Round Optimal Secure Multisignature Schemes from Lattice with Public Key Aggregation and Signature Compression

Meenakshi Kansal(✉) and Ratna Dutta

Indian Institute of Technology Kharagpur, Kharagpur, India
{kansal,ratna}@maths.iitkgp.ernet.in

Abstract. This paper presents the *first* construction for an efficient multisignature (MS) in the *lattice* setting, achieving *signature compression* and *public key aggregation* simultaneously with *single round* signature generation. The multisignature size in our construction is the same as that of a single signature. The verification of a multisignature can be performed with the aggregated public key and the verifier gets convinced that the message has been signed by all the signers. More positively, our aggregated public key size is also the same as that of a single signer.

Additionally, we extend our multisignature to an accountable subgroup multisignature (ASM) that permits any subset of potential signers to sign a common message with the property that the signature reveals the identities of the signers to any verifier. Our ASM scheme enjoys the same efficiency as that of our MS scheme without incurring any loss in the security reduction. We design our schemes in the plain public key model where there is no need to verify individual public keys. Our constructions are built in the standard lattice and are proven to be secure under the hardness of the short integer solution (SIS) problem in the random oracle model.

Keywords: Multisignature · Accountable subgroup multisignature · Public key aggregation · Lattice · Short integer solution

1 Introduction

Multisignature. In today's digital world, reducing bandwidth is a desirable and challenging task, especially for low energy devices. For instance, sensors and cell phones have restricted battery life. Multisignature is a powerful cryptographic primitive that helps to reduce the bandwidth taken by N signatures from $\mathcal{O}(N)$ to $\mathcal{O}(1)$. A multisignature scheme provides a group of signers the ability to sign collaboratively a common message in such a way that the size of the multisignature remains the same as that of a single signature and the verifier gets convinced that the message has been signed by all the signers. Multisignature becomes more efficient when the public keys can be aggregated to have size

A. Nitaj and A. Youssef (Eds.): AFRICACRYPT 2020, LNCS 12174, pp. 281–300, 2020.
https://doi.org/10.1007/978-3-030-51938-4_14

asymptotically equivalent to that of the public key of an individual signer and can be verified with the aggregated public key.

Accountable Subgroup Multisignature. Accountable subgroup multisignature, introduced by Micali et al. [17], enables a subset S of a set of potential signers G to jointly produce a multisignature on a given message such that it satisfies *flexibility* and *accountability*. Flexibility means that any subset S of G can sign the document and the verification is then upto the verifier whether the subset S is sufficient to approve the document (message) which is signed jointly by the signers in S. For instance, consider a case as taken in [17], when a company X signs a contract of a company Y. Suppose a subset S of X containing chief operating officer, chief financial officer and chief marketing officer sign the contract and sends the signature to Y. If Y prefers to have the signature of the chief executive officer then Y may reject the signature. Accountability refers to the fact that the set S is known to the verifier.

Application of Multisignature and Accountable Subgroup Multisignature. Multisignatures find applications in areas where storage and bandwidth costs are subject to minimization. Recently, multisignature has gained attention due to the popularity of the distributed applications that supports decentralize trust such as blockchain. Blockchain is a promising technology in the new financial era where digital currency like Bitcoin is the central currency with no intermediaries trusted parties such as bank to process transactions. Multisignature can reduce the size of blockchains [16]. In blockchain, a number of users agree (sign) on a specific message and put the signature to a block. It is desirable to aggregate these signatures into a single signature to reduce the size of the block. Furthermore, since all the public keys need to be written to the blockchain, it is also required to aggregate all the public keys into a single public key such that the aggregated public key has the same size as that of a single public key.

In Bitcoin, multisig is the hash of l public keys and a number k with $1 \leq k \leq l$. Multisignature can reduce the multisig Bitcoin address. The multisig in real life offers a feature that participation of all the l signers is not required to spend funds from the multisig address, but a sufficient number k of participation is sufficient. Accountable subgroup multisignature is a solution that allows a subset S of k signers take part in the signature generation instead of all l signers where $\binom{l}{k}$ is large [4]. The subset S may be decided by the verifier from the flexibility property of the accountable subgroup multisignature [17].

Our Contribution. As pointed by Micali et al. [17], many proposed multisignature schemes are vulnerable to rogue key attacks (for instance Harn [10], Li et al. [14]) or their security requires trusted generation of each key (for instance Ohta et al. [19], Ohta et al. [20]). They constructed the first multisignature scheme in [17] without trusted key generation. However, it requires an interactive initialization session among all the signers where each signer proves to the other signers that it possesses the secret key for the given public key. This model does not support dynamic setting and is not suited for large groups. Later, Boldyreva [3] introduced the concept of *knowledge of secret key* (KOSK)

to overcome the interactive initialization round in the key registration process. The KOSK assumption utilizes *non-interactive zero knowledge proof of knowledge* (ZKPoK) involving heavy computation. Consequently, it is highly desirable to construct multisignature scheme in the plain public key model where the special registration of public key is not required.

Table 1. Comparative summary of multisignature resistant to rogue key attack and secure in the ROM

MS	Communication		R_s	Storage		Computation		Security assumption	Model
	$\|\mathsf{apk}\|$	$\|\mathsf{msig}\|$		$\|\mathsf{pk}\|$	$\|\mathsf{sk}\|$	Sign	Verify		
[4]	$\|\mathbb{G}_2\|$	$\|\mathbb{G}_1\|$	1	$\|\mathbb{G}_2\|$	$\|\mathbb{Z}_q\|$	1E	2P	co-DH	PPK
[5]	$\|\mathbb{G}\|$	$2\|\mathbb{G}\| + 3\|\mathbb{Z}_q\|$	2	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\|\mathbb{Z}_q\|$	5E	6E	DL	PoP
[6]	$\|\mathbb{G}\|$	$2\|\mathbb{G}\|$	1	$\|\mathbb{G}\| + 2\|\mathbb{Z}_q\|$	$\mathcal{O}((\log T)^2)$	4E	3P + 1E	l-wBDHI$^*_3$	PoP
[16]	$\|\mathbb{G}\|$	$\|\mathbb{G}\| + \|\mathbb{Z}_q\|$	3	$\|\mathbb{G}\|$	$\|\mathbb{Z}_q\|$	2E	1E	DL	PPK
[7]	–	$\mathcal{O}(n)$	3	$\mathcal{O}(n)$	$\mathcal{O}(n)$	2PM	$(N+1)$PM	Ring-SIS	PPK
Ours	$\mathcal{O}(n^2)$	$\tilde{\mathcal{O}}(n^2)$	1	$\tilde{\mathcal{O}}(n^2)$	$\tilde{\mathcal{O}}(n^2)$	2MM	2MM	SIS	PPK

Table 2. Comparative summary of accountable subgroup multisignature resistant to rogue key attack and secure in the ROM

ASM	Communication		Rounds		Storage			Computation		Security assumption	Model
	$\|\mathsf{apk}\|$	$\|\mathsf{msig}\|$	R_s	R_g	$\|\mathsf{pk}\|$	$\|\mathsf{sk}\|$	$\|\mathsf{mk}\|$	Sign	Verify		
[4]	$\|\mathbb{G}_2\|$	$\|\mathbb{G}_1\| + \|\mathbb{G}_2\|$	1	1	$\|\mathbb{G}_2\|$	$\|\mathbb{Z}_q\|$	$\|\mathbb{Z}_q\|$	1E	3P	ψ-co-DH	PPK
Ours	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	1	1	$\tilde{\mathcal{O}}(n^2)$	$\tilde{\mathcal{O}}(n^2)$	$\mathcal{O}(n^2)$	1MM	$(2+L)$MM	SIS	PPK

$|\mathsf{apk}|$: size of the aggregated public key, $|\mathsf{msig}|$: size of the compressed signature, $|\mathsf{pk}|$: size of a public key, $|\mathsf{sk}|$: size of a secret key, $|\mathsf{mk}|$: size of group membership key, co-DH: computational Diffie-Hellman, DL: discrete logarithm, l-wBDHI$^*_3$: weak bilinear Diffie-Hellman inversion problem for type-3 pairings, SIS: short integer solution, $\mathbb{G}$, $\mathbb{G}_1$, $\mathbb{G}_2$ are groups of prime order q, $|\mathbb{G}|$: bit size of an element of the group $\mathbb{G}$, T: max number of time periods in forward secrecy, λ: security parameter, $n = \mathcal{O}(\lambda)$, R_s: number of rounds in the signature generation algorithm, PPK: plain public key model, PoP: proof of possession, E: number of exponentiations, P: number of pairings, R_g: number of rounds in the group membership key generation algorithm, N: number of signers, L: size of the subgroup, Model: model to prevent rogue key attack, PM: number of polynomial multiplications, MM: number of matrix multiplications.

This paper constructs the *first* lattice based multisignature scheme supporting public key aggregation in the *plain public key model*. Specifically, we design a multisignature scheme MS and an accountable subgroup multisignature scheme ASM that exhibit signature compression as well as public key aggregation. The verifier only requires an aggregated public key instead of all the public keys to verify a multisignature. Each signer in MS takes part in the multisignature generation and uses public keys of all the participating signers. On the other hand, each signer in our ASM uses aggregated public key along with a group membership key to issue a multisignature. We require only a single round interactive protocol among all the participating signers in a group G to generate a group membership key which can be used to issue an accountable subgroup multisignature for any subset of signers $S \subseteq G$. Both our constructions achieve simulation

based security in the plain public key model against adversaries making bounded number of queries to signatures and hashes. The security of our MS and ASM is derived under the hardness of *short integer solution* (SIS) problem following the security model of Boneh et al. [4].

As shown in Table 1, 2, our MS and ASM schemes are computationally efficient as we have used matrix addition and multiplication. These are linear operations and are very efficient compared to exponentiations and pairings used in [4–6,16]. Our construction enjoys the same round complexity as in the work of Boneh et al. [4]. Similar to the existing works, the multisignature size in our designs are independent of the number of signers involved. Since, our designs are based on lattice, the storage and communication overheads are more (see Table 1, 2) compared to the pairing based multisignature schemes [4–6,16].

The only lattice based multisignature scheme is by Bansarkhani et al. [7] which compresses signature but does not support public key aggregation. It is based on the signature scheme of Guneysu et al. [8,9]. The verifier requires public keys of all the signers. The scheme uses ideal lattice and chooses secret keys from polynomial rings where coefficients are bounded. The scheme is secure under the hardness of ring-SIS problem. It involves three rounds of communication between a signer and cosigner to generate a multisignature. In contrast, our scheme requires only one round of communication between a signer and the designated signer, is built on standard lattice, the verifier requires only an aggregated public key instead of public keys of all the signers and is proven to be secure under the hardness of SIS problem.

