i,j}, \mathbf{h}_{i,j}\right) \xleftarrow{\$} \mathcal{R}_d^q \times \mathcal{R}_d^q$ and compute $\mathbf{r}_{i,j} = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{g}_{i,j}\\ \mathbf{h}_{i,j}\end{pmatrix} = \mathbf{a} \cdot \mathbf{g}_{i,j} + \mathbf{h}_{i,j} \ mod\ q$ and query $\mathbf{t}_{i,j} = H_1\left(\mathbf{r}_{i,j}\right)$ for $1 \leqslant j \leqslant \alpha$.
- Send to signer k ($k \neq i$): $\mathbf{t}_{i,1}, \mathbf{t}_{i,2}, \ldots, \mathbf{t}_{i,\alpha}$.

Round 2:

- Receive from signer k ($k \neq i$): $\mathbf{t}_{k,1}, \mathbf{t}_{k,2}, \ldots, \mathbf{t}_{k,\alpha}$.
- Send to signer k ($k \neq i$): $\mathbf{r}_{i,1}, \mathbf{r}_{i,2}, \ldots, \mathbf{r}_{i,\alpha}$.

Round 3:

- Receive from signer k ($k \neq i$): $\mathbf{r}_{k,1}, \mathbf{r}_{k,2}, \ldots, \mathbf{r}_{k,\alpha}$.
- Computation: Check that $\mathbf{t}_{k,j} = H_1\left(\mathbf{r}_{k,j}\right)$ for $1 \leqslant k \leqslant l$, $k \neq i$, $1 \leqslant j \leqslant \alpha$. Abort the protocol with local output $\perp$ if this is not the case. For each $1 \leqslant f \leqslant 2^{\alpha}$, θ denotes the occurrence number of 1

in $[f]_2^\alpha$, compute $\mathbf{r}_{k,f} = \left(\mathbf{r}_{k,1}\ \mathbf{r}_{k,2}\ \ldots\ \mathbf{r}_{k,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant k \leqslant \ell$ and $\mathbf{r}_f = \sum_{k=1}^{\ell} \mathbf{r}_{k,f}\ mod\ q$ and query $\mathbf{c}_f = H_2\left(\mathbf{apk}, \mathbf{r}_f, m\right)$ and then compute $\mathbf{z}_{i,f} = \left(\mathbf{w}_{i,f}, \mathbf{x}_{i,f}\right) = \mathbf{c}_f \cdot \mathbf{u}_i \cdot \mathbf{sk}_i + \left(\mathbf{y}_{i,1}\ \mathbf{y}_{i,2}\ \ldots\ \mathbf{y}_{i,\alpha}\right) \cdot [f]_2^\alpha$ [2]. If $\mathbf{z}_{i,f} \in \mathcal{R}_{\theta d-1024}^q \times \mathcal{R}_{\theta d-1024}^q$, put the index f into the set $S_i = \left\{f \mid \mathbf{z}_{i,f} \in \mathcal{R}_{\theta d-1024}^q \times \mathcal{R}_{\theta d-1024}^q\right\}$.

- Send to signer k $(k \neq i)$: S_i.

Round 4:

- Receive from signer k $(k \neq i)$: S_k.
- Computation: Compute the intersection of all sets $S_1, S_2, \ldots, S_\ell$. **If the intersection is empty, restart the signing protocol again.** Let $\bar{f}$ be the smallest index in this intersection.
- Send to signer k $(k \neq i)$: $\mathbf{z}_{i,\bar{f}}$.

At last, the designed combiner collects all $\mathbf{z}_{k,\bar{f}}$ for $1 \leqslant k \leqslant \ell$, computes $\mathbf{z}_{\bar{f}} = \sum_{k=1}^{\ell} \mathbf{z}_{k,\bar{f}}$ and outputs the multisignature $\sigma = \left(\mathbf{z}_{\bar{f}}, \mathbf{c}_{\bar{f}}\right)$. This designed combiner can be one of the signers in the set L.

MVerify. Given an aggregated public key **apk** and a candidate multisignature $\sigma = (\mathbf{z}, \mathbf{c})$ on the message m, the verifier computes $\mathbf{r} = \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \mathbf{c} \cdot \mathbf{apk}$ and accepts the candidate signature if $\mathbf{c} = H_2\left(\mathbf{apk}, \mathbf{r}, m\right)$ and $\mathbf{z} \in \mathcal{R}_{\ell\cdot(\alpha d-1024)}^q \times \mathcal{R}_{\ell\cdot(\alpha d-1024)}^q$, and rejects otherwise.

Correctness. The intuition behind our ELMS scheme is as follows. If each signer executes the protocol honestly, then the multisignature $\sigma = (\mathbf{z}, \mathbf{c})$ is derived from $\mathbf{z} = \sum_{i=1}^{\ell} \mathbf{z}_i$, $\mathbf{c} = H_2\left(\mathbf{apk}, \mathbf{r}, m\right)$ and $\mathbf{r} = \sum_{i=1}^{\ell} \mathbf{r}_i \mod q$. Therefore,

$$
\begin{aligned}
\mathbf{r} &= \left(\mathbf{a} \quad \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \mathbf{c} \cdot \mathbf{apk} \\
&= \left(\mathbf{a} \quad \mathbf{1}\right) \cdot \begin{pmatrix}\sum_{i=1}^{\ell} \mathbf{w}_i\\ \sum_{i=1}^{\ell} \mathbf{x}_i\end{pmatrix} - \mathbf{c} \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i \\
&= \left(\mathbf{a} \quad \mathbf{1}\right) \cdot \begin{pmatrix}\sum_{i=1}^{\ell} \mathbf{c}\mathbf{u}_i\mathbf{s}_i + \left(\mathbf{g}_{i,1} \ldots \mathbf{g}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^\alpha\\ \sum_{i=1}^{\ell} \mathbf{c}\mathbf{u}_i\mathbf{v}_i + \left(\mathbf{h}_{i,1} \ldots \mathbf{h}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^\alpha\end{pmatrix} \\
&\quad - \mathbf{c} \sum_{i=1}^{\ell} \mathbf{u}_i \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{s}_i\\ \mathbf{v}_i\end{pmatrix} \\
&= \sum_{i=1}^{\ell} \left(\mathbf{a}\ \mathbf{1}\right) \left(\mathbf{y}_{i,1} \ldots \mathbf{y}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^\alpha \\
&= \sum_{i=1}^{\ell} \left(\mathbf{r}_{i,1} \ldots \mathbf{r}_{i,\alpha}\right) \cdot \left[\bar{f}\right]_2^\alpha \\
&= \mathbf{r}_{\bar{f}}
\end{aligned}
$$

Hence,

$$\mathbf{c} = H_2\left(\mathbf{apk}, \left(\mathbf{a}\ \mathbf{1}\right) \cdot \begin{pmatrix}\mathbf{w}\\ \mathbf{x}\end{pmatrix} - \mathbf{c} \cdot \mathbf{apk}, m\right)$$

which indicates that the verification equation holds. It remains to prove the security of the scheme.

[2]Please pay attention here. We do not reduct modulo q because all involved polynomials in this step have small parameters.

V. SECURITY

In order to prove the security of our constructions, we use the general forking lemma which is introduced briefly by [6]. Here we just give some simple explanations about the security of our multisignature schemes. The detailed security proofs of the PLMS scheme and the ELMS scheme are described in the appendix A and B, respectively.

A. THE GENERAL FORKING LEMMA

Forking lemma plays an important role in proving the security of FS-like digital signature schemes. Here, we first review the general forking lemma introduced by Bellare and Neven [6] to prove the security of our multisignature scheme.

Lemma 1 (General Forking Lemma [6]): Fix an integer $q \geqslant 1$ and a set H of size $h \geqslant 2$. Let A be a randomized algorithm that on input $x, h_1, \ldots, h_q$ returns a pair, the first element of which is an integer in the range $0, \ldots, q$ and the second element of which we refer to as a side output. Let IG be a randomized algorithm that we call the input generator. The accepting probability of A, denoted acc, is defined as the probability that $J \geqslant 1$ in the experiment

$$
\begin{aligned}
x &\xleftarrow{\$} IG; \quad h_1, \ldots, h_q \xleftarrow{\$} H; \\
(J, \delta) &\xleftarrow{\$} A(x, h_1, \ldots, h_q)
\end{aligned}
$$

The general forking algorithm GF_A associated with A is the randomized algorithm that takes input x proceeds as follows:

Algorithm $GF_A(x)$

Pick coins ρ for A at random
$h_1, \ldots, h_q \xleftarrow{\$} H$
$(I, \delta) \longleftarrow A(x, h_1, \ldots, h_q; \rho)$
If $I = 0$ then return $(0, \varepsilon, \varepsilon)$
$h_I', \ldots, h_q' \xleftarrow{\$} H$
$(I', \delta') \longleftarrow A(x, h_1 \ldots, h_{I-1}, h_I', \ldots, h_q'; \rho)$
If $(I = I'$ and $h_I \neq h_I')$ then return $(1, \delta, \delta')$
Else return $(0, \varepsilon, \varepsilon)$.

