Accountable Multi-Signatures with Constant Size Public Keys

Dan Boneh[1], Aditi Partap[1], and Brent Waters[2]

[1] Stanford University, {dabo,aditi712}@cs.stanford.edu
[2] NTT Research and University of Texas, Austin, bwaters@cs.utexas.edu

Abstract. A multisignature scheme is used to aggregate signatures by multiple parties on a common message m into a single short signature on m. Multisignatures are used widely in practice, most notably, in proof-of-stake consensus protocols. In existing multisignature schemes, the verifier needs the public keys of all the signers in order to verify a multisignature issued by some subset of signers. We construct new practical multisignature schemes with three properties: (i) the verifier only needs to store a *constant* size public key in order to verify a multisignature by an arbitrary subset of parties, (ii) signature size is constant beyond the description of the signing set, and (iii) signers generate their secret signing keys locally, that is, without a distributed key generation protocol. Existing schemes satisfy properties (ii) and (iii). The new capability is property (i) which dramatically reduces the verifier's memory requirements from linear in the number of signers to constant. We give two pairing-based constructions: one in the random oracle model and one in the plain model. We also show that by relaxing property (iii), that is, allowing for a simple distributed key generation protocol, we can further improve efficiency while continuing to satisfy properties (i) and (ii). We give a pairing-based scheme and a lattice-based scheme in this relaxed model.

1 Introduction

An n-party accountable multisignature scheme [MOR01] is a tuple of five algorithms. A key generation algorithm $\mathsf{LocalKeyGen}(1^\lambda) \to (\mathsf{pk}_i, \mathsf{sk}_i)$ generates a key pair for each of the n parties. Let $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ be the resulting vector of public keys. A $\mathsf{Sign}(\mathsf{sk}_i, m) \to \sigma_i$ algorithm lets party i sign a message m. An aggregation algorithm $\mathsf{SigAgg}(\{\sigma_i\}_{i\in\mathcal{J}}) \to \sigma$ aggregates the signatures generated by parties $\mathcal{J} \subseteq [n]$ on a common message m into a short aggregate signature $\sigma = (\sigma', \mathcal{J})$, where $\mathcal{J}$ is a description of the signing set, and σ' is additional signature data. We require that σ' is constant size, that is, independent of the size of $\mathcal{J}$. A verification algorithm $\mathsf{Vf}(\mathsf{pk}, m, \sigma)$ verifies the aggregate signature using the list of n public keys in pk. Finally, it is convenient to include an explicit tracing algorithm $\mathsf{Trace}(m, \sigma) \to \mathcal{J}$ that traces an aggregate signature σ to the set of parties that generated it. This algorithm simply parses σ as $(\sigma', \mathcal{J})$ and outputs $\mathcal{J} \subseteq [n]$. Informally, the signature scheme is secure if a coalition of $n-1$ corrupt parties cannot produce a valid aggregate signature σ that frames the remaining party for signing a message m which it did not sign. Note that the corrupt parties may choose their public keys adversarially in what is called a rogue public key attack. We give precise definitions in the next section. As shorthand, we will refer to an n-party accountable multisignature scheme simply as a multisignature scheme.

The tracing algorithm provides accountability: when a valid signature σ on some rogue message m is found, the tracing algorithm will reveal the subset of parties that generated σ so that they can be held accountable. This property is required in proof-of-stake consensus [DGKR18, GHM+17, Smi22] so that the parties who generated the rogue signature will lose their committed stake. Accountability implies that a signature must encode the signing set $\mathcal{J}$. In all our constructions the signature contains an explicit description of the signing set $\mathcal{J}$. This lets the verifier choose

the signing sets for which it accepts the signature as valid: the verifier could enforce a (weighted) threshold requirement on the signing set, or it could choose to implement a more sophisticated validity policy on $\mathcal{J}$.

Accountable multisignatures were defined by Micali, Ohta, and Reyzin [MOR01] and since then several constructions have been proposed. Some are based on discrete-log [BN06, NRS21], some are based on pairings [Bol03, BGLS03, LOS+06, RY07, BDN18, BCG+23], some are based on lattices [FH20, DOTT21, BTT22], and some are based on generic SNARKs [DFKP16]. The discrete-log and lattice constructions require rounds of interaction among the signers and do not support post-signing aggregation. In the pairing based constructions, aggregation does not require the original signers to be present, so that anyone can aggregate signatures. Moreover, the aggregate signature data σ' is constant size. This also holds for the SNARK-based construction, but its performance is much worse than the pairing-based schemes. The most widely used multisignatures in proof-of-stake consensus are based on BLS signatures [BLS01].

In all these algebraic constructions, the verifier needs to store the list of all n signer public keys to verify a multisignature. For example, in BLS multisignatures, the verifier uses a description of the signing set $\mathcal{J}$ to compute an aggregate public key apk by computing $apk := \prod_{i\in\mathcal{J}} \mathsf{pk}_i$, and uses this apk to verify the aggregate signature. Computing apk for an arbitrary set $\mathcal{J}$ requires storing the entire vector $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. Interestingly, the SNARK-based multisignature scheme [DFKP16] can be easily adapted so that the verifier only needs a constant size verification key: a collision-resistant hash of pk along with a constant-size SNARK verification key. However, as mentioned above, these multisignatures are difficult to use even for a moderately large n.

Our results. We define a new type of multisignature scheme that greatly reduces the amount of information that the verifier needs to store. In particular, we introduce a new key aggregation algorithm, $\mathsf{KeyAgg}(\mathsf{pk}) \to (\mathsf{pkc}, \mathsf{vk})$, that takes as input $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and outputs a public signature aggregation key pkc and a *constant size* verification key vk.

- The key pkc is used by the signature aggregation algorithm SigAgg which is now invoked as $\mathsf{SigAgg}(\mathsf{pkc}, \{\sigma_i\}_{i\in\mathcal{J}})$ to form the aggregate signature σ.
- The short verification key vk is used by algorithm $\mathsf{Vf}(\mathsf{vk}, m, \sigma)$ to verify signatures.

In other words, once the verifier runs KeyAgg on $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ it no longer needs to store this vector of keys. It suffices to only store vk which is constant size, and can be used to verify signatures generated by any set $\mathcal{J} \subseteq [n]$. Alternatively, a trusted party could compute vk from pk, and give vk to the verifier, in which case the verifier never needs to see the vector pk. In our first two schemes, the aggregation key pkc has linear size in n, which is fine because it is only used by the signature aggregator who anyhow handles a linear number of signature shares from the signers. In our third and fourth schemes pkc is empty, but these schemes require a simple distributed key generation protocol (DKG).

Our first scheme is a new pairing-based multisignature whose security is proved in the random oracle model based on a variant of the Bilinear Diffie-Hellman (BDH) problem. As mentioned above, pkc is linear size, while vk and the final signature data σ' are constant size. Our multisignature scheme is based on the signature scheme due to Boneh and Boyen [BB04, BB11], called the BB signature scheme (obtained from the first IBE system in their paper). We generalize this scheme to make it into a multisignature with a constant size verifier key vk.

Our second scheme modifies the first scheme to prove security in the plain model (i.e., without relying on random oracles), while preserving the properties of the first scheme. Our first scheme uses two random oracles, and both need to be instantiated concretely. We instantiate one with an

algebraic hash using an approach similar to Boneh and Boyen [BB04, BB11] or Waters [Wat05]. We instantiate the second random oracle by using suitable values derived from the underlying complexity assumption. This approach shows that the random oracle proof was only using minimal properties of this second random oracle, in a way reminiscent of the work of Hofheinz and Kiltz [HK08, HK12]. We prove security in the plain model based on an assumption we call the n-BDH assumption.

Next, we look at multisignatures with a constant size verifier key, but where we allow for a distributed key generation (DKG) protocol among the signers at setup (in our first two schemes the parties generated their keys locally; there was no need for a DKG among them). In Section 5 we describe such a pairing-based scheme where the benefit of introducing a DKG is that there is no longer a need for a signature aggregation key pkc. Signature aggregation is done using only the supplied (constant-size) signatures from the signers.

Once we introduce a DKG, the security model changes and becomes more closely related to the security model for a threshold signature scheme (e.g., as in [BS23]). After setup the adversary can fully corrupt some parties by requesting their signing keys, and is provided with a signing oracle for the remaining parties. As before, the scheme is secure if the adversary cannot create a valid aggregate signature on a message m that traces to a non-corrupt party who did not sign m. We prove security of the scheme in Section 5 in a fully adaptive model where the adversary can adaptively choose which parties to corrupt. The proof is based on the BDH assumption in the random oracle model. We note that several recent works prove adaptive security for standard threshold signatures [BL22, CKM23, DR23].

Our final scheme, presented in Section 6, is another multisignature with a constant size verifier key and a DKG, but this time based on lattices. Our starting point is a recent multisignature scheme due to Damgård, Orlandi, Takahashi, and Tibouchi [DOTT21]. We adapt the scheme to make it have a constant-size verification key. However, unlike our earlier schemes, the signing process requires interaction among the signers, and the signers need to know the signing set $\mathcal{J}$.

Related work. Two recent papers, one by Das et al. [DCX+23] and one by Garg et al. [GJM+23], also presented a multisignature scheme with a short verification key. While the schemes presented in these papers are not accountable (there is no way to trace a valid signature), they can be made accountable by increasing the length of the signature. Their constructions are based on BLS multisignatures, where the work to compute the aggregate public key $apk := \prod_{i \in \mathcal{J}} \mathsf{pk}_i$ is shifted from the verifier to the aggregator. The aggregator includes apk in the aggregate signature along with a proof that it was computed correctly. As a result, their verifier key vk is longer than ours, as is the aggregate signature which includes apk and a proof that apk is valid. Both schemes rely on a one-time trusted setup to generate a structured reference string (SRS) used to prove validity of apk. The schemes in this paper do not require a trusted setup. Table 1 compares our constructions to other pairing-based multisignature schemes.

Proactive refresh. A recent result [BPR22] shows how to proactively refresh the key shares of an accountable threshold signature scheme without changing the public key. Our third multisignature scheme (Section 5) fits well into their two-level construction, to obtain a proactively refreshable threshold scheme that achieves their strongest level of accountable security. We discuss this further in Appendix B.

Multisignatures vs. aggregate signatures. Multisignatures, the topic of this paper, compress signatures by multiple parties *on the same message* into a single short signature. Aggregate signatures [BGLS03, GR06, BNN07] can compress signatures by multiple parties on possibly different

	sig size overhead	vk size	pkc size	DKG	RO	one-time trusted setup	KeyAgg time	SigAgg time
BLS multisigs [Bol03]	$1\mathbb{G}$	$n\mathbb{G}$	0	no	yes	no	0	$O(\|\mathcal{J}\|)$
BB multisigs [LOS+06]	$2\mathbb{G}$	$n\mathbb{G}$	0	no	no	no	0	$O(\|\mathcal{J}\|)$
SNARK [Gro16, DFKP16]	$3\mathbb{G}$	$7\mathbb{G}$	$\text{poly}(\lambda, n)$	no	yes	yes	$O(n)$	$\text{poly}(\lambda, n)$
Das et al. [DCX+23]	$8\mathbb{G}$	$7\mathbb{G}$	$n\mathbb{G}$	no	yes	yes	$O(n^2)$	$O(n)$
Garg et al. [GJM+23]	$9\mathbb{G}$	$6\mathbb{G}$	$n\mathbb{G}$	no	yes	yes	$O(n^2)$	$O(n)$
This work (Section 3)	$2\mathbb{G}$	$\mathbf{1}\mathbb{G}$	$n\mathbb{G}$	no	yes	no	$O(n^2)$	$O(\|\mathcal{J}\|)$
This work (Section 4)	$2\mathbb{G}$	$\mathbf{1}\mathbb{G}$	$n\mathbb{G}$	no	no	no	$O(n^2)$	$O(\|\mathcal{J}\|)$
This work (Section 5)	$2\mathbb{G}$	$\mathbf{1}\mathbb{G}$	0	yes	yes	no	0	$O(\|\mathcal{J}\|)$
Remark 6 (Section 3)	$2\mathbb{G}$	$\sqrt{n}\,\mathbb{G}$	$n\mathbb{G}$	no	yes	no	$O(n^{1.5})$	$O(\|\mathcal{J}\|)$

Table 1. Comparing pairing-based multisignature constructions. The numbers indicate the number of pairing group elements. For signature size we measure the signature data overhead beyond the description of the signing set $\mathcal{J}$. The DKG column indicates whether a distributed key generation is needed to generate keys, the RO column indicates whether the proof of security is set in the random oracle model, and the setup column indicates whether a one-time trusted setup is needed to generate a reference string. The columns KeyAgg time and SigAgg time indicate the time to aggregate n public keys into pkc and the time to generate the final aggregate signature, respectively. The SNARK row corresponds to a multisignature built from the Groth16 SNARK [Gro16]. The numbers for the [DCX+23, GJM+23] rows are for their non-accountable scheme; adding accountability makes the parameters worse.

messages into a single short signature. In either case there is a preference for schemes where aggregation can be done after the signers have generated their signatures. That is, there is no need to involve the signers in the aggregation process. A weaker notion of aggregate signatures, called *sequential aggregate signatures* [LMRS04, LOS+06, FLS12, BGR12, GOR18, EB14], enables n signers to sign n messages in sequence, one after the other. For $i = 2, \ldots, n$, signer number i receives the latest aggregate from signer $(i-1)$, adds its signature on message m_i to the aggregate, and passes the resulting aggregate to the next signer. The last signer, signer number n, obtains the final short aggregate signature, and this aggregate signature commits signer i to message m_i for all $i \in [n]$.

2 Preliminaries

Notation. For an integer $n \in \mathbb{N}$ we use $[n]$ to denote the set $\{1, \ldots, n\}$. For a distribution X we denote by $x \leftarrow\!\$\, X$ the process of sampling a value x from the distribution X. Similarly, for a finite set $\mathcal{X}$ we denote by $x \leftarrow\!\$\, \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$. We denote matrices by boldface capital letters, e.g. $\boldsymbol{A}$, and vectors in boldface lower-case letters, e.g. $\boldsymbol{w}$. We may use a non-bold capital letter, e.g. A or V, to describe a matrix or a vector, when we wish to emphasize that this matrix or vector is being treated as a random variable.

2.1 Multisignatures with Local Key Generation

We begin by defining secure multisignature schemes, where each signer generates its signing key locally. Similarly, signature shares are generated without interacting with the other signers. In Appendix A we define schemes that allow interaction during signing. This is needed for our lattice-based scheme in Section 6.

Syntax. A **multisignature scheme** (MS) with local key generation is a tuple of PPT algorithms $\mathsf{MS} = (\mathsf{Setup}, \mathsf{LocalKeyGen}, \mathsf{KeyAgg}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace})$, where:

- $\mathsf{Setup}(1^\lambda, n) \to \mathsf{pp}$: a one-time global setup algorithm that takes as input the number of parties $n \le n(\lambda)$, for some polynomial $n(\cdot)$, and outputs public parameters pp, which are an implicit input to all the remaining algorithms.

- $\mathsf{LocalKeyGen}(i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: takes as input an index $i \in [n]$ and outputs a public key pk_i and a corresponding secret key sk_i.
- $\mathsf{KeyAgg}\big(\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)\big) \to (\mathsf{pkc}, \mathsf{vk})$: a deterministic algorithm that aggregates n public keys into a public aggregation key pkc and a verifier key vk. It may output $\perp$ if any of the input public keys are invalid.
- $\mathsf{Sign}(\mathsf{sk}_i, m) \to s_i$: signs the message m using key sk_i and outputs signature share s_i.
- $\mathsf{SigAgg}(\mathsf{pkc}, \{s_i\}_{i \in \mathcal{J}}) \to \sigma$: a deterministic algorithm that combines the given signature shares into a complete signature σ, possibly using a public aggregation key pkc. Outputs $\perp$ if any of the input signatures are invalid.
- $\mathsf{Vf}(\mathsf{vk}, m, \sigma)$: a deterministic verification algorithm that outputs 1 (implying acceptance) or 0 (implying rejection).
- $\mathsf{Trace}(m, \sigma) \to \mathcal{J}$: a deterministic algorithm that traces a signature σ to a subset $\mathcal{J} \subseteq [n]$ that generated σ.

These algorithms must satisfy the following *verification correctness* and *tracing correctness* properties: for all messages m in the message space $\mathcal{M}$, all polynomials $n = n(\lambda)$, and all non-empty subsets $\mathcal{J} \subseteq [n]$, it holds that

$$\Pr\left[\begin{array}{c}\mathsf{Vf}\big(\mathsf{vk}, m, \mathsf{SigAgg}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j, m)\}_{j \in \mathcal{J}})\big) = 1, \\ \mathsf{pkc} \neq \perp, \quad \mathsf{vk} \neq \perp\end{array}\right] = 1,$$

$$\Pr\left[\begin{array}{c}\mathsf{Trace}\big(m, \mathsf{SigAgg}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j, m)\}_{j \in \mathcal{J}})\big) = \mathcal{J}, \\ \mathsf{pkc} \neq \perp, \quad \mathsf{vk} \neq \perp\end{array}\right] = 1,$$

where the probability is over the random variables $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{LocalKeyGen}(i)$ for all $i \in [n]$, $(\mathsf{pkc}, \mathsf{vk}) \leftarrow \mathsf{KeyAgg}((\mathsf{pk}_1, \ldots, \mathsf{pk}_n))$, and the random coins of Sign.

Security Notions. A multisignature scheme with local key generation should be unforgeable, even in the face of rogue public key attacks. An adversary is allowed to generate public keys for all but one honest party, and is allowed to query signatures of this honest party on any message of its choice. Unforgeability requires that such an adversary should not be able to produce a valid signature on a message m on behalf of a subset $\mathcal{J}$ that includes this honest party, without observing its secret key or its signature share on m.

For a multisignature scheme MS, the unforgeability requirement is captured by the security game, Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS}}$, in Figure 1. The adversary first sends the number of signers n to the challenger, which then runs Setup and sends the public parameters pp to the adversary. The adversary responds with the index $i^* \in [n]$ for the honest party. This is followed by the challenger sampling a key for the honest party using the $\mathsf{LocalKeyGen}(i^*)$ procedure. The adversary then issues signature queries for messages of its choice, and receives back signatures on these messages with respect to the secret key of party i^*. Finally, the adversary should produce (i) a list of n valid public keys that includes the honest public key at index i^* and (ii) a valid forgery, that is, a message m^* for which it did not issue a signature query, and a valid signature σ^* on m^*.

We denote by $\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS},\mathcal{A}}(\lambda)$ the output of Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS}}$ when executed with an adversary $\mathcal{A}$ and security parameter λ. It is a random variable defined over the random bits of both $\mathcal{A}$ and the random choices of the game's main procedure and oracles. With this notation, the following definition captures security for a multisignature scheme.

Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS}}$ with respect to an adversary $\mathcal{A}$ and security parameter λ

1 : $(\mathsf{st}, n) \leftarrow \mathcal{A}(\lambda)$
2 : $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n)$; $\mathcal{Q}_{\mathsf{sig}} \leftarrow \emptyset$
3 : $(\mathsf{st}, i^*) \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pp})$ // $i^* \in [n]$ is the user index to attack
4 : $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow\!\!\$\ \mathsf{LocalKeyGen}(i^*)$ // pk^* is the public key of party i^*
5 : $\big((\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*), m^*, \sigma^*\big) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{SignO}(\cdot)}(\mathsf{st}, \mathsf{pk}^*)$ // The forgery from the adversary; we require on line 9 that $\mathsf{pk}_{i^*}^* = \mathsf{pk}^*$.
6 : $(\mathsf{pkc}^*, \mathsf{vk}^*) \leftarrow \mathsf{KeyAgg}\big((\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)\big)$
7 : **if** $\mathsf{pkc}^* = \bot \ \vee\ \mathsf{vk}^* = \bot$ **then** // if key aggregation failed
8 : **return** 0
9 : **if** $\mathsf{pk}^* \neq \mathsf{pk}_{i^*}^* \ \vee\ \mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 0 \ \vee\ m^* \in \mathcal{Q}_{\mathsf{sig}}$ **then** // if forgery is invalid
10 : **return** 0
11 : **if** $i^* \notin \mathsf{Trace}(m^*, \sigma^*)$ **then**
12 : **return** 0
13 : **return** 1

Oracle $\mathsf{SignO}(m)$

1 : $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}^*, m)$
2 : $\mathcal{Q}_{\mathsf{sig}} \leftarrow \mathcal{Q}_{\mathsf{sig}} \cup \{m\}$
3 : **return** σ

Fig. 1. The security game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS}}$ for a multisignature scheme $\mathsf{MS} = (\mathsf{Setup}, \mathsf{LocalKeyGen}, \mathsf{KeyAgg}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace})$.

Definition 1. *A Multisignature scheme* MS *is said to be secure if for all* PPT *adversaries* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{MS},\mathcal{A}}(\lambda) := \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{MS},\mathcal{A}}(\lambda) = 1\right].$$

The security definition above extends to the random oracle model by granting all algorithms, including the adversary $\mathcal{A}$, oracle access to a function H chosen uniformly at random from a family $\mathcal{H}$ of functions. In the correctness and security requirements (Definition 1), all probabilities are then also taken over the random choice of H.