Overview of Our Technique. In our MS construction, a trusted third party generates the public parameter set $\mathcal{Y}$ that contains a public matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ along with hash functions $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_q^{m\times n}$, $H_1 : \{0,1\}^* \rightarrow D_{\mathbb{Z}_q,\sigma}^{m\times n}$ and $H_2 : \{0,1\}^* \rightarrow D_{\mathbb{Z}_q,\sigma}^{n\times n}$ modeled as random oracles in the security proof. Here $D_{\mathbb{Z}_q,\sigma}^{k\times l} = \{\mathbf{M} \in \mathbb{Z}_q^{k\times l} : ||\mathbf{M}|| \leq \sigma\sqrt{k}\}$. Each user generates its own public-secret key pair $(\mathsf{pk}, \mathsf{sk})$. The signer i chooses a short matrix $\mathbf{V}_i \in \mathbb{Z}_q^{m\times m}$ with $||\mathbf{V}_i|| \leq \sigma\sqrt{m}$ as its secret key sk_i and sets its own public key as $\mathsf{pk}_i = \mathbf{Y}_i = \mathbf{A} \cdot \mathbf{V}_i \in \mathbb{Z}_q^{n\times m}$ where σ is specified in the public parameter set $\mathcal{Y}$. Note that finding $\mathsf{sk}_i = \mathbf{V}_i$ from $\mathsf{pk}_i = \mathbf{Y}_i$ is the SIS problem. As each signer generates its own public-secret key pair, the adversary is allowed to generate public and secret keys of users in the security game except for the challenged signer i^*. The adversary is given access to the signing oracle corresponding to the signer $i^* \in G$. Let G be a group of signers involved in generating a multisignature on a message M and $\mathcal{PK}$ is the set of public key of the signers in G who have participated in this multisignature generation. Each signer $i \in G$ uses its secret key sk_i together with the public keys of all the signers in G to generate a signature $\mathbf{T}_{i,M} = H_0(M, \mathcal{PK}) + \mathsf{sk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_2(M)$ on M and sends $\mathbf{T}_{i,M}$ to the designated signer. The designated signer aggregates all the received signatures $\mathbf{T}_{i,M}$ into a multisignature $\mathbf{T}_M = \sum_{i\in G} \mathbf{T}_{i,M}$ and outputs $\mathsf{msig}_{\mathcal{PK},M} = (\mathbf{T}_M, \mathsf{pkag}_{\mathcal{PK}}, G, M)$. Anyone can aggregate the public keys in $\mathcal{PK}$ into an aggregated public key

$\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in G} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \in \mathbb{Z}_q^{n \times n}$. A verifier verifies a multisignature $\mathsf{msig}_{\mathcal{PK},M} = (\mathbf{T}_M, \mathsf{pkag}_{\mathcal{PK}}, G, M)$ using the aggregated public key $\mathsf{pkag}_{\mathcal{PK}}$. It outputs 1 if $\mathbf{A} \cdot \mathbf{T}_M = \mathbf{A} \cdot |G| \cdot H_0(M, \mathcal{PK}) + \mathsf{pkag}_{\mathcal{PK}} \cdot H_2(M)$ and $||\mathbf{T}_M|| \leq |G| \cdot (||H_0(M, \mathcal{PK})|| + \sigma^3 m\sqrt{n})$. Otherwise, it outputs 0.

While the adversary makes a signature generation query, the simulator simulates the signature for the challenged signer i^*. The ranges of H_1 and H_2 have been specified with bounds to preserve the security. While simulating the signature $\mathbf{T}_{i^*,M}$ for i^* without knowing its secret key, the simulator calls for $H_1(\mathsf{pk}_{i^*}, \mathcal{PK})$ query, $H_2(M)$ query, chooses $\mathbf{T}_{i^*,M} \in \mathbb{Z}_q^{n \times m}$ and finds the value of $H_0(M, \mathcal{PK})$ satisfying the equation $\mathbf{A} \cdot \mathbf{T}_{i^*,M} = \mathbf{A} \cdot H_0(M, \mathcal{PK}) + \mathsf{pk}_{i^*} \cdot H_1(\mathsf{pk}_{i^*}, \mathcal{PK}) \cdot H_2(M)$. As there is no bound restriction on the range of H_0, one can find $H_0(M, \mathcal{PK})$ using the Gauss elimination method or any other linear algebra method. Using the generalized forking lemma, we finally show that if the adversary is able to forge a multisignature, then the simulator finds $\mathbf{V}^* \in \mathbb{Z}_q^{m \times m}$ satisfying $\mathbf{A} \cdot \mathbf{V}^* = 0 \bmod q$ with $||\mathbf{V}^*|| \leq \sigma\sqrt{m}$. Thus the simulator solves an instance of SIS problem and we have the following theorem.

Theorem 1 *(Informal). The scheme* MS *is unforgeable in the random oracle model if the* SIS *problem is hard.*

The public parameter set $\mathcal{Y}$ in our ASM scheme uses a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and hash functions $H_0 : \{0,1\}^* \to \mathbb{Z}_q^{m \times n}$, $H_1 : \{0,1\}^* \to D_{\mathbb{Z}_q,\sigma}^{m \times n}$, $H_2 : \{0,1\}^* \to \mathbb{Z}_q^{n \times n}$ and $H_3 : \{0,1\}^* \to D_{\mathbb{Z}_q,\sigma}^{n \times n}$ where H_0, H_1, H_2 are modeled as random oracles in the security proof. The key generation and the key aggregation are performed as in our MS scheme. All the members in a group of signers G take part in the group membership key protocol. Let $\mathcal{PK}$ be the set of public keys of the signers in G. Each member $i \in G$ uses its secret key sk_i together with the public keys of other signers in G, computes $\mathbf{M}_{j,i} = H_2(\mathsf{pkag}_{\mathcal{PK}}, j) + \mathsf{sk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_3(j)$ for all $j \in G$ and sends $\mathbf{M}_{j,i}$ to all $j \in G$ parallely where $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in G} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \in \mathbb{Z}_q^{n \times n}$. After receiving $\mathbf{M}_{i,j}$ from all signers $j \in G$, the i-th signer generates its group membership key $\mathsf{mk}_{i,\mathcal{PK}} = \sum_{j \in G} \mathbf{M}_{i,j}$. Let S be a subset of G and L be the set of all public keys in S. Each signer $i \in S$ using its secret key sk_i together with the public keys of all the signers in G computes $\mathbf{T}_{i,M} = \mathsf{sk}_i \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + \mathsf{mk}_{i,\mathcal{PK}}$ and sends $\mathbf{T}_{i,M}$ to the designated signer. The designated signer aggregates all the received signatures $\mathbf{T}_{i,M}$ into a multisignature $\mathbf{T}_M = \sum_{i \in G} \mathbf{T}_{i,M}$ and outputs $\mathsf{accmsig}_{L,M} = (\mathbf{T}_M, \mathsf{spkag}_L, \mathsf{pkag}_{\mathcal{PK}}, G, M, S)$ where $\mathsf{spkag}_L = \sum_{i \in S} \mathsf{pk}_i \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M)$ is the aggregated subgroup public key. The verifier using the aggregated public key $\mathsf{pkag}_{\mathcal{PK}}$ and aggregated subgroup public key spkag_L, outputs 1 if $\mathbf{A} \cdot \mathbf{T}_M = \mathsf{spkag}_L + |G| \cdot \sum_{i \in S} \mathbf{A} \cdot H_2(\mathsf{pkag}_{\mathcal{PK}}, i) + \mathsf{pkag}_{\mathcal{PK}} \cdot \sum_{i \in S} H_3(i)$ and $||\mathbf{T}_M|| \leq |S| \cdot \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |G| \cdot \max_{i \in S} ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |S| \cdot |G| \cdot \sigma^3 m\sqrt{n}$. Otherwise, it outputs 0.

The adversary is given access to the group membership key query for the challenged signer $i^* \in G$. The simulator and the adversary take part in the group membership key generation protocol where the simulator simulates the group membership key for the challenged signer i^*. The ranges of H_1 and H_3 have been specified with bounds to preserve the security. While simulating $\mathbf{M}_{j,i^*}$ for i^* without knowing its secret key, the simulator chooses $\mathbf{M}_{j,i^*}$ such that $||\mathbf{M}_{j,i^*}|| \leq \sigma\sqrt{m}$ for each $j \in G$, queries to $H_1(\mathsf{pk}_j, \mathcal{PK})$ oracle for each $j \in G$, finds $H_2(\mathsf{pkag}_{\mathcal{PK}}, j)$ satisfying $\mathbf{A}\cdot\mathbf{M}_{j,i^*} = \mathbf{A}\cdot H_2(\mathsf{pkag}_{\mathcal{PK}}, j) + \mathbf{Y}_{i^*}\cdot H_1(\mathsf{pk}_{i^*}, \mathcal{PK})\cdot H_3(j)$. There is no bound on the range of H_2 and thus can be found using any linear algebra method. The adversary is also given signature oracle to query for the challenged signer $i^* \in G$. The simulator models the random oracle H_0 to simulate the signature for the challenged signer i^*. Finally, we apply generalized forking lemma to show that forging an accountable subgroup multisignature yields a solution to an SIS instance and proved the following theorem.

Theorem 2 *(Informal). The scheme* ASM *is unforgeable in the random oracle model if the* SIS *problem is hard.*

Related Work. The first construction of multisignature was presented by Itakura and Nakamura [12]. Multisignature schemes require homomorphic properties of arithmetic operations involved in standard signatures. Unfortunately, the same homomorphic properties that permits aggregation of signatures into multisignatures can enable a rogue key attack on such schemes. Infact, the multisignature schemes in early literature [10,11,13,14,18–20] were broken mostly by mounting a rogue-key-attack. In this attack, a cheating group member sets its public key as a function of the public key of an honest signer of the group enabling it to forge multisignature easily. Many solutions were proposed to prevent rogue key attack like *key registration model*, *knowledge of secret key* (KOSK) assumption, *proof of possession* (PoP) assumption etc. These approaches have higher complexity and are unrealistic assumptions on the public key infrastructure (PKI). The key registration model is parameterized by the key registration and the adversarial behaviour is restricted by the security game based on the successful or unsuccessful registration. In this model, the client registers with the certifying authority through the key registration protocol and the adversary can access the key registration oracle. Okamoto [19] and Micali [17] developed proper security framework for multisignature. They also built constructions for multisignatures and analyzed the security in the respective proposed models. In contrast to [19], the security model of [17] addresses attacks in the key generation phase. To prevent rogue key attack, Micali et al. [17] allows all the signers to engage in an interactive protocol to generate public and secret keys. This scheme is not dynamic in the sense that all the signers require to be fixed at the setup phase.

The constructions in Boldyreva et al. [3], Lu et al. [15], on the other hand, use KOSK assumption to achieve security against rogue key attack. When the adversary provides a public key for a signer, it is required to provide a matching secret key. In KOSK setting, a user has to prove the knowledge of secret key to

the certifying authority during public key registration. However, PKI has yet not realized the KOSK assumption. Bellare and Neven [2] pointed out that a scheme is secure under the KOSK assumption face the upgradation of existing PKI as it would require client and certifying authority to possess *zero knowledge proof of knowledge* (ZKPoK) with *extraction guarantees* in *fully concurrent settings*. The utilization of non interactive zero knowledge proof of knowledge requires heavy computation.