Let

$$frk = Pr\left[b = 1 : x \xleftarrow{\$} IG;\ (b, \delta, \delta') \xleftarrow{\$} GF_A(x)\right]$$

Then

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

B. THE SECURITY OF OUR SCHEMES

To prove the security of our PLMS scheme, we first wrap the forger $\mathcal{F}$ into an algorithm $\mathcal{A}$ which is used in the forking lemma. The algorithm $\mathcal{A}$ runs $\mathcal{F}$ with simulating the random oracles H_1 and H_2 and the signing oracle, and returns a forgery with some other information about the forger execution. Then we construct an algorithm $\mathcal{B}$ that runs $GF_{\mathcal{A}}$ to obtain a solution to a Ring-SIS instance. The security of the PLMS scheme is as follows.

TABLE 2. PLMS scheme parameters.

Parameters	SetI	SetII	SetIII	SetIV
Lattice dimension n	512	512	1024	1024
Modulus q	134136833	258001921	258001921	536856577
Group signers ℓ	5	10	5	10
Range d	6554	13104	13104	26206
Basis number α	10	10	10	10
Expected number of repetitions E	1.001	1.001	1.001	1.001
Private key bit size	1623	1623	3246	3246
Public key bit size	13824	14307	28613	29696
Multisignature bit size $\|\|\sigma\|\|$	20585	23433	42418	47314
Root Hermite factor δ	1.00395	1.00499	1.00215	1.00265

Theorem 1: In the random oracle model, suppose there exist a polynomial-time forger $\mathcal{F}$, who makes at most q_H queries to random oracles (including H_1, H_2), initiates at most q_S signing protocols with the honest signer involving at most ℓ_{max} public keys, and succeeds in providing a forgery of our PLMS multisignature scheme with probability δ. Then there exists the same time complexity algorithm $\mathcal{B}$ that for a given $\mathbf{A} = (\mathbf{a}, \mathbf{1}) \xleftarrow{\$} \mathcal{R}^q \times \{\mathbf{1}\}$ finds non-zero vectors $\mathbf{o}_1, \mathbf{o}_2$ in $\mathcal{R}^q$ such that $\mathbf{a}\cdot\mathbf{o}_1+\mathbf{o}_2 = 0$ and $\|\mathbf{o}_i\|_\infty \leqslant 2l_{max}\cdot(\alpha d - 32)$ with probability at least

$$\left(\frac{1}{2} - 2^{-100}\right)\left(\delta - \frac{q_A}{q^n}\right)\left(\frac{\delta - q_A/q^n}{q_T} - \frac{1}{|D_{H_2}|}\right)$$

where $q_A = (q_H + \ell_{max}\cdot q_S\cdot\alpha)^2 + (q_H + l_{max}\cdot q_S)^2$ and $q_T = q_H + q_S$.

To prove the security of our ELMS scheme, we follow the double forking proof technique of [13], and firstly use the forking lemma to $H_2(\mathbf{apk}, \mathbf{r}, m)$ to obtain an equation about the aggregated public key $\mathbf{apk}$. Next, we use the forking lemma to $H_0(\mathbf{pk}, \mathbf{pk}_i)$ to solve a Ring-SIS instance. Concretely, we first wrap the forger $\mathcal{F}$ into an algorithm $\mathcal{A}$ which is used in the forking lemma. The algorithm $\mathcal{A}$ runs $\mathcal{F}$ with simulating the random oracles H_0, H_1, and H_2 and the signing oracle, and returns a forgery with some other information about the forger execution unless a couple of bad events happen. Then we construct an algorithm $\mathcal{B}$ that runs $GF_{\mathcal{A}}$ to obtain an equation about the aggregated public key $\mathbf{apk}$. Finally, we construct an algorithm $\mathcal{D}$ that runs $GF_{\mathcal{B}}$ to obtain a solution to a Ring-SIS instance. The security of the ELMS scheme is as follows.

Theorem 2: In the random oracle model, suppose there exist a polynomial-time forger $\mathcal{F}$, who makes at most q_H queries to random oracles (including H_0, H_1, H_2), initiates at most q_S signing protocols with the honest signer involving at most ℓ_{max} public keys, and succeeds in providing a forgery of our ELMS multisignature scheme with probability δ. Then there exists the same time complexity algorithm $\mathcal{D}$ that for a given $\mathbf{A} = (\mathbf{a}, \mathbf{1}) \xleftarrow{\$} \mathcal{R}^q \times \{\mathbf{1}\}$ finds non-zero vectors $\mathbf{o}_1, \mathbf{o}_2$ in $\mathcal{R}^q$ such that $\mathbf{a}\cdot\mathbf{o}_1 + \mathbf{o}_2 = 0$ and $\|\mathbf{o}_i\|_\infty \leqslant 256l_{max}\cdot(\alpha d - 1024)$ with probability at least

$$\frac{2^{99}-1}{2^{100}}\left(\frac{q_B^2}{q_T} - \frac{q_B}{|D_{H_2}|}\right)\left(\frac{q_B^2/q_T - q_B/|D_{H_2}|}{q_T} - \frac{1}{|D_{H_0}|}\right)$$

where $q_T = q_H + q_S$, $q_B = \delta - \frac{q_A}{q^n}$ and $q_A = (q_H + \ell_{max}\cdot q_S\cdot 2^\alpha)^2 + (q_H + \ell_{max}\cdot q_S\cdot\alpha)^2 + \ell_{max}\cdot q_S\cdot\alpha + (q_H + l_{max}\cdot q_S)^2$.

VI. PERFORMANCE ANALYSIS AND EXPERIMENTS

In the following section, we will explain how to select parameters appropriately for our multisignature schemes and give some sets of suggested parameters. Then we describe the platform information of our implementation between BS scheme and PLMS scheme and provide some experimental data.

A. PARAMETERS CHOICES

In our PLMS scheme, security depends on three things: the hardness of $\mathbf{DCK}_{q,n}$ problems, the hardness of ring version short integer solution (Ring-SIS) problem and the hardness of finding pre-images in the random oracle H. The output of random oracle H_2 is in D_{32}^n, which implies that finding pre-images is 160 bits hard. As for the security of lattice problems, we use the extensive experiments introduced by [34], [35] to analyze the hardness of lattice reductions and to obtain the root Hermite factor. In Table 2, we present four sets of parameters.

Now we give a detailed explanation about parameter choices. n is a positive integer that is a power of 2. The modulus q is chosen to be a prime and convergent to 1 modulo $2n$. The number of signers is denoted by ℓ which is a small integer impacting the communication overheads of the third-round interaction. It is reasonable to keep the parameter ℓ smaller than 20. The coefficient range parameter d for the random polynomial $\mathbf{y}_{i,j}$ is extremely important and controls the trade-off among the security, the sizes of signatures and the number of repetitions of signing protocol execution. If the parameter d is too small, then the single signature $\mathbf{z}_{i,f}$ lies in $\mathcal{R}^q_{\theta d-32}$ with an extremely small probability. On the other hand, if d is too big, the sizes of public key and signature will be too large. The parameter d in the BS scheme is the

same as αd in the PLMS scheme, which means that the final multisignatue $\mathbf{z}$ is in the same range. When it comes to the range of rejection sampling, $\mathcal{R}^q_{d-32}$ and $\mathcal{R}^q_{\theta d-32}$ are suitable for the BS scheme and PLMS scheme, respectively. Follow the work [15], the exact probability that $\mathbf{z}_1$, $\mathbf{z}_2$ will be in $\mathcal{R}^{p^n}_{d-32}$ is $\left(1-\frac{64}{2d+1}\right)^{2n}$. The basis number α requires to meet the condition $2^\alpha \geq \left[\frac{1}{\left(1-\frac{64}{2\theta d+1}\right)^{2n}}\right]^\ell$, so α can be represented with $\alpha \geq \ell \left\lceil log^{\frac{1}{\left(1-\frac{64}{2\theta d+1}\right)^{2n}}} \right\rceil$. θ denotes the number of 1 in $[f]_2^\alpha$, so $\theta \in [1, \alpha]$. We choose $\theta = \frac{\alpha}{2}$ to show the average case among the expected number of repetitions. Actually, the probability that the signature $\mathbf{z}_i$ will be $\mathcal{R}^q_{\frac{\alpha}{2}d-32}$ in the PLMS scheme is smaller than that of the BS scheme because the probability formula $\left(1-\frac{64}{2d+1}\right)^{2n}$ depicted in [15] is an increasing function. The probability of not restarting the MSign protocol can be denoted by

$$Prob\,[success] = 1 - \left[1 - \left(1 - \frac{64}{2 \cdot \frac{\alpha}{2} \cdot d + 1}\right)^{2n\ell}\right]^{2^\alpha}$$

yielding $E = 1/Prob\,[success]$ expected number of trials. According to the recommended parameter setI listed in Table 2, each execution of the MSign protocol will produce a multisignature with probability 0.999.

Next, we explain how to calculate the private key, public key, and signature sizes. We will take the concrete parameters from setI listed in Table 2 as an example. The private key $\mathbf{sk}_i$ consists of two polynomials $\mathbf{s}_i$, $\mathbf{v}_i$ chosen uniformly at random from $\mathcal{R}^q_1$, so it can be represented with $2n\lceil log^3 \rceil = 1623$ bits. The public key $\mathbf{pk}_i$ is in $\mathcal{R}^q$, so the size of $\mathbf{pk}_i$ can be represented with $n\lceil log^q \rceil = 13824$ bits. The signature σ consists of polynomials $\mathbf{z}$ and $\mathbf{c}_1 \cdots \mathbf{c}_l$. Since $\mathbf{z}$ is in $\mathcal{R}^q_{\ell\cdot(\alpha d-32)} \times \mathcal{R}^q_{\ell\cdot(\alpha d-32)}$, so it can be represented with $2n\left\lceil log^{2\ell\cdot(\alpha d-32)+1}\right\rceil = 19785$ bits. And each $\mathbf{c}_i$ for $1 \leq i \leq \ell$ is 160 bits, the total signature size is 20585 bits.