Remark 1 (User adaptivity). The adversary in Figure 1 is required to choose the user index i^* to attack at the beginning of the game (Line 3). This does not limit the adversary's adaptivity. The game can be easily modified so that the adversary sends i^* along with its forgery on Line 5. The two definitions are equivalent because the challenger can guess i^* on Line 3 and abort if it guessed incorrectly. This adds at most a factor of $O(n)$ to the adversary's advantage. We chose to use the definition where i^* is selected on Line 3 since it eliminates the need to guess an i^* in the security proofs.

3 An Efficient Multisignature Scheme With a Short Verification Key

We now describe a multisignature scheme with constant size verification key vk and short signatures. The scheme uses a linear size aggregation key pkc.

The scheme is described in Figure 2. The public parameters include an asymmetric bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$, where $\mathbb{G}_1, \mathbb{G}_2$ are cyclic groups of prime order p generated by g_1, g_2 respectively, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable non-degenerate bilinear map. We use $\mathsf{GroupGen}(1^\lambda) \to (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$ to denote a bilinear group generator that outputs an asymmetric bilinear group. Our scheme also relies on two hash functions $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$ where $\mathcal{M}$ is the message space, and $\mathsf{H}_1 : [n] \to \mathbb{G}_1$, where n is an upper bound on the number of signers. We will model these hash functions as random oracles in the proof of security.

Remark 2 (On synchronizing slot numbers). The parties in this scheme generate their secret keys locally. However, each of the n parties needs to choose a unique slot number in $\{1, \ldots, n\}$. This can be done by publishing a counter on a public bulletin board, initialized at zero. Every party that joins the system increments the counter by one and uses its current value as its slot number.

Remark 3 (Adding and removing parties). Whenever a party joins the system as a signer, it will publish its pk_i. The signature aggregator will check validity of pk_i (step 2 of KeyAgg in Figure 2) and if valid, it will aggregate this pk_i into its current signature aggregation key pkc. Every verifier will similarly check validity of pk_i and aggregate $\mathsf{pk}_{i,0}$ into its vk. When party j leaves the system, and no longer issues signatures, the only change is that it tells the signature aggregators to delete cell number j of pkc, since it is no longer needed.

Correctness. First, we see why honestly generated public keys are valid. Observe that for every $i \in [n]$ and $j \in [n] \setminus \{i\}$, we have that

$$e(\mathsf{pk}_{i,j}, g_2) = e(\mathsf{H}_1(j)^{\alpha_i}, g_2) = e(\mathsf{H}_1(j), g_2^{\alpha_i}) = e(\mathsf{H}_1(j), \mathsf{pk}_{i,0}).$$

Hence, for honestly generated keys, the procedure KeyAgg will output a valid pkc and vk. Next, observe that for an honestly generated signature $(\sigma_0, \sigma_1, \mathcal{J})$, we have,

$$\begin{aligned}
e(\sigma_1, g_2) &= e(\prod_{j \in \mathcal{J}} (\sigma_{j,1} \cdot \mathsf{pkc}_j), g_2) \\
&= e\left(\prod_{j \in \mathcal{J}} \left(\mathsf{H}_1(j)^{\alpha_j} \cdot \mathsf{H}_0(m)^{r_{j,m}} \cdot \prod_{i \in [n] \setminus \{j\}} \mathsf{H}_1(j)^{\alpha_i}\right), g_2\right) \\
&= e\left(\left(\prod_{j \in \mathcal{J}} \mathsf{H}_1(j)\right)^{\Sigma_{i \in [n]} \alpha_i} \cdot \mathsf{H}_0(m)^{\Sigma_{j \in \mathcal{J}} r_{j,m}}, g_2\right) \\
&= e\left(\prod_{j \in \mathcal{J}} \mathsf{H}_1(j), g_2^{\Sigma_{i \in [n]} \alpha_i}\right) \cdot e\left(\mathsf{H}_0(m), g_2^{\Sigma_{j \in \mathcal{J}} r_{j,m}}\right) \\
&= e\left(\prod_{j \in \mathcal{J}} \mathsf{H}_1(j), \mathsf{vk}\right) \cdot e\left(\mathsf{H}_0(m), \sigma_0\right)
\end{aligned}$$

The Multisignature scheme SIG_1 with local key generation

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. Sample a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p) \leftarrow\!\!\$\ \mathsf{GroupGen}(1^\lambda)$.
2. Output $\mathsf{pp} \leftarrow (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$.
3. The system uses two hash functions: $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$ and $\mathsf{H}_1 : [n] \to \mathbb{G}_1$.

$\underline{\mathsf{LocalKeyGen}(i \in [n])}$:

1. Sample $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p$. Set $\mathsf{sk}_i \leftarrow \alpha_i$.
2. For all $j \in [n] \setminus \{i\}$, compute $\mathsf{pk}_{i,j} \leftarrow \mathsf{H}_1(j)^{\alpha_i}$.
3. Output $\mathsf{pk}_i \leftarrow (g_2^{\alpha_i}, \{\mathsf{pk}_{i,j}\}_{j\in[n]\setminus\{i\}}) \in \mathbb{G}_2 \times \mathbb{G}_1^{n-1}$.

$\underline{\mathsf{KeyAgg}(\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))}$:

1. For each $i \in [n]$, parse pk_i as $(\mathsf{pk}_{i,0}, \{\mathsf{pk}_{i,j}\}_{j\in[n]\setminus\{i\}})$.
2. For each $i \in [n]$ and $j \in [n] \setminus \{i\}$, if $e(\mathsf{H}_1(j), \mathsf{pk}_{i,0}) \neq e(\mathsf{pk}_{i,j}, g_2)$, output $\perp$.
3. Compute $\mathsf{pkc}_i \leftarrow \prod_{j\in[n]\setminus\{i\}} \mathsf{pk}_{j,i} \in \mathbb{G}_1$, for all $i \in [n]$.
4. Output $(\mathsf{pkc} \leftarrow \{\mathsf{pkc}_i\}_{i\in[n]} \in \mathbb{G}_1^n,\ \ \mathsf{vk} \leftarrow \prod_{i\in[n]} \mathsf{pk}_{i,0} \in \mathbb{G}_2)$.

   // observe that $\mathsf{pkc}_i = \mathsf{H}_1(i)^{\Sigma_{j\in[n]\setminus\{i\}}\alpha_j} \in \mathbb{G}_1$ and $\mathsf{vk} = g_2^{\Sigma_{j\in[n]}\alpha_j} \in \mathbb{G}_2$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i,\ m \in \mathcal{M})}$:

1. Parse sk_i as α_i.
2. Sample $r_{i,m} \leftarrow\!\!\$\ \mathbb{Z}_p$. Compute $\sigma_{i,0} \leftarrow g_2^{r_{i,m}}$ and $\sigma_{i,1} \leftarrow \mathsf{H}_1(i)^{\alpha_i} \cdot \mathsf{H}_0(m)^{r_{i,m}}$.
3. Output $\sigma_i = (\sigma_{i,0}, \sigma_{i,1}) \in \mathbb{G}_2 \times \mathbb{G}_1$.

$\underline{\mathsf{SigAgg}(\mathsf{pkc}, \{\sigma_j\}_{j\in\mathcal{J}})}$:

1. For each $j \in \mathcal{J}$, parse σ_j as $(\sigma_{j,0}, \sigma_{j,1})$. Parse pkc as $\{\mathsf{pkc}_i\}_{i\in[n]}$.
2. Compute $\sigma_0 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,0}$.
3. For each $j \in \mathcal{J}$, compute $\hat{\sigma}_{j,1} \leftarrow \sigma_{j,1} \cdot \mathsf{pkc}_j$.
4. Output $\sigma = (\sigma_0,\ \ \sigma_1 \leftarrow \prod_{j\in\mathcal{J}} \hat{\sigma}_{j,1},\ \ \mathcal{J})$.

   // observe that $\hat{\sigma}_{j,1}$ is equal to $\mathsf{H}_1(j)^{\Sigma_{i\in[n]}\alpha_i} \cdot \mathsf{H}_0(m)^{r_{j,m}}$.

$\underline{\mathsf{Vf}(\mathsf{vk}, m, \sigma)}$:

1. Parse σ as $(\sigma_0, \sigma_1, \mathcal{J})$.
2. Output 1 if $e(\sigma_1, g_2) = e(\mathsf{H}_0(m), \sigma_0) \cdot e(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j), \mathsf{vk})$.

$\underline{\mathsf{Trace}(m, \sigma)}$:

1. Parse σ as $(\sigma_0, \sigma_1, \mathcal{J})$ and output $\mathcal{J} \subseteq [n]$.

Fig. 2. The Multisignature scheme SIG_1 with local key generation

We see that an honestly generated signature will pass the checks in Vf, implying correctness and trace correctness.

Security. We next prove security of the multisignature scheme SIG_1 using the co-Bilinear Diffie-Helman assumption, assuming H_0 and H_1 are modeled as random oracles. Let us first define the co-Bilinear Diffie-Helman assumption.

Definition 2. *Let $\mathcal{G}$ be an asymmetric bilinear group generator. The co-BDH assumption holds with respect to $\mathcal{G}$ if, for all probabilistic polynomial time adversaries $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}^{\text{co-bdh}}_{\mathcal{G},\mathcal{A}}(\lambda) := \Pr\left[\mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, p, g_1, g_2, g_1^a, g_2^a, g_1^b, g_2^b, g_1^c, g_2^c) = e(g_1, g_2)^{abc}\right]$$

where the probability is taken over the random choice of generators g_1, g_2 of $\mathbb{G}_1, \mathbb{G}_2$ respectively, the random choice of exponents $a, b, c \in \mathbb{Z}_p$, and the random bits used by $\mathcal{A}$.

Theorem 1 below reduces the security of SIG_1 to the hardness of the co-Bilinear Diffie-Helman problem. If we were using a symmetric pairing, then it would be possible to reduce security to the Computational Diffie-Helman (CDH) problem in the source group. But when using an asymmetric pairing, it is more convenient to use co-Bilinear Diffie-Helman.

Theorem 1. *Let $\mathcal{G}$ be an asymmetric bilinear group generator. Then, for every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ with about the same running time as $\mathcal{A}$ such that,*

$$\mathsf{Adv}^{\text{uf}}_{\mathsf{SIG}_1,\mathcal{A}}(\lambda) \leq 2e(q_S + 1) \cdot \mathsf{Adv}^{\text{co-bdh}}_{\mathcal{G},\mathcal{B}}(\lambda) \tag{1}$$

where $q_H = q_H(\lambda)$ and $q_S = q_S(\lambda)$ are a bound on the number of queries issued by $\mathcal{A}$ to H_0 and SignO oracles respectively, and $e \approx 2.71$. We assume that $2(q_S + q_H) < p$, where $p = p(\lambda)$ is the size of the groups output by $\mathcal{G}$.

Remark 4 (Proof of possession). Theorem 1 shows that our multisignature scheme SIG_1 is secure against rogue key attacks, where the adversary chooses its public keys adversarially. There are a number of techniques for preventing these attacks: message augmentation [BGLS03, BNN07], proof of possession [RY07], and linear combinations [BDN18]. In the scheme SIG_1, the public key pk_i that party i generates implicitly contains multiple proofs of possession of the secret key. This is used in a crucial way in the proof of Theorem 1 to obtain a solution to the co-BDH problem from the adversary's forged multisignature.

Proof of Theorem 1. We construct a co-BDH algorithm $\mathcal{B}$. This $\mathcal{B}$ takes as input an asymmetric group description $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$ along with $\big(g_{a,1} = g_1^a,\ g_{a,2} = g_2^a,\ g_{b,1} = g_1^b,\ g_{b,2} = g_2^b,\ g_{c,1} = g_1^c,\ g_{c,2} = g_2^c\big)$ from its challenger. It then invokes $\mathcal{A}$ and plays the role of challenger to $\mathcal{A}$ in the game $\mathbf{G}^{\text{uf}}_{\mathsf{SIG}_1}$ as follows:

1. Receive n from $\mathcal{A}$ and send $\mathsf{pp} \leftarrow (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p)$ to $\mathcal{A}$.
2. Receive $i^* \in [n]$ from $\mathcal{A}$.
3. Sample $\beta_1, \ldots, \beta_{i^*-1}, \beta_{i^*+1}, \ldots, \beta_n \leftarrow\!\!\$\ \mathbb{Z}_p$.
4. Set $\mathsf{pk}^* \leftarrow (g_{a,2},\ \{g_{a,1}^{\beta_j}\}_{j \in [n] \setminus \{i^*\}})$. Send pk^* to $\mathcal{A}$. This means that sk^* is equal to a, but this value is unknown to $\mathcal{B}$. We will show that this pk^* is valid once we describe how we program the random oracle H_1.

Next, $\mathcal{A}$ issues a sequence of random oracle and signing queries. We use q_S and q_H to denote an upper bound on the number of signing queries and random oracle queries on H_0 issued by $\mathcal{A}$ respectively. Note that the number of queries to H_1 is bounded by n, the size of the domain of H_1. $\mathcal{B}$ responds to these queries by maintaining the following data structures:

- $\mathcal{B}$ initializes the simulated oracles by setting $\mathsf{H}_1(j) \leftarrow \perp$ for all $j \in [n]$, and $\mathsf{H}_0(m) \leftarrow \perp$ for all $m \in \mathcal{M}$.
- $\mathcal{B}$ maintains two mappings $R : \mathcal{M} \to \mathbb{Z}_p$ and $R' : \mathcal{M} \to \{0,1\}$ to track auxiliary information used to answer H_0 queries. Both R and R' are initialized with $\perp$ for all messages.
- For signing queries, $\mathcal{B}$ stores a set $\mathcal{Q}_{\mathsf{sig}}$ to store all the messages m for which $\mathcal{A}$ queries $\mathsf{SignO}(m)$. This set is initialized with $\emptyset$.
- $\mathcal{B}$ samples a random message $\delta \leftarrow\!\!\$\ \mathcal{M}$.

We next explain how $\mathcal{B}$ responds to each of $\mathcal{A}$'s queries:

- A query for $\mathsf{H}_0(m)$. If $m = \delta$, then $\mathcal{B}$ aborts. Otherwise, if $\mathsf{H}_0(m)$ has been queried before, i.e. if $\mathsf{H}_0(m) \neq \perp$, then, $\mathcal{B}$ returns the value $\mathsf{H}_0(m)$. If not, $\mathcal{B}$ samples $x_m \leftarrow\!\!\$\ [q_S + 1]$ and $\gamma_m \leftarrow\!\!\$\ \mathbb{Z}_p$ uniformly at random. We use x_m as a biased coin that is equal to 1 with probability $1/(q_S+1)$.
  - If $x_m \neq 1$, then it sets $R(m) \leftarrow \gamma_m$, $R'(m) \leftarrow 0$ and $\mathsf{H}_0(m) \leftarrow g_1^{\gamma_m} \cdot g_{a,1}^{m-\delta}$.
  - If $x_m = 1$, it sets $R(m) \leftarrow \gamma_m$, $R'(m) \leftarrow 1$ and $\mathsf{H}_0(m) \leftarrow g_1^{\gamma_m}$.

  $\mathcal{B}$ returns $\mathsf{H}_0(m)$ to $\mathcal{A}$ and continues the game. Note that since γ_m is sampled randomly from $\mathbb{Z}_p$, the value $\mathsf{H}_0(m)$ is uniform in $\mathbb{G}_1$ as required.
- A query for $\mathsf{H}_1(j)$. If $j \neq i^*$, then $\mathcal{B}$ returns $g_1^{\beta_j}$. Otherwise, $\mathcal{B}$ returns $g_{b,1}$. Note that since all the β_j values are sampled randomly, $\mathsf{H}_1(j)$ will be indistinguishable from random for $\mathcal{A}$. Also, observe that the pk^*, as defined in Step 4 above, is indeed valid because for all $j \in [n] \setminus \{i^*\}$, $\mathsf{pk}^*_{i^*,j} = \mathsf{H}_1(j)^{\alpha_{i^*}}$, which is equal to $(g_1^{\beta_j})^{\alpha_{i^*}}$ and since $\mathsf{sk}^* = \alpha_{i^*} = a$, we get that $(g_1^{\beta_j})^{\alpha_{i^*}} = g_1^{a\beta_j} = g_{a,1}^{\beta_j}$.
- A query for $\mathsf{SignO}(m)$. $\mathcal{B}$ first queries $\mathsf{H}_0(m)$ and aborts if $R'(m) = 1$. At this point we can assume that $m \neq \delta$, since otherwise $\mathcal{B}$ would have aborted when querying for $\mathsf{H}_0(m)$. Next, $\mathcal{B}$ samples $r \leftarrow\!\!\$\ \mathbb{Z}_p$ and generates a signature as follows:

$$(\sigma_{m,0} \leftarrow g_2^r \cdot g_{b,2}^{\frac{-1}{m-\delta}},\ \ \sigma_{m,1} \leftarrow \mathsf{H}_0(m)^r \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}})$$

  Here, $R(m)$ is the value of γ_m that was sampled by $\mathcal{B}$ when responding to the $\mathsf{H}_0(m)$ query. Note that $R(m)$ cannot be $\perp$ since $\mathcal{B}$ queries $\mathsf{H}_0(m)$ during any signing query on m.
  We claim that this signature is distributed identically to a real signature generated by sk^*. To see why, set $\tilde{r} \leftarrow r - \frac{b}{m-\delta} \in \mathbb{Z}_p$. Then,

$$\begin{aligned}
\sigma_{m,0} &= g_2^r \cdot g_{b,2}^{\frac{-1}{m-\delta}} = g_2^r \cdot g_2^{\frac{-b}{m-\delta}} = g_2^{\tilde{r}} \\
\sigma_{m,1} &= \mathsf{H}_0(m)^r \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}} = \mathsf{H}_0(m)^{\tilde{r}} \cdot \mathsf{H}_0(m)^{\frac{b}{m-\delta}} \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}} \\
&= \mathsf{H}_0(m)^{\tilde{r}} \cdot (g_1^{\gamma_m} \cdot g_{a,1}^{m-\delta})^{\frac{b}{m-\delta}} \cdot g_{b,1}^{\frac{-\gamma_m}{m-\delta}} \\
&= \mathsf{H}_0(m)^{\tilde{r}} g_1^{\frac{b\gamma_m}{m-\delta}} g_1^{ab} \cdot g_1^{\frac{-b\gamma_m}{m-\delta}} = \mathsf{H}_0(m)^{\tilde{r}} \cdot g_1^{ab}
\end{aligned}$$

Next, observe that $\mathsf{sk}^* = a$, and since $\mathsf{H}_1(i^*) = g_{b,1}$, we get that $\mathsf{H}_1(i^*)^a = g_{b,1}^a = g_1^{ab}$. This means that, the above signature is of the form $(g_2^{\tilde{r}},\ \mathsf{H}_0(m^*)^{\tilde{r}} \cdot \mathsf{H}_1(i^*)^{\mathsf{sk}^*})$, and $\tilde{r}$ is uniform in $\mathbb{Z}_p$ as required. Hence, it is a valid response to the signing query. Lastly, $\mathcal{B}$ adds m to the list $\mathcal{Q}_{\mathsf{sig}}$.

Eventually, $\mathcal{A}$ outputs a list of public keys $(\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)$ and a forgery $(m^*, (\mathcal{J}^*, \sigma_0^*, \sigma_1^*))$. Our $\mathcal{B}$ examines $\mathcal{A}$'s output and aborts if $\mathsf{pk}^* \neq \mathsf{pk}_{i^*}^*$, or if $m^* \in \mathcal{Q}_{\mathsf{sig}}$, or if $i^* \notin \mathcal{J}^*$. Note that $\mathcal{A}$'s forgery is invalid if either of these conditions are true. Our $\mathcal{B}$ also aborts if $R'(m^*) = 0$.
Otherwise, $\mathcal{B}$ runs the key aggregation procedure to get

$$(\mathsf{pkc}^*, \mathsf{vk}^*) \leftarrow \mathsf{KeyAgg}((\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)).$$

It aborts if pkc^* or vk^* are $\perp$ or if $\mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 0$. Observe that $\mathcal{A}$'s forgery is invalid if either of these conditions are true. Lastly, $\mathcal{B}$ computes $\beta^* \leftarrow \Sigma_{k \in \mathcal{J}^* \setminus \{i^*\}} \beta_k$ and responds with:

$$W := \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{R(m^*)}, \sigma_0^*) \cdot \prod_{j \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{j,i^*}^*, g_{c,2}) \cdot e(g_{c,1}^{\beta^*}, \mathsf{vk}^*)} \in \mathbb{G}_T. \tag{2}$$

This completes the description of the co-BDH adversary $\mathcal{B}$.