To avoid the KOSK assumption for preventing rogue key attack, Ristenpart and Yilek [21] modified the multisignature scheme of Boldyreva et al. [3] and proved it is secure under the PoP assumption. Unlike KOSK, the PoP setting does not ask to prove the knowledge of secret key, but it attests that a client has the access to the public and secret key pair. One of the simplest ways to achieve PoP in signature schemes is by sending the signature on the message requested by the certifying authority.

Bellare and Neven [2] had overcome the KOSK assumption and proposed a multisignature scheme in the plain public key model. In plain public key model, the users do not need to prove the knowledge or possession of their secret keys. The multisignature scheme of Micali et al. [17] is the first scheme that is secure in the plain public key model. Downfall of this scheme is that the set of the potential signers becomes static once the key setup phase is done. On the other hand, the multisignature of Bellare and Neven [2] does not require a dedicated key setup algorithm and is secure in the plain public key model. However, this scheme requires several rounds of communication between the signers.

Recently, many multisignature schemes [4–6,16] have been proposed. The scheme by Boneh et al. [4] is the first compact multisignature scheme secure under the computational co-Diffie-Hellman problem with both signature compression and public key aggregation. Further, they have constructed the first short accountable subgroup multisignature scheme under the hardness of computational Ψ-co-Diffie-Hellman problem in the *random oracle model* (ROM). Drijvers et al. [6] proposed a construction for pairing based multisignature secure under a variant of the bilinear Diffie-Hellman inversion problem in the ROM. The work in Drijvers et al. [5] pointed out serious issues in the two round multisignature schemes without pairings and presented a variant of Bagherzandi et al. [1] scheme secure under the discrete logarithm assumption in the ROM. Maxwell [16] gave the first multisignature scheme secure in the palin public key model. It is based on Schnorr signature and is secure under the hardness of discrete logarithm problem. All the aforementioned schemes are secure only on the classical machine and are not quantum computer resistant. The construction of Bansarkhani et al. [7] is the only multisignature scheme that is secure under the hardness of computational problems from lattice that are not succeptiable to quantum attacks. The scheme is secure in the ROM under the ring-SIS problem. However, the scheme is interactive involving three rounds during the signature generation and does not support public key aggregation.

Drijvers et al. [6] proposed a multisignature scheme with forward secrecy to address adaptive corruption. The adversary can corrupt committee members

after they have certified (signed) a message and use their signing keys to certify (sign) a different message. Forward secure multisignatures prevent this attack and enables signers to update their secret keys over time without changing their respective verification keys.

2 Preliminaries

Notation. We provide below some of the notation that will be used: $\mathbf{a} \in \Delta^n$ means that $\mathbf{a}$ is a column vector of dimension $n \times 1$ with elements from the set Δ. For a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \Delta^n$, $||\mathbf{x}|| = \sqrt{x_1^2 + \ldots + x_n^2}$ denotes the Euclidean norm. Let $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ be a matrix with n columns in Δ^m then $||\mathbf{X}|| = \max_{1 \leq k \leq n} ||\mathbf{x}_k||$. We say that a function f is negligible in λ if $f = \lambda^{-\omega(1)}$.

Definition 1 (Lattice). *A full rank matrix $\boldsymbol{B} \in \mathbb{Z}_q^{n \times m}$ is a basis of an m dimensional lattice Λ if $\Lambda = \{\boldsymbol{y} \in \mathbb{Z}^m \mid \exists\ \boldsymbol{x} \in \mathbb{Z}^m, \boldsymbol{y} = \boldsymbol{B} \cdot \boldsymbol{x}\}$. For any integer $q \geq 2$, a matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$ and a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$ define $\Lambda_q^{\perp}(\boldsymbol{A}) = \{\boldsymbol{v} \in \mathbb{Z}_q^m | \boldsymbol{A} \cdot \boldsymbol{v} = \boldsymbol{0} \bmod q\}$ and $\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}) = \{\boldsymbol{v} \in \mathbb{Z}_q^m | \boldsymbol{A} \cdot \boldsymbol{v} = \boldsymbol{u} \bmod q\}$.*

Definition 2 (Discrete Gaussian Distribution). *The discrete Gaussian distribution over a lattice Λ with center $\boldsymbol{c} \in \mathbb{R}^m$ and parameter σ is $D_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) = \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$ for all $\boldsymbol{y} \in \Lambda$. Here $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y}) = exp(-\pi \frac{||\boldsymbol{y}-\boldsymbol{c}||^2}{\sigma^2})$ and $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{y} \in \Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})$. If $c = 0$, we simply denote it by $D_{\Lambda,\sigma}$.*

Definition 3 (Short Integer Solution (SIS) Problem). *Given a uniformly random matrix $\boldsymbol{A} \in \mathbb{Z}_q^{n \times m}$ and a real number β, the SIS problem is to find a vector $\boldsymbol{v} \in \mathbb{Z}_q^m$ such that $\boldsymbol{A} \cdot \boldsymbol{v} = \boldsymbol{0} \bmod q$ and $||\boldsymbol{v}|| \leq \beta$.*

Generalized Forking Lemma [5]**.** Let us consider an algorithm $\mathcal{A}$ that takes $\mathsf{in}_{\mathcal{A}}$ as input and interacts with a random oracle $\mathcal{O}$. Let $\Omega = \{r | r = (\hat{r}, h_1, h_2, \ldots, h_{q_H})\}$ be the randomness space and let $r|_j = (h_1, h_2, \ldots, h_{j-1})$. Here $\hat{r}$ is the random tape of $\mathcal{A}$, q_H is the maximum allowable number of random oracle queries and h_j is the response to j-th random oracle query. The execution of $\mathcal{A}$ is termed success if it outputs $(I, \{\mathsf{out}_i\}_{i \in I})$ where I is a non empty subset of $\{1, 2, \ldots, q_H\}$. The input $\mathsf{in}_{\mathcal{A}}$ is generated by the input generator IG. The working of the algorithm $\mathcal{FL}_{\mathcal{A}}$ is explained below in Algorithm 1. We say that $\mathcal{FL}_{\mathcal{A}}$ succeeds if it does not output *fail*.

Lemma 1 (Generalized Forking Lemma [5]). *Let IG be a randomized input generation algorithm and $\mathcal{A}$ be a randomized algorithm running in time τ with access to a random oracle $\mathcal{O}$ such that $\mathcal{A}$ succeeds with probability ϵ. If $q > \frac{8nq_H}{\epsilon}$, then $\mathcal{FL}_{\mathcal{A}}(\mathsf{in}_{\mathcal{A}})$ runs in time atmost $\tau \cdot \frac{8n^2 q_H}{\epsilon} \cdot ln(\frac{8n}{\epsilon})$ and succeeds with probability atleast $\frac{\epsilon}{8}$, where the probability is over the choice of $\mathsf{in}_{\mathcal{A}} \leftarrow \mathsf{IG}$ and over the coins of $\mathcal{FL}_{\mathcal{A}}$.*

Algorithm 1. Generalized Forking Algorithm $\mathcal{FL}_{\mathcal{A}}(\mathsf{in}_{\mathcal{A}})$

$(I, \{\mathsf{out}_i\}_{i\in I}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{in}_{\mathcal{A}}, r)$ where $r = (\hat{r}h_1, h_2, \ldots, h_{q_H})$;
if $I = \emptyset$ **then**
  output *fail*;
else
  $\mathsf{Aux} \leftarrow aux$;
  Let $I = \{i_1, i_2, \ldots, i_n\}$ such that $i_1 \leq i_2 \leq \ldots \leq i_n$;
  for $t = 1$ to n **do**
    $success_t \leftarrow 0$; $k_t \leftarrow 0$; $k_{max} \leftarrow \frac{8nq_H}{\epsilon} \cdot ln(\frac{8n}{\epsilon})$;
    repeat
      $r'' \in \Omega$ such that $r''|_{i_t} = r|_{i_t}$;
      Let $r'' = (\hat{r}, h_1, h_2, \ldots, h_{i_t-1}, h''_{i_t}, \ldots, h''_{q_H})$; (Note that $h''_j \neq h_j$ for $j = i_t$ to q_H)
      $(I'', \{\mathsf{out}''_i\}_{i\in I''}, aux) \leftarrow \mathcal{A}^{\mathcal{O}}(\mathsf{in}_{\mathcal{A}}, r'')$;
      $\mathsf{Aux} \leftarrow Aux \cup aux$;
    if ($h''_{i_t} \neq h_{i_t}$ and $I'' \neq \emptyset$ and $i_t \in I''$) **then**
      $\mathbf{out}'_{i_t} \leftarrow \mathsf{out}''_{i_t}$; $success_t \leftarrow 1$;
    end if
    $k_t \leftarrow k_t + 1$;
    until $success_t = 1$ or $k_t > k_{max}$
  end for
  if ($success_t = 1$ for all $t = 1, 2, \ldots, n$) **then**
    output $(I, \{\mathsf{out}_i\}_{i\in I}, \{\mathsf{out}'_i\}_{i\in I}, \mathsf{Aux})$
  else
    output *fail*
  end if
end if

2.1 Multisignature - Syntax, Definition and Security Model

Syntax of Multisignature. The multisignature scheme allows a group of signers with public keys $\{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$ to issue a multisignature 'msig' on a message M in such a way that the verifier agrees that all the N signers have signed the message M. Let there be a designated signer who combines all the signatures of the signers into a single multisignature. The designated signer may be one of the signers or an external party.

At high level, we define a multisignature scheme $\mathsf{MS} = \{\mathsf{pg}, \mathsf{kg}, \mathsf{kag}, \mathsf{sg}, \mathsf{vrf}\}$ as consisting of *parameter generation algorithm* pg, *key generation algorithm* kg and *key aggregation algorithm* kag together with an interactive *signature generation* protocol sg and a deterministic *verification* algorithm vrf. A trusted third party, called the *key generation center* (KGC), generates the public parameter set $\mathcal{Y} \leftarrow \mathsf{MS.pg}$. A user generates its public-secret key pair (pk, sk)$\leftarrow\mathsf{MS.kg}$. The public keys are made public while the secret keys are kept secret to the users. The signer i uses secret key sk_i to generate signature $\mathbf{T}_{i,M}$ on a message M and sends $\mathbf{T}_{i,M}$ to the "*designated signer*". The designated signer aggregates all the received signatures $\mathbf{T}_{i,M}$ on the message M into a multisignature $\mathsf{msig}_{\mathcal{PK},M}$. Here $\mathcal{PK}$ is the set of public key of the signers participated in this multisignature generation. The key aggregation algorithm $\mathsf{MS.kag}$ can be run by anyone to aggregate the public keys in a set $\mathcal{PK}$ into a single public key $\mathsf{pkag}_{\mathcal{PK}}$. The verifier using the aggregated public key $\mathsf{pkag}_{\mathcal{PK}}$, runs the algorithm $\mathsf{MS.vrf}$ and returns 0, indicating the multisignature $\mathsf{msig}_{\mathcal{PK},M}$ is not properly generated or 1, assuring that $\mathsf{msig}_{\mathcal{PK},M}$ is correct. More concretely, we have the following.