Using the above quantities, we can calculate the root Hermite factor which depends on the quality of the lattice-reduction algorithm being used. According to the experiments in [34], [35], a factor of currently best lattice-reduction algorithms is around 1.01, and we can obtain the concrete root Hermite factor of our scheme with different sets of parameters by using the equation $\delta = \left(\beta/\sqrt{q}\right)^{\frac{1}{2n}}$.

B. EXPERIMENT

We implement both BS scheme and PLMS scheme with five clients using three different clouds in the setting of Ubuntu 16.04 64 Bit. Experiments are performed on the following machines: Two clients in Huawei Cloud with Intel(R) Xeon(R) Gold 6278C CPU at 2.60GHz and 4GB RAM, two clients in Tianyi Cloud with Intel(R) Xeon(R) Gold 6161 CPU at 2.20GHz and 2GB RAM, and one client in Alibaba Cloud with Intel(R) Xeon(R) CPU E5-2682 v4 at 2.50GHz and 2GB RAM. All software has complied with gcc-5.4.0 and compiler flags -g++-g -O2 -std= $c++11$ -march=native. We use NTL[3] for the cryptographic primitives and data structures, and socket for sending or receiving data on a computer network. We report the average time of 1000 signature generations for message sizes of 100 bytes including broadcasting the data on the internet in Table 3.

TABLE 3. Time comparisons between BS scheme and PLMS scheme.

Scheme \ Time (s)	MKeyGen	MSign	MVerify
BS	1.172831	40.547804	0.007695
PLMS	1.268363	13.052704	0.007794

Remark 2: We make comparisons between the BS scheme and our PLMS scheme with parameters which are listed in Table 2 **SetI**, in terms of running time in MKeyGen algorithm, MSign algorithm and MVerify algorithm.

As depicted in Table 3, the timings for MSign in BS scheme are 40.547804s, which is over 3 times of those of our PLMS scheme. On the other hand, the timings for MKeyGen algorithm and MVerify algorithm between the BS scheme and our PLMS scheme are close. Although the repetitions of BS is over 12 times of PLMS, each execution of PLMS has to compute more matrix multiplications. It is reasonable that the overall running time of PLMS to produce a multisignature is less than one third of that of BS. Therefore, our scheme PLMS beats the scheme BS in terms of theoretical analysis and experimental results.

VII. CONCLUSION

Multisignature is a special kind of digital signature scheme which allows a group of signers to produce a compact signature cooperatively on a common message. In this paper, we proposed a practical lattice-based multisignature scheme and extend it to support public key aggregation with acceptable performance. In the random oracle model, our schemes have been proven to be secure under the hardness of the ring version short integer solution problem. Our schemes give a better tradeoff between parameters and communications, as we use slightly larger parameters than GLP scheme and avoid the restart of signing protocol, and we provide corresponding experimental data. It is convenient to use our schemes for small signer group. How to construct lattice-based multisignature schemes for large signer group with almost the same performance as ours needs further research.

APPENDIXES

APPENDIX A

THE PROOF OF THEOREM 1

Proof: Let $x = (\mathcal{R}^q, \mathbf{A}, d)$, where $\mathbf{A} = (\mathbf{a}, \mathbf{1})$ and $\mathbf{a}$ is a random polynomial chosen uniformly from $\mathcal{R}^q$. Also, let $D_{H_1} = \mathcal{R}^q$ and $D_{H_2} = \{\mathbf{l} : \mathbf{l} \in \{-1, 0, 1\}^\kappa, \|\mathbf{l}\| \leqslant \kappa\}$ denote

[3]https://www.shoup.net/ntl/

the ranges of random oracles H_1 and H_2. Given a forger $\mathcal{F}$ that can break the security of our multisignature scheme, consider the following algorithm $\mathcal{A}$.

On inputs x, random oracle responses $\lambda_1, \ldots \lambda_{\alpha \cdot q_H}$ which corresponds to H_2, and the random coin $\rho_{\mathcal{A}}$, the algorithm $\mathcal{A}$ selects two polynomials $\mathbf{s}^*$ and $\mathbf{v}^*$ from $\mathcal{R}_1^q$ uniformly at random. It sets $\mathbf{sk}^* = (\mathbf{s}^*, \mathbf{v}^*)$ as the private key and computes the public key $\mathbf{pk}^* = \mathbf{A} \cdot \mathbf{sk}^*$. Then algorithm $\mathcal{A}$ runs the forger $\mathcal{F}$ on input the parameter x and the target public key $\mathbf{pk}^*$ by simulating the random oracle queries and signing queries as follows.

Hash Function Queries: To simulate random oracle queries, the algorithm $\mathcal{A}$ initializes associated lists $L_1[\cdot], L_2[\cdot]$ to store answered values for random oracle queries on H_1, H_2 respectively and the associated list $L_3[\cdot]$ to store different public keys by using the unique indices $0 \leqslant i \leqslant (q_H + q_S)\ell_{max}$. Let $L_3\left[\mathbf{pk}^*\right] \longleftarrow 0$, which means the public key of the honest signer is identified by index 0. The algorithm $\mathcal{A}$ also prepares one counter *ctr* (initially zero) to record the number of queries of H_2.

- When a query $H_1(\mathbf{r}_{i,j})$ is asked, the algorithm $\mathcal{A}$ checks the content of $L_1\left[\mathbf{r}_{i,j}\right]$. If $L_1\left[\mathbf{r}_{i,j}\right]$ has not been defined before, $\mathcal{A}$ selects a random polynomial from $\mathcal{R}^q$ uniformly and sets it to $L_1\left[\mathbf{r}_{i,j}\right]$. Finally, $\mathcal{A}$ returns $L_1\left[\mathbf{r}_{i,j}\right]$ to the forger $\mathcal{F}$.
- When a query $H_2\left(\mathbf{pk}_i, \mathbf{r}_f, \mathbf{pk}, m\right)$ is asked, the algorithm $\mathcal{A}$ checks the content of $L_2\left[\mathbf{pk}_i, \mathbf{r}_f, \mathbf{pk}, m\right]$. If it has not been defined before, $\mathcal{A}$ increases *ctr* and sets $L_2\left[\mathbf{pk}_i, \mathbf{r}_f, \mathbf{pk}, m\right] = \lambda_{ctr}$. Then, $\mathcal{A}$ returns $L_2\left[\mathbf{pk}_i, \mathbf{r}_f, \mathbf{pk}, m\right]$ to the forger $\mathcal{F}$.

Signing Queries. During the simulation, when the forger $\mathcal{F}$ requests a multisignature on the message m with public key set $\mathbf{pk}$, $\mathcal{A}$ answers as follows.

Let $\alpha \geqslant \ell \left\lceil log^{\frac{1}{\left(1-\frac{64}{2\theta d+1}\right)^{2n}}} \right\rceil$. In the first round of **MSign**, the algorithm $\mathcal{A}$ checks whether $\mathbf{pk}^* \in \mathbf{pk}$ at first. If it is not the case, $\mathcal{A}$ outputs $\perp$, else parses $\mathbf{pk} = \left\{\mathbf{pk}^* = \mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$. Subsequently, the algorithm $\mathcal{A}$ chooses $\mathbf{z}_{1,j} \xleftarrow{\$} \mathcal{R}^q_{\alpha d-32} \times \mathcal{R}^q_{\alpha d-32}$ and sets $\mathbf{c}_{1,j} = \lambda_{ctr+j}$, and computes $\mathbf{r}_{1,j} = \mathbf{A}\mathbf{z}_{1,j} - \mathbf{pk}_1 \cdot \mathbf{c}_{1,j}$, then queries $\mathbf{t}_{1,j} = H_1\left(\mathbf{r}_{1,j}\right)$ and for $1 \leqslant j \leqslant \alpha$. At last, the algorithm $\mathcal{A}$ sets $ctr = ctr + \alpha$ and sends $\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,\alpha}$ to other signers.