We claim that if $\mathcal{B}$ did not abort, then the quantity W output by $\mathcal{B}$ is a correct response to the given co-BDH challenge, namely $W = e(g_1, g_2)^{abc}$. To see this, observe the following. First, if $\mathcal{A}$'s forgery is valid, namely $\mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 1$, then

$$\begin{aligned}
e(\sigma_1^*, g_2) &= e(\mathsf{H}_0(m^*), \sigma_0^*) \cdot e(\prod_{j \in \mathcal{J}^*} \mathsf{H}_1(j), \mathsf{vk}^*) \\
&= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1} \cdot \prod_{j \in \mathcal{J}^* \setminus \{i^*\}} g_1^{\beta_j}, \mathsf{vk}^*) \\
&= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1} \cdot g_1^{\beta^*}, g_{a,2} \cdot \prod_{j \in [n] \setminus \{i^*\}} \mathsf{pk}_{j,0}^*) \\
&= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1}, g_{a,2}) \cdot e(g_{b,1}, \prod_{j \in [n] \setminus \{i^*\}} \mathsf{pk}_{j,0}^*) \cdot e(g_1^{\beta^*}, \mathsf{vk}^*)
\end{aligned} \tag{3}$$

Next, since KeyAgg output a pkc^* this is not $\perp$, we know that all n public keys output by $\mathcal{A}$ are valid. Hence, for all $j \in [n] \setminus \{i^*\}$, we have that,

$$e(g_{b,1}, \mathsf{pk}_{j,0}^*) = e(\mathsf{H}_1(i^*), \mathsf{pk}_{j,0}^*) = e(\mathsf{pk}_{j,i^*}^*, g_2).$$

Using this relation in (3) gives:

$$e(\sigma_1^*, g_2) = e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1}, g_{a,2}) \cdot \prod_{j \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{j,i^*}^*, g_2) \cdot e(g_1^{\beta^*}, \mathsf{vk}^*).$$

Solving for $e(g_{b,1}, g_{a,2})$ we get

$$e(g_{b,1}, g_{a,2}) = \frac{e(\sigma_1^*, g_2)}{e(g_1^{R(m^*)}, \sigma_0^*) \cdot \prod_{j \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{j,i^*}^*, g_2) \cdot e(g_1^{\beta^*}, \mathsf{vk}^*)}$$

Raising both sides to the power c gives us:

$$e(g_1, g_2)^{abc} = e(g_{b,1}, g_{a,2})^c = \frac{e(\sigma_1^*, g_2)^c}{e(g_1^{R(m^*)}, \sigma_0^*)^c \cdot \prod_{j \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{j,i^*}^*, g_2)^c \cdot e(g_1^{\beta^*}, \mathsf{vk}^*)^c}$$

The right hand side is exactly the value W computed by $\mathcal{B}$ in (2). This proves that if $\mathcal{B}$ does not abort, then $\mathcal{B}$ responds correctly to the given co-BDH challenge.

We next bound the probability that $\mathcal{B}$ aborts. Let Abort be the event in which $\mathcal{B}$ aborts. We have that,

$$\mathsf{Adv}^{\text{co-bdh}}_{\mathcal{G},\mathcal{B}}(\lambda) = \Pr\left[\ \overline{\mathsf{Abort}}\ \right].$$

We next analyse the abort probability. Let $\mathsf{E}_{\mathcal{A}}$ be the event that $\mathcal{B}$ aborts because $\mathsf{pk}^* \neq \mathsf{pk}^*_{i^*}$ or $m^* \in \mathcal{Q}_{\mathsf{sig}}$ or $i^* \notin \mathcal{J}^*$ or $\mathsf{pkc}^* = \perp$ or $\mathsf{vk}^* = \perp$ or $\mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 0$. We have that, $\Pr[\overline{\mathsf{E}}_{\mathcal{A}}] \geq \mathsf{Adv}^{\text{uf}}_{\mathsf{SIG}_1,\mathcal{A}}(\lambda)$ because the adversary $\mathcal{A}$ can win only if none of these conditions hold. Let E_1 be the event that $\mathcal{B}$ aborts because $\mathcal{A}$ queries H_0 on δ. Let E_3 be the event that $\mathcal{B}$ aborts during a signing query because $R'(m) = 1$. Let E_4 be the event that $\mathcal{B}$ aborts because $R'(m^*) = 0$. Observe that $\Pr[\overline{\mathsf{E}}_1] \geq 1 - \frac{q_S + q_H}{p}$, since $\mathcal{B}$ implicitly queries $\mathsf{H}_0(m)$ whenever $\mathcal{A}$ queries $\mathsf{SignO}(m)$. Next, $\Pr[\overline{\mathsf{E}}_3 | \overline{\mathsf{E}}_1] = \left(1 - \frac{1}{q_S+1}\right)^{q_S} \geq \frac{1}{e}$, since $R'(m)$ is set to 1 with a probability $\frac{1}{q_S+1}$ for all $m \in \mathcal{M}$. Similarly, we have that $\Pr[\overline{\mathsf{E}}_4 | \overline{\mathsf{E}}_1] = \frac{1}{q_S+1}$. This gives us,

$$\begin{aligned}
\mathsf{Adv}^{\text{co-bdh}}_{\mathcal{G},\mathcal{B}}(\lambda) &= \Pr\left[\overline{\mathsf{Abort}}\right] \\
&= \Pr\left[\overline{\mathsf{E}}_1 \wedge \overline{\mathsf{E}}_3 \wedge \overline{\mathsf{E}}_4 \wedge \overline{\mathsf{E}}_{\mathcal{A}}\right] \\
&= \Pr\left[\overline{\mathsf{E}}_1\right] \cdot \Pr[\overline{\mathsf{E}}_3 | \overline{\mathsf{E}}_1] \cdot \Pr[\overline{\mathsf{E}}_4 | \overline{\mathsf{E}}_1] \cdot \Pr\left[\overline{\mathsf{E}}_{\mathcal{A}}\right] \\
&\geq \left(1 - \frac{q_S + q_H}{p}\right) \cdot \frac{1}{e} \cdot \frac{1}{q_S + 1} \cdot \mathsf{Adv}^{\text{uf}}_{\mathsf{SIG}_1,\mathcal{A}}(\lambda)
\end{aligned}$$

Lastly, since we assumed that $2(q_S + q_H) < p$, we have that,

$$\left(1 - \frac{(q_S + q_H)}{p}\right) \geq \frac{1}{2}$$

which proves the bound in (1) and completes the proof. □

Remark 5 (On the runtime of KeyAgg*).* The runtime of the key aggregation procedure KeyAgg is $O(n^2)$ since it verifies that the terms $\mathsf{pk}_{i,j}$ for all $i \neq j$ are valid. At first glance, it might seem that verifying only one term for each party would be sufficient. But one can show that there is an attack that allows the adversary to forge a signature and blame an honest party i^* if there is at least one party j for which the i^*th term pk_{j,i^*} is not verified in KeyAgg. The attack is similar to the rogue public key attack. This shows the necessity for KeyAgg to verify all the $O(n^2)$ terms contained in the n public keys given to it as input. However, the number of pairing computations needed can be significantly reduced: only two pairings are needed to verify each public key. For each provided public key pk_i, algorithm KeyAgg samples a random vector of length $n-1$ as $r \leftarrow\!\!\$\ [B]^{n-1}$ where $B \ll p$, and then combines all the checks for party i into a single equation requiring only *two* pairings:

$$e\left(\prod_{j \in [n] \setminus \{i\}} \mathsf{H}_1(j)^{r[j]},\ \mathsf{pk}_{i,0}\right) = e\left(\prod_{j \in [n] \setminus \{i\}} \mathsf{pk}_{i,j}^{r[j]},\ g_2\right).$$

This modification increases the advantage of the adversary by at most an additive factor of $O(n/B)$, which follows from [BGR98, Th 3.3]. As a result, adding a new user to the system requires only two pairing computations.

Remark 6 (A faster KeyAgg *procedure).* We can improve the running time of the KeyAgg procedure by dividing the parties into blocks of size $b \in [n]$, for some parameter b such as $b := \lceil \sqrt{n} \rceil$. We assign party i to block number $\lceil i/b \rceil$. Then, in $\mathsf{LocalKeyGen}(i)$, party i computes $\mathsf{pk}_{i,j} = \mathsf{H}_1(j)^{\alpha_i}$ only for parties $j \in [n] \setminus \{i\}$ that are in the same block as i. This means that pk_i now contains only b group elements (as opposed to n group elements in Figure 2). In KeyAgg, we now only need to verify these b group elements for each party, meaning that KeyAgg now runs in time $O(n \cdot b)$. Entry i in pkc is now computed by multiplying $\mathsf{pk}_{j,i}$ terms for all parties $j \in [n] \setminus \{i\}$ that are in the same block as i.

We obtain a faster KeyAgg procedure, but this comes at the cost of a larger verification key vk, which must now contain $\lceil n/b \rceil$ group elements (as opposed to a single group element in Figure 2). Specifically, for each block $k \in \lceil n/b \rceil$, let $\mathcal{B}_k = \{i : i \in [n] \wedge \lceil i/b \rceil = k\}$ be the set of all parties in this block. Then, the verification key will have an element $\mathsf{vk}_k = \prod_{i \in \mathcal{B}_k} \mathsf{pk}_{i,0}$ for each block $k \in \lceil n/b \rceil$.

The Sign and SigAgg procedures remain the same as in Figure 2. The Vf procedure accepts a signature (σ_0, σ_1) on a message m if

$$e(\sigma_1, g_2) = e(\mathsf{H}_0(m), \sigma_0) \cdot \prod_{k \in \lceil n/b \rceil} e\Big(\prod_{j \in \mathcal{B}_k \cap \mathcal{J}} \mathsf{H}_1(j),\ \mathsf{vk}_k \Big).$$

This means that Vf now needs to do upto $\min(|\mathcal{J}|, \lceil n/b \rceil) + 2$ pairings to verify a signature, with the exact number depending upon how many blocks have at least one party in common with the signer set $\mathcal{J}$. The proof of correctness and security for this scheme is similar to the proof of Theorem 1 using the same complexity assumption.

To summarize, the parameter b provides a tunable trade-off between the runtime of KeyAgg and the size of the verification key vk. For example, setting $b = \lceil \sqrt{n} \rceil$ gives an n-user KeyAgg with a runtime of $O(n^{1.5})$, as opposed to the $O(n^2)$ runtime in Figure 2, but at the cost of a verification key of $\lceil \sqrt{n} \rceil$ group elements.

Remark 7 (An Improved Multiverse Threshold Signature). Baird et al. in [BGJ+23] define the notion of a multiverse threshold signature scheme (MTS) where they refer to a set of signers as a universe, and multiple "universes" can co-exist, each having its own threshold, and containing a subset of all signers. Our signature scheme SIG_1 can be used as the per-universe threshold signature scheme to instantiate an accountable MTS. This scheme has the following properties: (i) the secret key of each signer is a single field element, and the signer needs to sign only once, independent of the number of universes it is part of, (ii) the per-universe aggregate signature would be two group elements and the signer set description, (iii) each party can generate its own key-pair, which will be re-usable across different universes, and lastly, (iv) we can support any access structure for each universe, since the signer set is included in the signature. The security analysis of [BGJ+23] extends to our multisignature scheme.

4 A Multisignature without Random Oracles

We next describe a scheme SIG_2 with similar efficiency as the scheme SIG_1 from the previous section, but without relying on random oracles. Recall that SIG_1 used two random oracles: H_0 and H_1. We

instantiate the H_0 random oracle with an algebraic hash, using an approach similar to Boneh and Boyen [BB04, BB11]. That is, we define $\mathsf{H}_0(m) = v_0^m h_0$ for random group elements $v_0, h_0 \in \mathbb{G}_1$ chosen at setup. We instantiate $\mathsf{H}_1(i)$ using elements provided by our complexity assumption, which we call the n-Bilinear Diffie Helman assumption (Definition 3). In essence, we set $\mathsf{H}_1(i)$ to be some precomputed generator of $\mathbb{G}_1$, for all $i \in [n]$.

To keep the verifier's memory to a constant we need to ensure that both the scheme's public parameters pp and verification key vk are constant size. In the previous section we used the random oracle $\mathsf{H}_1(\cdot)$ to map the integers $1, \ldots, n$ to *random* group generators $u_1, \ldots, u_n \in \mathbb{G}_1$. In the security proof we had to program the random oracle H_1 to make it possible to reduce security to the co-BDH assumption. In this section, we cannot do that. Instead, we make the bilinear group generator output the description of the bilinear groups $\mathbb{G}_1$ and $\mathbb{G}_2$ along with *fixed* generators $u_1, \ldots, u_n \in \mathbb{G}_1$. These generators can be computed as needed using a concrete hash function, say $u_i := \text{MapToGroup}(\text{SHA256}(i))$ for $i \in [n]$, and are not part of pp or vk. As a result, we now have a problem in the security proof: the reduction can no longer choose these generators during setup because they are fixed by the group generator. Consequently, we can no longer reduce security to the co-BDH assumption. Instead, we define a stronger assumption called the n-Bilinear Diffie-Helman assumption that makes it possible to prove security.

First, let us define two extended $\mathsf{GroupGen}(\cdot)$ procedures called $\mathsf{NGroupGen}$ and $\mathsf{FullNGroupGen}$:

- $\mathsf{NGroupGen}(1^\lambda, n) \to (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{u_i\}_{i\in[n]})$ outputs an asymmetric bilinear group along with n fixed generators $u_1, \ldots, u_n$ in $\mathbb{G}_1$. These fixed generators are not stored anywhere; they are computed as needed.
- $\mathsf{FullNGroupGen}(1^\lambda, n) \to (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{(u_i, u_{i,2})\}_{i\in[n]})$ outputs an asymmetric bilinear group along with n pairs of fixed generators $\{(u_i, u_{i,2}) \in \mathbb{G}_1 \times \mathbb{G}_2\}_{i\in[n]}$, where for all $i \in [n]$ it holds that $\text{dlog}_{g_1}(u_i) = \text{dlog}_{g_2}(u_{i,2})$. When invoked with the same inputs and random tape as $\mathsf{NGroupGen}(\cdot)$, it outputs the same n generators $u_1, \ldots, u_n$ in $\mathbb{G}_1$. This algorithm is only used in the security analysis (see also Remark 8).

In what follows, it is convenient to split the Setup procedure into two sub-procedures called $\mathsf{GlobalSetup}$ and $\mathsf{SystemSetup}$. The Global setup is a one-time setup that uses $\mathsf{NGroupGen}$ to output global public parameters pp_g which can be re-used across multiple signature systems. Algorithm $\mathsf{SystemSetup}$ runs after the bilinear group has been generated and outputs additional parameters that become an explicit part of the public parameters pp.

The multisignature scheme SIG_2 is described in Figure 3. The proof that the scheme has verification and tracing correctness is the same as the proof of correctness in the previous section.

Security. We now prove the unforgeability of our multisignature scheme SIG_2. As mentioned above, the proof is based on the hardness of the n-Bilinear Diffie-Helman assumption, defined below.

Definition 3. *Let $n = n(\lambda)$ be a parameter for some polynomial $n(\cdot)$. Let $\mathcal{G} := \mathsf{FullNGroupGen}$ be an extended asymmetric bilinear group generator that also outputs n pairs of fixed generators in both groups, as defined above. Define the following security game $\mathbf{G}^{\text{n-bdh}}$ with respect to an adversary $\mathcal{A}$:*

1. *Sample a bilinear group along with n pairs of fixed generators*

$$\left(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{(u_i, u_{i,2})\}_{i\in[n]}\right) \leftarrow\!\!\$\ \mathsf{FullNGroupGen}(1^\lambda, n).$$

The Multisignature scheme SIG_2 with local key generation, and no random oracles

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. $\mathsf{GlobalSetup}(1^\lambda, n)$: Sample a bilinear group along with n fixed generators $(\mathbb{G}_1,\ \mathbb{G}_2,\ \mathbb{G}_T,\ e,\ g_1,\ g_2,\ \{u_i\}_{i\in[n]} \in \mathbb{G}_1,\ p) \leftarrow\!\!\$\ \mathsf{NGroupGen}(1^\lambda, n)$.
   Output $\mathsf{pp}_g \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, \{u_i\}_{i\in[n]}, p)$.
2. $\mathsf{SystemSetup}(\mathsf{pp}_g)$: Sample $v_0, h_0 \leftarrow\!\!\$\ \mathbb{G}_1$. Output $\mathsf{pp} \leftarrow (\mathsf{pp}_g, v_0, h_0)$.
   // pp is constant size since the generators $\{u_i\}_{i\in[n]}$ are not stored, they are computed as needed.

$\underline{\mathsf{LocalKeyGen}(i)}$:

1. Sample $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p$. Set $\mathsf{sk}_i \leftarrow \alpha_i$.
2. For all $j \in [n] \setminus \{i\}$, compute $\mathsf{pk}_{i,j} \leftarrow u_j^{\alpha_i}$. Set $\mathsf{pk}_i \leftarrow (g_2^{\alpha_i}, \{\mathsf{pk}_{i,j}\}_{j\in[n]\setminus\{i\}})$.

$\underline{\mathsf{KeyAgg}(\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n))}$:

1. For each $i \in [n]$, parse pk_i as $(\mathsf{pk}_{i,0}, \{\mathsf{pk}_{i,j}\}_{j\in[n]\setminus\{i\}})$.
2. For any $i \in [n], j \in [n] \setminus \{i\}$, if $e(u_j, \mathsf{pk}_{i,0}) \neq e(\mathsf{pk}_{i,j}, g_2)$, output $\perp$.
3. Compute $\mathsf{pkc}_i \leftarrow \prod_{j\in[n]\setminus\{i\}} \mathsf{pk}_{j,i}$ for all $i \in [n]$.
4. Output $(\mathsf{pkc} \leftarrow \{\mathsf{pkc}_i\}_{i\in[n]},\quad \mathsf{vk} \leftarrow \prod_{i\in[n]} \mathsf{pk}_{i,0})$.
   // observe that pkc_i is $u_i^{\Sigma_{j\in[n]\setminus\{i\}}\alpha_j}$ and vk is $g_2^{\Sigma_{j\in[n]}\alpha_j}$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i,\ m \in \mathbb{Z}_p)}$:

1. Parse sk_i as α_i.
2. Sample $r_{i,m} \leftarrow\!\!\$\ \mathbb{Z}_p$. Compute $\sigma_{i,0} \leftarrow g_2^{r_{i,m}}$ and $\sigma_{i,1} \leftarrow u_i^{\alpha_i} \cdot (v_0^m h_0)^{r_{i,m}}$.
3. Output $\sigma_i = (\sigma_{i,0}, \sigma_{i,1})$.

$\underline{\mathsf{SigAgg}(\mathsf{pkc}, \{\sigma_j\}_{j\in\mathcal{J}})}$:

1. For each $j \in \mathcal{J}$, parse σ_j as $(\sigma_{j,0}, \sigma_{j,1})$. Parse pkc as $\{\mathsf{pkc}_i\}_{i\in[n]}$.
2. Compute $\sigma_0 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,0}$.
3. For each $j \in \mathcal{J}$, compute $\hat{\sigma}_{j,1} \leftarrow \sigma_{j,1} \cdot \mathsf{pkc}_j$.
4. Output $\sigma = (\sigma_0,\quad \sigma_1 \leftarrow \prod_{j\in\mathcal{J}} \hat{\sigma}_{j,1},\quad \mathcal{J})$.
   // observe that $\hat{\sigma}_{j,1}$ is equal to $u_j^{\Sigma_{i\in[n]}\alpha_i} \cdot (v_0^m h_0)^{r_{j,m}}$.

$\underline{\mathsf{Vf}(\mathsf{vk}, m, \sigma)}$:

1. Parse σ as $(\sigma_0, \sigma_1, \mathcal{J})$.
2. Output 1 if $e(\sigma_1, g_2) = e(v_0^m h_0, \sigma_0) \cdot e(\prod_{j\in\mathcal{J}} u_j, \mathsf{vk})$.

$\underline{\mathsf{Trace}(m, \sigma)}$:

1. Parse σ as $(\sigma_0, \sigma_1, \mathcal{J})$ and output $\mathcal{J} \subseteq [n]$.

Fig. 3. The Multisignature scheme SIG_2 with no random oracles

2. *Send the group description and the generators* $u_1, \ldots, u_n$ *in* $\mathbb{G}_1$ *to the adversary, that is, send to* $\mathcal{A}$ *the tuple*

$$\Big(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{u_i\}_{i\in[n]}\Big).$$

The adversary responds with an index $j^* \in [n]$*. This lets the adversary choose the generator in the list* $u_1, \ldots, u_n \in \mathbb{G}_1$ *that it wants to attack.*

3. *Sample* $a, c \leftarrow\!\!\$\ \mathbb{Z}_p$ *and send to* $\mathcal{A}$ *the n-BDH challenge*

$$\big(\{u_i^a, u_i^c\}_{i\in[n]\setminus\{j^*\}},\ u_{j^*,2},\ g_1^a,\ g_2^a,\ g_1^c,\ g_2^c\big). \tag{4}$$

The adversary responds with a candidate BDH solution $W \in \mathbb{G}_T$*.*

4. *Output 1 if* $W = e(u_{j^*}, g_2)^{ac}$ *and 0 otherwise.*

We say that n-BDH is hard if, for all PPT *algorithms* $\mathcal{A}$*, the following function is negligible in* λ*:*

$$\mathsf{Adv}^{\text{n-bdh}}_{\mathcal{G},n,\mathcal{A}}(\lambda) := \Pr\big[\mathbf{G}^{\text{n-bdh}}_{\mathcal{G},n,\mathcal{A}}(\lambda) = 1\big]$$

where the probability is taken over the random bits of FullNGroupGen*, the random choice of exponents* $a, c \in \mathbb{Z}_p$*, and the random bits used by* $\mathcal{A}$*.*

Our instantiation of H_0 using the Boneh-Boyen algebraic hash $\mathsf{H}_0(m) = v_0^m h_0$ lets us prove selective unforgeability (rather than existential unforgeability) of the resulting scheme. As such, we modify the unforgeability game in Fig. 1 into a selective game. We modify Lines 1 to 3 as follows:

- the adversary first outputs n, the total number of parties (as in Fig. 1),
- the environment then runs the global setup $\mathsf{pp}_g \leftarrow\!\!\$\ \mathsf{GlobalSetup}(1^\lambda, n)$ and send pp_g to the adversary,
- the adversary responds with an index $i^* \in [n]$ and the target message m^* for which it will forge a signature,
- the environment runs $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{SystemSetup}(\mathsf{pp}_g)$ and sends pp to the adversary.