- $\mathsf{MS.pg}(1^\lambda) \rightarrow \mathcal{Y}$. It is a probabilistic polynomial time (PPT) algorithm run on a security parameter λ and outputs the public parameter set $\mathcal{Y}$.
- $\mathsf{MS.kg}(\mathcal{Y}, i) \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$. For each user i, this PPT algorithm returns the public and secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ on input the user i and the public parameter set $\mathcal{Y}$. The public key pk_i is made public while the secret key sk_i is kept secret to the user.
- $\mathsf{MS.kag}(\mathcal{Y}, \mathcal{PK}) \rightarrow \mathsf{pkag}_{\mathcal{PK}}$. Let $\mathcal{PK}$ be the set of public keys of signers. This is a deterministic algorithm and it aggregates the public keys in $\mathcal{PK}$ into a single public key $\mathsf{pkag}_{\mathcal{PK}}$. It outputs the aggregated public key $\mathsf{pkag}_{\mathcal{PK}}$ which asymptotically has the same size as a single public key.
- $\mathsf{MS.sg}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}, M) \rightarrow \mathsf{msig}_{\mathcal{PK},M}$. With input the public parameter set $\mathcal{Y}$, the set of public and secret keys $(\mathcal{PK}, \mathcal{SK})$ of the signers and a message M, this single round protocol executes as follows. Let $\mathcal{PK} = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$, $\mathcal{SK} = \{\mathsf{sk}_{i_1}, \mathsf{sk}_{i_2}, \ldots, \mathsf{sk}_{i_l}\}$ and $I_{\mathcal{PK}} = \{i_1, i_2, \ldots, i_l\}$. The signer $i \in I_{\mathcal{PK}}$ uses $\mathcal{PK}$ along with its secret key sk_i to generate a signature $\mathbf{T}_{i,M}$ on M and sends $\mathbf{T}_{i,M}$ to the designated signer. The designated signer aggregates all the signatures $\mathbf{T}_{i,M}, i \in I_{\mathcal{PK}}$ on M into a single multisignature $\mathsf{msig}_{\mathcal{PK},M}$.
- $\mathsf{MS.vrf}(\mathcal{Y}, \mathsf{msig}_{\mathcal{PK},M}) \rightarrow$ (0 or 1). On input the public parameter set $\mathcal{Y}$, a multisignature $\mathsf{msig}_{\mathcal{PK},M}$, this deterministic algorithm returns 1 if $\mathsf{msig}_{\mathcal{PK},M}$ is valid. Otherwise, it returns 0.

Completeness. A multisignature scheme should satisfy completeness. That is, for any $\mathcal{Y} \leftarrow \mathsf{MS.pg}(1^\lambda)$, for any N, if we have $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MS.kg}(\mathcal{Y}, i)$ for $i = 1, 2, \ldots, N$, for any message M and for any set of public keys $\mathcal{PK} = \{\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_N\}$ with corresponding set of secret keys $\mathcal{SK} = \{\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_N\}$, if $\mathsf{msig}_{\mathcal{PK},M} \leftarrow \mathsf{MS.sg}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}, M)$ then $\mathsf{MS.vrf}(\mathcal{Y}, \mathsf{msig}_{\mathcal{PK},M})$ outputs 1.

Security Under Unforgeability. The unforgeability experiment $\mathsf{Exp}^{\mathsf{unforg}}_{\mathcal{F}}(\lambda)$ between a simulator $\mathcal{S}$ and a forger $\mathcal{F}$ is described in Fig. 1 following the model of Boneh et al. [4] that considers the infeasibility to forge multisignature with atleast one honest signer. The forger has given polynomially many access to the signature queries on any message M with any set of public keys $\mathcal{PK}$.

Definition 4. *We say that a multisignature is unforgeable if* $\mathsf{Adv}^{\mathsf{unforg}}_{\mathcal{F}}(\lambda) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{unforg}}_{\mathcal{F}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *for every PPT adversary* $\mathcal{F}$ *in the experiment* $\mathsf{Exp}^{\mathsf{unforg}}_{\mathcal{F}}(\lambda)$ *defined in Fig. 1 where* $\mathsf{negl}(\lambda)$ *is a negligible function in* λ.

3 The MS

Our multisignature $\mathsf{MS} = (\mathsf{pg}, \mathsf{kg}, \mathsf{kag}, \mathsf{sg}, \mathsf{vrf})$ works as follows.

- $\mathsf{MS.pg}(1^\lambda) \rightarrow \mathcal{Y}$. A trusted third party, called key generation center (KGC), generates the system parameters $\mathcal{Y} \leftarrow (n, q, m, \sigma, H_0, H_1, H_2, \mathbf{A})$.
  - choose n of size $\mathcal{O}(\lambda)$, q of size $\mathcal{O}(n^3)$ and $m \geq 2n\lceil \log q \rceil$,
  - pick the standard deviation σ of the discrete Gaussian distribution $D_{\Lambda,\sigma}$ of size $\Omega(\sqrt{n \log q} \log n)$,

1. The simulator $\mathcal{S}$ generates system parameters $\mathcal{Y}$ and a challenge public key pk_{i^*} for user i^*. The simulator $\mathcal{S}$ runs the forger $\mathcal{F}$ on $(\mathcal{Y}, \mathsf{pk}_{i^*})$.
2. The forger $\mathcal{F}$ is allowed to make signature queries to $\mathcal{S}$ on $(M_l, \mathcal{PK}_l)$, $1 \leq l \leq q_s$ where M_l is a message and $\mathcal{PK}_l$ is a set of public keys with $\mathsf{pk}_{i^*} \in \mathcal{PK}_l$ i.e., $\mathcal{F}$ has access to the oracle $O^{(\mathcal{Y},\cdot,\cdot,\cdot)}$ that simulates the honest signer i^* with the public keys in $\mathcal{PK}_l$ and produce a signature $\mathbf{T}_{i^*,M}$ on M.
3. Finally, $\mathcal{F}$ outputs a forgery $\mathsf{msig}^*_{\mathcal{PK},M}$ on a message M for a set of public keys $\mathcal{PK}$.
4. The simulator $\mathcal{S}$ returns 1 if the following conditions hold:
   (a) $\mathsf{MS.vrf}(\mathcal{Y}, \mathsf{msig}^*_{\mathcal{PK},M}) \rightarrow 1$,
   (b) $\mathsf{pk}_{i^*} \in \mathcal{PK}$.
   (c) $M \neq M_l$ for $1 \leq l \leq q_s$.
   Otherwise, $\mathcal{S}$ returns 0.
5. The forger $\mathcal{F}$ wins the game if $\mathcal{S}$ returns 1.

Fig. 1. Unforgeability game $\mathsf{Exp}^{\mathsf{unforg}}_{\mathcal{F}}(\lambda)$

– select a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ over $\mathbb{Z}_q$ and sample cryptographically secure hash functions $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_q^{m \times n}$, $H_1 : \{0,1\}^* \rightarrow D^{m \times n}_{\mathbb{Z}_q,\sigma}$ and $H_2 : \{0,1\}^* \rightarrow D^{n \times n}_{\mathbb{Z}_q,\sigma}$ where $D^{k \times l}_{\mathbb{Z}_q,\sigma} = \{\mathbf{M} \in \mathbb{Z}_q^{k \times l} : ||\mathbf{M}|| \leq \sigma\sqrt{k}\}$.

- $\mathsf{MS.kg}(\mathcal{Y}, i) \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$. The signer i runs this algorithm using $\mathcal{Y}$ to generate its own public and secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ by performing the following steps.
  – choose a short matrix $\mathbf{V}_i \in D^{m \times m}_{\mathbb{Z}_q,\sigma}$ and compute $\mathbf{Y}_i = \mathbf{A} \cdot \mathbf{V}_i \bmod q \in \mathbb{Z}_q^{n \times m}$,
  – set the public key $\mathsf{pk}_i = \mathbf{Y}_i \in \mathbb{Z}_q^{n \times m}$ and secret key $\mathsf{sk}_i = \mathbf{V}_i \in D^{m \times m}_{\mathbb{Z}_q,\sigma}$.
  The public key pk_i is made public and the secret key sk_i is kept secret to the signer i.
- $\mathsf{MS.kag}(\mathcal{Y}, \mathcal{PK}) \rightarrow \mathsf{pkag}_{\mathcal{PK}}$. This deterministic algorithm outputs the aggregated public key $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \in \mathbb{Z}_q^{n \times n}$ by extracting H_1 from $\mathcal{Y}$ where $\mathcal{PK} = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$ and $I_{\mathcal{PK}} = \{i_1, i_2, \ldots, i_l\}$ is the index set of $\mathcal{PK}$.
- $\mathsf{MS.sg}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}, M) \rightarrow \mathsf{msig}_{\mathcal{PK},M}$. It is an interactive protocol among the signers $i \in I_{\mathcal{PK}}$ where $\mathcal{PK} = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$ is the set of public keys of the signers with $\mathsf{pk}_i = \mathbf{Y}_i$, $\mathcal{SK} = \{\mathsf{sk}_{i_1}, \mathsf{sk}_{i_2}, \ldots, \mathsf{sk}_{i_l}\}$ is the corresponding set of secret keys with $\mathsf{sk}_i = \mathbf{V}_i$ and $I_{\mathcal{PK}} = \{i_1, i_2, \ldots, i_l\}$ is the index set of $\mathcal{PK}$. The protocol executes the following steps where $\mathbf{A}, n, m, \sigma, H_0, H_1, H_2$ are extracted from $\mathcal{Y}$.
  – each signer $i \in I_{\mathcal{PK}}$ generates a signature $\mathbf{T}_{i,M}$ on a message $M \in \{0,1\}^*$ using its secret key $\mathsf{sk}_i = \mathbf{V}_i$ as $\mathbf{T}_{i,M} = H_0(M, \mathcal{PK}) + \mathsf{sk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_2(M)$ and sends $\mathbf{T}_{i,M}$ to the designated signer. Note that

$$\begin{aligned} ||\mathbf{T}_{i,M}|| &\leq ||H_0(M, \mathcal{PK})|| + ||\mathsf{sk}_i|| \cdot ||H_1(\mathsf{pk}_i, \mathcal{PK})|| \cdot ||H_2(M)|| \\ &\leq ||H_0(M, \mathcal{PK})|| + \sigma^3 m\sqrt{n} \end{aligned}$$

  as $\mathsf{sk}_i \in D^{m \times m}_{\mathbb{Z}_q,\sigma}$, $H_1(\mathsf{pk}_i, \mathcal{PK}) \in D^{m \times n}_{\mathbb{Z}_q,\sigma}$ and $H_2(M) \in D^{n \times n}_{\mathbb{Z}_q,\sigma}$,

  - the designated signer in turn verifies whether $||\mathbf{T}_{i,M}|| \leq ||H_0(M, \mathcal{PK})|| + \sigma^3 m\sqrt{n}$ and $\mathbf{A} \cdot \mathbf{T}_{i,M} = \mathbf{A} \cdot H_0(M, \mathcal{PK}) + \mathbf{Y}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_2(M)$,
  - if the verification fails, the designated signer does not accept the signature and returns $\perp$. Otherwise, it issues the multisignature $\mathsf{msig}_{\mathcal{PK},M} = \big(\mathbf{T}_M, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, M\big)$ where $\mathbf{T}_M = \sum_{i \in I_{\mathcal{PK}}} \mathbf{T}_{i,M} \bmod q$.