In the second round of **MSign**, after receiving $\mathbf{t}_{k,1}$, $\mathbf{t}_{k,2}, \ldots, \mathbf{t}_{k,\alpha}$ from the signer k for $2 \leqslant k \leqslant \ell$, the algorithm $\mathcal{A}$ checks the content of $L_1\left[\mathbf{r}_{k,j}\right]$ for each $\mathbf{t}_{k,j}$. If any $\mathbf{t}_{k,j}$ can not be found in $L_1\left[\mathbf{r}_{k,j}\right]$ for $2 \leqslant k \leqslant \ell$ and $1 \leqslant j \leqslant \alpha$, $\mathcal{A}$ sets the flag **alter** to be *true* and sends another tuples of $\mathbf{r}_{1,1}, \mathbf{r}_{1,2}, \ldots, \mathbf{r}_{1,\alpha}$ which are chosen uniformly at random from $\mathcal{R}^q$ to other signers. If the associated list L_1 contains more than one $\mathbf{t}_{k,j}$ such that $H_1\left(\mathbf{r}_{k,j}\right) = H_1\left(\mathbf{r}_{k,j'}\right) = \mathbf{t}_{k,j}$ for $\mathbf{r}_{k,j} \neq \mathbf{r}_{k,j'}$, the event $\mathbf{bad}_1$ occured and $\mathcal{A}$ stops the protocol by returning $\perp$. Otherwise, $\mathcal{A}$ computes $\mathbf{r}_{k,f} = \left(\mathbf{r}_{k,1}\, \mathbf{r}_{k,2}\, \ldots\, \mathbf{r}_{k,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant k \leqslant \ell$ and $1 \leqslant f \leqslant 2^\alpha$, and $\mathbf{r}_f = \sum_{k=1}^{\ell} \mathbf{r}_{k,f}\ mod\ q$. Then, the algorithm $\mathcal{A}$ computes $\mathbf{c}_{1,f} = \left(\mathbf{c}_{1,1}\, \mathbf{c}_{1,2}\, \ldots \mathbf{c}_{1,\alpha}\right) \cdot [f]_2^\alpha\ mod\ q$, puts $\mathbf{c}_{1,f}$ in $L_2\left[\mathbf{pk}_1, \mathbf{r}_f, \mathbf{pk}, m\right]$, chooses random values from $\mathcal{R}^q$ and puts them in $L_2\left[\mathbf{pk}_k, \mathbf{r}_f, \mathbf{pk}, m\right]$ for $1 \leqslant f \leqslant 2^\alpha$ and $2 \leq k \leq \ell$. Subsequently, the algorithm $\mathcal{A}$ computes $\mathbf{z}_{1,f} = \left(\mathbf{z}_{1,1}\, \mathbf{z}_{1,2}\, \ldots \mathbf{z}_{1,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant f \leqslant 2^\alpha$ and puts the appropriate tuple of subscript f into the set S_1. Finally, $\mathcal{A}$ broadcasts $\mathbf{r}_{1,1}, \mathbf{r}_{1,2}, \ldots, \mathbf{r}_{1,\alpha}$ to other signers.

In the third round of **MSign**, after receiving $\mathbf{r}_{k,1}\, \mathbf{r}_{k,2}\, \ldots\, \mathbf{r}_{k,\alpha}$ from the signer k for $2 \leqslant k \leqslant \ell$, $\mathcal{A}$ checks whether $\mathbf{t}_{k,j} = H_1\left(\mathbf{r}_{k,j}\right)$. If there is any $\mathbf{r}_{k,j}$ such that $H_1\left(\mathbf{r}_{k,j}\right) \neq \mathbf{t}_{k,j}$, $\mathcal{A}$ aborts the protocol by returning $\perp$. If all $\mathbf{r}_{k,j}$ are satisfied $H_1\left(\mathbf{r}_{k,j}\right) = \mathbf{t}_{k,j}$ for $2 \leqslant k \leqslant \ell$, $1 \leqslant j \leqslant 2^\alpha$ and **alter** $=$ *true*, the event $\mathbf{bad}_2$ occurred and $\mathcal{A}$ aborts the protocol by returning $\perp$. Otherwise, the algorithm $\mathcal{A}$ sends S_1 to other signers.

In the fourth round of **MSign**, after receiving all sets $S_2, S_3, \ldots, S_\ell$ from all other signers, $\mathcal{A}$ computes the intersection of all sets $S_1, S_2, \ldots, S_\ell$. If the intersection is not empty, $\mathcal{A}$ chooses the smallest index $\bar{f}$ and broadcasts $\mathbf{z}_{1,\bar{f}}, \mathbf{c}_{1,\bar{f}}$ to other signers. Otherwise, it restarts the signing protocol.

After receiving all $\mathbf{z}_{2,\bar{f}}, \mathbf{c}_{2,\bar{f}}, \ldots, \mathbf{z}_{\ell,\bar{f}}, \mathbf{c}_{\ell,\bar{f}}$ from other signers, $\mathcal{A}$ computes $\mathbf{z}_{\bar{f}} = \sum_{k=1}^{\ell} \mathbf{z}_{k,\bar{f}} mod\ q$ and returns the multisignature $\sigma = \left(\mathbf{z}_{\bar{f}}, \mathbf{c}_{1,\bar{f}}, \cdots, \mathbf{c}_{\ell,\bar{f}}\right)$.

If $\mathcal{F}$ returns $\perp$, $\mathcal{A}$ outputs $\perp$ as well. Otherwise, the forger $\mathcal{F}$ outputs a multisignature $\sigma = (\mathbf{z}, \mathbf{c}_1, \cdots, \mathbf{c}_\ell)$ on message m under the public key set $\mathbf{pk}$. The algorithm $\mathcal{A}$ parses $\mathbf{pk}$ as $\left\{\mathbf{pk}_1 = \mathbf{pk}^*, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$, computes $\mathbf{r} = \mathbf{A}\mathbf{z} - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i$ and checks whether $\|\mathbf{z}\|_\infty \leqslant \ell \cdot (\alpha d - 32)$ and $H_2\left(\mathbf{pk}_i, \mathbf{r}, \mathbf{pk}, m\right) = \mathbf{c}_i$. Let j denote the index such that $H_2\left(\mathbf{pk}_1, \mathbf{r}, \mathbf{pk}, m\right) = \lambda_j$. If the forgery $\sigma = (\mathbf{z}, \mathbf{c}_1, \cdots, \mathbf{c}_\ell)$ is valid, the algorithm $\mathcal{A}$ returns $(j, (\mathbf{z}, \mathbf{c}_1, \cdots, \mathbf{c}_\ell, \mathbf{r}, \mathbf{pk}))$. If not, $\mathcal{A}$ halts with outputs $(0, \varepsilon)$.

Now we consider the accepting probability of $\mathcal{A}$ as defined in the general forking lemma [6]. When these event $\mathbf{bad}_i$ $(1 \leqslant i \leqslant 2)$ do not occur, the simulation of $\mathcal{A}$ is indistinguishable from the real environment, then

$$acc(\mathcal{A}) \geqslant \delta - (q_H + \ell_{max} \cdot q_S \cdot \alpha)^2 / q^n - \ell_{max} \cdot q_S \cdot \alpha / q^n$$

Now we construct an algorithm $\mathcal{B}$ on input x, runs the general forking algorithm $GF_{\mathcal{A}}$ to obtain the output

$$\left(j, (\mathbf{z}, \mathbf{c}_1, \cdots, \mathbf{c}_\ell, \mathbf{r}, \mathbf{pk}); j', \left(\mathbf{z}', \mathbf{c}_1', \cdots, \mathbf{c}_\ell', \mathbf{r}'\mathbf{pk}'\right)\right)$$

In both executions of $GF_{\mathcal{A}}$, the forking point is located in the $H_2\left(\mathbf{pk}_1, \mathbf{r}, \mathbf{pk}, m\right)$. This means that up to this point, the environments simulated by the algorithm $\mathcal{A}$ are identical in the first and second execution. Therefore, all these arguments in the query H_2 are identical in both executions, which implies that $\mathbf{r} = \mathbf{r}'$,$\mathbf{pk} = \mathbf{pk}'$ and $m = m'$. Let ℓ^* denote the times of occurrences of $\mathbf{pk}_1$ in $\mathbf{pk}$. According to the output of $GF_{\mathcal{A}}$, we can obtain $\mathbf{A}\mathbf{z} - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i = \mathbf{A}\mathbf{z}' - \sum_{i=1}^{\ell} \mathbf{pk}_i \cdot \mathbf{c}_i'$, which implies that

$$\mathbf{A}\left[\mathbf{z} - \mathbf{z}' + |\ell^*| \cdot \mathbf{sk}^* \left(\mathbf{c}_1{}' - \mathbf{c}_1\right)\right] = 0$$

Due to $\|\mathbf{z}\|_\infty$, $\left\|\mathbf{z}'\right\|_\infty \leqslant \ell \cdot (\alpha d - 32)$ and $\left\|\mathbf{sk}^* \mathbf{c}_1\right\|_\infty$, $\left\|\mathbf{sk}^* \mathbf{c}_1'\right\|_\infty \leqslant 32$, we have $\left\|\mathbf{z} - \mathbf{z}' + |\ell^*| \cdot \mathbf{sk}^* \left(\mathbf{c}' - \mathbf{c}\right)\right\|_\infty \leqslant 2\ell \cdot (\alpha d - 32) + 64\,|\ell^*|$.