The rest of the game remains unchanged. We refer to this modified game as $\mathbf{G}^{\text{sa-uf}}_{\mathsf{MS}}$, for selective unforgeability. The advantage function of an adversary $\mathcal{A}$ in this game is defined as

$$\mathsf{Adv}^{\text{sa-uf}}_{\mathsf{MS},n_{\max},\mathcal{A}}(\lambda) := \Pr\left[\mathbf{G}^{\text{sa-uf}}_{\mathsf{MS},n_{\max},\mathcal{A}}(\lambda) = 1\right].$$

This advantage is parameterized by a polynomial $n_{\max} = n_{\max}(\lambda)$ that denotes an upper bound on the number of parties n that the adversary sends in Step 1 of the game. We say that a multisignature scheme is *selectively secure* if this advantage is negligible in λ for all PPT adversaries $\mathcal{A}$. The scheme SIG_2 can be extended to be existentially unforgeable by instantiating the hash function $\mathsf{H}_0(m)$ using the Waters algebraic hash [Wat05].

Theorem 2 below proves the selective security of SIG_2 using n-Bilinear Diffie-Helman assumption. The proof is presented in Appendix C.1.

Theorem 2. *Let* $\mathcal{G}$ *be an extended asymmetric bilinear group generator. Then, for every adversary* $\mathcal{A}$*, there exists an adversary* $\mathcal{B}$ *with about the same runtime as* $\mathcal{A}$*, such that,*

$$\mathsf{Adv}^{\text{sa-uf}}_{\mathsf{SIG}_2,n_{\max},\mathcal{A}}(\lambda) \le \mathsf{Adv}^{\text{n-bdh}}_{\mathcal{G},n_{\max},\mathcal{B}}(\lambda)$$

where $n_{\max} = n_{\max}(\lambda)$ *is a bound on the number of parties that* $\mathcal{A}$ *outputs in Step 1 of the game* $\mathbf{G}^{\text{sa-uf}}_{\mathsf{MS}}$*.*

Remark 8 (on the n-BDH assumption). The n-BDH assumption is much simpler to state using a symmetric pairing where $\mathbb{G}_1 = \mathbb{G}_2$ and $g_1 = g_2$. In particular, for a symmetric pairing, the n-BDH challenge in (4) contains half as many terms, and the fixed generator pairs $(u_i, u_{i,2}) \in \mathbb{G}_1 \times \mathbb{G}_2$ become a single fixed generator by setting $u_{i,2} = u_i$. This is conceptually important because in the asymmetric settings used in practice, it is not known how to efficiently generate these fixed pairs due to the requirement that $\mathrm{dlog}_{g_1}(u_i) = \mathrm{dlog}_{g_2}(u_{i,2})$. When using a symmetric pairing we simply set $u_{i,2} = u_i$. Consequently, in symmetric setting the assumption is refutable in sense of [Nao03], while in the asymmetric settings it is not refutable. Nevertheless, we chose to present our construction in the asymmetric settings because that is how pairings are most commonly used in practice.

5 A Multisignature with a Short Verifier Key and an Efficient DKG

The multisignature schemes in the previous two sections have a short verification key, but require a linear size combiner key pkc. In this section, we present a new multisignature scheme with a short verification key and an empty pkc. The cost of eliminating the pkc is a trusted setup for key generation, or alternatively, a simple distributed key generation protocol (DKG) to generate the party's secret keys. To simplify the presentation, we present our schemes using a trusted setup, but the trusted setup can be converted into a simple DKG using standard techniques.

5.1 Multisignatures with Trusted Setup

We first define the syntax and security for a multisignature scheme with a trusted setup.

Syntax. A multisignature with a trusted setup is defined as in Section 2.1, but instead of the $\mathsf{LocalKeyGen}(\cdot)$ and $\mathsf{KeyAgg}(\cdot)$ procedures, we have a single probabilistic algorithm KeyGen that is executed by a trusted party as $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$. The algorithm takes as input the public parameters (that encode the number of parties n), and outputs keys for all the parties. The scheme must satisfy *verification correctness* and *tracing correctness* properties, similar to those defined in Section 2.1. The only difference is that now, KeyGen is called to generate the keys for all n parties (instead of running $\mathsf{LocalKeyGen}(i)$ for all parties $i \in [n]$).

Security. To define unforgeability we consider an adversary that can adaptively corrupt up to $n-1$ parties, and can query for signature shares from any party on any message. The scheme is unforgeable if such an adversary cannot produce a valid signature on a message m on behalf of a set $\mathcal{J} \subseteq [n]$ that includes some non-corrupt party, without requesting its signature on m.

Figure 4 defines the security game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS}}$ that captures unforgeability for a multisignature scheme with a trusted setup. The adversary first specifies the number of signers n, which is followed by challenger running Setup and sampling keys for all n parties using the KeyGen procedure. The adversary then interacts with the challenger using two types of queries: Secret-key queries and signature queries. A secret-key query for signer i reveals to the adversary the secret key of signer i. A signature query for (m, i) provides the adversary with an honestly-generated signature share on m with respect to signer i's secret key. Finally, the adversary should produce a valid forgery; that is, a valid pair (m^*, σ^*) that traces to some party $i \in [n]$ such that the adversary did not request the secret key for i and did not ask for a signature on m^* from i.

Definition 4. *A multisignature scheme with trusted setup* TMS *is secure if for all* PPT *adversaries* $\mathcal{A}$ *the following function is negligible:*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{TMS},\mathcal{A}}(\lambda) := \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS},\mathcal{A}}(\lambda) = 1\right].$$

Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS}}$ with respect to an adversary $\mathcal{A}$ and security parameter λ

$(\mathsf{st}, n) \leftarrow \mathcal{A}(\lambda)$
$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, n)$
$\mathcal{Q}^{\mathsf{sk}} \leftarrow \emptyset\ ;\ \ \forall\ m \in \mathcal{M}, \mathcal{Q}^{\mathsf{sig}}(m) \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow\!\!\$\ \mathsf{KeyGen}()$ // pp is an implicit argument to KeyGen
$(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot), \mathsf{SignO}(\cdot,\cdot)}(\mathsf{st}, \mathsf{pp}, \mathsf{pk}, \mathsf{pkc}, \mathsf{vk})$
if $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ **then**
  return 0
$\mathcal{J} \leftarrow \mathsf{Trace}(m^*, \sigma^*) \subseteq [n]$
if $\mathcal{J} \not\subseteq \mathcal{Q}^{\mathsf{sk}} \cup \mathcal{Q}^{\mathsf{sig}}(m^*)$ **then** // the forgery traces to a set containing an honest party
  return 1

Oracle $\mathsf{skO}(i)$
$\mathcal{Q}^{\mathsf{sk}} \leftarrow \mathcal{Q}^{\mathsf{sk}} \cup \{i\}$
return sk_i

Oracle $\mathsf{SignO}(m, i)$
$\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, m)$
$\mathcal{Q}^{\mathsf{sig}}(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}(m) \cup \{i\}$
return σ_i

Fig. 4. The security game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS}}$ for a Multisignature scheme with trusted setup $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace})$.

5.2 The SIG_3 Multisignature

We next present our multisignature scheme with trusted setup and a constant size vk. The scheme is described in Figure 5. The scheme uses two hash functions $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$, where $\mathcal{M}$ is the message space, and $\mathsf{H}_1 : [n] \to \mathbb{G}_1$, where n is an upper bound on the number of signers. We will model these hash functions as random oracles in the proof of security.

Correctness. An honestly generated signature $(\sigma_0, \sigma_1, \mathcal{J})$ verifies correctly because

$$
\begin{aligned}
e(\sigma_1, g_2) &= e\Big(\prod_{j\in\mathcal{J}} (\mathsf{H}_1(j)^\alpha \cdot \mathsf{H}_0(m)^{r_j})\ ,\ g_2\Big) \\
&= e\left(\Big(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j)\Big)^\alpha \cdot \mathsf{H}_0(m)^{\Sigma_{j\in\mathcal{J}} r_j}, g_2\right) = e\Big(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j), g_2^\alpha\Big) \cdot e\left(\mathsf{H}_0(m), g_2^{\Sigma_{j\in\mathcal{J}} r_j}\right) \\
&= e\Big(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j),\ h\Big) \cdot e(\mathsf{H}_0(m),\ \sigma_0).
\end{aligned}
$$

Security. Theorem 3 below reduces the security of SIG_3 to the hardness of the co-Bilinear Diffie-Hellman problem from Definition 2. The proof models H_0 and H_1 as random oracles, and is provided in Appendix C.2.

Theorem 3. *Let $\mathcal{G}$ be an asymmetric bilinear group generator. Then, for every adversary $\mathcal{A}$ there exists another adversary $\mathcal{B}$ with roughly the same runtime as $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{SIG}_3,\mathcal{A}}(\lambda) \le 2e \cdot n_{\max} \cdot (q_S + 1) \cdot \mathsf{Adv}^{\text{co-bdh}}_{\mathcal{G},\mathcal{B}}(\lambda)$$

The Multisignature SIG_3 with a Trusted setup

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. Sample a bilinear group $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p) \leftarrow\!\$\ \mathsf{GroupGen}(1^\lambda)$.
2. Output $\mathsf{pp} \leftarrow (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p)$.
3. The system uses two hash functions: $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}_1$ and $\mathsf{H}_1 : [n] \to \mathbb{G}_1$.

$\underline{\mathsf{KeyGen}()}$: // implemented as a distributed key generation protocol

1. Sample $\alpha \leftarrow\!\$\ \mathbb{Z}_p$. Set $h \leftarrow g_2^\alpha$ and $\mathsf{sk}_i \leftarrow \mathsf{H}_1(i)^\alpha$ for all $i \in [n]$.
2. Output $\big(\mathsf{pk} = h,\ \mathsf{pkc} = \perp,\ \mathsf{vk} = h,\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\big)$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i, m)}$:

1. Sample $r_i \leftarrow\!\$\ \mathbb{Z}_p$.
2. Compute $\sigma_{i,0} \leftarrow g_2^{r_i} \in \mathbb{G}_2$ and $\sigma_{i,1} \leftarrow \mathsf{sk}_i \cdot \mathsf{H}_0(m)^{r_i} \in \mathbb{G}_1$.
3. Output $\sigma_i = (\sigma_{i,0}, \sigma_{i,1})$.

$\underline{\mathsf{SigAgg}(\mathsf{pkc}, (\sigma_{i_1}, \ldots, \sigma_{i_{|\mathcal{J}|}}))}$:

1. For $j \in \mathcal{J}$: Parse σ_j as $(\sigma_{j,0}, \sigma_{j,1})$.
2. Compute $\sigma_0 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,0} \in \mathbb{G}_2$ and $\sigma_1 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,1} \in \mathbb{G}_1$.
3. Output $\sigma = (\sigma_0, \sigma_1, \mathcal{J})$.

   // observe that σ_1 is equal to $\Big(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j)\Big)^\alpha \cdot \mathsf{H}_0(m)^{\sum_{j\in\mathcal{J}} r_{j,m}}$.

$\underline{\mathsf{Vf}(\mathsf{vk}, m, \sigma)}$:

1. Parse vk as (h) and σ as $(\sigma_0, \sigma_1, \mathcal{J})$.
2. Output 1 if $e(\sigma_1, g_2) = e(\mathsf{H}_0(m), \sigma_0) \cdot e(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j), h)$.

$\underline{\mathsf{Trace}(m, \sigma)}$:

1. Parse σ as $(\sigma_0, \sigma_1, \mathcal{J})$ and output $\mathcal{J} \subseteq [n]$.

Fig. 5. The Multisignature scheme SIG_3 with constant size public keys and signatures and a trusted setup

where $n_{\max} = n_{\max}(\lambda)$ is a bound on the number of signers, $q_H = q_H(\lambda)$, $q_S = q_S(\lambda)$ are a bound on the number of queries issued by $\mathcal{A}$ to H_0 and SignO respectively, and $e \approx 2.71$. We assume that $2(q_S + q_H) < p$, where $p = p(\lambda)$ is the size of the groups output by $\mathcal{G}$.

Remark 9 (Proactive Refresh). [BPR22] presented a generic construction for an accountable threshold signature scheme with proactive refresh, called PRATS, that uses two building blocks: a threshold signature scheme with proactive refresh, and an accountable threshold scheme. Since SIG_3 has a short public key and an efficient DKG protocol, we can instantiate PRATS with SIG_3 to get a concretely efficient threshold signature scheme that is both accountable and proactively refreshable. This is the first such pairing-based scheme that satisfies the strongest security properties from [BPR22]. In Appendix B we further optimize this construction.

Distributed Key Generation. Although we described the multisignature scheme SIG_3 using a trusted setup, the public and secret keys of SIG_3 can be generated via a simple distributed key generation protocol, along the lines discussed in other works, such as [GJKR99, GJKR07, CS04,

Gro21] to name a few. As a simple illustration, when the parties are semi-honest, and are connected to each other via a private authenticated channel, the following simple protocol is sufficient:

1. Each signer $i \in [n]$ samples a uniformly random element $\alpha_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes $Y_{i,i} \leftarrow \mathsf{H}_1(i)^{\alpha_i}$.
2. For each $j \in [n] \setminus \{i\}$, signer i sends $Y_{i,j} \leftarrow \mathsf{H}_1(j)^{\alpha_i}$ to signer j. It also sends $X_i \leftarrow g_2^{\alpha_i}$ to all other signers in $[n] \setminus \{i\}$.
3. Upon receiving $Y_{1,i}, \ldots, Y_{n,i}$ and $X_1, \ldots, X_n$ from all other signers, each signer sets the public key to be $\mathsf{pk} \leftarrow \prod_{j\in[n]} X_j$, and its secret key as $\mathsf{sk}_i \leftarrow \prod_{j\in[n]} Y_{j,i}$.

Observe that if we write $\alpha := \Sigma_{j\in[n]}\alpha_j$, then $\mathsf{pk} = g_2^{\alpha}$, and $\mathsf{sk}_i = \mathsf{H}_1(i)^{\alpha}$, as required. For a DKG among malicious parties one would need to use one of the protocols cited above.

6 A lattice based Multisignature with short public keys

In this section, we extend our techniques from Section 5.2 to get a lattice-based multisignature with a short public key and a two-round interactive signing protocol. We start with preliminaries.

6.1 Preliminaries

Polynomial Rings. As standard, we identify $\mathbb{Z}_q$ for a prime q with the set $(-q/2, \ldots, q/2]$, and we define the absolute value of an element $x \in \mathbb{Z}_q$ as $|x| = \{\min |y| : y \in \mathbb{Z}, y = x \pmod q\}$. For $N, q \in \mathbb{N}$, we define $R = \mathbb{Z}[X]/f(X)$ and $R_q = \mathbb{Z}_q[X]/f(X)$, where q is a prime modulus and N is a power of two defining the degree of $f(X)$. Specifically, $f(X) = X^N + 1$ is the $2N$th cyclotomic polynomial. We define the norm of elements in these rings to be the norm of their coefficient vector in $\mathbb{Z}^N$, which is also called the coefficient embedding.

Following [DOTT21], we define a key set $S_\eta \subseteq R$ parameterized by $\eta \geq 0$ consisting of small polynomials:

$$S_\eta = \{x \in R : ||x||_\infty \leq \eta\}$$

We define a challenge set $C \subseteq R$ parameterized by κ, consisting of small and sparse polynomials:

$$C = \{x \in R : ||x||_1 = \kappa \wedge ||x||_\infty = 1\}$$

The discrete Gaussian distribution over R^m is defined as follows.

Definition 5. *For $\boldsymbol{x} \in R^m$, $L \subseteq R^m$, let $\rho_{\boldsymbol{v},s}(\boldsymbol{x}) = exp(-\pi||\boldsymbol{x} - \boldsymbol{v}||_2^2/s^2)$ be a Gaussian function of parameters $\boldsymbol{v} \in R^m$ and $s \in \mathbb{R}$. The discrete Gaussian distribution $D^m_{L,\boldsymbol{v},s}$ centered at $\boldsymbol{v}$ is*

$$D^m_{L,\boldsymbol{v},s}(\boldsymbol{x}) := \rho_{\boldsymbol{v},s}(\boldsymbol{x})/\rho_{\boldsymbol{v},s}(L)$$

where $\rho_{\boldsymbol{v},s}(L) = \Sigma_{\boldsymbol{y}\in L}\rho_{\boldsymbol{v},s}(\boldsymbol{y})$.

We omit the subscript $\boldsymbol{v}$ if $\boldsymbol{v} = \boldsymbol{0}$, and omit L if $L = R^m$. We assume s exceeds the smoothing parameter (as defined in [DOTT21]), and hence D^m_s behaves like a continuous Gaussian with standard deviation $\sigma = s/\sqrt{2\pi}$.

We now define ring lattices, namely discrete subgroups of R^m.

Definition 6. *For a prime q, $\boldsymbol{A} \in R_q^{k\times\ell}$ and $\boldsymbol{u} \in R_q^k$, define:*

$$\Lambda_q^{\perp}(\boldsymbol{A}) = \{\boldsymbol{e} \in R^\ell : \boldsymbol{A}\boldsymbol{e} = \boldsymbol{0} \ mod\ q\}$$

$$\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}) = \{\boldsymbol{e} \in R^\ell : \boldsymbol{A}\boldsymbol{e} = \boldsymbol{u} \ mod\ q\}$$

Lattice-based Assumptions. We define two standard lattice problems over rings: Module Short Integer Solution (MSIS) and Module Learning With Errors (MLWE).

Definition 7 ($\mathsf{MSIS}_{q,k,\ell,\beta}$)**.** *For prime $q = q(\lambda)$, $k = k(\lambda), \ell = \ell(\lambda) \in \mathbb{N}$, $\beta \in \mathbb{R}$, the $\mathsf{MSIS}_{q,k,\ell,\beta}$ problem is said to be hard if for all* PPT *adversaries, the following function is negligible in λ:*

$$\mathsf{Adv}_{\mathsf{MSIS}_{q,k,\ell,\beta},\mathcal{A}}(\lambda) = \Pr\left[[\boldsymbol{A}|\boldsymbol{I}] \cdot \boldsymbol{x} = \boldsymbol{0} \wedge ||\boldsymbol{x}||_2 \leq \beta : \begin{array}{c} \boldsymbol{A} \leftarrow\$ R_q^{k\times\ell} \\ \boldsymbol{x} \leftarrow\$ \mathcal{A}(\boldsymbol{A}) \end{array}\right]$$

Definition 8 ($\mathsf{MLWE}_{q,k,\ell,\eta}$)**.** *For prime $q = q(\lambda)$, $k = k(\lambda), \ell = \ell(\lambda) \in \mathbb{N}$, $\eta \in \mathbb{R}$, the $\mathsf{MLWE}_{q,k,\ell,\eta}$ problem is said to be hard if for all* PPT *adversaries $\mathcal{A}$, the following function is negligible in λ:*

$$\mathsf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta},\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{t}) = 1] - \Pr[\mathcal{A}(\boldsymbol{A}, [\boldsymbol{A}|\boldsymbol{I}] \cdot \boldsymbol{s}) = 1]|$$

where $\boldsymbol{A} \leftarrow\$ R_q^{k\times\ell}$, $\boldsymbol{s} \leftarrow\$ S_\eta^\ell \times S_\eta^k$, $\boldsymbol{t} \leftarrow\$ R_q^k$.

Note that, in the MLWE definition, the latter k elements of $\boldsymbol{s}$ correspond to the error term of MLWE, sampled from S_η.

Lattice-based Trapdoors. Past works [ABB10, GM18, GPV08] have shown how to sample an essentially uniform matrix $\boldsymbol{A} \in R_q^{k\times\ell}$ along with a trapdoor $\mathcal{T}_A$. This trapdoor can be used to sample $\boldsymbol{x} \in \Lambda_q^{\boldsymbol{u}}(\boldsymbol{A})$ drawn from a distribution statistically close to the Gaussian distribution over the lattice, i.e. $D^\ell_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}),\sigma}$ for σ more than a certain threshold. We now formally define these procedures:

Lemma 1. *Let q be a prime, and $k, \ell \in \mathbb{N}$. Then,*

- *There is a probabilistic polynomial time algorithm* $\mathsf{TrapGen}(q, k, \ell)$ *that outputs a pair $(\boldsymbol{A} \in R_q^{k\times\ell}, \mathcal{T}_A)$ such that $\boldsymbol{A}$ is statistically close to a uniform matrix in $R_q^{k\times\ell}$, and $\mathcal{T}_A$ is a trapdoor for $\boldsymbol{A}$, which can be used in the following procedure.*
- *There is a probabilistic polynomial time algorithm* $\mathsf{SampleGaussian}(\boldsymbol{A}, \boldsymbol{B}, \mathcal{T}_A, \boldsymbol{u}, s)$ *that takes as input a matrix and its trapdoor $(\boldsymbol{A} \in R_q^{k\times\ell}, \mathcal{T}_A)$ along with $\boldsymbol{u} \in R^k$, $s \in \mathbb{R}, \boldsymbol{B} \in R^{k\times m}$ and outputs $\boldsymbol{x} \in \Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}|\boldsymbol{B})$ sampled from a distribution statistically close to $D^{\ell+m}_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}|\boldsymbol{B}),s}$.*
- *For any $\boldsymbol{u} \in R^k$, $\boldsymbol{A} \in R_q^{k\times\ell}, \boldsymbol{B} \in R^{k\times m}$, $s \in \mathbb{R}$,*

$$\Pr[x \sim D^{\ell+m}_{\Lambda_q^{\boldsymbol{u}}(\boldsymbol{A}|\boldsymbol{B}),s} : ||x|| > s\sqrt{(\ell+m)}] < negl(k)$$

Rejection Sampling. We now state the rejection sampling algorithm which is used as a subroutine in our signature scheme.