- $\mathsf{MS.vrf}\big(\mathcal{Y}, \mathsf{msig}_{\mathcal{PK},M}\big) \rightarrow$ (0 or 1). On input the multisignature $\mathsf{msig}_{\mathcal{PK},M} = (\mathbf{T}_M, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, M)$, it outputs 1 if $\mathbf{A} \cdot \mathbf{T}_M = \mathbf{A} \cdot |I_{\mathcal{PK}}| \cdot H_0(M, \mathcal{PK}) + \mathsf{pkag}_{\mathcal{PK}} \cdot H_2(M)$, $||\mathbf{T}_M|| \leq |I_{\mathcal{PK}}| \cdot (||H_0(M, \mathcal{PK})|| + \sigma^3 m\sqrt{n})$. Otherwise, it returns 0.

The proof of the following Theorem 3 is immediate from the construction.

Theorem 3. *The scheme* MS *is complete.*

3.1 Security Proof

Theorem 4. *The scheme* MS *is* $(t_{\mathcal{F}}, q_s, q_H, \epsilon_{\mathcal{F}})$*-unforgeable in the random oracle model if* SIS *problem is* $((t_{\mathcal{F}} + t_{q_H} + t_{q_s} + t_{extra}) \cdot 8q_H^2 \cdot \epsilon_{\mathcal{F}} \cdot \log(8q_H/\epsilon_{\mathcal{F}}), \frac{\epsilon_{\mathcal{F}}}{8q_H})$*-hard. In other words, suppose that there exists a forger* $\mathcal{F}$ *running in time* $t_{\mathcal{F}}$ *can break the security under unforgeability of our scheme* MS *with non-negligible advantage* $\epsilon_{\mathcal{F}}$ *making* q_s *signature queries and* q_H *hash queries. Then there exists an algorithm* $\mathcal{S}$ *running in time* $(t_{\mathcal{F}} + t_{q_H} + t_{q_s} + t_{extra}) \cdot 8q_H^2 \cdot \epsilon_{\mathcal{F}} \cdot \log(8q_H/\epsilon_{\mathcal{F}})$*, that for a given* $\mathbf{P} \in \mathbb{Z}_q^{n \times m}$ *finds a nonzero* $\boldsymbol{V} \in \mathbb{Z}_q^{m \times m}$ *satisfying* $||\boldsymbol{V}|| \leq \sigma\sqrt{m}$ *and* $\mathbf{P} \cdot \boldsymbol{V} = \boldsymbol{0} \bmod q$ *with non negligible advantage* $\frac{\epsilon_{\mathcal{F}}}{8q_H}$*. Here* $m \geq 2n\lceil \log q \rceil$*,* σ *is of size* $\Omega(\sqrt{n \log q} \log n)$*,* q *is of size* $\mathcal{O}(n^3)$*,* t_{q_H}*,* t_{q_s} *respectively denote the time taken to answer hash and signature queries and* t_{extra} *is extra time taken by the algorithm* $\mathcal{S}$*.*

Proof. We assume that there exists a forger $\mathcal{F}$ that wins the unforgeability game played with a simulator $\mathcal{S}$ given in Definition 4 with probability $\epsilon_{\mathcal{F}}$.

1. Given an SIS instance $\mathbf{P} \in \mathbb{Z}_q^{n \times m}$ with $m \geq 2n\lceil \log q \rceil$, q is of size $\mathcal{O}(n^3)$, σ is of size $\Omega(\sqrt{n \log q} \log n)$, the simulator $\mathcal{S}$ sets $\mathcal{Y} = (n, q, m, \sigma, H_0, H_1, H_2, \mathbf{A})$ by setting $\mathbf{A} = \mathbf{P}$ and $H_0 : \{0,1\}^* \rightarrow \mathbb{Z}_q^{m \times n}$, $H_1 : \{0,1\}^* \rightarrow D_{\mathbb{Z}_q,\sigma}^{m \times n}$ and $H_2 : \{0,1\}^* \rightarrow D_{\mathbb{Z}_q,\sigma}^{n \times n}$ where $D_{\mathbb{Z}_q,\sigma}^{k \times l} = \{\mathbf{M} \in \mathbb{Z}_q^{k \times l} : ||\mathbf{M}|| \leq \sigma\sqrt{k}\}$. The simulator $\mathcal{S}$ generates a random matrix $\mathsf{pk}_{i^*} = \mathbf{Y}_{i^*}$ and randomness $\rho = \{\xi, \mathcal{C}\}$ where $\mathcal{C} = \{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{q_H}\}$, $\xi \in \mathbb{Z}_q^{m \times n}$ and each $\mathbf{C}_i \in D_{\mathbb{Z}_q,\sigma}^{m \times n}$ for $i = 1, 2, \ldots, q_H$. The simulator $\mathcal{S}$ speculates a random index $k \in \{1, 2, \ldots, q_H\}$. More precisely, $\mathcal{S}$ guesses that $\mathcal{F}$ makes k-th H_2 query on a message that is used by $\mathcal{F}$ to output a valid forgery. It then runs $\mathcal{F}$ on input $\mathsf{pk}_{i^*} \in \mathbb{Z}_q^{n \times m}$, randomness ρ and system parameters $\mathcal{Y} = (n, q, m, \sigma, H_0, H_1, H_2, \mathbf{A})$.
2. The forger $\mathcal{F}$ is allowed to make q_H many hash and q_s many signature queries which are simulated as follows.

H_1 **Query.** The simulator $\mathcal{S}$ maintains a list L_{H_1} containing elements of the form $(x, H_1(x))$. If the tuple $x = (\mathsf{pk}_i, \mathcal{PK}_l)$, $1 \leq l \leq q_H$ is already answered then $\mathcal{S}$ returns from the list L_{H_1}. If it is queried for the first time, $\mathcal{S}$ chooses a random value from the set $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{q_H}\}$ for $H_1(\mathsf{pk}_i, \mathcal{PK})$ for $\mathsf{pk}_{i^*} \in \mathcal{PK}$ if $\mathsf{pk}_{i^*} \in \mathcal{PK}$ and $i = i^*$. Otherwise, it returns random value from $D^{m\times n}_{\mathbb{Z}_q,\sigma}$. Finally, the simulator stores $((\mathsf{pk}_i, \mathcal{PK}), H_1(\mathsf{pk}_i, \mathcal{PK}))$ in the list L_{H_1}.

H_2 **Query.** On receiving the query on a message M_l, $1 \leq l \leq q_H$ if it already queried then the simulator returns from the list L_{H_2}. Otherwise, $\mathcal{S}$ honestly generates and returns $H_2(M_l)$ to the forger $\mathcal{F}$. The simulator stores $(M_l, H_2(M_l))$ in the list L_{H_2}.

H_0 **Query.** The simulator maintains a list L_{H_0} containing elements of the form $(x, H_0(x))$ where $x = (M_l, \mathcal{PK}_l)$. If the message has already been queried then it returns from the list L_{H_0}. Otherwise, the simulator performs the following steps to answer H_0 query on any message M_l.

- choose $\mathbf{T}_{i^*,M_l} \in D^{m\times n}_{\mathbb{Z}_q,\sigma}$ uniformly,
- query $H_1(\mathsf{pk}_{i^*}, \mathcal{PK}_l)$ and $H_2(M)$ to the random oracles H_1 and H_2 respectively,
- find $\mathbf{B} \in \mathbb{Z}_q^{m\times n}$ (using Gauss elimination method or any linear algebra method) satisfying the equation $\mathbf{A} \cdot \mathbf{B} = \mathbf{A} \cdot \mathbf{T}_{i^*,M_l} - \mathbf{Y}_{i^*} \cdot H_1(\mathsf{pk}_{i^*}, \mathcal{PK}_l) \cdot H_2(M_l)$,
- return $H_0(M_l, \mathcal{PK}_l) = \mathbf{B}$ to $\mathcal{F}$ and store $((M_l, \mathcal{PK}_l), H_0(M_l, \mathcal{PK}_l))$ in the list L_{H_0} and $((M_l, \mathcal{PK}_l), \mathbf{T}_{i^*,M_l})$ in the list L_{good}.

The distribution of $\mathbf{T}_{i^*,M_l}$ is identical to the real protocol. Note that in the real protocol, $||\mathbf{T}_{i,M_l}|| \leq \sigma^3 m\sqrt{n}$ and as we have chosen $\mathbf{T}_{i^*,M_l} \in D^{m\times n}_{\mathbb{Z}_q,\sigma}$ giving $||\mathbf{T}_{i^*,M_l}|| \leq \sigma\sqrt{m} \leq \sigma^3 m\sqrt{n}$.

Signature Generation Query. When $\mathcal{F}$ makes a signature query on a message M_l, with signers $\mathcal{PK}_l$, $1 \leq l \leq q_s$ the simulator firstly checks whether $\mathsf{pk}_{i^*} \in \mathcal{PK}_l$. If not, it aborts. Otherwise, $\mathcal{S}$ checks whether $(M_l, H_2(M_l)) \in L_{H_2}$ with $l = k$ where $k \in \{1, 2, \ldots, q_H\}$ is fixed at the beginning of the game. If yes, it aborts. Otherwise, $\mathcal{S}$ checks whether $((M_l, \mathcal{PK}_l), \mathbf{T}_{i^*,M_l}) \in L_{\mathsf{good}}$. If yes, then return $\mathbf{T}_{i^*,M_l}$. If not, then $\mathcal{S}$ calls H_0 query on $(M_l, \mathcal{PK}_l)$ and return $\mathbf{T}_{i^*,M_l}$.

3. With the above knowledge, the forger $\mathcal{F}$ outputs a forgery $\mathsf{msig}^*_{\mathcal{PK},M} = (\mathbf{T}^*_M, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, M)$ on a message M. If it is a valid forgery then $\mathbf{A} \cdot \mathbf{T}^*_M = \mathbf{A} \cdot |I_{\mathcal{PK}}| \cdot H_0(M, \mathcal{PK}) + \sum_{i\in\mathcal{PK}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_2(M)$.
4. The algorithm $\mathcal{S}$ returns *fail* if (a) $\mathsf{msig}^*_{\mathcal{PK},M}$ is not a valid forgery. (b) $\mathsf{pk}_{i^*} \notin \mathcal{PK}$. (c) $M = M_l$ for some $1 \leq l \leq q_s$.

As $\mathsf{pk}_{i^*} \in \mathcal{PK}$ for a valid forgery, $H_1(\mathsf{pk}_{i^*}, \mathcal{PK}) = \mathbf{C}_t$ for some t, $1 \leq t \leq q_H$. The simulator $\mathcal{S}$ computes $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i\in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK})$ where pk_i is the public key corresponding to the signer $i \in I_{\mathcal{PK}}$ and $H_1(\mathsf{pk}_i, \mathcal{PK})$ are simulated as in the H_1 query for each $\mathsf{pk}_i \in \mathcal{PK}$. Let $\mathbf{E}_j = H_1(\mathsf{pk}_j, \mathcal{PK})$ for $\mathsf{pk}_j \in \mathcal{PK}$.