Now we proceed to show that

$$\mathbf{z} - \mathbf{z}' + \left|\ell^*\right| \cdot \mathbf{sk}^* \left(\mathbf{c}_1{}' - \mathbf{c}_1\right) \neq 0.$$

Following the idea introduced in [17], we use $\mathbf{sk}'$ with slightly larger coefficients (so that there exists another $\mathbf{sk}''$ such that $\mathbf{A} \cdot \mathbf{sk}' = \mathbf{A} \cdot \mathbf{sk}''$). If

$$\mathbf{z} - \mathbf{z}' + \left|\ell^*\right| \cdot \mathbf{sk}' \left(\mathbf{c}_1{}' - \mathbf{c}_1\right) = 0,$$

then there exists another $\mathbf{sk}''$ such that

$$\mathbf{z} - \mathbf{z}' + \left|\ell^*\right| \cdot \mathbf{sk}'' \left(\mathbf{c}_1{}' - \mathbf{c}_1\right) \neq 0.$$

Since we never use the private key to generate signatures during the simulation, the forger does not know whether we know a secret key like $\mathbf{sk}'$ or like $\mathbf{sk}''$. So, we will get a non-zero answer with probability at least $\frac{1}{2}$, since each key has an equal probability of being chosen. Therefore, this solves Ring-SIS$_{2n,q,\beta}$ for $\beta \leqslant 2\ell \cdot (\alpha d - 32) + 64 \left|\ell^*\right|$, which is assumed to be hard.

Finally, the accepting probability of the algorithm $\mathcal{B}$ is *blacksquare*

$$acc\left(\mathcal{B}\right) \geqslant acc\left(\mathcal{A}\right) \cdot \left(\frac{acc\left(\mathcal{A}\right)}{q_T} - \frac{1}{\left|D_{H_2}\right|}\right)$$

■

APPENDIX B
THE PROOF OF THEOREM 2

Proof: Let $D_{H_0} = \{\mathbf{l} : \mathbf{l} \in \{-1, 0, 1\}^\kappa, \|\mathbf{l}\| \leqslant \kappa\}, D_{H_1} = \mathcal{R}^q$, and $D_{H_2} = \{\mathbf{l} : \mathbf{l} \in \{-1, 0, 1\}^\kappa, \|\mathbf{l}\| \leqslant \kappa\}$ denote the ranges of random oracles H_0, H_1, and H_2. Also let $x = (\mathcal{R}^q, \mathbf{A}, d)$, where $\mathbf{A} = (\mathbf{a}, \mathbf{1})$ and $\mathbf{a}$ is a random polynomial chosen uniformly from $\mathcal{R}^q$. Given a forger $\mathcal{F}$ that can break the security of our multisignature scheme, consider the following algorithm $\mathcal{A}$.

On inputs x, random oracle responses $\lambda_{0,1}, \ldots \lambda_{0,q_H}$ and $\lambda_{2,1}, \ldots \lambda_{2,q_H}$ which correspond to H_0 and H_2 respectively, and the random coin $\rho_{\mathcal{A}}$, the algorithm $\mathcal{A}$ selects two polynomials $\mathbf{s}^*$ and $\mathbf{v}^*$ from $\mathcal{R}_1^q$ uniformly at random. It sets $\mathbf{sk}^* = (\mathbf{s}^*, \mathbf{v}^*)$ as the private key and computes the public key $\mathbf{pk}^* = \mathbf{A} \cdot \mathbf{sk}^*$. Then algorithm $\mathcal{A}$ runs the forger $\mathcal{F}$ on input the parameter x and the target public key $\mathbf{pk}^*$ by simulating the random oracle queries and signing queries as follows.

Hash Function Queries. To simulate random oracle queries, the algorithm $\mathcal{A}$ initializes associated lists $L_0[\cdot]$, $L_1[\cdot], L_2[\cdot]$ to store answered values for random oracle queries on H_0, H_1, H_2 respectively, and the associated list $L_3[\cdot]$ to store different public keys by using the unique indices $0 \leqslant i \leqslant (q_H + q_S)\ell_{max}$. Let $L_3\left[\mathbf{pk}^*\right] \longleftarrow 0$, which means the public key of the honest signer is identified by index 0. The algorithm $\mathcal{A}$ also prepares two counters ctr_0 and ctr_1 (initially zero) to record the number of queries of H_0 and H_2 respectively.

- When a query $H_0(\mathbf{pk}, \mathbf{pk}_i)$ is asked, the algorithm $\mathcal{A}$ checks the content of $L_3\left[\mathbf{pk}_i\right]$ at first. If it has not been defined yet, $\mathcal{A}$ puts $\mathbf{pk}_i$ in to the list $L_3\left[\mathbf{pk}_i\right]$. Then the algorithm $\mathcal{A}$ parses the set of public keys $\mathbf{pk}$ as $\left\{\mathbf{pk}_1, \mathbf{pk}_2, \cdots, \mathbf{pk}_\ell\right\}$ and checks whether each public key in $\mathbf{pk}$ has been defined in the list L_3. For each $\mathbf{pk}_j \in \mathbf{pk}$ $(1 \leqslant j \leqslant \ell)$, after ensuring that $L_3\left[\mathbf{pk}_j\right]$ is defined, the algorithm $\mathcal{A}$ checks the content of $L_0\left[\mathbf{pk}, \mathbf{pk}_j\right]$. If it has not been defined before, $\mathcal{A}$ assigns $L_0\left(\mathbf{pk}, \mathbf{pk}_j\right)$ according to the following three types:
  (1) $\mathbf{pk}_j = \mathbf{pk}^*$ and $\mathbf{pk}^* \in \mathbf{pk}$.
  (2) $\mathbf{pk}_j \neq \mathbf{pk}^*$ and $\mathbf{pk}^* \in \mathbf{pk}$.
  (3) $\mathbf{pk}^* \notin \mathbf{pk}$.
  The algorithm $\mathcal{A}$ increases ctr_0 and assigns $L_0\left[\mathbf{pk}, \mathbf{pk}_j\right] = \lambda_{0,ctr_0}$ in type (1). As for type (2) and (3), $\mathcal{A}$ assigns random elements selected from D_{32}^n to $L_0\left[\mathbf{pk}, \mathbf{pk}_j\right]$. After assigning $L_0\left(\mathbf{pk}, \mathbf{pk}_j\right)$ for $1 \leqslant j \leqslant \ell$, $\mathcal{A}$ computes the aggregated public key $\mathbf{apk}$. If the forger $\mathcal{F}$ already made any random oracle query $H_2\left(\mathbf{apk}, \cdot, \cdot\right)$ or signing query which contains $\mathbf{apk}$, then the event $\mathbf{bad}_1$ occurred and $\mathcal{A}$ aborts the protocol by returning $\perp$. Otherwise, if $\mathbf{pk}_i \in \mathbf{pk}$ then the algorithm $\mathcal{A}$ returns $L_0\left[\mathbf{pk}, \mathbf{pk}_i\right]$ to the forger $\mathcal{F}$, else it returns a random element selected from D_{32}^n and updates $L_0\left[\mathbf{pk}, \mathbf{pk}_i\right]$ accordingly.
- When a query $H_1(\mathbf{r}_{i,j})$ is asked, the algorithm $\mathcal{A}$ checks the content of $L_1\left[\mathbf{r}_{i,j}\right]$. If $L_1\left[\mathbf{r}_{i,j}\right]$ has not been defined before, $\mathcal{A}$ selects a random polynomial from $\mathcal{R}^q$ uniformly and sets it to $L_1\left[\mathbf{r}_{i,j}\right]$. Finally, $\mathcal{A}$ returns $L_1\left[\mathbf{r}_{i,j}\right]$ to the forger $\mathcal{F}$.
- When a query $H_2\left(\mathbf{apk}, \mathbf{r}_f, m\right)$ is asked, the algorithm $\mathcal{A}$ checks the content of $L_2\left[\mathbf{apk}, \mathbf{r}_f, m\right]$. If it has not been defined before, $\mathcal{A}$ increases ctr_1 and sets $L_2\left[\mathbf{apk}, \mathbf{r}_f, m\right] = \lambda_{2,ctr_1}$. Then, $\mathcal{A}$ returns $L_2\left[\mathbf{apk}, \mathbf{r}_f, m\right]$ to the forger $\mathcal{F}$.

Signing Queries. During the simulation, when the forger $\mathcal{F}$ requests a multisignature on the message m with public key set $\mathbf{pk}$, $\mathcal{A}$ answers as follows.

In the first round of **MSign**, the algorithm $\mathcal{A}$ checks whether $\mathbf{pk}^* \in \mathbf{pk}$ at first. If it is not the case, $\mathcal{A}$ outputs $\perp$, else parses $\mathbf{pk} = \left\{\mathbf{pk}^* = \mathbf{pk}_1, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$. Let $\alpha \geqslant \ell \left\lceil log^{\frac{1}{\left(1 - \frac{2048}{2^{\theta} d + 1}\right)^{2n}}} \right\rceil$. Then the algorithm $\mathcal{A}$ computes the aggregated public key $\mathbf{apk} = \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i \ mod\ q$ by querying $H_0\left(\mathbf{pk}, \mathbf{pk}_i\right)$ to obtain $\mathbf{u}_i = L_0\left[\mathbf{pk}, \mathbf{pk}_i\right]$ for $1 \leqslant j \leqslant \ell$. Subsequently, the algorithm $\mathcal{A}$ chooses $\mathbf{z}_{1,j} \xleftarrow{\$} \mathcal{R}^q_{\alpha d - 1024} \times \mathcal{R}^q_{\alpha d - 1024}$ and $\mathbf{c}_j = \lambda_{2,ctr1+j}$, and computes $\mathbf{r}_{1,j} = \mathbf{A}\mathbf{z}_{1,j} - \mathbf{c}_j \cdot \mathbf{u}_1 \cdot \mathbf{pk}_1$, then queries $\mathbf{t}_{1,j} = H_1\left(\mathbf{r}_{1,j}\right)$ and for $1 \leqslant j \leqslant \alpha$. At last, the algorithm $\mathcal{A}$ sets $ctr_1 = ctr_1 + \alpha$ and sends $\mathbf{t}_{1,1}, \mathbf{t}_{1,2}, \ldots, \mathbf{t}_{1,\alpha}$ to other signers.