Lemma 2. *(Theorem 4.6 of [Lyu12]) For $m \in \mathbb{N}$, $R = \mathbb{Z}[X]/f(X)$ with N being the degree of $f(X)$, let V be a subset of R^m, in which all elements have norm less than T. Let $s \in \mathbb{R}$ such that $s = \omega(T\sqrt{\log(m)})$, and $h : V \to \mathbb{R}$ be a probability distribution. Then, for constants M, α, t where $M = e^{t/\alpha+1/(2\alpha^2)}$, $s = \alpha T\sqrt{2\pi}$ and $t = \omega(\sqrt{\log(mN)})$, we have that the distribution of the following algorithm $\mathcal{A}$:*

1. *Sample $\boldsymbol{v} \leftarrow\$ h$, $\boldsymbol{z} \leftarrow\$ D^m_{\boldsymbol{v},s}$.*
2. *Output $(\boldsymbol{z}, \boldsymbol{v})$ with probability $p = \min\left(1, \frac{D^m_s(\boldsymbol{z})}{MD^m_{\boldsymbol{v},s}(\boldsymbol{z})}\right)$ (and output $\perp$ with probability $1 - p$).*

is within statistical distance $\frac{2^{-\omega(\log(m))}}{M}$ of the distribution of the following algorithm $\mathcal{F}$:

1. *Sample* $\boldsymbol{v} \leftarrow\$ h$, $\boldsymbol{z} \leftarrow\$ D_s^m$.
2. *Output* $(\boldsymbol{z}, \boldsymbol{v})$ *with probability* $1/M$ *(and output* $\perp$ *with probability* $1 - 1/M$*).*

Moreover, the probability that $\mathcal{A}$ *does not output* $\perp$ *is at least* $\frac{1-2^{-\omega(\log(m))}}{M}$.

Trapdoor Commitment Scheme. A trapdoor commitment scheme is a tuple of PPT algorithms $\mathsf{Com} = (\mathsf{Setup}, \mathsf{CGen}, \mathsf{Commit}, \mathsf{Open}, \mathsf{TCGen}, \mathsf{TCommit}, \mathsf{Eqv})$, where:

- $\mathsf{Setup}(1^\lambda) \rightarrow cpp$ outputs public parameters containing S_{ck}, S_m, S_r, S_{com}, S_{td} which define the space of commitment keys, messages, randomness, the commitments and the trapdoors respectively. It also outputs $D(S_r)$, a distribution over S_r.
- $\mathsf{CGen}(cpp) \rightarrow ck$ outputs a commitment key $ck \in S_{ck}$.
- $\mathsf{Commit}(ck, m; r) \rightarrow com$, takes a commitment key $ck \in S_{ck}$, message $m \in S_m$ and randomness $r \in S_r$ as input and outputs a commitment com to m.
- $\mathsf{Open}(ck, com, m, r) \rightarrow \{0, 1\}$ outputs 1 if com is a valid commitment to m with opening r, with respect to ck.
- $\mathsf{TCGen}(cpp) \rightarrow (tck, td)$ is the trapdoor key generation algorithm, and outputs a commitment key $tck \in S_{ck}$ along with a trapdoor $td \in S_{td}$.
- $\mathsf{TCommit}(tck, td) \rightarrow com$, the trapdoor commitment algorithm outputs a commitment com.
- $\mathsf{Eqv}(tck, td, com, m) \rightarrow r$ is the equivocation algorithm that outputs $r \in S_r$.

A secure trapdoor commitment scheme must satisfy *correctness*, *hiding*, *binding* and *secure trapdoor*, which we define below:

Correctness. A commitment scheme Com is correct if, for all messages $m \in S_m$,

$$\Pr\left[\mathsf{Open}(ck, com, m, r) = 1 : \begin{array}{c} cpp \leftarrow\$ \mathsf{Setup}(1^\lambda); ck \leftarrow\$ \mathsf{CGen}(cpp) \\ r \leftarrow\$ D(S_r); com \leftarrow \mathsf{Commit}(ck, m; r) \end{array}\right] = 1$$

Hiding. A commitment scheme Com is perfectly (or computationally) hiding if the following probability is negligible in λ for a probabilistic (or PPT) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$:

$$\epsilon_h := \left| \Pr\left[b = b' : \begin{array}{c} cpp \leftarrow\$ \mathsf{Setup}(1^\lambda); ck \leftarrow\$ \mathsf{CGen}(cpp) \\ (m_0, m_1) \leftarrow\$ \mathcal{A}_1(cpp, ck) \\ b \leftarrow\$ \{0, 1\}; com \leftarrow\$ \mathsf{Commit}(ck, m_b) \\ b' \leftarrow\$ \mathcal{A}_2(com) \end{array}\right] - \frac{1}{2} \right|$$

Binding. A commitment scheme Com is computationally binding if the following probability is negligible in λ for any probabilistic polynomial time algorithm $\mathcal{A}$:

$$\epsilon_{bind} := \Pr\left[\begin{array}{c} m \neq m' \\ \mathsf{Open}(ck, com, m, r) = 1 \\ \mathsf{Open}(ck, com, m', r') = 1 \end{array} : \begin{array}{c} cpp \leftarrow\$ \mathsf{Setup}(1^\lambda) \\ ck \leftarrow\$ \mathsf{CGen}(cpp) \\ (com, m, r, m', r') \leftarrow\$ \mathcal{A}(cpp, ck) \end{array}\right]$$

Secure Trapdoor. Com has a secure trapdoor if for any message $m \in S_m$, the statistical distance ϵ_{td} between (ck, m, r, com) and (tck, m, r', com') is negligible in λ where $cpp \leftarrow\$ \mathsf{Setup}(1^\lambda)$, $ck \leftarrow\$ \mathsf{CGen}(cpp)$, $r \leftarrow\$ D(S_r)$, $com \leftarrow \mathsf{Commit}(ck, m, r)$ and $(tck, td) \leftarrow\$ \mathsf{TCGen}(cpp)$, $com' \leftarrow\$ \mathsf{TCommit}(tck, td)$ and $r' \leftarrow \mathsf{Eqv}(tck, td, com', m)$.

Our protocol also requires that the commitment scheme be additively homomorphic and uniform key. We now define these properties:

Uniform Key. A commitment scheme Com is uniform key if the output of $\mathsf{CGen}(cpp)$ follows the uniform distribution over S_{ck}.

Additively Homomorphic. A commitment scheme Com is additively homomorphic if for all $m, m' \in S_m$,

$$\Pr\left[\mathsf{Open}(ck, com + com', m + m', r + r') = 1 : \begin{array}{c} cpp \leftarrow\$ \mathsf{Setup}(1^\lambda) \\ ck \leftarrow\$ \mathsf{CGen}(cpp) \\ r, r' \leftarrow\$ D(S_r) \\ com \leftarrow \mathsf{Commit}(ck, m, r) \\ com' \leftarrow \mathsf{Commit}(ck, m', r') \end{array}\right] = 1$$

Any lattice-based trapdoor commitment scheme that satisfies all of the above properties can be used to instantiate our scheme. The scheme in Fig.15 in [DOTT21] is one such example.

6.2 A lattice based Multisignature with short public keys

We now extend our techniques from Sections 5 to build a lattice-based Multisignature scheme which has a short public key. Our starting point is a two-round multisignature scheme MS_2 from [DOTT21]. This scheme uses Fiat Shamir with aborts and rejection sampling. It relies on the module LWE and SIS assumptions, and an additively homomorphic trapdoor commitment scheme Com which is statistically hiding and computationally binding. The message space is defined as $\mathcal{M} \leftarrow \{0,1\}^*$. The public key in [DOTT21] includes a uniformly random matrix $\boldsymbol{A} \in R_q^{k \times \ell}$, n vectors $\boldsymbol{t}_i \in R^k$ such that $[\boldsymbol{A}|\boldsymbol{I}_k] \cdot \mathsf{sk}_i = \boldsymbol{t}_i$ for all $i \in [n]$, and two hash functions $\mathsf{H}_{\mathsf{chal}} : \{0,1\}^* \to C$ and $\mathsf{H}_{\mathsf{ck}} : \{0,1\}^* \to S_{ck}$, where S_{ck} is the space of commitment keys for the commitment scheme Com. The main modification we make is that, for each party i, the secret key is now a short vector $\boldsymbol{s}_i$, such that $\boldsymbol{A}\boldsymbol{s}_i = \mathsf{H}_1(i)$ where $\mathsf{H}_1 : [n] \to R_q^k$ is a hash function. The public key now only needs to contain the matrix $\boldsymbol{A}$. We describe the full scheme in Figure 6.

Correctness. Observe that for any honestly generated signature $(\boldsymbol{z}, r, \hat{c}, \mathcal{J})$, we have:

$$\begin{aligned} \bar{\boldsymbol{A}}\boldsymbol{z} - \Sigma_{j \in \mathcal{J}}(d_j \cdot \mathsf{H}_1(j)) &= \bar{\boldsymbol{A}} \cdot (\Sigma_{j \in \mathcal{J}}(d_j \cdot \mathsf{sk}_j + \boldsymbol{y}_j)) - \Sigma_{j \in \mathcal{J}}(d_j \cdot \mathsf{H}_1(j)) \\ &= \Sigma_{j \in \mathcal{J}}\left(d_j \cdot \bar{\boldsymbol{A}} \cdot \mathsf{sk}_j + \bar{\boldsymbol{A}} \cdot \boldsymbol{y}_j - d_j \cdot \mathsf{H}_1(j)\right) \\ &= \Sigma_{j \in \mathcal{J}}\left(d_j \cdot \mathsf{H}_1(j) + \bar{\boldsymbol{A}} \cdot \boldsymbol{y}_j - d_j \cdot \mathsf{H}_1(j)\right) \\ &= \Sigma_{j \in \mathcal{J}} \bar{\boldsymbol{A}} \cdot \boldsymbol{y}_j \\ &= \Sigma_{j \in \mathcal{J}} \boldsymbol{w}_j \end{aligned}$$

The above equations follow from the fact that $\bar{\boldsymbol{A}} \cdot \mathsf{sk}_j = \mathsf{H}_1(j)$ for all parties. Next, observe that $\hat{c} = \Sigma_{j \in \mathcal{J}} c_j$, where c_j is a commitment to $\boldsymbol{w}_j$ with randomness r_j. Hence, by the homomorphism of the commitment scheme, we have that $(\boldsymbol{w} = \Sigma_{j \in \mathcal{J}} \boldsymbol{w}_j; r = \Sigma_{j \in \mathcal{J}} r_j)$ is a valid opening to the commitment $\hat{c}$. Lastly, observe that $\boldsymbol{z}$ is the sum of $|\mathcal{J}| \leq n$ Gaussian variables. Hence, its norm would only be $\sqrt{n}$ times larger than the norm of each $\boldsymbol{z}_i$ vector. Since we set B to be $\sqrt{n}$ times the tail bound on $||\boldsymbol{z}_i||_2 \leq \gamma\sigma\sqrt{N(\ell + k)}$ (as stated in Lemma 2 of [DOTT21], with $\sigma = s/\sqrt{2\pi}$), the Vf algorithm will pass for an honestly generated signature.

The lattice-based Multisignature scheme LSIG_3

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. Run $cpp \leftarrow\!\!\$\ \mathsf{Com.Setup}(1^\lambda)$. Output $\mathsf{pp} \leftarrow (n, cpp)$.
2. The system uses three hash functions, $\mathsf{H_{chal}} : \{0,1\}^* \to C$, $\mathsf{H}_1 : [n] \to R_q^k$, and $\mathsf{H_{ck}} : \{0,1\}^* \to S_{ck}$.

$\underline{\mathsf{KeyGen}()}$:

1. Sample $(\boldsymbol{A}, \mathcal{T}_A) \leftarrow\!\!\$\ \mathsf{TrapGen}(k, \ell)$. Define $\bar{\boldsymbol{A}} \leftarrow [\boldsymbol{A}|\boldsymbol{I}_k]$.
2. For all $i \in [n]$, set $\mathsf{sk}_i \leftarrow \mathsf{SampleGaussian}(\boldsymbol{A}, \boldsymbol{I}_k, \mathcal{T}_A, \mathsf{H}_1(i), \hat{\eta})$.
3. Output $(\mathsf{pk} = \bar{\boldsymbol{A}}, \mathsf{pkc} = \bar{\boldsymbol{A}}, \mathsf{vk} = \bar{\boldsymbol{A}}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i, \mathsf{pk}, \mathcal{J}, m)}$:

– **First Round:**
  1. Parse pk as $\bar{\boldsymbol{A}}$. Sample $\boldsymbol{y}_i \leftarrow\!\!\$\ D_s^{k+\ell}$, set $\boldsymbol{w}_i \leftarrow \bar{\boldsymbol{A}}\boldsymbol{y}_i$. Sample $r_i \leftarrow\!\!\$\ D(S_r)$.
  2. Compute $ck \leftarrow \mathsf{H_{ck}}(m, \mathsf{pk}), c_i \leftarrow \mathsf{Com.Commit}(ck, \boldsymbol{w}_i; r_i)$
  3. Send $\mathsf{msg}_{i,1} \leftarrow c_i$ to all parties in $\mathcal{J}$.
– **Second Round:**
  1. Upon receiving c_j from all $j \in \mathcal{J} \setminus \{i\}$, set $c \leftarrow \Sigma_{j \in \mathcal{J}} c_j$.
  2. Derive a challenge $d_i \leftarrow \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, c, i)$.
  3. Compute signature share $\boldsymbol{z}_i \leftarrow d_i \cdot \mathsf{sk}_i + \boldsymbol{y}_i$.
  4. Run Rejection sampling on $(\boldsymbol{z}_i, d_i\, \mathsf{sk}_i)$. Specifically, output $s_i \leftarrow (m, \boldsymbol{z}_i, r_i, c, \{c_j\}_{j \in \mathcal{J}\setminus\{i\}})$ with probability $p = \min\left(1, \frac{D_s^{k+\ell}(\boldsymbol{z}_i)}{M \cdot D_{d_i\,\mathsf{sk}_i, s}^{k+\ell}(\boldsymbol{z}_i)}\right)$ and output $\perp$ with probability $1 - p$.

$\underline{\mathsf{SigAgg}(\mathsf{pkc}, (s_{i_1}, \ldots, s_{i_{|\mathcal{J}|}}))}$:

1. For each $j \in [|\mathcal{J}|]$, parse s_{i_j} as $(m_{i_j}, \boldsymbol{z}_{i_j}, r_{i_j}, \hat{c}_{i_j}, \{\hat{c}_{i_j,k}\}_{k \in \mathcal{J}\setminus\{i_j\}})$. Let $\hat{c} \leftarrow \hat{c}_{i_1}$, $m \leftarrow m_{i_1}$. If for any $j \in [|\mathcal{J}|]$, $\hat{c}_{i_j} \neq \hat{c}$ or $m_{i_j} \neq m$, then output $\perp$.
2. Parse pkc as $\bar{\boldsymbol{A}} = \mathsf{pk}$. Set $ck \leftarrow \mathsf{H_{ck}}(m, \mathsf{pk})$.
3. For each $j \in \mathcal{J}$, compute $\boldsymbol{w}_j \leftarrow \bar{\boldsymbol{A}}\boldsymbol{z}_j - \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, \hat{c}, j) \cdot \mathsf{H}_1(j)$.
4. If, for any $j, k \in \mathcal{J}$, $\mathsf{Com.Open}(ck, \hat{c}_{k,j}, \boldsymbol{w}_j, r_j) \neq 1$, output $\perp$.
5. Compute $\boldsymbol{z} \leftarrow \Sigma_{j \in \mathcal{J}} \boldsymbol{z}_j$ and $r \leftarrow \Sigma_{j \in \mathcal{J}} r_j$. Output $\sigma \leftarrow (\boldsymbol{z}, r, \hat{c}, \mathcal{J})$.
   // observe that $\boldsymbol{z} = \Sigma_{i \in \mathcal{J}} d_i \cdot \mathsf{sk}_i + \Sigma_{i \in \mathcal{J}} \boldsymbol{y}_i$ where $\Sigma_{i \in \mathcal{J}} d_i \cdot \mathsf{sk}_i$ is the "collective" secret key of subset $\mathcal{J}$.

$\underline{\mathsf{Vf}(\mathsf{vk}, m, \sigma)}$:

1. Parse vk as $\bar{\boldsymbol{A}} = \mathsf{pk}$ and σ as $(\boldsymbol{z}, r, \hat{c}, \mathcal{J})$. Compute $ck \leftarrow \mathsf{H_{ck}}(m, \mathsf{pk})$.
2. For each $j \in \mathcal{J}$, compute $d_j \leftarrow \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, \hat{c}, j)$.
3. Compute $\boldsymbol{w} \leftarrow \bar{\boldsymbol{A}}\boldsymbol{z} - \Sigma_{j \in \mathcal{J}} d_j \cdot \mathsf{H}_1(j)$.
4. Output 1 if $||\boldsymbol{z}||_2 \leq B$ and $\mathsf{Com.Open}(ck, \hat{c}, \boldsymbol{w}, r) = 1$.

$\underline{\mathsf{Trace}(m, \sigma)}$:

1. Parse σ as $(\boldsymbol{z}, r, \hat{c}, \mathcal{J})$ and output $\mathcal{J} \subseteq [n]$.

Fig. 6. The lattice-based Multisignature LSIG_3 with Trusted Setup

Security. Theorem 4 below proves the unforgeability of LSIG_3, with respect to the security game defined in Fig. 7. It is based on the hardness of MSIS and MLWE assumptions (as defined in Definitions 7 and 8) and the security of the commitment scheme Com. The proof follows the proof of Theorem 2 in [DOTT21], and models the hash functions as random oracles. It is provided in Appendix C.3. The parameters of our scheme are the same as those in Table 2 of [DOTT21], with $\hat{\eta} = \eta$.

Theorem 4. *For every* PPT *adversary* $\mathcal{A}$*, there exist adversaries* $\mathcal{B}_1$*,* $\mathcal{B}_2$ *such that,*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{LSIG}_3,\mathcal{A}}(\lambda) \leq e \cdot n \cdot (q_H + q_S + 1) \cdot \left(\begin{array}{c} (q_H + q_S)\epsilon_{td} + \frac{2q_S e^{-t^2/2}}{M} + \\ (n-1) \cdot \mathsf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta},\mathcal{B}_1}(\lambda) + \frac{q_H+q_S+1}{|C|} + \\ \sqrt{(q_H + q_S + 1) \cdot (\epsilon_{bind} + \mathsf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta},\mathcal{B}_2}(\lambda)} \end{array} \right)$$

where $n = n(\lambda)$*,* $q_S = q_S(\lambda)$ *and* $q_H = q_H(\lambda)$ *are an upper bound on the number of signers, the number of signing and random oracle queries by* $\mathcal{A}$ *respectively, and* $e \approx 2.71$*. Also,* $\epsilon_{td} = \epsilon_{td}(\lambda)$ *and* $\epsilon_{bind} = \epsilon_{bind}(\lambda)$ *are defined with respect to* Com*, as in Section 6.1.* t, M *are parameters of the rejection sampling algorithm as defined in Lemma 2,* C *is the challenge space as defined in Section 6.1, and* q, k, ℓ, β, η *are parameters of the scheme as described in Table 2 of [DOTT21].*

On aborts and Signature size. The protocol only outputs a signature after two rounds if the rejection sampling step goes through for all the signers, which happens with probability $1/M^{|\mathcal{J}|} \geq 1/M^n$, where M is a parameter of the rejection sampling algorithm (discussed in Lemma 2). As explained in [DOTT21], for M^n to be constant, lets say 3, we would need to scale $\alpha \geq 11n$ and hence σ now needs to be $\geq 11nT$. This increases the norm bound B on the aggregate signature, which is proportional to $\sqrt{n} \cdot \sigma$, and hence grows with $n^{3/2}$. The signature size would increase by a factor of roughly $O(\log(n))$. We leave further optimizations and concrete parameter analysis as future work.

Round Complexity. The probability that we get a valid signature at the end of two rounds is $1/M^{|\mathcal{J}|} \geq 1/M^n$. The parameters can be adjusted so that this probability is constant, lets say at least $1/3$, which would mean that the expected number of rounds to get a signature is $2M^n = 6$. Alternatively, as also suggested in [DOTT21], all the parties could run multiple executions of the protocol in parallel, then, the probability that at least one execution outputs a signature would be overwhelmingly high.

Semi-Honest Distributed Key Generation. The public and secret keys of LSIG_3 can be generated via a distributed protocol using techniques given in [BKP13]. Specifically, both the trapdoor generation and Gaussian sampling algorithms can be executed as a distributed protocol among all the parties.