Then the algorithm $\mathcal{S}^{\mathcal{F}}(\mathsf{in}_{\mathcal{S}} = \mathbf{P}, \rho)$ outputs $(\{t\}, \{(\mathsf{msig}_{\mathcal{PK},M}, \mathcal{PK}, \mathsf{pkag}_{\mathcal{PK}}, \mathbf{E}_1, \cdots, \mathbf{E}_{|I_{\mathcal{PK}}|})\})$ where $\rho = (\xi, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{q_H})$.

We prove the theorem by constructing an algorithm $\mathcal{B}$ that, on input an SIS instance $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and the above constructed simulator $\mathcal{S}$, solves the SIS problem. Particularly, $\mathcal{B}$ runs the generalized forking lemma $\mathcal{FL}_{\mathcal{S}}$ on $\mathcal{S}^{\mathcal{F}}(\mathsf{in}_{\mathcal{S}} = \mathbf{P}, \rho)$ given in Sect. 2. The algorithm $\mathcal{B}$ outputs fail if $\mathcal{FL}_{\mathcal{S}}$ outputs $(0, \perp)$. On the other hand, $\mathcal{B}$ outputs a solution $\mathbf{V}^*$ of the SIS instance as follows if $\mathcal{FL}_{\mathcal{S}}$ outputs $(\{t\}, \{\mathsf{out}_1\}, \{\mathsf{out}_2\})$. Here $\mathsf{out}_1 = \{(\mathsf{msig}^*_{\mathcal{PK},M}, \mathcal{PK}, \mathbf{E}_1, \ldots, \mathbf{E}_{|I_{\mathcal{PK}}|})\}$, $\mathsf{out}_2 = (\{(\mathsf{msig}^{'}_{\mathcal{PK}',M}, \mathcal{PK}', \mathbf{E}'_1, \ldots, \mathbf{E}'_{|I_{\mathcal{PK}'}|})\})$ with $\mathsf{msig}^*_{\mathcal{PK},M} = (\mathbf{T}^*_M, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, M)$ and $\mathsf{msig}'_{\mathcal{PK},M} = (\mathbf{T}'_M, \mathsf{pkag}_{\mathcal{PK}'}, I_{\mathcal{PK}'}, M)$ are obtained from two executions of $\mathcal{S}$ with randomness ρ and ρ' such that $\rho|_t = \rho'|_t$ i.e., $\rho = (\xi, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{t-1}, \mathbf{C}_t, \ldots, \mathbf{C}_{q_H})$ and $\rho' = (\xi, \mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{t-1}, \mathbf{C}'_t, \ldots, \mathbf{C}'_{q_H})$. In other words, the arguments of this query are identical ($\mathcal{PK} = \mathcal{PK}'$) but $\mathbf{E}_{i^*} = H_1(\mathsf{pk}_{i^*}, \mathcal{PK}) = \mathbf{C}_t$ and $\mathbf{E}'_{i^*} = H_1(\mathsf{pk}_{i^*}, \mathcal{PK}') = \mathbf{C}'_t$ with $\mathbf{E}_{i^*} \neq \mathbf{E}'_{i^*}$. Also $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot \mathbf{E}_i$ and $\mathsf{pkag}'_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot \mathbf{E}'_i$. Since $\mathbf{E}_j = \mathbf{E}'_j$ for all $j \in I_{\mathcal{PK}}$ except $j = i^*$ before the forking point and therefore $\mathsf{pkag}_{\mathcal{PK}} - \mathsf{pkag}'_{\mathcal{PK}} = \mathsf{pk}_{i^*}\mathbf{E}_{i^*} - \mathsf{pk}'_{i^*}\mathbf{E}'_{i^*}$.

$\mathcal{B}$ extracts $\mathbf{T}^*_M$ and $\mathbf{T}'_M$ from $\mathsf{msig}^*_{\mathcal{PK},M}$ and $\mathsf{msig}'_{\mathcal{PK},M}$ respectively, sets $\mathbf{V}^* = \mathbf{T}^*_M - \mathbf{T}'_M = \mathsf{sk}_{i^*} \cdot (\mathbf{E}_{i^*} - \mathbf{E}'_{i^*}) \cdot H_2(M)$ where $||\mathbf{V}^*|| \leq \sigma\sqrt{m}$, $||\mathbf{E}_{i^*}|| \leq \sigma\sqrt{m}$, $||\mathbf{E}'_{i^*}|| \leq \sigma\sqrt{m}$ and $||H_2(M)|| \leq \sigma\sqrt{n}$. Thus $||\mathbf{V}^*|| \leq \sigma^4 m\sqrt{n}$. Also note that $\mathbf{A} \cdot \mathbf{T}^*_M = \mathbf{A} \cdot \mathbf{T}'_M \bmod q$. This implies $\mathbf{A} \cdot \mathbf{V}^* = 0 \bmod q$. Hence, $\mathbf{V}^*$ is a solution to the SIS instance.

The probability of success of $\mathcal{S}$ is the probability that (i) $\mathcal{F}$ succeeds to output a valid forgery with probability $\epsilon_{\mathcal{F}}$ and (ii) $(M_k, H_2(M_k)) \in L_{H_2}$ with $M_k = M$ i.e., $\mathcal{F}$ has asked the k-th H_2 query on M. Here the index k is guessed at prior by $\mathcal{S}$ before H_2 queries are made. The algorithm $\mathcal{S}$ chooses the correct index with probability $\frac{1}{q_H}$. Thus the success probability of $\mathcal{S}$ is $\frac{\epsilon_{\mathcal{F}}}{q_H}$.

The running time of $\mathcal{S}$ is that of $\mathcal{F}$ plus the time taken to answer the queries and the additional computation $\mathcal{S}$ makes. Let t_{q_H}, t_{q_s} be the time taken to answer hash and sign queries. Let t_{extra} be extra time taken by $\mathcal{S}$. Therefore, the run time of $\mathcal{S}$ is $t_{\mathcal{F}} + t_{q_H} + t_{q_s} + t_{extra}$. By the generalized forking lemma, if $q > \frac{8q_H}{\epsilon_{\mathcal{F}}}$, the running time of $\mathcal{B}$ is $(t_{\mathcal{F}} + t_{q_H} + t_{q_s} + t_{extra}) \cdot 8q_H^2/\epsilon_{\mathcal{F}} \cdot ln(\frac{8q_H}{\epsilon_{\mathcal{F}}})$ and the success probability of $\mathcal{B}$ is atleast $\frac{\epsilon_{\mathcal{F}}}{8q_H}$. □

4 Accountable Subgroup Multisignature

Syntax of Accountable Subgroup Multisignature. Let $\mathcal{PK} = \{\mathsf{pk}_1, \mathsf{pk}_2, \ldots, \mathsf{pk}_l\}$ denotes the set of public keys of a group of signers $I_{\mathcal{PK}} = \{1, 2, \ldots, l\}$ and $\mathcal{SK}_{\mathcal{PK}} = \{\mathsf{sk}_1, \mathsf{sk}_2, \ldots, \mathsf{sk}_l\}$ be the set of corresponding secret keys of the set $\mathcal{PK}$. Let $L = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_k}\}$ be the set of public keys of a subgroup of signers $I_L = \{i_1, i_2, \ldots, i_k\}$ and $\mathcal{SK}_L = \{\mathsf{sk}_{i_1}, \mathsf{sk}_{i_2}, \ldots, \mathsf{sk}_{i_k}\}$ be the set of corresponding secret keys of the set L. The accountable subgroup multisignature scheme allows a subgroup $I_L \subseteq I_{\mathcal{PK}}$ to issue an accountable subgroup multisignature $\mathsf{accmsig}$ on a message M in such a way that the verifier agrees that all

the k signers in I_L have signed the message M. Let there be a designated signer who combines all the signatures of signers into a single accountable subgroup multisignature. The designated signer may be one of the signers or an external party.

An accountable subgroup multisignature scheme $\mathsf{ASM} = \{\mathsf{pg}, \mathsf{kg}, \mathsf{kag}, \mathsf{gmk}, \mathsf{sg}, \mathsf{vrf}\}$ consists of *parameter generation algorithm* pg, *key generation algorithm* kg and *key aggregation algorithm* kag together with an interactive *group membership key protocol* gmk, *signature generation* protocol sg and a deterministic *verification* algorithm vrf. A trusted third party, called the *key generation center* (KGC), generates the public parameter set $\mathcal{Y} \leftarrow \mathsf{AMS.pg}$. A user generates its public-secret key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{ASM.kg}$. The public keys are made public while the secret keys are kept secret to the users. All signer $i \in I_{\mathcal{PK}}$ with its own secret key sk_i execute the protocol $\mathsf{ASM.gmk}$ among themselves and generates a group membership key $\mathsf{mk}_{i,\mathcal{PK}}$ $i \in I_{\mathcal{PK}}$. In signature generation protocol $\mathsf{ASM.sg}$, each signer $i \in I_L \subseteq I_{\mathcal{PK}}$ uses its secret key sk_i and group membership key $\mathsf{mk}_{i,\mathcal{PK}}$ to generate signature $\mathbf{T}_{i,M}$ on a message M and sends $\mathbf{T}_{i,M}$ to the designated signer. The designated signer aggregates all the received signatures $\mathbf{T}_{i,M}$ for $i \in I_L$ on the message M into an accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M}$. The key aggregation algorithm $\mathsf{ASM.kag}$ can be run by anyone to aggregate the public keys in a set $\mathcal{PK}$ into a single public key $\mathsf{pkag}_{\mathcal{PK}}$. The verifier runs the algorithm $\mathsf{ASM.vrf}$ and returns 0, if the multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M}$ is not properly generated or 1 if $\mathsf{accmsig}_{\mathcal{PK},L,M}$ is correct. More concretely, description of these algorithms are given below.