In the second round of **MSign**, after receiving $\mathbf{t}_{k,1}, \mathbf{t}_{k,2}, \ldots, \mathbf{t}_{k,\alpha}$ from the signer k for $2 \leqslant k \leqslant \ell$, the algorithm $\mathcal{A}$ checks the content of $L_1\left[\mathbf{r}_{k,j}\right]$ for each $\mathbf{t}_{k,j}$. If any $\mathbf{t}_{k,j}$ can not be found in $L_1\left[\mathbf{r}_{k,j}\right]$ for $2 \leqslant k \leqslant \ell$ and $1 \leqslant j \leqslant \alpha$, $\mathcal{A}$ sets the flag **alter** to be *true* and sends another tuples of $\mathbf{r}_{1,1}, \mathbf{r}_{1,2}, \ldots, \mathbf{r}_{1,\alpha}$ which are chosen uniformly at random from $\mathcal{R}^q$ to other signers. If the associated list L_1 contains more than one $\mathbf{t}_{k,j}$ such that $H_1\left(\mathbf{r}_{k,j}\right) = H_1\left(\mathbf{r}_{k,j}{}'\right) = \mathbf{t}_{k,j}$ for $\mathbf{r}_{k,j} \neq \mathbf{r}_{k,j}{}'$, the event $\mathbf{bad}_2$ occured and $\mathcal{A}$ stops the protocol by returning $\perp$. Otherwise, $\mathcal{A}$ computes

$\mathbf{r}_{k,f} = \left(\mathbf{r}_{k,1}\ \mathbf{r}_{k,2}\ \ldots\ \mathbf{r}_{k,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant k \leqslant \ell$ and $1 \leqslant f \leqslant 2^\alpha$, and $\mathbf{r}_f = \sum_{k=1}^{\ell} \mathbf{r}_{k,f}\ mod\ q$. Then, the algorithm $\mathcal{A}$ computes $\mathbf{c}_f = (\mathbf{c}_1\ \mathbf{c}_2\ \ldots \mathbf{c}_\alpha) \cdot [f]_2^\alpha\ mod\ q$ and puts $\mathbf{c}_f$ in $L_2\left[\mathbf{apk}, \mathbf{r}_f, m\right]$ for $1 \leqslant f \leqslant 2^\alpha$. Subsequently, the algorithm $\mathcal{A}$ computes $\mathbf{z}_{1,f} = \left(\mathbf{z}_{1,1}\ \mathbf{z}_{1,2}\ \ldots \mathbf{z}_{1,\alpha}\right) \cdot [f]_2^\alpha$ for $1 \leqslant f \leqslant 2^\alpha$ and puts the appropriate tuple of subscript f into the set S_1. Finally, $\mathcal{A}$ broadcasts $\mathbf{r}_{1,1}, \mathbf{r}_{1,2}, \ldots, \mathbf{r}_{1,\alpha}$ to other signers.

In the third round of **MSign**, after receiving $\mathbf{r}_{k,1}\ \mathbf{r}_{k,2}\ \ldots\ \mathbf{r}_{k,\alpha}$ from the signer k for $2 \leqslant k \leqslant \ell$, $\mathcal{A}$ checks whether $\mathbf{t}_{k,j} = H_1\left(\mathbf{r}_{k,j}\right)$. If there is any $\mathbf{r}_{k,j}$ such that $H_1\left(\mathbf{r}_{k,j}\right) \neq \mathbf{t}_{k,j}$, $\mathcal{A}$ aborts the protocol by returning $\perp$. If all $\mathbf{r}_{k,j}$ are satisfied $H_1\left(\mathbf{r}_{k,j}\right) = \mathbf{t}_{k,j}$ for $2 \leqslant k \leqslant \ell$, $1 \leqslant j \leqslant 2^\alpha$ and **alter** $= true$, the event $\mathbf{bad}_3$ occurred and $\mathcal{A}$ aborts the protocol by returning $\perp$. Otherwise, the algorithm $\mathcal{A}$ sends S_1 to other signers.

In the fourth round of **MSign**, after receiving all sets $S_2, S_3, \ldots, S_\ell$ from all other signers, $\mathcal{A}$ computes the intersection of all sets $S_1, S_2, \ldots, S_\ell$. If the intersection is not empty, $\mathcal{A}$ chooses the smallest index $\bar{f}$ and broadcasts $\mathbf{z}_{1,\bar{f}}$ to other signers. Otherwise, it restarts the signing protocol.

After receiving all $\mathbf{z}_{2,\bar{f}}, \mathbf{z}_{3,\bar{f}}, \ldots, \mathbf{z}_{\ell,\bar{f}}$ from other signers, $\mathcal{A}$ computes $\mathbf{z}_{\bar{f}} = \sum_{k=1}^{\ell} \mathbf{z}_{k,\bar{f}} mod\ q$ and returns the multisignature $\sigma = \left(\mathbf{z}_{\bar{f}}, \mathbf{c}_{\bar{f}}\right)$.

If $\mathcal{F}$ returns $\perp$, $\mathcal{A}$ outputs $\perp$ as well. Otherwise, the forger $\mathcal{F}$ outputs a multisignature $\sigma = (\mathbf{z}, \mathbf{c})$ on message m under the public key set **pk**. The algorithm $\mathcal{A}$ parses **pk** as $\left\{\mathbf{pk}_1 = \mathbf{pk}^*, \mathbf{pk}_2, \ldots, \mathbf{pk}_\ell\right\}$, queries $H_0\left(\mathbf{pk}, \mathbf{pk}_i\right) = \mathbf{u}_i$ and computes the aggregated public key $\mathbf{apk} = \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i\ mod\ q$. Then, $\mathcal{A}$ computes $\mathbf{r} = \mathbf{Az} - \mathbf{c} \cdot \mathbf{apk}$ and checks whether $\|\mathbf{z}\|_\infty \leqslant \ell \cdot (\alpha d - 1024)$ and $H_2\left(\mathbf{apk}, \mathbf{r}, m\right) = \mathbf{c}$. Let i denote the index such that $H_0\left(\mathbf{pk}, \mathbf{pk}^*\right) = \lambda_{0,i}$ and assume that the equation $H_2\left(\mathbf{apk}, \mathbf{r}, m\right) = \mathbf{c}$ occurred in the j-th query. If the forgery $\sigma = (\mathbf{z}, \mathbf{c})$ is valid, the algorithm $\mathcal{A}$ returns $(j, (i, \mathbf{z}, \mathbf{c}, \mathbf{r}, \mathbf{apk}, \mathbf{pk}, \mathbf{u}_1, \ldots \mathbf{u}_\ell))$. If not, $\mathcal{A}$ halts with outputs $(0, \varepsilon)$.

Now we consider the accepting probability of $\mathcal{A}$ as defined in the general forking lemma [6]. When these event $\mathbf{bad}_i$ $(1 \leqslant i \leqslant 3)$ and **AKColl**[4] do not occur, the simulation of $\mathcal{A}$ is indistinguishable from the real environment, then

$$acc\left(\mathcal{A}\right) = Pr\left[\ the\ forgery\ is\ valid\ \right] - \sum_{i=1}^{3} Pr\left[\mathbf{bad}_i\right] - Pr\left[\mathbf{AKColl}\right]$$

Subsequently, we give explanations about the meanings of above four events and methods to calculate these probabilities as follows:

$\mathbf{bad}_1$: $\mathcal{F}$ has known the aggregated public key **apk** before the assignment $L_0\left[\mathbf{pk}, \mathbf{pk}_i\right]$ finished. Let ℓ^* denote the occurrences of $\mathbf{pk}^*$ in **pk**. In the i-th set of L_0 assignments, the aggregated public key **apk** is computed by $\mathbf{apk} = |\ell^*|\,\lambda_{0,i} \cdot \mathbf{pk}^* + \sum_{\mathbf{pk}_i \in \mathbf{pk}, \mathbf{pk}_i \neq \mathbf{pk}^*} \mathbf{u}_i \cdot \mathbf{pk}_i\ mod\ q$, where $\lambda_{0,i}$ and $\mathbf{u}_i$ are uniformly random in D_{32}^n. Hence, **apk** is uniformly at random in a set of q^n vectors. Since there are at most $(q_H + \ell_{max} \cdot q_S \cdot 2^\alpha)$ defined entries in L_2 and at most $(q_H + \ell_{max} \cdot q_S \cdot 2^\alpha)$ sets of assignments, the event happens with probability $(q_H + \ell_{max} \cdot q_S \cdot 2^\alpha)^2 / q^n$.

$\mathbf{bad}_2$: there is at least one collision occurred in H_1 such that $H_1\left(\mathbf{r}_{k,j}\right) = H_1\left(\mathbf{r}_{k,j}{}'\right) = \mathbf{t}_{k,j}$. All responses of H_1 are chosen uniformly at random from $\mathcal{R}^q$ and there are at most $q_H + \ell_{max} \cdot q_S \cdot \alpha$ queries to H_1, so the event $\mathbf{bad}_2$ happens with probability $\binom{q_H + \ell_{max} \cdot q_S \cdot \alpha}{2} / q^n \leqslant (q_H + \ell_{max} \cdot q_S \cdot \alpha)^2 / q^n$.