Proactive Refresh. Similar to our techniques in Appendix B, we can add proactive refresh to our lattice-based multisignature LSIG_3 by setting the secret key of party i in epoch e to be a short vector such that $\boldsymbol{A} \cdot \mathsf{sk}_i^e = \mathsf{H}_1(i, e)$. The Update protocol can use techniques given in [BKP13] to run the $\mathsf{SampleGaussian}$ protocol in a distributed fashion to refresh the secret keys for each epoch.

On requiring a Trapdoor Commitment Scheme. Our technique can also be used to extend the 3-round protocol in [DOTT21]. The resulting scheme will be a multisignature with short public keys, and will only require an additively homomorphic commitment scheme. Additionally, we believe

that the 2-round scheme given in [BTT22] can also be extended using our technique, which will result in a 2-round multisignature with a short public key without requiring a trapdoor commitment scheme.

7 Conclusion and Future Directions

We presented a number of multisignature schemes with a short verification key. Some of our schemes support local key generation for the signers while others require a distributed key generation protocol. Some are set in the plain model while others are set in the random oracle model. Our last scheme is based on lattices, while the first three are pairing based.

An interesting direction for future work is to extend the ideas from the schemes SIG_1 and SIG_2 to the lattice setting, to get multisignatures with short public keys and local key generation. Constructing such a scheme that requires only a single round of interaction to generate a signature (as opposed to two) is an important open problem.

Acknowledgments. This work was funded by NSF, DARPA, the Simons Foundation, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

ABB10. S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT 2010*, *LNCS* 6110, pages 553–572, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

BB04. D. Boneh and X. Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *EUROCRYPT 2004*, *LNCS* 3027, pages 223–238, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

BB11. D. Boneh and X. Boyen. Efficient selective identity-based encryption without random oracles. *Journal of Cryptology*, 24(4):659–693, October 2011.

BCG+23. F. Baldimtsi, K. K. Chalkias, F. Garillot, J. Lindstrom, B. Riva, A. Roy, A. Sonnino, P. Waiwitlikhit, and J. Wang. Subset-optimized bls multi-signature with key aggregation. Cryptology ePrint Archive, Paper 2023/498, 2023. https://eprint.iacr.org/2023/498.

BDN18. D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018, Part II*, *LNCS* 11273, pages 435–464, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.

BGJ+23. L. Baird, S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. Threshold signatures in the multiverse. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1454–1470, 2023.

BGLS03. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT 2003*, *LNCS* 2656, pages 416–432, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

BGR98. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *EUROCRYPT'98*, *LNCS* 1403, pages 236–250, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany.

BGR12. K. Brogle, S. Goldberg, and L. Reyzin. Sequential aggregate signatures with lazy verification from trapdoor permutations - (extended abstract). In *ASIACRYPT 2012*, *LNCS* 7658, pages 644–662, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.

BKP13. R. Bendlin, S. Krehbiel, and C. Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In *ACNS 13*, *LNCS* 7954, pages 218–236, Banff, AB, Canada, June 25–28, 2013. Springer, Heidelberg, Germany.

BL22. R. Bacho and J. Loss. On the adaptive security of the threshold BLS signature scheme. In *ACM CCS 2022*, pages 193–207, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

BLS01. D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *ASIACRYPT 2001*, *LNCS* 2248, pages 514–532, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

BN06. M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS 2006*, pages 390–399, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press.

BNN07. M. Bellare, C. Namprempre, and G. Neven. Unrestricted aggregate signatures. In *ICALP 2007*, *LNCS* 4596, pages 411–422, Wroclaw, Poland, July 9–13, 2007. Springer, Heidelberg, Germany.

Bol03. A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *PKC 2003*, *LNCS* 2567, pages 31–46, Miami, FL, USA, January 6–8, 2003. Springer, Heidelberg, Germany.

BPR22. D. Boneh, A. Partap, and L. Rotem. Proactive refresh for accountable threshold signatures. Cryptology ePrint Archive, Paper 2022/1656, 2022. https://eprint.iacr.org/2022/1656.

BS23. D. Boneh and V. Shoup. *A graduate course in applied cryptography (version 0.6)*. 2023. cryptobook.us.

BTT22. C. Boschini, A. Takahashi, and M. Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In *CRYPTO 2022, Part II*, *LNCS* 13508, pages 276–305, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

CKM23. E. Crites, C. Komlo, and M. Maller. Fully adaptive schnorr threshold signatures. Cryptology ePrint Archive, Paper 2023/445, 2023. https://eprint.iacr.org/2023/445.

CS04. J. F. Canny and S. Sorkin. Practical large-scale distributed key generation. In *EUROCRYPT 2004*, *LNCS* 3027, pages 138–152, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

DCX+23. S. Das, P. Camacho, Z. Xiang, J. Nieto, B. Bunz, and L. Ren. Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. Cryptology ePrint Archive, Paper 2023/598, 2023. https://eprint.iacr.org/2023/598.

DFKP16. A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *2016 IEEE Symposium on Security and Privacy*, pages 235–254, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

DGKR18. B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT 2018, Part II*, *LNCS* 10821, pages 66–98, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

DOTT21. I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In *PKC 2021, Part I*, *LNCS* 12710, pages 99–130, Virtual Event, May 10–13, 2021. Springer, Heidelberg, Germany.

DR23. S. Das and L. Ren. Adaptively secure bls threshold signatures from ddh and co-cdh. Cryptology ePrint Archive, Paper 2023/1553, 2023. https://eprint.iacr.org/2023/1553.

EB14. R. El Bansarkhani and J. Buchmann. Towards lattice based aggregate signatures. In *AFRICACRYPT 14*, *LNCS* 8469, pages 336–355, Marrakesh, Morocco, May 28–30, 2014. Springer, Heidelberg, Germany.

FH20. M. Fukumitsu and S. Hasegawa. A lattice-based provably secure multisignature scheme in quantum random oracle model. In *ProvSec 2020*, *LNCS* 12505, pages 45–64, Singapore, November 29 – December 1, 2020. Springer, Heidelberg, Germany.

FLS12. M. Fischlin, A. Lehmann, and D. Schröder. History-free sequential aggregate signatures. In *SCN 12*, *LNCS* 7485, pages 113–130, Amalfi, Italy, September 5–7, 2012. Springer, Heidelberg, Germany.

GHM+17. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Report 2017/454, 2017. https://eprint.iacr.org/2017/454.

GJKR99. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *EUROCRYPT'99*, *LNCS* 1592, pages 295–310, Prague, Czech Republic, May 2–6, 1999. Springer, Heidelberg, Germany.

GJKR07. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

GJM+23. S. Garg, A. Jain, P. Mukherjee, R. Sinha, M. Wang, and Y. Zhang. hints: Threshold signatures with silent setup. Cryptology ePrint Archive, Paper 2023/567, 2023. https://eprint.iacr.org/2023/567.

GM18. N. Genise and D. Micciancio. Faster Gaussian sampling for trapdoor lattices with arbitrary modulus. In *EUROCRYPT 2018, Part I*, *LNCS* 10820, pages 174–203, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

GOR18. C. Gentry, A. O'Neill, and L. Reyzin. A unified framework for trapdoor-permutation-based sequential aggregate signatures. In *PKC 2018, Part II*, *LNCS* 10770, pages 34–57, Rio de Janeiro, Brazil, March 25–29, 2018. Springer, Heidelberg, Germany.

GPV08. C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

GR06. C. Gentry and Z. Ramzan. Identity-based aggregate signatures. In *PKC 2006*, *LNCS* 3958, pages 257–273, New York, NY, USA, April 24–26, 2006. Springer, Heidelberg, Germany.

Gro16. J. Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016, Part II*, *LNCS* 9666, pages 305–326, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

Gro21. J. Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021. https://eprint.iacr.org/2021/339.

HK08. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. In *CRYPTO 2008*, *LNCS* 5157, pages 21–38, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

HK12. D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012.

LMRS04. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In *EUROCRYPT 2004*, *LNCS* 3027, pages 74–90, Interlaken, Switzerland, May 2–6, 2004. Springer, Heidelberg, Germany.

LOS+06. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT 2006*, *LNCS* 4004, pages 465–485, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

Lyu12. V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT 2012*, *LNCS* 7237, pages 738–755, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

MOR01. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In *ACM CCS 2001*, pages 245–254, Philadelphia, PA, USA, November 5–8, 2001. ACM Press.

Nao03. M. Naor. On cryptographic assumptions and challenges (invited talk). In *CRYPTO 2003*, *LNCS* 2729, pages 96–109, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.

NRS21. J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In *CRYPTO 2021, Part I*, *LNCS* 12825, pages 189–221, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany.

RY07. T. Ristenpart and S. Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT 2007*, *LNCS* 4515, pages 228–245, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

Smi22. C. Smith. Ethereum proof of stake, 2022. link.

Wat05. B. R. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT 2005*, *LNCS* 3494, pages 114–127, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

A Interactive Multisignatures

In this section, we present our definitions for interactive multisignature schemes. We focus on schemes in which the signing protocol is made up of two rounds of communication among the signers. These definitions capture our lattice based multisignature from Section 6.

The syntax for interactive schemes closely follows that of non-interactive schemes, with the following exception. In interactive schemes, the signature algorithm Sign is now an interactive protocol, made up of two sub-algorithms $(\mathsf{Sign}_1, \mathsf{Sign}_2)$, where:

- Sign_1 is a randomized algorithm which takes as input a secret key sk_i, a message m, the public key pk and a subset $\mathcal{J} \subseteq [n]$ of indices. It outputs a state $\mathsf{st}_{i,1}$ and a first message $\mathsf{msg}_{i,1}$ to be sent in round 1 of the protocol to all signers in $\mathcal{J} \setminus \{i\}$.
- Sign_2 is a deterministic algorithm which takes as input a state $\mathsf{st}_{i,1}$ and incoming messages $\{\mathsf{msg}_{j,1}\}_{j \in \mathcal{J} \setminus \{i\}}$. It outputs a signature share s_i.

The syntax above requires knowledge of m and $\mathcal{J}$ at the beginning of the protocol (these are given as inputs to Sign_1). This captures the standard scenario in which a subset $\mathcal{J}$ of signers initiates

the signing protocol in a coordinated manner in order to sign a particular message m. Defining the syntax in this manner also has the advantage of being general enough to capture protocols that require knowledge of m and $\mathcal{J}$ already in the onset of the protocol.

The correctness definitions for an interactive multisignature scheme are naturally extended from the non-interactive case, by replacing the non-interactive signing algorithm with an honest execution of the interactive signing protocol.

Security. Unforgeability of two-round multisignatures are defined via a natural generalization of the analogous definitions for non-interactive schemes (presented in Figure 4). Importantly, the signing oracle is now replaced with two separate oracles, one for each of the sub-routines making up the signing protocol. The adversary may query any of these oracles on inputs and ordering of its choice. In particular, this allows the adversary to interact with honest signers in many concurrent sessions of the signing protocol, arbitrarily interleaving between them.

For a two-round multisignatures scheme $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign} = (\mathsf{Sign}_1, \mathsf{Sign}_2), \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace})$, the security game capturing its unforgeability is defined in Figure 7. The advantage is defined as in Definition 4.

Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS}}$ with respect to an adversary $\mathcal{A}$ and security parameter λ

1 : $(\mathsf{st}, n) \leftarrow \mathcal{A}(\lambda)$
2 : $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, n)$
3 : $\mathcal{Q}^{\mathsf{sk}} \leftarrow \emptyset\ ;\ \ \forall\ m \in \mathcal{M}, \mathcal{Q}^{\mathsf{sig}}(m) \leftarrow \emptyset$
4 : $(\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow\!\$\ \mathsf{KeyGen}()$
5 : **for** $i = \{1, \ldots, n\}$ **do**
6 : $\quad \mathsf{sid}_i \leftarrow 0, \mathcal{S}_{i,1} \leftarrow \emptyset$
7 : $(m^*, \sigma^*) \leftarrow\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot), \mathsf{Sign}(\cdot)}(\mathsf{st}, \mathsf{pp}, \mathsf{pk}, \mathsf{pkc}, \mathsf{vk})$
8 : **if** $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ **then**
9 : $\quad$ **return** 0
10 : $\mathcal{J} \leftarrow \mathsf{Trace}(m^*, \sigma^*) \subseteq [n]$
11 : **if** $\mathcal{J} \not\subseteq \mathcal{Q}^{\mathsf{sk}} \cup \mathcal{Q}^{\mathsf{sig}}(m^*)$ **then** // the forgery traces to a set containing an honest party
12 : $\quad$ **return** 1

Oracle $\mathsf{skO}(i)$:
1 : $\mathcal{Q}^{\mathsf{sk}} \leftarrow \mathcal{Q}^{\mathsf{sk}} \cup \{i\}$
2 : **return** sk_i

Oracle $\mathsf{Sign}_1\mathsf{O}(i, \mathcal{J}, m)$:
1 : $\mathsf{sid}_i \leftarrow \mathsf{sid}_i + 1$
2 : $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \cup \{\mathsf{sid}_i\}$
3 : $(\mathsf{st}^{\mathsf{sid}_i}_{i,1}, \mathsf{msg}^{\mathsf{sid}_i}_{i,1}) \leftarrow\!\$\ \mathsf{Sign}_1(\mathsf{sk}_i, m, \mathcal{J})$
4 : **return** $\mathsf{msg}^{\mathsf{sid}_i}_{i,1}$

Oracle $\mathsf{Sign}_2\mathsf{O}(i, \mathsf{sid}, (\widetilde{\mathsf{msg}_{i_1,1}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,1}}))$:
1 : **if** $\mathsf{sid} \notin \mathcal{S}_{i,1}$ **then**
2 : $\quad$ **return** $\perp$
3 : **fi**
4 : $\mathcal{Q}^{\mathsf{sig}}(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}(m) \cup \{i\}$
5 : $s^{\mathsf{sid}}_i \leftarrow\!\$\ \mathsf{Sign}_2(\mathsf{st}^{\mathsf{sid}}_{i,1}, \widetilde{\mathsf{msg}_{i_1,1}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,1}})$
6 : $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \setminus \{\mathsf{sid}\}$
7 : **return** s^{sid}_i

Fig. 7. The security game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{TMS}}$ for a two-round multisignature scheme $\mathsf{TMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace})$. In line 7, we write $\mathsf{Sign}(\cdot)$ as a short hand for denoting that $\mathcal{A}$ has oracle access to the two oracles $\mathsf{Sign}_1\mathsf{O}(\cdot,\cdot,\cdot)$ and $\mathsf{Sign}_2\mathsf{O}(\cdot,\cdot,\cdot)$.

B Efficient Proactive Refresh for the SIG_3 Multisignature scheme

In this section, we present an efficient proactive refresh mechanism for the SIG_3 scheme. We start by providing the syntactic additions for proactive refresh, and defining the correctness and security notions for such schemes.

B.1 Multisignatures with Proactive Refresh

Syntax. A Multisignature scheme with proactive refresh (MS-PR) is a Multisignature scheme that is additionally equipped with a key-update procedure, whose role is to refresh the signers' secret keys without modifying the public key in any way. As also described in [BPR22], we can envision the key-update procedure as dividing time into epochs. An epoch starts once one execution of the key-update procedure ends (or, for the first epoch right after the invocation key generation algorithm), and ends when the next execution of the key-update procedure ends. Following [BPR22], we define the key-update procedure as a pair $\mathsf{Update} = (\mathsf{Update}_0, \mathsf{Update}_1)$ of algorithms:

- Update_0 is a randomized algorithm that takes in a secret key sk_i^e of signer i in epoch e and the public key pk, and outputs a vector $(\delta_{i,1}^e, ..., \delta_{i,n}^e)$ of update messages. Each signer i sends $\delta_{i,j}^e$ to the jth signer, for all $j \neq i$.
- Update_1 is a deterministic algorithm that takes in a secret key sk_i^e and n update messages $\delta_{1,i}^e, \ldots, \delta_{n,i}^e$. It outputs an updated secret key sk_i^{e+1} for epoch $e+1$ for signer i.

For succinctness, we may write $(\mathsf{sk}_1^{e+1}, \ldots, \mathsf{sk}_n^{e+1}) \leftarrow\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^e, \ldots, \mathsf{sk}_n^e)$ as a shorthand for the random process of first invoking $\mathsf{Update}_0(\mathsf{sk}_i^e, \mathsf{pk})$ for every $i \in [n]$ to randomly sample n^2 update messages $\{\delta_{i,j}\}_{i,j\in[n]}$; and then running $\mathsf{Update}_1(\mathsf{sk}_i^e, (\delta_{1,i}^e, \ldots, \delta_{n,i}))$ to obtain sk_i^{e+1} for every $i \in [n]$.

Correctness. These algorithms must satisfy *verification correctness* and *tracing correctness*, similar to those defined in Section 2.1. Informally, Vf should accept honestly-generated signatures in all epochs, and Trace should trace to the set of signing parties, for honestly-generated signatures in all epochs. More formally, for all messages m in the message space, all polynomials $n = n(\lambda)$, all positive integers $e \leq e(\lambda)$ and all non-empty subsets $\mathcal{J} \subseteq [n]$, it holds that,

$$\Pr\left[\mathsf{Vf}\big(\mathsf{vk}, m, \mathsf{SigAgg}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j^e, m)\}_{j\in\mathcal{J}})\big) = 1\right] = 1,$$

$$\Pr\left[\mathsf{Trace}\big(m, \mathsf{SigAgg}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j^e, m)\}_{j\in\mathcal{J}})\big) = \mathcal{J}\right] = 1,$$

where the probability is over the random variables $\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, n)$, $(\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\!\$\ \mathsf{KeyGen}()$, $(\mathsf{sk}_1^{i+1}, \ldots, \mathsf{sk}_n^{i+1}) \leftarrow\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^i, \ldots, \mathsf{sk}_n^i)$ for $i = 1, \ldots, e-1$, and the random coins of Sign.

Security. A multisignature with proactive refresh should satisfy unforgeability. As defined in Section 5.1, the traditional unforgeability property states that an adversary should not be able to produce a valid signature on a message m on behalf of a subset $\mathcal{J}$ of signers without observing the secret key or signature share on m of all the signers in $\mathcal{J}$. In the proactive refresh setting, we require that this restriction on the adversary should hold in each epoch (thus allowing them to observe secret keys/signature shares on m of all signers in $\mathcal{J}$ across different epochs). In other words, the adversary is allowed to corrupt up to $n-1$ parties in each epoch. Fig. 8 extends the security game given in Section 5.1 to capture this unforgeability notion. The adversary first specifies the number of parties n, which is followed by the challenger running Setup and KeyGen to sample keys for all

parties. The challenger sends the public parameters to the adversary and gets E, the number of epochs. The challenger then runs the Update protocol $E-1$ times to sample keys for epochs 2 to E. The adversary then interacts with the challenger using two types of queries: Secret-key queries and signature queries. A secret-key query (e, i) reveals to the adversary the secret key of signer i in epoch e. A signature query (m, e, i) provides the adversary with an honestly-generated signature share on m with respect to signer i's secret key in epoch e. Finally, the adversary should produce a valid forgery; that is, a message m^* and a signature σ^* that passes verification. Note that this definition is similar to the uf-1 $\wedge$ acc-1 notion defined in [BPR22].

Game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRMS}}$ with respect to an adversary $\mathcal{A}$ and security parameter λ

$(\mathsf{st}, n) \leftarrow \mathcal{A}(\lambda)$
$\mathsf{pp} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, n)$
$(\mathsf{st}, E) \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{pp})$
$\forall\ e \in [E], \mathcal{Q}^{\mathsf{sk}}_e \leftarrow \emptyset$
$\forall\ m \in \mathcal{M}, e \in [E], \mathcal{Q}^{\mathsf{sig}}_e(m) \leftarrow \emptyset$
$(\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, \mathsf{sk}^1_1, \ldots, \mathsf{sk}^1_n) \leftarrow\!\$\ \mathsf{KeyGen}()$
for $e = \{2, \ldots, E\}$ **do**
  $(\mathsf{sk}^e_1, \ldots, \mathsf{sk}^e_n) \leftarrow\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}^{e-1}_1, \ldots, \mathsf{sk}^{e-1}_n)$
$(m^*, \sigma^*) \leftarrow\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot),\mathsf{SignO}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc}, \mathsf{vk})$
if $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ **then**
  return 0
if $\forall e \in [E], \mathsf{Trace}(m^*, \sigma^*) \not\subseteq \mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)$ **then return** 1

Oracle $\mathsf{skO}(e, i)$
$\mathcal{Q}^{\mathsf{sk}}_e \leftarrow \mathcal{Q}^{\mathsf{sk}}_e \cup \{i\}$
return sk^e_i

Oracle $\mathsf{SignO}(m, e, i)$
$\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}^e_i, m)$
$\mathcal{Q}^{\mathsf{sig}}_e(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}_e(m) \cup \{i\}$
return σ_i

Fig. 8. The security game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRMS}}$ for a Multisignature scheme with proactive refresh $\mathsf{PRMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$. Note that if the adversary queries skO or SignO for an epoch $e > E$, then the game outputs 0.

Definition 9 below defines the advantage of an adversary $\mathcal{A}$ in the game defined in Figure 8 as the probability that the game outputs 1 when executed with $\mathcal{A}$.