- $\mathsf{ASM.pg}(1^\lambda) \rightarrow \mathcal{Y}$. It is a PPT algorithm run by a KGC on a security parameter λ to generate the public parameter set $\mathcal{Y}$.
- $\mathsf{ASM.kg}(\mathcal{Y}, i) \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$. Each user i runs this algorithm with input the public parameter set $\mathcal{Y}$ to generate the public and secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$. The secret key sk_i is kept secret to the user i while the public key pk_i is made publicly available.
- $\mathsf{ASM.kag}(\mathcal{Y}, \mathcal{PK}) \rightarrow \mathsf{pkag}_{\mathcal{PK}}$. This is a deterministic algorithm and it aggregates the public keys in $\mathcal{PK}$ into a single public key $\mathsf{pkag}_{\mathcal{PK}}$. It outputs the aggregated public key $\mathsf{pkag}_{\mathcal{PK}}$ which asymptotically has the same size as a single public key.
- $\mathsf{ASM.gmk}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}_{\mathcal{PK}}) \rightarrow \mathsf{mk}_{i,\mathcal{PK}}$. With input the public parameter set $\mathcal{Y}$, the set of public keys $\mathcal{PK}$ of the signers, the set of secret keys $\mathcal{SK}_{\mathcal{PK}}$ of the signers in $I_{\mathcal{PK}}$, this interactive protocol runs among all signers in $I_{\mathcal{PK}}$ and generates group membership key $\mathsf{mk}_{i,\mathcal{PK}}$ for each $i \in I_{\mathcal{PK}}$.
- $\mathsf{ASM.sg}(\mathcal{Y}, L, \mathcal{PK}, \mathcal{SK}_L, \mathcal{G}_L, M) \rightarrow \mathsf{accmsig}_{\mathcal{PK},L,M}$. With input the public parameter set $\mathcal{Y}$, the set of public keys $\mathcal{PK}$ of signers, the set of secret keys $\mathcal{SK}_L$ of signers in I_L, the set of group membership keys $\mathcal{G}_L = \{\mathsf{mk}_{i,\mathcal{PK}} | i \in I_L \subseteq I_{\mathcal{PK}}\}$ and a message M, this interactive protocol works as follows. The signer $i \in I_L$ uses its secret key sk_i and group membership key $\mathsf{mk}_{i,\mathcal{PK}}$ to generate a signature $\mathbf{T}_{i,M}$ on M and sends $\mathbf{T}_{i,M}$ to the designated signer. The designated signer aggregates all the signatures $\mathbf{T}_{i,M}$ for $i \in I_L$ on a message M into a single accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M}$.

- $\mathsf{ASM.vrf}(\mathcal{Y}, \mathsf{accmsig}_{\mathcal{PK},L,M}) \rightarrow$ (0 or 1). On input the public parameter set $\mathcal{Y}$ and an accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M}$, this deterministic algorithm returns 1 if the accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M}$ is valid. Otherwise, it returns 0.

1. The simulator $\mathcal{S}$ generates system parameters $\mathcal{Y}$ and a challenge public key pk_{i^*}. The simulator $\mathcal{S}$ runs the forger $\mathcal{F}$ on $(\mathcal{Y}, \mathsf{pk}_{i^*})$.

2a. The forger $\mathcal{F}$ is allowed to make group membership key queries on a set of public keys $\mathcal{PK}_l$, $1 \leq l \leq q_m$ where $\mathcal{PK}_l$ is a set of public keys with $\mathsf{pk}_{i^*} \in \mathcal{PK}_l$ i.e., $\mathcal{F}$ has access to the oracle $O^{(\mathcal{Y},\mathcal{PK}_l,\cdot)}$ in which $\mathcal{S}$ plays the role of the honest signer i^*. The simulator stores the resulting membership key $\mathsf{mk}_{i^*,\mathcal{PK}_l}$ but does not return it to $\mathcal{F}$.

2b. The forger $\mathcal{F}$ is allowed to make signature queries on $(M_l, \mathcal{PK}_l)$ where M_l is a message and $\mathcal{PK}_l$ is a set of public keys with $\mathsf{pk}_{i^*} \in \mathcal{PK}_l$, $1 \leq l \leq q_s$. That is, $\mathcal{F}$ has access to the oracle $O^{(\mathcal{Y},\cdot,\mathcal{PK}_l,\cdot,\cdot,M_l)}$. The simulator plays the role of the honest signer i^* and produce a signature $\mathbf{T}_{i^*,M_l}$ on M_l.

3. Finally, $\mathcal{F}$ outputs a forgery $\mathsf{accmsig}^*_{\mathcal{PK},L,M}$ on a message M for $L \subseteq \mathcal{PK}$ where $\mathcal{PK}$ is a set of public keys of a group of signers.

4. The simulator $\mathcal{S}$ returns 1 if the following conditions hold:
   (a) $\mathsf{ASM.vrf}(\mathcal{Y}, \mathsf{accmsig}^*_{\mathcal{PK},L,M}) \rightarrow 1$,
   (b) $\mathsf{pk}_{i^*} \in L$,
   (c) $M \neq M_l$ $1 \leq l \leq q_s$.
   Otherwise, $\mathcal{S}$ returns 0.

5. The forger $\mathcal{F}$ wins the game if $\mathcal{S}$ returns 1.

Fig. 2. Unforgeability game $\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{F}}(\lambda)$

Completeness. An accountable subgroup multisignature scheme should satisfy completeness. That is, for any $\mathcal{Y} \leftarrow \mathsf{ASM.pg}(1^\lambda)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{ASM.kg}(\mathcal{Y}, i)$ with $i \in I_{\mathcal{PK}}$ where $I_{\mathcal{PK}}$ is the index set for the set of public keys $\mathcal{PK}$, $\mathcal{SK}_{\mathcal{PK}}$ is the corresponding set of secret keys, any message M, any subset $L \subset \mathcal{PK}$ with the set of secret keys $\mathcal{SK}_L$, group membership keys $\mathcal{G}_L = \{\mathsf{mk}_{i,\mathcal{PK}} | i \in I_L \subseteq I_{\mathcal{PK}}\}$ where $\mathsf{mk}_{i,\mathcal{PK}} \leftarrow \mathsf{ASM.gmk}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}_{\mathcal{PK}})$, if $\mathsf{accmsig}_{\mathcal{PK},L,M} \leftarrow \mathsf{ASM.sg}(\mathcal{Y}, L, \mathcal{PK}, \mathcal{SK}_L, \mathcal{G}_L, M)$ then $\mathsf{ASM.vrf}(\mathcal{Y}, \mathsf{accmsig}_{\mathcal{PK},L,M})$ outputs 1.

Security Model. We consider the infeasibility to forge accountable subgroup multisignature with atleast one honest signer following the security model of Boneh et al. [4]. The forger has given access to q_g many group membership key queries along with q_s many signature queries on any message with any set of public keys $\mathcal{PK}$ and any subgroup of signers $I_L \subseteq I_{\mathcal{PK}}$. The unforgeability game $\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{F}}(\lambda)$ between a forger $\mathcal{F}$ and a simulator $\mathcal{L}$ is described in Fig. 2.

Definition 5. *We say that an accountable subgroup multisignature is unforgeable if* $\mathsf{Adv}^{\mathsf{unf}}_{\mathcal{F}}(\lambda) = \mathsf{Pr}[\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{F}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$ *for every PPT adversary* $\mathcal{F}$ *in the experiment* $\mathsf{Exp}^{\mathsf{unf}}_{\mathcal{F}}(\lambda)$ *defined in Fig. 2 where* $\mathsf{negl}(\lambda)$ *is a negligible function in* λ.

4.1 The ASM

We describe our accountable subgroup multisignature $\mathsf{ASM} = \{\mathsf{pg}, \mathsf{kg}, \mathsf{kag}, \mathsf{gmk}, \mathsf{sg}, \mathsf{vrf}\}$ below.

- $\mathsf{ASM.pg}(1^\lambda) \rightarrow \mathcal{Y}$. The key generation center (KGC) generates the system parameters $\mathcal{Y} \leftarrow (n, q, m, \sigma, H_0, H_1, H_2, H_3, \mathbf{A})$ as follows.
  - parameters $n, q, m, \sigma, H_0, H_1, \mathbf{A}$ are generated as in the algorithm $\mathsf{MS.pg}(1^\lambda)$ of Sect. 3,
  - sample cryptographically secure hash functions $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^{n \times n}$ and $H_3 : \{0,1\}^* \rightarrow D_{\mathbb{Z}_q,\sigma}^{n \times n}$ where $D_{\mathbb{Z}_q,\sigma}^{k \times l} = \{\mathbf{M} \in \mathbb{Z}_q^{k \times l} : ||\mathbf{M}|| \leq \sigma\sqrt{k}\}$.
- $\mathsf{ASM.kg}(\mathcal{Y}, i) \rightarrow (\mathsf{pk}_i, \mathsf{sk}_i)$. The signer i generates its own public and secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ same as in the algorithm $\mathsf{MS.kg}(\mathcal{Y}, i)$ of Sect. 3. The secret key $\mathsf{sk}_i = \mathbf{V}_i \in D_{\mathbb{Z}_q,\sigma}^{m \times m}$ is kept secret to the signer i and the public key $\mathsf{pk}_i = \mathbf{Y}_i \in \mathbb{Z}_q^{n \times m}$ is made public. Note that $\mathbf{Y}_i = \mathbf{A} \cdot \mathbf{V}_i \bmod q$.
- $\mathsf{ASM.kag}(\mathcal{Y}, \mathcal{PK}) \rightarrow \mathsf{pkag}_{\mathcal{PK}}$. This deterministic algorithm outputs the aggregated public key $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \in \mathbb{Z}_q^{n \times n}$.
- $\mathsf{ASM.gmk}(\mathcal{Y}, \mathcal{PK}, \mathcal{SK}_{\mathcal{PK}}) \rightarrow \mathsf{mk}_{i,\mathcal{PK}}$. It is a single round protocol between the signers in $I_{\mathcal{PK}}$ where $\mathcal{PK} = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$ is a set of public keys with $\mathsf{pk}_i = \mathbf{Y}_i$ and $\mathcal{SK}_{\mathcal{PK}} = \{\mathsf{sk}_{i_1}, \mathsf{sk}_{i_2}, \ldots, \mathsf{sk}_{i_l}\}$ is the collection of corresponding secret keys with $\mathsf{sk}_i = \mathbf{V}_i$. All signers $i \in I_{\mathcal{PK}}$ utilize the public parameter set $\mathcal{Y} = (n, q, m, \sigma, H_0, H_1, H_2, H_3, \mathbf{A})$ and parallely execute the following.
  - generate $\mathsf{pkag}_{\mathcal{PK}} \leftarrow \mathsf{ASM.kag}(\mathcal{Y}, \mathcal{PK})$ where $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK})$,
  - compute $\mathbf{M}_{j,i} = H_2(\mathsf{pkag}_{\mathcal{PK}}, j) + \mathsf{sk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK}) \cdot H_3(j)$ for all $j \in I_{\mathcal{PK}}$,
  - send $\mathbf{M}_{j,i}$ to signer j with $||\mathbf{M}_{j,i}|| \leq ||H_2(\mathsf{pkag}_{\mathcal{PK}}, j)|| + \sigma^3 m\sqrt{n}$.
  - On receiving $\mathbf{M}_{i,j} \setminus \{i\}$ from all signers $j \in I_{\mathcal{PK}}$, the i-th signer verifies $||\mathbf{M}_{i,j}|| \leq ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + \sigma^3 m\sqrt{n}$. If the verification fails, it returns $\perp$. Otherwise, it computes the group membership key $\mathsf{mk}_{i,\mathcal{PK}} = \sum_{j \in I_{\mathcal{PK}}} \mathbf{M}_{i,j} = \sum_{j \in I_{\mathcal{PK}}} \left[H_2(\mathsf{pkag}_{\mathcal{PK}}, i) + \mathsf{sk}_j \cdot H_1(\mathsf{pk}_j, \mathcal{PK}) \cdot H_3(i)\right]$
- $\mathsf{ASM.sg}(\mathcal{Y}, L, \mathcal{PK}, \mathcal{SK}_L, \mathcal{G}_L, M) \rightarrow \mathsf{accmsig}_{\mathcal{PK},L,M}$. It is a one round protocol run between the members of the set I_L where $L \subseteq \mathcal{PK} = \{\mathsf{pk}_{i_1}, \mathsf{pk}_{i_2}, \ldots, \mathsf{pk}_{i_l}\}$ is the set of public keys of the signers in I_L with $\mathsf{pk}_i = \mathbf{Y}_i$. The set $\mathcal{SK}_L$ is the collection of corresponding secret keys of the signers in I_L with $\mathsf{sk}_i = \mathbf{V}_i$. Each signer $i \in I_L$ performs the following steps by extracting $(n, q, m, \sigma, H_0, H_1, H_2, H_3, \mathbf{A})$ from $\mathcal{Y}$.
  - generate $\mathsf{pkag}_{\mathcal{PK}} \leftarrow \mathsf{ASM.kag}(\mathcal{Y}, \mathcal{PK})$ where $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK})$,
  - compute $\mathbf{T}_{i,M} = \mathsf{sk}_i \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + \mathsf{mk}_{i,\mathcal{PK}}$ with $||\mathbf{T}_{i,M}|| \leq \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |I_{\mathcal{PK}}| \cdot \sigma^3 m\sqrt{n}$,
  - send $\mathbf{T}_{i,M}$ to the designated signer.