$\mathbf{bad}_3$: $\mathcal{F}$ obtains at least one value of $H_1\left(\mathbf{r}_{i,j}\right)$ for $1 \leqslant i \leqslant \ell$ and $1 \leqslant j \leqslant \alpha$ just by predicting rather than querying. The output of $H_1\left(\mathbf{r}_{i,j}\right)$ is uniformly at random in a set of q^n vectors. Therefore, the event $\mathbf{bad}_3$ happens with probability $\ell_{max} \cdot q_S \cdot \alpha / q^n$.

AKColl: in the i-th query, the forger $\mathcal{F}$ obtains a same aggregated public key **apk** from two different public key multisets **pk** and **pk**′ such that $\sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i = \sum_{i=1}^{\ell} \mathbf{u}_i{}' \cdot \mathbf{pk}_i{}'$. Since each aggregated public key **apk** is uniform in a set of q^n ring elements and independent from other aggregated public keys, the event happens with the probability at most $(q_H + \ell_{max} \cdot q_S)^2 / q^n$.

Therefore, the accepting probability of the algorithm $\mathcal{A}$ is

$$\begin{aligned} acc\left(\mathcal{A}\right) \geqslant \delta &- \left(q_H + \ell_{max} \cdot q_S \cdot 2^\alpha\right)^2 / q^n \\ &- \left(q_H + \ell_{max} \cdot q_S \cdot \alpha\right)^2 / q^n \\ &- \ell_{max} \cdot q_S \cdot \alpha / q^n \\ &- \left(q_H + \ell_{max} \cdot q_S\right)^2 / q^n \end{aligned}$$

Now we construct a polynomial algorithm $\mathcal{B}$ on input $\left(x, \lambda_{0,1}, \ldots, \lambda_{0,q_H}, \rho_{\mathcal{B}}\right)$, where $\lambda_{0,1}, \ldots, \lambda_{0,q_H}$ corresponds the random oracle responses of H_0 and $\rho_{\mathcal{B}}$ is the random coin of the algorithm $\mathcal{B}$, runs the general forking algorithm $GF_{\mathcal{A}}$ to obtain the output

$$\left(j, out; j', out'\right),$$

such that $j = j'$,$out = (i, \mathbf{z}, \mathbf{c}, \mathbf{r}, \mathbf{apk}, \mathbf{pk}, \mathbf{u}_1, \ldots, \mathbf{u}_\ell)$, and $out' = \left(i', \mathbf{z}', \mathbf{c}', \mathbf{r}', \mathbf{apk}', \mathbf{pk}', \mathbf{u}_1', \ldots, \mathbf{u}_{\ell'}'\right)$ In both executions of $GF_{\mathcal{A}}$, the forking point is located in the j-th $H_2\left(\mathbf{apk}, \mathbf{r}, m\right)$. This means that up to this point, the environments simulated by the algorithm $\mathcal{A}$ are identical in the first and second execution. Therefore, all these arguments in the query H_2 are identical in both executions, which implies that $\mathbf{apk} = \mathbf{apk}'$, $\mathbf{r} = \mathbf{r}'$ and $m = m'$. The step of querying $H_0\left(\mathbf{pk}, \mathbf{pk}_i\right)$ occurred before the forking point, so the public key sets and the responses of $H_0\left(\mathbf{pk}, \mathbf{pk}_i\right)$ are identical in both executions. Hence, $i = i'$, $\ell = \ell'$, $\mathbf{pk} = \mathbf{pk}'$ and $\mathbf{u}_j = \mathbf{u}_j'$ for $1 \leqslant j \leqslant \ell$. According to the output of $GF_{\mathcal{A}}$, we can obtain $\mathbf{Az} - \mathbf{c} \cdot \mathbf{apk} = \mathbf{Az}' - \mathbf{c}' \cdot \mathbf{apk}'$, which implies that

$$\mathbf{A}\left(\mathbf{z} - \mathbf{z}'\right) + \left(\mathbf{c}' - \mathbf{c}\right) \cdot \mathbf{apk} = 0 \tag{1}$$

Then the algorithm $\mathcal{B}$ outputs (i, ot) such that $ot = \left(\mathbf{z}, \mathbf{z}', \mathbf{c}, \mathbf{c}', \mathbf{apk}, \mathbf{pk}, \mathbf{u}_1, \ldots, \mathbf{u}_\ell\right)$. The accepting probability

[4] If the forger is able to find two different multisets of public keys **pk** and **pk**′ which correspond the same aggregated public key **apk**, the event **AKColl** occured and $\mathcal{A}$ aborts the protocol by returning $\perp$.

of the algorithm $\mathcal{B}$ is

$$acc(\mathcal{B}) \geqslant acc(\mathcal{A}) \cdot \left(\frac{acc(\mathcal{A})}{q_T} - \frac{1}{|D_{H_2}|} \right)$$

Now we construct an algorithm $\mathcal{D}$ on input x, runs the forking algorithm $GF_{\mathcal{B}}$ to obtain the output

$$\left(i, ot; \tilde{i}, \tilde{ot}\right)$$

such that $i = \tilde{i}$, $ot = \left(\mathbf{z}, \mathbf{z}', \mathbf{c}, \mathbf{c}', \mathbf{apk}, \mathbf{pk}, \mathbf{u}_1, \ldots, \mathbf{u}_\ell\right)$, and $\tilde{ot} = \left(\tilde{\mathbf{z}}, \tilde{\mathbf{z}}', \tilde{\mathbf{c}}, \tilde{\mathbf{c}}', \tilde{\mathbf{apk}}, \tilde{\mathbf{pk}}, \tilde{\mathbf{u}_1}, \ldots, \tilde{\mathbf{u}_\ell}\right)$. In both executions of $GF_{\mathcal{B}}$, the forking point is located in the i-th query of $H_0\left(\mathbf{pk}, \mathbf{pk}^*\right)$. This means that up to this point, the environments simulated by the algorithm $\mathcal{A}$ are identical in both executions. Therefore, all these arguments in the query $H_0\left(\mathbf{pk}, \mathbf{pk}^*\right)$ are identical, which implies that $\mathbf{pk} = \tilde{\mathbf{pk}}$ and $\mathbf{pk}^* = \tilde{\mathbf{pk}}^*$. Furthermore, each $H_0(\mathbf{pk}, \mathbf{pk}_j)$ $(\mathbf{pk}_j \neq \mathbf{pk}^*)$ is assigned the same value in both executions, according the description of the algorithm $\mathcal{A}$. Hence, we have $\mathbf{u}_i = \tilde{\mathbf{u}}_i$ for $\mathbf{pk}_i \neq \mathbf{pk}^*$ and $\mathbf{u}_i \neq \tilde{\mathbf{u}}_i$ for $\mathbf{pk}_i = \mathbf{pk}^*$. So $\mathbf{apk} \neq \tilde{\mathbf{apk}}$. According to the output of the algorithm $GF_{\mathcal{B}}$, we can obtain two equations about the aggregated public key $\mathbf{apk} = \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{pk}_i \ mod\ q$ and $\tilde{\mathbf{apk}} = \sum_{i=1}^{\ell} \tilde{\mathbf{u}}_i \cdot \tilde{\mathbf{pk}}_i \ mod\ q$. Let ℓ^* denote the times of occurrences of $\mathbf{pk}^*$ in $\mathbf{pk}$. Combining the equation 1 , we obtain the following equations

$$\mathbf{A}\left[\mathbf{z} - \mathbf{z}' + \left(\mathbf{c}' - \mathbf{c}\right) \sum_{i=1}^{\ell} \mathbf{u}_i \cdot \mathbf{sk}_i\right] = 0 \tag{2}$$

$$\mathbf{A}\left[\tilde{\mathbf{z}} - \tilde{\mathbf{z}}' + \left(\tilde{\mathbf{c}}' - \tilde{\mathbf{c}}\right) \sum_{i=1}^{\ell} \tilde{\mathbf{u}}_i \cdot \mathbf{sk}_i\right] = 0 \tag{3}$$

Through elimination, we obtian

$$\mathbf{A}\left[\mathbf{Z}\tilde{\mathbf{C}} - \tilde{\mathbf{Z}}\mathbf{C} + \left|\ell^*\right| \cdot \mathbf{sk}^*\mathbf{C}\tilde{\mathbf{C}}\mathbf{U}\right] = 0 \tag{4}$$

such that $\mathbf{Z} = \mathbf{z} - \mathbf{z}'$, $\mathbf{C} = \mathbf{c}' - \mathbf{c}$, $\tilde{\mathbf{Z}} = \tilde{\mathbf{z}} - \tilde{\mathbf{z}}'$, $\tilde{\mathbf{C}} = \tilde{\mathbf{c}}' - \tilde{\mathbf{c}}$, and $\mathbf{U} = \mathbf{u}_1 - \tilde{\mathbf{u}_1}$. Due to $\|\mathbf{z}\|_\infty$, $\left\|\mathbf{z}'\right\|_\infty$, $\left\|\tilde{\mathbf{z}}\right\|_\infty$, $\left\|\tilde{\mathbf{z}}'\right\|_\infty \leqslant \ell \cdot (\alpha d - 1024)$, we have $\left\|\mathbf{z} - \mathbf{z}'\right\|_\infty$, $\left\|\tilde{\mathbf{z}} - \tilde{\mathbf{z}}'\right\|_\infty \leqslant 2\ell \cdot (\alpha d - 1024)$. Therefore, we obtain

$$\begin{aligned}&\left\|\mathbf{Z}\tilde{\mathbf{C}} - \tilde{\mathbf{Z}}\mathbf{C} + \left|\ell^*\right| \cdot \mathbf{sk}^*\mathbf{C}\tilde{\mathbf{C}}\mathbf{U}\right\|_\infty \\ &\quad \leqslant 256l \cdot (\alpha d - 1024) + \left|\ell^*\right| \cdot 64^3\end{aligned}$$