Definition 9. *Let* $\mathsf{PRMS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *be a Multisignature scheme with proactive refresh. It is said to be secure if the following function is negligible in* λ *for all* PPT *adversaries* $\mathcal{A}$*:*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{PRMS},\mathcal{A}}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRMS},\mathcal{A}}(\lambda) = 1\right].$$

B.2 The Multisignature scheme PRSIG_3 with proactive refresh

Recall that the secret key of party i in SIG_3 (defined in Section 5.2) is a BLS signature on (i) with a secret key α. To refresh the keys, we simply set the secret keys to be BLS signatures on (i, e) instead. Formally, we set the secret key for party i for epoch e to be $(\alpha_{i,e}, \mathsf{H}_1(i,e)^{\alpha})$, where $\{\alpha_{i,e}\}_{i\in[n]}$ is an n-out-of-n additive secret sharing of the BLS secret key α. The $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}$ functions can be extended directly. But now, to verify a signature, we would also need to know which epoch was this signature generated in. To resolve this, we include the epoch number in the signature. Hence, a signature in PRSIG_3 is the tuple of e and the SIG_3 signature on m with respect to epoch e.

To update the secret keys, all the parties jointly generate BLS signatures on $(i, e+1)$ for all $i \in [n]$. To do this, party i simply signs $(j, e+1)$ using its secret key share $\alpha_{i,e}$, and sends this partial BLS signature $\mathsf{H}_1(j, e+1)^{\alpha_{i,e}}$ as the update message, to party j, for all $j \neq i$. Each party j then multiplies all the update messages received and also signs $(j, e+1)$ using its own share of the secret key $\alpha_{j,e}$, to get $\mathsf{sk}'_{j,e+1} \leftarrow \mathsf{H}_1(j, e+1)^{\alpha_{j,e}+\Sigma_{i\in[n]\setminus\{j\}}\alpha_{i,e}}$. Secondly, all the parties jointly sample a random n-out-of-n additive sharing of 0, i.e. a vector of n values $\{a_i\}_{i\in[n]}$ that add up to zero. Then, the new secret key of party j becomes a 3-tuple containing $(e+1)$, $\alpha_{j,e} + a_j$ and $\mathsf{sk}'_{j,e+1}$.

To simplify the presentation, we present our scheme using a trusted setup, but the trusted setup can be converted into a simple DKG using standard techniques. We use $\mathsf{SymGroupGen}(1^\lambda) \to (\mathbb{G}, \mathbb{G}_T, e, g, p)$ to denote a bilinear group generator that outputs a symmetric bilinear group. We formally describe our scheme in Figure 9 where we use a symmetric bilinear group for simplicity. The scheme can be easily generalized to use an asymmetric bilinear group.

Correctness. Correctness of PRSIG_3 follows directly from correctness of the SIG_3 scheme as explained in Section 5.2.

Security. Theorem 5 below reduces the security of PRSIG_3 to the unforgeability of SIG_3 and the BLS signature scheme.

Theorem 5. *For any adversary $\mathcal{A}$ there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ with about the same runtime as $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) \leq E \cdot \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{SIG}_3,\mathcal{B}_1}(\lambda) + \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS},\mathcal{B}_2}(\lambda)$$

where $E = E(\lambda)$ is a bound on the number of epochs requested by $\mathcal{A}$ in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3}$.

Proof. Let $\mathcal{A}$ be an adversary playing the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3}$. Let (m^*, σ^*) denote the message and signature that $\mathcal{A}$ outputs at the end of the game. We assume without loss of generality that with probability 1, the signature σ^* is a 4-tuple containing $(e^*, \sigma^*_0, \sigma^*_1, \mathcal{J}^*)$. Let E denote the number of epochs specified by $\mathcal{A}$ in the beginning of the game. By total probability,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1 \wedge e^* \leq E\right] \\ + \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1 \wedge e^* > E\right]\end{aligned}$$

Theorem 5 now follows from Lemma 3 and Lemma 4.

Lemma 3. *There exists an adversary $\mathcal{B}_1$ such that,*

$$\Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1 \wedge e^* \leq E\right] \leq E \cdot \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{SIG}_3,\mathcal{B}_1}(\lambda)$$

The Multisignature scheme PRSIG_3 with Proactive Refresh

$\underline{\mathsf{Setup}(1^\lambda, n)}$:

1. Sample a symmetric bilinear group $(\mathbb{G}, \mathbb{G}_T, e, g, p) \leftarrow\!\!\$\ \mathsf{SymGroupGen}(1^\lambda)$ and output $\mathsf{pp} \leftarrow (n, \mathbb{G}, \mathbb{G}_T, e, g, p)$.
2. The system uses two hash functions, $\mathsf{H}_0 : \mathcal{M} \to \mathbb{G}$ and $\mathsf{H}_1 : [n] \to \mathbb{G}$.

$\underline{\mathsf{KeyGen}()}$:

1. Sample $\alpha_1, \ldots, \alpha_n \leftarrow\!\!\$\ \mathbb{Z}_p$. Set $\alpha \leftarrow \Sigma_{i\in[n]}\alpha_i$, $h \leftarrow g^\alpha$ and $\mathsf{sk}_i \leftarrow (1, \alpha_i, \mathsf{H}_1(i,1)^\alpha)$ for all $i \in [n]$.
2. Output $(\mathsf{pk} = h, \mathsf{pkc} = \perp, \mathsf{vk} = h, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\underline{\mathsf{Sign}(\mathsf{sk}_i, m)}$:

1. Parse sk_i as $(e_i, \alpha_i, \mathsf{sk}'_i)$. Sample $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$.
2. Compute $\sigma_{i,0} \leftarrow g^{r_i} \in \mathbb{G}$ and $\sigma_{i,1} \leftarrow \mathsf{sk}'_i \cdot \mathsf{H}_0(m)^{r_i} \in \mathbb{G}$.
3. Output $\sigma_i = (e_i, \sigma_{i,0}, \sigma_{i,1})$.

$\underline{\mathsf{SigAgg}(\mathsf{pkc}, (\sigma_{i_1}, \ldots, \sigma_{i_{|\mathcal{J}|}}))}$:

1. For $j \in \mathcal{J}$: Parse σ_j as $(e_j, \sigma_{j,0}, \sigma_{j,1})$.
2. Let $e = e_{i_1}$. If for some $j \in \mathcal{J}$, it holds that $e_j \neq e$, output $\perp$.
3. Compute $\sigma_0 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,0}$ and $\sigma_1 \leftarrow \prod_{j\in\mathcal{J}} \sigma_{j,1}$.
4. Output $\sigma = (e, \sigma_0, \sigma_1, \mathcal{J})$.

$\underline{\mathsf{Vf}(\mathsf{vk}, m, \sigma)}$:

1. Parse vk as (h) and σ as $(e, \sigma_0, \sigma_1, \mathcal{J})$.
2. Output 1 if $e(\sigma_1, g) = e(\mathsf{H}_0(m), \sigma_0) \cdot e(\prod_{j\in\mathcal{J}} \mathsf{H}_1(j, e), h)$.

$\underline{\mathsf{Trace}(m, \sigma)}$:

1. Parse σ as $(e, \sigma_0, \sigma_1, \mathcal{J})$ and output $\mathcal{J} \subseteq [n]$.

$\underline{\mathsf{Update}_0(\mathsf{pk}, \mathsf{sk}_i)}$:

1. Parse pk as (h) and sk_i as $(e, \alpha_{i,e}, \mathsf{sk}'_{i,e})$.
2. Sample $a_1, \ldots, a_{n-1} \leftarrow\!\!\$\ \mathbb{Z}_p$. Set $a_n \leftarrow -\Sigma_{j\in[n-1]} a_j$.
3. Set $\sigma_{j,e+1} \leftarrow \mathsf{H}_1(j, e+1)^{\alpha_{i,e}}$ for all $j \in [n]$.
4. Output $\{\delta_{i,j} \leftarrow (a_j, \sigma_{j,e+1})\}_{j\in[n]}$.

$\underline{\mathsf{Update}_1(\mathsf{pk}, \mathsf{sk}_j, \delta_{1,j}, \ldots, \delta_{n,j})}$:

1. Parse $\delta_{i,j}$ as $(a_{i,j}, \sigma_{i,j})$ for all $i \in [n]$ and sk_j as $(e, \alpha_{j,e}, \mathsf{sk}'_{j,e})$.
2. Compute $e' = e + 1$, $\alpha_{j,e+1} \leftarrow \alpha_{j,e} + \Sigma_{i\in[n]} a_{i,j}$ and $\mathsf{sk}'_{j,e+1} \leftarrow \prod_{i\in[n]} \sigma_{i,j}$.
3. Output $(e', \alpha_{j,e+1}, \mathsf{sk}'_{j,e+1})$.

Fig. 9. The Multisignature scheme PRSIG_3 with Proactive Refresh

Proof (of Lemma 3). Consider the following adversary $\mathcal{B}_1$ playing the game $\mathbf{G}^{\text{uf}}_{\mathsf{SIG}_3}$. $\mathcal{B}_1$ can issue signature queries, secret key queries and random oracle queries to its challenger. To distinguish the random oracles in $\mathbf{G}^{\text{uf}}_{\mathsf{SIG}_3}$ from the ones in $\mathbf{G}^{\text{uf}}_{\mathsf{PRSIG}_3}$, we denote queries by $\mathcal{B}_1$ to its challenger as $\hat{\mathsf{H}}_0(m)$ and $\hat{\mathsf{H}}_1(i)$.

$\mathcal{B}_1$ invokes $\mathcal{A}$ and simulates the game $\mathbf{G}^{\text{uf}}_{\mathsf{PRSIG}_3}$ as follows:

1. Receive (n) from $\mathcal{A}$. Forward (n) to its challenger.
2. Receive $\mathsf{pp}, \mathsf{pk} = (h), \mathsf{pkc} = \bot, \mathsf{vk} = h$ from its challenger and forward pp to $\mathcal{A}$. Let us use α to denote the secret key of the SIG_3 challenger, which is unknown to $\mathcal{B}_1$. So, $h = g^\alpha$.
3. Receive E from $\mathcal{A}$, and forward $\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}$ to $\mathcal{A}$.
4. Guess $\hat{e} \leftarrow\!\!\$\ [E]$.
5. For all $e \in [E] \setminus \{\hat{e}\}$ and all $i \in [n]$, sample $\gamma_{i,e} \leftarrow\!\!\$\ \mathbb{Z}_p$. This will be used by $\mathcal{B}_1$ to respond to random oracle queries.

Next, $\mathcal{A}$ issues a sequence of queries. $\mathcal{B}_1$ initializes the simulated oracles $\mathsf{H}_0(m) \leftarrow \bot$ for all m and $\mathsf{H}_1(i, e) \leftarrow \bot$ for all $i \in [n], e \in [E]$.

$\mathcal{B}_1$ maintains a set $\mathcal{Q}^{\mathsf{sk}}_e$ for each $e \in [E]$, to store all the parties for which $\mathcal{A}$ calls the secret key oracle in epoch e. It also stores a mapping $S_e : [n] \to \mathbb{Z}_p$ for all $e \in [E]$, to record auxiliary information for responding to skO queries in epoch e. We now discuss how $\mathcal{B}_1$ responds to each of $\mathcal{A}$'s queries:

- $\mathsf{H}_0(m)$. $\mathcal{B}_1$ simply queries its challenger for $\hat{\mathsf{H}}_0(m)$, sets $\mathsf{H}_0(m) \leftarrow \hat{\mathsf{H}}_0(m)$ and returns this value to $\mathcal{A}$.
- $\mathsf{H}_1(i, e)$. If $\mathsf{H}_1(i, e) \neq \bot$, the $\mathcal{B}$ outputs $\mathsf{H}_1(i, e)$. Otherwise, if $e > E$, $\mathcal{B}_1$ samples $\gamma_{i,e} \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $\mathsf{H}_1(i, e) \leftarrow g^{\gamma_{i,e}}$ and returns this value.
  Otherwise, if $e \neq \hat{e}$, $\mathcal{B}_1$ responds with $g^{\gamma_{i,e}}$ ($\gamma_{i,e}$ was sampled in Step 5 above). If $e = \hat{e}$, $\mathcal{B}_1$ queries its challenger for $\hat{\mathsf{H}}_1(i)$, sets $\mathsf{H}_1(i, e) \leftarrow \hat{\mathsf{H}}_1(i)$ and forwards this to $\mathcal{A}$.
- $\mathsf{skO}(j, e)$. If $e > E$, $\mathcal{B}_1$ aborts. Otherwise, it first adds j to $\mathcal{Q}^{\mathsf{sk}}_e$. If $|\mathcal{Q}^{\mathsf{sk}}_e| = n$, then $\mathcal{B}_1$ aborts. Next, if $S_e(j) = \bot$, $\mathcal{B}_2$ samples $S_e(j) \leftarrow\!\!\$\ \mathbb{Z}_p$.
  If $e \neq \hat{e}$, then $\mathcal{B}_1$ returns $(e, S_e(j), h^{\gamma_{j,e}})$. Note that this secret key is distributed identically to a real secret key, since $\mathsf{H}_1(j, e) = g^{\gamma_{j,e}}$ implies that $\mathsf{sk}'_{j,e} = (g^{\gamma_{j,e}})^\alpha = (g^\alpha)^{\gamma_{j,e}} = h^{\gamma_{j,e}}$.
  Lastly, if $e = \hat{e}$, $\mathcal{B}_1$ queries its challenger for $\mathsf{sk}'_{j,\hat{e}} \leftarrow \mathsf{skO}(j)$, and returns $(e, S_e(j), \mathsf{sk}'_{j,\hat{e}})$.
  Note that, sampling $S_e(j)$ uniformly randomly is distributed identically to real secret keys because, for an adversary that only observes up to $n - 1$ secret keys in a single epoch e, all the $\alpha_{i,e}$ values appear uniformly random.
- $\mathsf{SignO}(j, m, e)$. If $e > E$, $\mathcal{B}_1$ aborts. $\mathcal{B}_1$ then calls $\mathsf{H}_0(m)$. Next, if $e \neq \hat{e}$, $\mathcal{B}_1$ samples $r \leftarrow\!\!\$\ \mathbb{Z}_p$, and returns $(e, g^r, h^{\gamma_{i,e}} \cdot \mathsf{H}_0(m)^r)$. This is distributed identically to a real signature since $\mathcal{B}_1$ knows the secret key for all parties $j \in [n]$, for all epochs $e \neq \hat{e} : \mathsf{sk}'_{j,e} \leftarrow h^{\gamma_{j,e}}$.
  If $e = \hat{e}$, $\mathcal{B}_1$ queries $\sigma_{j,m,\hat{e}} \leftarrow \mathsf{SignO}(j, m)$ from its challenger. It returns $(\hat{e}, \sigma_{j,m,\hat{e}})$ to $\mathcal{A}$.

Eventually, $\mathcal{A}$ outputs a forgery (m^*, σ^*), where $\sigma^* = (e^*, \sigma^*_0, \sigma^*_1, \mathcal{J}^*)$. $\mathcal{B}_1$ aborts if $\hat{e} \neq e^*$ or if $e^* > E$. $\mathcal{B}_1$ also aborts if this is not a valid forgery, i.e. $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ or, for some $i \in [E]$, $\mathsf{Trace}(m^*, \sigma^*) \subseteq (\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*))$.

Otherwise, $\mathcal{B}_1$ forwards $(m^*, (\sigma^*_0, \sigma^*_1, \mathcal{J}^*))$ to its challenger. We claim that if $\mathcal{B}_1$ does not abort, then $\mathcal{B}_1$ sends a valid forgery. To see this, observe that a valid forgery means that, in each epoch e, $\mathsf{Trace}(m^*, \sigma^*) \not\subseteq (\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*))$. Specifically this means that for epoch e^*, $\mathsf{Trace}(m^*, \sigma^*) \not\subseteq$

$(\mathcal{Q}^{\mathsf{sk}}_{e^*} \cup \mathcal{Q}^{\mathsf{sig}}_{e^*}(m^*))$. Hence, $\mathcal{B}_1$ also gets a valid forgery since it only forwards queries for epoch $e^* = \hat{e}$ to its challenger.

Let Abort denote the event in which $\mathcal{B}_1$ aborts prematurely. Let $\mathsf{E}_{\mathcal{A}}$ denote the event in which $\mathcal{A}$ returns a valid forgery.

We now analyse the abort probability. Let E_1 be the event where $\mathcal{B}_1$ aborts due to $|\mathcal{Q}^{\mathsf{sk}}_e| = n$ for some e. Let E_2 be the event where $\hat{e} \neq e^*$. Let E_3 be the event where $e^* > E$. Let E_4 be the event that $\mathcal{B}_1$ aborts due to a skO or a SignO query for an epoch $e > E$. Let E_5 be the event where $\mathcal{B}_1$ aborts because $\mathcal{A}$ returns an invalid forgery. Observe that $\Pr\left[\overline{\mathsf{E}_2} \mid \overline{\mathsf{E}_3}\right] = \frac{1}{E}$ since $\mathcal{B}_1$ randomly guesses the forgery epoch e^* in $[E]$. Also note that for the forgery returned by $\mathcal{A}$ to be valid, $\mathcal{A}$ can only query upto $n-1$ secret keys within any epoch, meaning that the event $\overline{\mathsf{E}_1}$ is implied by $\mathsf{E}_{\mathcal{A}}$. Additionally, $\mathcal{A}$ is not allowed to query secret keys or signatures for $e > E$, hence, $\mathsf{E}_{\mathcal{A}}$ also implies $\overline{\mathsf{E}}_4$. We also have that $\mathcal{A}$ can win only if $\overline{\mathsf{E}}_5$ occurs. Combining the above, we get that,

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{SIG}_3,\mathcal{B}_1}(\lambda) &\geq \Pr\left[\overline{\mathsf{Abort}}\right] \\
&\geq \Pr\left[\overline{\mathsf{E}}_1 \wedge \overline{\mathsf{E}}_2 \wedge \overline{\mathsf{E}}_3 \wedge \overline{\mathsf{E}}_4 \wedge \overline{\mathsf{E}}_5\right] \\
&\geq \Pr\left[\overline{\mathsf{E}}_2 \wedge \overline{\mathsf{E}}_3 \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1)\right] \\
&\geq \frac{1}{E} \cdot \Pr\left[\overline{\mathsf{E}}_3 \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1)\right] \\
&\geq \frac{1}{E} \cdot \Pr\left[(e^* \leq E) \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1)\right]
\end{aligned}$$

This concludes the proof.

Lemma 4. *There exists an adversary $\mathcal{B}_2$ such that*

$$\Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1 \wedge e^* > E\right] \leq \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS},\mathcal{B}_2}(\lambda)$$

Proof (of Lemma 4). Consider the following adversary $\mathcal{B}_2$ playing the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{BLS}}$. $\mathcal{B}_2$ can issue signature, secret key and random oracle queries to its BLS challenger. To distinguish the random oracle in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{BLS}}$ from the one in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3}$, we denote queries by $\mathcal{B}_2$ to its challenger as $\hat{\mathsf{H}}(\cdot)$.

$\mathcal{B}_2$ invokes $\mathcal{A}$ and simulates the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{PRSIG}_3}$ as follows:

- Receive (n) from $\mathcal{A}$.
- Receive $\mathsf{pp}, \mathsf{pk} = (h), \mathsf{pkc} = \perp$ from its challenger. We use α to denote the secret key of the BLS challenger, i.e. $h = g^{\alpha}$. Note that α is unknown to $\mathcal{B}_2$.
- Send pp to $\mathcal{A}$, and receive E. Forward $\mathsf{pk}, \mathsf{pkc}, \mathsf{vk} = h$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries. $\mathcal{B}_2$ initializes the simulated oracles $\mathsf{H}_0(m) \leftarrow \perp$ for all messages, and $\mathsf{H}_1(i, e) \leftarrow \perp$ for all $i \in [n], e \in [E]$.

$\mathcal{B}_2$ maintains the following metadata: (a) a mapping $R : \mathcal{M} \to \mathbb{Z}_p$ to store auxiliary information for responding to H_0 queries, (b) a set $\mathcal{Q}^{\mathsf{sk}}_e$ for each $e \in [E]$, to store all the parties for which $\mathcal{A}$ calls the secret key oracle in epoch e and (c) a mapping $S_e : [n] \to \mathbb{Z}_p$ for each $e \in [E]$, to store auxiliary information for answering skO queries in epoch e.

We now discuss how $\mathcal{B}_2$ responds to each of $\mathcal{A}$'s queries.

– $\mathsf{H}_0(m)$. If $\mathsf{H}_0(m)$ has been determined, i.e. $\mathsf{H}_0(m) \neq \perp$, then $\mathcal{B}_2$ returns $\mathsf{H}_0(m)$. Otherwise, $\mathcal{B}_2$ samples $\delta_m \leftarrow\$ \mathbb{Z}_p$, sets $\mathsf{H}_0(m) \leftarrow g^{\delta_m}$, $R(m) \leftarrow \delta_m$ and returns $\mathsf{H}_0(m)$.

– $\mathsf{H}_1(i, e)$. $\mathcal{B}_2$ queries its challenger for $\hat{\mathsf{H}}(i, e)$, and returns this value.

– $\mathsf{skO}(i, e)$. If $e > E$, $\mathcal{B}_2$ aborts (note that this is in line with the security game, wherein $\mathcal{A}$ is not allowed to query secret keys or signatures for epochs $e > E$). Next, $\mathcal{B}_2$ adds i to $\mathcal{Q}_e^{\mathsf{sk}}$, and aborts if $|\mathcal{Q}_e^{\mathsf{sk}}| = n$.