Note that $||\mathbf{T}_{i,M}|| \leq \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |I_{\mathcal{PK}}| \cdot \sigma^3 m\sqrt{n}$.
The designated signer verifies whether $||\mathbf{T}_{i,M}|| \leq \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |I_{\mathcal{PK}}| \cdot \sigma^3 m\sqrt{n}$. If not, it aborts and returns $\perp$. Otherwise, the designated signer combines all the signatures $\mathbf{T}_{i,M}, i \in I_L$ to produce $\mathbf{T}_M = \sum_{i \in I_L} \mathbf{T}_{i,M}$ with $||\mathbf{T}_M|| \leq |I_L| \cdot \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot \max_{i \in I_L} ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |I_L| \cdot |I_{\mathcal{PK}}| \cdot \sigma^3 m\sqrt{n}$. The designated combiner also aggregates the public keys in L and generates aggregated subgroup public key $\mathsf{spkag}_L = \sum_{i \in I_L} \mathsf{pk}_i$. It finally returns the accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M} = (\mathbf{T}_M, \mathsf{spkag}_L, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, I_L, M)$.

- $\mathsf{ASM.vrf}(\mathcal{Y}, \mathsf{accmsig}_{\mathcal{PK},L,M}) \rightarrow (0 \text{ or } 1)$. On receiving an accountable subgroup multisignature $\mathsf{accmsig}_{\mathcal{PK},L,M} = (\mathbf{T}_M, \mathsf{spkag}_L, \mathsf{pkag}_{\mathcal{PK}}, I_{\mathcal{PK}}, I_L, M)$, a verifier runs this deterministic algorithm using the public parameter set $\mathcal{Y}$ and returns 1 if
  - $\mathbf{A} \cdot \mathbf{T}_M = \mathsf{spkag}_L \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot \sum_{i \in I_L} \mathbf{A} \cdot H_2(\mathsf{pkag}_{\mathcal{PK}}, i) + \mathsf{pkag}_{\mathcal{PK}} \cdot \sum_{i \in I_L} H_3(i)$ where $\mathsf{spkag}_L = \sum_{i \in I_L} \mathsf{pk}_i$ and $\mathsf{pkag}_{\mathcal{PK}} = \sum_{i \in I_{\mathcal{PK}}} \mathsf{pk}_i \cdot H_1(\mathsf{pk}_i, \mathcal{PK})$
  - $||\mathbf{T}_M|| \leq |I_L| \cdot \sigma\sqrt{m} \cdot H_0(\mathsf{pkag}_{\mathcal{PK}}, M) + |I_{\mathcal{PK}}| \cdot \max_{i \in I_L} ||H_2(\mathsf{pkag}_{\mathcal{PK}}, i)|| + |I_L| \cdot |I_{\mathcal{PK}}| \cdot \sigma^3 m\sqrt{n}$. Otherwise, the verifier returns 0.

The proof of the following theorem is immediate from the construction.

Theorem 5. *The scheme* ASM *described above is complete.*

Theorem 6. *The scheme ASM is unforgeable in the random oracle model if the* SIS *problem is hard.*

Proof **(Sketch).** We assume that there exists a forger $\mathcal{F}$ that wins the unforgeability game played with a simulator $\mathcal{S}$ given in Definition 5 with probability $\epsilon_{\mathcal{F}}$.

1. This step is similar to the step 1 of the Theorem 4 in Sect. 3.1.
2. We give the hints of the simulation of H_0, H_1 and group membership key queries. The H_1-query on $x = (\mathsf{pk}_i, \mathcal{PK})$ is simulated from the already chosen random values $\{\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_{q_H}\}$. for $\mathsf{pk}_i \in \mathcal{PK}$ if $i = i^*$. Otherwise a random value is returned. Let $\mathbf{bad}_1$ be the event that a query to random oracles H_0 or H_2 is made involving $\mathsf{pkag}_{\mathcal{PK}}$ before making H_1 query on $(\mathsf{pk}_i, \mathcal{PK})$ for some pk_i. The simulator $\mathcal{S}$ aborts when the event $\mathbf{bad}_1$ occurs as it cannot simulate the queries without knowing the public keys used to form $\mathsf{pkag}_{\mathcal{PK}}$. The group membership key query on $\mathcal{PK}$ is simulated only if $\mathsf{pk}_{i^*} \in \mathcal{PK}$ by finding $\mathbf{B} \in \mathbb{Z}_q^{n \times n}$ satisfying $\mathbf{A} \cdot \mathbf{M}_{i,i^*} = \mathbf{A} \cdot \mathbf{B} + \mathsf{pk}_{i^*} \cdot H_1(\mathsf{pk}_{i^*}, \mathcal{PK}) \cdot H_3(i)$ and sets $H_2(\mathsf{pkag}_{\mathcal{PK}}, i) = \mathbf{B}$. Here $\mathbf{M}_{i,i^*} \in \mathbb{Z}_q^{m \times n}$ is randomly chosen such that $||\mathbf{M}_{i,i^*}|| \leq \sigma\sqrt{m}$. Let $\mathbf{bad}_2$ be the event that a query to random oracle H_0 is made involving $\mathsf{pkag}_{\mathcal{PK}}$ before making group membership key query on $\mathcal{PK}$. The simulator $\mathcal{S}$ aborts when the event $\mathbf{bad}_2$ occurs as it cannot

simulate the H_0 query without knowing $\mathsf{mk}_{i^*,\mathcal{PK}}$. Also, H_0-query on $x = (\mathsf{pkag}_{\mathcal{PK}}, M)$ is simulated by finding $\mathbf{B} \in \mathbb{Z}_q^{n\times n}$ satisfying $\mathbf{A}\cdot\mathbf{M}_{i,i^*} = \mathbf{A}\cdot\mathbf{B} + \mathbf{Y}_{i^*}\cdot H_1(\mathsf{pk}_{i^*}, \mathcal{PK})\cdot H_3(i)$ and sets $H_2(\mathsf{pkag}_{\mathcal{PK}}, i) = \mathbf{B}$ where $\mathbf{T}_{i^*,M} \in \mathbb{Z}_q^{m\times n}$ is randomly chosen such that $||\mathbf{T}_{i^*,M}|| \leq \sigma\sqrt{m}$.

3. With the view of all the allowed queries, $\mathcal{F}$ outputs a valid forgery.
4. The simulator $\mathcal{S}$ applies the generalized forking lemma (on H_1 query) and solves the SIS instance as we have done in the Theorem 4 in Sect. 3.1. □

The complete proof will be provided in the full version of the paper.

References

1. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 449–458. ACM (2008)
2. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 390–399. ACM (2006)
3. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_3
4. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11273, pp. 435–464. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03329-3_15
5. Drijvers, M., et al.: On the security of two-round multi-signatures. In: On the Security of Two-Round Multi-signatures. IEEE (2019)
6. Drijvers, M., Gorbunov, S., Neven, G., Wee, H.: Pixel: multi-signatures for consensus
7. El Bansarkhani, R., Sturm, J.: An efficient lattice-based multisignature scheme with applications to bitcoins. In: Foresti, S., Persiano, G. (eds.) CANS 2016. LNCS, vol. 10052, pp. 140–155. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-48965-0_9
8. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: a signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33027-8_31
9. Güneysu, T., Oder, T., Pöppelmann, T., Schwabe, P.: Software speed records for lattice-based signatures. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 67–82. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_5
10. Harn, L.: Group-oriented (t, n) threshold digital signature scheme and digital multisignature. IEE Proc.-Comput. Digital Tech. **141**(5), 307–313 (1994)
11. Horster, P., Michels, M., Petersen, H.: Meta-multisignature schemes based on the discrete logarithm problem. Information Security — The Next Decade. IAICT, pp. 128–142. Springer, Boston (1995). https://doi.org/10.1007/978-0-387-34873-5_11

12. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC Res. Dev. **71**, 1–8 (1983)
13. Langford, S.K.: Weaknesses in some threshold cryptosystems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 74–82. Springer, Heidelberg (1996). https://doi.org/10.1007/3-540-68697-5_6
14. Li, C.-M., Hwang, T., Lee, N.-Y.: Threshold-multisignature schemes where suspected forgery implies traceability of adversarial shareholders. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 194–204. Springer, Heidelberg (1995). https://doi.org/10.1007/BFb0053435
15. Lu, S., Ostrovsky, R., Sahai, A., Shacham, H., Waters, B.: Sequential aggregate signatures, multisignatures, and verifiably encrypted signatures without random oracles. J. Cryptol. **26**(2), 340–373 (2012). https://doi.org/10.1007/s00145-012-9126-5
16. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to bitcoin. Des. Codes Cryptogr. **87**(9), 2139–2164 (2019). https://doi.org/10.1007/s10623-019-00608-x
17. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures. In: Proceedings of the 8th ACM Conference on Computer and Communications Security, pp. 245–254. ACM (2001)
18. Michels, M., Horster, P.: On the risk of disruption in several multiparty signature schemes. In: Kim, K., Matsumoto, T. (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 334–345. Springer, Heidelberg (1996). https://doi.org/10.1007/BFb0034859
19. Ohta, K., Okamoto, T.: A digital multisignature scheme based on the Fiat-Shamir scheme. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT 1991. LNCS, vol. 739, pp. 139–148. Springer, Heidelberg (1993). https://doi.org/10.1007/3-540-57332-1_11
20. Ohta, K., Okamoto, T.: Multi-signature schemes secure against active insider attacks. IEICE Trans. Fund. Electron. Commun. Comput. Sci. **82**(1), 21–31 (1999)
21. Ristenpart, T., Yilek, S.: The power of proofs-of-possession: securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 228–245. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72540-4_13