Now we proceed to show that

$$\mathbf{Z}\tilde{\mathbf{C}} - \tilde{\mathbf{Z}}\mathbf{C} + \left|\ell^*\right| \cdot \mathbf{sk}^*\mathbf{C}\tilde{\mathbf{C}}\mathbf{U} \neq 0$$

Following the idea introduced in [17], we use $\mathbf{sk}'$ with slightly larger coefficients (so that there exists another $\mathbf{sk}''$ such that $\mathbf{A} \cdot \mathbf{sk}' = \mathbf{A} \cdot \mathbf{sk}''$). If

$$\mathbf{Z}\tilde{\mathbf{C}} - \tilde{\mathbf{Z}}\mathbf{C} + \left|\ell^*\right| \cdot \mathbf{sk}^*\mathbf{C}\tilde{\mathbf{C}}\mathbf{U} = 0$$

then there exists another $\mathbf{sk}''$ such that

$$\mathbf{Z}\tilde{\mathbf{C}} - \tilde{\mathbf{Z}}\mathbf{C} + \left|\ell^*\right| \cdot \mathbf{sk}^*\mathbf{C}\tilde{\mathbf{C}}\mathbf{U} \neq 0$$

Since we never use the private key to generate signatures during the simulation, the forger does not know whether we know a secret key like $\mathbf{sk}'$ or like $\mathbf{sk}''$. So, we will get a non-zero answer with probability at least $\frac{1}{2}$, since each key has an equal probability of being chosen. Therefore, this solves Ring-SIS$_{2n,q,\beta}$ for $\beta \leqslant 256\ell \cdot (\alpha d - 1024) + |\ell^*| \cdot 64^3$, which is assumed to be hard.

Finally, the accepting probability of the algorithm $\mathcal{D}$ is

$$acc(\mathcal{D}) \geqslant acc(\mathcal{B}) \cdot \left(\frac{acc(\mathcal{B})}{q_T} - \frac{1}{|D_{H_0}|} \right)$$

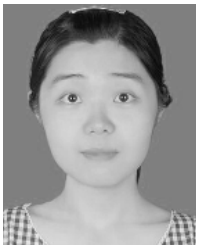
■

REFERENCES

[1] K. Itakura and K. Nakamura, "A public-key cryptosystem suitable for digital multisignatures," *NEC Res. Develop.*, vol. 71, pp. 1–8, Jan. 1983.
[2] S. Goldwasser, S. Micali, and R. L. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM J. Comput.*, vol. 17, no. 2, pp. 281–308, 1988.
[3] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, Mar. 2009.
[4] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie–Hellman-group signature scheme," in *Proc. Int. Workshop Public Key Cryptogr.* Berlin, Germany: Springer, 2003, pp. 31–46.
[5] S. Micali, K. Ohta, and L. Reyzin, "Accountable-subgroup multisignatures," in *Proc. 8th ACM Conf. Comput. Commun. Secur.*, 2001, pp. 245–254.
[6] M. Bellare and G. Neven, "Multi-signatures in the plain public-key model and a general forking lemma," in *Proc. 13th ACM Conf. Comput. Commun. Secur.*, 2006, pp. 390–399.
[7] A. Bagherzandi, J.-H. Cheon, and S. Jarecki, "Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma," in *Proc. 15th ACM Conf. Comput. Commun. Secur.*, 2008, pp. 449–458.
[8] C. Ma, J. Weng, Y. Li, and R. Deng, "Efficient discrete logarithm based multi-signature scheme in the plain public key model," *Des., Codes Cryptogr.*, vol. 54, no. 2, pp. 121–133, 2010.
[9] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Jun. 2017, pp. 557–564.
[10] C.-Y. Li, X.-B. Chen, Y.-L. Chen, Y.-Y. Hou, and J. Li, "A new lattice-based signature scheme in post-quantum blockchain network," *IEEE Access*, vol. 7, pp. 2026–2033, 2019.
[11] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille, "Simple Schnorr multi-signatures with applications to bitcoin," *Des., Codes Cryptogr.*, vol. 87, no. 9, pp. 2139–2164, 2019.
[12] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs, "On the security of two-round multi-signatures," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 780–797.
[13] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2018, pp. 435–464.
[14] R. El Bansarkhani and J. Sturm, "An efficient lattice-based multisignature scheme with applications to bitcoins," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Cham, Switzerland: Springer, 2016, pp. 140–155.
[15] T. Güneysu, V. Lyubashevsky, and T. Pöppelmann, "Practical lattice-based cryptography: A signature scheme for embedded systems," in *Proc. Int. Workshop Cryptograph. Hardw. Embedded Syst.* Berlin, Germany: Springer, 2012, pp. 530–547.
[16] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proc. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1986, pp. 186–194.
[17] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2012, pp. 738–755.
[18] M. Bellare and P. Rogaway, "The exact security of digital signatures-How to sign with RSA and Rabin," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1996, pp. 399–416.
[19] T. Okamoto, "A digital multisignature scheme using bijective public-key cryptosystems," *ACM Trans. Comput. Syst.*, vol. 6, no. 4, pp. 432–441, 1988.

[20] S. Park, S. Park, K. Kim, and D. Won, "Two efficient RSA multisignature schemes," in *Proc. Int. Conf. Inf. Commun. Secur.* Berlin, Germany: Springer, 1997, pp. 217–222.
[21] K. Ohta and T. Okamoto, "A digital multisignature scheme based on the Fiat–Shamir scheme," in *Proc. Int. Conf. Theory Appl. Cryptol.* Berlin, Germany: Springer, 1991, pp. 139–148.
[22] L. Harn, "RSA blocking and multisignature schemes with no bit expansion," *Electron. Lett.*, vol. 26, no. 18, pp. 1490–1491, Aug. 1990.
[23] L. Harn, "Group-oriented (t, n) threshold digital signature scheme and digital multisignature," *IEE Proc.—Comput. Digit. Techn.*, vol. 141, no. 5, pp. 307–313, Sep. 1994.
[24] C.-M. Li, T. Hwang, and N.-Y. Lee, "Threshold-multisignature schemes where suspected forgery implies traceability of adversarial shareholders," in *Proc. Workshop Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1994, pp. 194–204.
[25] M. Michels and P. Horster, "On the risk of disruption in several multiparty signature schemes," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 1996, pp. 334–345.
[26] K. Ohta and T. Okamoto, "Multi-signature schemes secure against active insider attacks," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 82, no. 1, pp. 21–31, 1999.
[27] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. 1st ACM Conf. Comput. Commun. Secur.*, 1993, pp. 62–73.
[28] A. Bagherzandi and S. Jarecki, "Multisignatures using proofs of secret key possession, as secure as the Diffie–Hellman problem," in *Proc. Int. Conf. Secur. Cryptogr. Netw.* Berlin, Germany: Springer, 2008, pp. 218–235.
[29] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities 'honest or bust' with decentralized witness cosigning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 526–545.
[30] F. Y. Kong, L. H. Diao, J. Yu, Y. L. Jiang, and D. S. Zhou, "Lattice-based multi-signature schemes," *Appl. Mech. Mater.*, vol. 411, pp. 3–6, Dec. 2013.
[31] R. Choi and K. Kim, "Lattice-based multi-signature with linear homomorphism," in *Proc. Symp. Cryptogr. Inf. Secur. (SCIS)*. Tokyo, Japan: The Institute of Electronics, Information and Communication Engineers, 2016.
[32] T. Güneysu, T. Oder, T. Pöppelmann, and P. Schwabe, "Software speed records for lattice-based signatures," in *Proc. Int. Workshop Post-Quantum Cryptogr.* Berlin, Germany: Springer, 2013, pp. 67–82.
[33] D. Micciancio, "Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, Nov. 2002, pp. 356–365.
[34] N. Gama and P. Q. Nguyen, "Predicting lattice reduction," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2008, pp. 31–51.
[35] Y. Chen and P. Q. Nguyen, "BKZ 2.0: Better lattice security estimates," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Berlin, Germany: Springer, 2011, pp. 1–20.

CHANGSHE MA received the Ph.D. degree in computer science from Shanghai Jiaotong University, in 2002. He is currently a Professor with the School of Computer Science, South China Normal University. His current research interests include lattice-based cryptography, privacy of data outsourcing, and the IoT security.



MEI JIANG is currently pursuing the master's degree with the School of Computer Science, South China of Normal University.