  Otherwise, if $S_e(i) = \perp$, then $\mathcal{B}_2$ samples $S_e(i) \leftarrow\$ \mathbb{Z}_p$. $\mathcal{B}_2$ then queries its challenger for $\mathsf{SignO}((i, e))$ i.e. a signature on the tuple (i, e). We denote the response as $\sigma_{i,e}$. $\mathcal{B}_2$ responds to $\mathcal{A}$ with $(e, S_e(i), \sigma_{i,e})$. Note that this is distributed identically to a real secret key because (i) for an adversary that only sees up to $n-1$ secret keys in an epoch, α_i values appear uniformly random, (ii) $\sigma_{i,e} = \hat{\mathsf{H}}(i, e)^{\alpha} = \mathsf{H}_1(i, e)^{\alpha}$ is indeed the correct secret key.

– $\mathsf{SignO}(i, m, e)$. If $e > E$, $\mathcal{B}_2$ aborts. Otherwise, $\mathcal{B}_2$ calls $\mathsf{H}_0(m)$. Next, $\mathcal{B}_2$ queries its challenger for $\mathsf{SignO}((i, e))$ to get $\sigma_{i,e} \leftarrow \mathsf{H}_1(i, e)^{\alpha}$. $\mathcal{B}_2$ then samples $r_{i,e} \leftarrow\$ \mathbb{Z}_p$ and returns $(e, g^{r_{i,e}}, \mathsf{H}_0(m)^{r_{i,e}} \cdot \sigma_{i,e})$. This is indeed a valid signature on m since $\sigma_{i,e}$ is the correct secret key for party i in epoch e.

Eventually, $\mathcal{A}$ outputs a forgery (m^*, σ^*) where $\sigma^* = (e^*, \sigma_0^*, \sigma_1^*, \mathcal{J}^*)$. Without loss of generality, we assume that $\mathcal{A}$ queried $\mathsf{H}_0(m^*)$ before outputting the forgery, meaning that $R(m^*) \neq \perp$.

$\mathcal{B}_2$ aborts if $\mathcal{A}$ does not return a valid forgery, i.e. if $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ or for some $e \in [E]$, $\mathsf{Trace}(m^*, \sigma^*) \subseteq (\mathcal{Q}_e^{\mathsf{sk}} \cup \mathcal{Q}_e^{\mathsf{sig}}(m^*))$. $\mathcal{B}_2$ also aborts if $e^* \leq E$.

Otherwise, since $\mathcal{A}$ returns a valid forgery and $e^* > E$, $\mathcal{A}$ could not have queried $\mathsf{skO}(i, e^*)$ or $\mathsf{SignO}(i, \cdot, e^*)$ for any i. Specifically, this means that $\mathcal{B}_2$ never asked its BLS challenger for a signature on (i, e^*) for any $i \in \mathcal{J}^*$. Let $\mathcal{J}^* = i_1, \ldots, i_{|\mathcal{J}^*|}$. Then, $\mathcal{B}_2$ queries its BLS challenger for $\sigma_{i_2,e^*} \leftarrow \mathsf{SignO}((i_2, e^*)), \ldots, \sigma_{i_{|\mathcal{J}^*|},e^*} \leftarrow \mathsf{SignO}((i_{|\mathcal{J}^*|}, e^*))$. $\mathcal{B}_2$ returns

$$\left((i_1, e^*), \frac{\sigma_1^*}{(\sigma_0^*)^{R(m^*)} \cdot \prod_{j \in \mathcal{J}^* \setminus \{i_1\}} \sigma_{j,e^*}}\right)$$

We claim that this is a valid BLS forgery. To prove that, we first observe that $\mathcal{B}_2$ never queried its challenger for a signature on (i_1^*, e^*). Secondly, if $\mathcal{A}$ returned a valid forgery, then we have

$$e(\sigma_1^*, g) = e(\mathsf{H}_0(m^*), \sigma_0^*) \cdot e\left(\prod_{j \in \mathcal{J}^*} \mathsf{H}_1(j, e^*), h\right)$$

We now use the above equation to get that,

$$
\begin{aligned}
e\left(\frac{\sigma_1^*}{(\sigma_0^*)^{R(m^*)} \cdot \prod_{j\in\mathcal{J}^*\setminus\{i_1\}} \sigma_{j,e^*}}, g\right) &= \frac{e(\sigma_1^*, g)}{e((\sigma_0^*)^{R(m^*)}, g) \cdot e\left(\prod_{j\in\mathcal{J}^*\setminus\{i_1\}} \sigma_{j,e^*}, g\right)} \\
&= \frac{e(\mathsf{H}_0(m^*), \sigma_0^*) \cdot e\left(\prod_{j\in\mathcal{J}^*} \mathsf{H}_1(j, e^*), h\right)}{e((\sigma_0^*)^{R(m^*)}, g) \cdot e\left(\prod_{j\in\mathcal{J}^*\setminus\{i_1\}} \sigma_{j,e^*}, g\right)} \\
&= \frac{e(g^{R(m^*)}, \sigma_0^*) \cdot e\left(\prod_{j\in\mathcal{J}^*} \mathsf{H}_1(j, e^*), h\right)}{e((\sigma_0^*)^{R(m^*)}, g) \cdot e\left(\prod_{j\in\mathcal{J}^*\setminus\{i_1\}} \sigma_{j,e^*}, g\right)} \\
&= \frac{\prod_{j\in\mathcal{J}^*} e(\mathsf{H}_1(j, e^*), h)}{\prod_{j\in\mathcal{J}^*\setminus\{i_1\}} e(\sigma_{j,e^*}, g)} \\
&= \frac{\prod_{j\in\mathcal{J}^*} e(\hat{\mathsf{H}}(j, e^*), h)}{\prod_{j\in\mathcal{J}^*\setminus\{i_1\}} e(\hat{\mathsf{H}}(j, e^*), h)} \qquad (5) \\
&= e(\hat{\mathsf{H}}(i_1, e^*), h) \qquad (6)
\end{aligned}
$$

Equation 5 follows from the validity of signatures returned by the BLS challenger and from our programming of the H_1 oracle – specifically, $\mathsf{H}_1(j, e) = \hat{\mathsf{H}}(j, e)$ for any j, e.

The above equation implies that the forgery returned by $\mathcal{B}_2$ is indeed valid. Let Abort denote the event in which $\mathcal{B}_2$ aborts prematurely. Let $\mathsf{E}_{\mathcal{A}}$ denote the event where $\mathcal{A}$ returns a valid forgery. Let E_1 be the event where $\mathcal{B}_2$ aborts due to $|\mathcal{Q}_e^{\mathsf{sk}}| = n$ for some e. Note that $\overline{\mathsf{E}}_1$ is contained in the event $\mathsf{E}_{\mathcal{A}}$, because $\mathcal{A}$ can return a valid forgery only if it queries less than n secret keys in each epoch. Next, let E_2 denote the event where $\mathcal{B}_2$ aborts due to $e^* \leq E$. Let E_3 be the event where $\mathcal{B}_2$ aborts due to a secret key or a signing oracle query for some $e > E$. $\overline{\mathsf{E}}_3$ is contained in the event $\mathsf{E}_{\mathcal{A}}$, because the adversary can win only if it never makes such a query. Let E_4 be the event that $\mathcal{B}_2$ aborts because $\mathcal{A}$ does not output a valid forgery.

Then by the above discussion, we have that, the event $\overline{\mathsf{Abort}}$ implies that $\mathcal{B}_2$ wins its game. Combining the above, we get that

$$
\begin{aligned}
\Pr\left[\mathbf{G}^{\text{uf}}_{\mathsf{BLS},\mathcal{B}_2}(\lambda) = 1\right] &\geq \Pr\left[\overline{\mathsf{Abort}}\right] \\
\mathsf{Adv}^{\text{uf}}_{\mathsf{BLS},\mathcal{B}_2}(\lambda) &\geq \Pr\left[\overline{\mathsf{E}}_1 \wedge \overline{\mathsf{E}}_2 \wedge \overline{\mathsf{E}}_3 \wedge \overline{\mathsf{E}}_4\right] \\
&\geq \Pr\left[\overline{\mathsf{E}}_2 \wedge (\mathbf{G}^{\text{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1)\right] \\
&\geq \Pr\left[(e^* > E) \wedge (\mathbf{G}^{\text{uf}}_{\mathsf{PRSIG}_3,\mathcal{A}}(\lambda) = 1)\right]
\end{aligned}
$$

This proves the lemma.

C Deferred Proofs

C.1 Proof of Theorem 2

Proof (of Theorem 2). Consider the following adversary $\mathcal{B}$ playing the game $\mathbf{G}^{\text{n-bdh}}_{\mathcal{G},n_{\max}}$. It gets input $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{u_i, u_{i,2}\}_{i\in[n_{\max}]})$ from its challenger. $\mathcal{B}$ invokes $\mathcal{A}$ and simulates the game $\mathbf{G}^{\text{sa-uf}}_{\mathsf{SIG}_2}$ as follows:

– Receive n from $\mathcal{A}$.
– Send $\mathsf{pp}_g \leftarrow (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2, p, \{u_i\}_{i\in[n]}, n)$ to $\mathcal{A}$.
– Receive (m^*, i^*) from $\mathcal{A}$. Forward i^* to its challenger.
– Receive $(u_{i^*,2}, g_{a,1} = g_1^a, g_{a,2} = g_2^a, g_{c,1} = g_1^c, g_{c,2} = g_2^c, \{(u_{i,a} = u_i^a, u_{i,c} = u_i^c)\}_{i\in[n]\setminus\{i^*\}})$ from its challenger.
– Sample $\beta_0 \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $v_0 \leftarrow g_{a,1}$ and $h_0 \leftarrow g_{a,1}^{-m^*} g_1^{\beta_0}$. Send $\mathsf{pp} \leftarrow (\mathsf{pp}_g, v_0, h_0)$ to $\mathcal{A}$.
– Set $\mathsf{pk}^*_{i^*,0} \leftarrow g_{a,2}$. This means that the secret key α_{i^*} is a, but this value is unknown to $\mathcal{B}$.
– Send $\mathsf{pk}^* \leftarrow (\mathsf{pk}^*_{i^*,0}, \{u_{i,a}\}_{i\in[n]\setminus\{i^*\}})$ to $\mathcal{A}$. Note that this is a valid public key because, for any $j \in [n] \setminus \{i^*\}$, $\mathsf{pk}_{i^*,j} = u_j^{\alpha_{i^*}} = u_j^a = u_{j,a}$.

Next, $\mathcal{A}$ issues a sequence of signing queries. We now discuss how $\mathcal{B}$ responds to $\mathcal{A}$'s queries:

$\mathsf{SignO}(m)$. If $m = m^*$, then $\mathcal{B}$ aborts. Note that since m^* is the forgery message, $\mathcal{A}$ will lose its game if it queries a signature on m^*.

Otherwise, $\mathcal{B}$ samples $r_m \leftarrow\!\!\$\, \mathbb{Z}_p$, and generates a signature as follows:

$$(\sigma_{m,0} \leftarrow g_2^r \cdot u_{i^*,2}^{\frac{-1}{m-m^*}}, \sigma_{m,1} \leftarrow (v_0^m h_0)^r \cdot u_{i^*}^{\frac{-\beta_0}{m-m^*}})$$

We claim that this signature is distributed identically to a real signature. To see this, let $\tilde{r} \leftarrow r - \frac{b}{m-m^*}$ where $g_1^b = u_{i^*}$ (which implies $u_{i^*,2} = g_2^b$ by definition). Then,

$$\begin{aligned}\sigma_{m,0} &= g_2^r \cdot u_{i^*,2}^{\frac{-1}{m-m^*}} \\ &= g_2^r \cdot g_2^{\frac{-b}{m-m^*}} \\ &= g_2^{\tilde{r}}\end{aligned}$$

$$\begin{aligned}\sigma_{m,1} &= (v_0^m h_0)^{\tilde{r}} \cdot (v_0^m h_0)^{\frac{b}{m-m^*}} \cdot u_{i^*}^{\frac{-\beta_0}{m-m^*}} \\ &= (v_0^m h_0)^{\tilde{r}} \cdot (g_{a,1}^{m-m^*} g_1^{\beta_0})^{\frac{b}{m-m^*}} \cdot g_1^{\frac{-b\beta_0}{m-m^*}} \\ &= (v_0^m h_0)^{\tilde{r}} \cdot g_{a,1}^b\end{aligned}$$

Next, observe that $\mathsf{sk}_{i^*} = a$, and since $u_{i^*} = g_1^b$, we get that $u_{i^*}^a = g_1^{ab} = g_{a,1}^b$. This means that the above signature is of the form $(g_2^{\tilde{r}}, u_{i^*}^a \cdot (v_0^m h_0)^{\tilde{r}})$, and $\tilde{r}$ is uniform in $\mathbb{Z}_p$ as required. Hence, this is a valid response to the signing query.

Eventually, $\mathcal{A}$ outputs a list of public keys $(\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)$ and a forgery $(m^*, (\mathcal{J}^*, \sigma_0^*, \sigma_1^*))$. $\mathcal{B}$ aborts if $\mathsf{pk}^* \neq \mathsf{pk}_{i^*}^*$, or if $m^* \in \mathcal{Q}_{\mathsf{sig}}$ or if $i^* \notin \mathcal{J}^*$. Note that $\mathcal{A}$ would lose the game if any of these three conditions are true.

$\mathcal{B}$ then runs the key aggregation procedure to get $(\mathsf{pkc}^*, \mathsf{vk}^* \leftarrow \mathsf{KeyAgg}((\mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*))$, and aborts if either of pkc^* or vk^* are $\perp$ or if $\mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 0$. Observe that $\mathcal{A}$'s forgery is invalid if any of these conditions are true.

Lastly, $\mathcal{B}$ responds to its challenger with the following:

$$W = \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{\beta_0}, \sigma_0^*) \cdot \prod_{j\in[n]\setminus\{i^*\}} e(\mathsf{pk}_{j,i^*}^*, g_{c,2}) \cdot e(\prod_{i\in\mathcal{J}^*\setminus\{i^*\}} u_{i,c}, \mathsf{vk}^*)}$$

We claim that if $\mathcal{B}$ did not abort, then $\mathcal{B}$ wins the n-BDH game. To see this, observe the following: First, the forgery being valid implies that,

$$\begin{aligned}
e(\sigma_1^*, g_2) &= e(v_0^{m^*} h_0, \sigma_0^*) \cdot e(\prod_{j \in \mathcal{J}^*} u_j, \mathsf{vk}^*) \\
&= e(g_{a,1}^{m^*} g_{a,1}^{-m^*} g_1^{\beta_0}, \sigma_0^*) \cdot e(u_{i^*} \cdot \prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_j, g_{a,2} \cdot \prod_{i \in [n] \setminus \{i^*\}} \mathsf{pk}_{i,0}^*) \\
&= e(g_1^{\beta_0}, \sigma_0^*) \cdot e(u_{i^*}, g_{a,2}) \cdot e(u_{i^*}, \prod_{i \in [n] \setminus \{i^*\}} \mathsf{pk}_{i,0}^*) \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_j, \mathsf{vk}^*)
\end{aligned}$$

Next, since KeyAgg output a pkc^* that is not $\perp$, we know that all n public keys output by $\mathcal{A}$ are valid. Specifically, this means that for all $i \in [n] \setminus \{i^*\}$, we have that,

$$e(u_{i^*}, \mathsf{pk}_{i,0}^*) = e(\mathsf{pk}_{i,i^*}^*, g_2)$$

Combining the two equations, we get that,

$$e(u_{i^*}, g_{a,2}) = \frac{e(\sigma_1^*, g_2)}{e(g_1^{\beta_0}, \sigma_0^*) \cdot \prod_{i \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{i,i^*}^*, g_2) \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_j, \mathsf{vk}^*)}$$

Raising both sides to power c gives us:

$$\begin{aligned}
e(u_{i^*}, g_{a,2})^c &= \frac{e(\sigma_1^*, g_2)^c}{e(g_1^{\beta_0}, \sigma_0^*)^c \cdot \prod_{i \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{i,i^*}^*, g_2)^c \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_j, \mathsf{vk}^*)^c} \\
e(u_{i^*}, g_2^a)^c &= \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{\beta_0}, \sigma_0^*) \cdot \prod_{i \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{i,i^*}^*, g_{c,2}) \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_{j,c}, \mathsf{vk}^*)} \\
e(u_{i^*}, g_2)^{ac} &= \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{\beta_0}, \sigma_0^*) \cdot \prod_{i \in [n] \setminus \{i^*\}} e(\mathsf{pk}_{i,i^*}^*, g_{c,2}) \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{i^*\}} u_{j,c}, \mathsf{vk}^*)}
\end{aligned}$$

This proves that if $\mathcal{B}$ does not abort, then $\mathcal{B}$ responds correctly to its challenger. Let Abort be the event that $\mathcal{B}$ aborts. This gives us,

$$\mathsf{Adv}^{\text{n-bdh}}_{\mathcal{G}, n_{\max}, \mathcal{B}}(\lambda) = \Pr[\overline{\mathsf{Abort}}]$$

Next, observe that $\mathcal{B}$ aborts if one of the following conditions hold: (i) $\mathsf{pk}^* \neq \mathsf{pk}_{i^*}^*$, (ii) $m^* \in \mathcal{Q}_{\mathsf{sig}}$, or (iii) $i^* \notin \mathcal{J}^*$ or (iv) $\mathsf{pkc}^* = \perp$ or $\mathsf{vk}^* = \perp$ or (v) $\mathsf{Vf}(\mathsf{vk}^*, m^*, \sigma^*) = 0$ or (vi) $\mathcal{A}$ queries $\mathsf{SignO}(m^*)$. Since the adversary $\mathcal{A}$ can win its game only if none of these conditions hold, we get that,

$$\Pr[\overline{\mathsf{Abort}}] \geq \mathsf{Adv}^{\text{sa-uf}}_{\mathsf{SIG}_2, \mathcal{A}}(\lambda)$$

Hence we get,

$$\mathsf{Adv}^{\text{n-bdh}}_{\mathcal{G}, n_{\max}, \mathcal{B}}(\lambda) \geq \mathsf{Adv}^{\text{sa-uf}}_{\mathsf{SIG}_2, \mathcal{A}}(\lambda)$$

This proves the theorem.

C.2 Proof of Theorem 3

Proof (of Theorem 3). Consider the following adversary $\mathcal{B}$ playing the game $\mathbf{G}^{\text{co-bdh}}_{\mathcal{G}}$. On input the group description $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p)$ and $(g_{a,1} = g_1^a, g_{a,2} = g_2^a, g_{b,1} = g_1^b, g_{b,2} = g_2^b, g_{c,1} = g_1^c, g_{c,2} = g_2^c)$ from its challenger, $\mathcal{B}$ invokes $\mathcal{A}$ and simulates the game $\mathbf{G}^{\text{uf}}_{\mathsf{SIG}_3}$ as follows:

1. Receive (n) from $\mathcal{A}$.
2. Set $\mathsf{pp} \leftarrow (n, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e, p)$.
3. Guess $j^* \leftarrow\!\$\ [n]$. Sample $\beta_1, \ldots, \beta_{j^*-1}, \beta_{j^*+1}, \ldots, \beta_n \leftarrow\!\$\ \mathbb{Z}_p$.
4. Set $h \leftarrow g_{a,2}$ and send $\mathsf{pp}, \mathsf{pk} = h, \mathsf{pkc} = \perp, \mathsf{vk} = h$ to $\mathcal{A}$. This means that, the value α is equal to a, and is unknown to $\mathcal{B}$.

Next, $\mathcal{A}$ issues a sequence of queries. We use q_S, q_H to denote a bound on the number of signing queries and random oracle queries on H_0 respectively.

$\mathcal{B}$ initializes the simulated oracle $\mathsf{H}_1(j) \leftarrow \perp$ for all j, and $\mathsf{H}_0(m) \leftarrow \perp$ for all values of m. $\mathcal{B}$ also samples a random value $\delta \leftarrow\!\$\ \mathcal{M}$.

$\mathcal{B}$ maintains two mappings $R : \mathcal{M} \to \mathbb{Z}_p$ and $R' : \mathcal{M} \to \{0,1\}$ to track auxiliary information used to answer H_0 queries. Both R and R' are initialized with $\perp$ for all messages.

For signing queries, $\mathcal{B}$ stores a list $\mathcal{Q}^{j^*}_{\mathsf{sig}}$, to store all the messages m for which $\mathcal{A}$ queries $\mathsf{SignO}(j^*, m)$. This list is initialized with $\perp$.

We now discuss how $\mathcal{B}$ responds to each of $\mathcal{A}$'s queries:

- $\mathsf{H}_0(m)$. If $m = \delta$, then $\mathcal{B}$ aborts. Otherwise, if $\mathsf{H}_0(m)$ has been determined, i.e. if $\mathsf{H}_0(m) \neq \perp$, then, $\mathcal{B}$ returns the value $\mathsf{H}_0(m)$. If not, $\mathcal{B}$ samples $x_m \leftarrow\!\$\ [q_S + 1]$ and $\gamma_m \leftarrow\!\$\ \mathbb{Z}_p$ uniformly randomly. We use x_m as a biased coin that is equal to 1 with probability $1/(q_S + 1)$.
  - If $x_m \neq 1$, then it sets $R(m) \leftarrow \gamma_m$, $R'(m) \leftarrow 0$ and $\mathsf{H}_0(m) \leftarrow g_1^{\gamma_m} \cdot g_{a,1}^{m-\delta}$.
  - If $x_m = 1$, it sets $R(m) \leftarrow \gamma_m$, $R'(m) \leftarrow 1$ and $\mathsf{H}_0(m) \leftarrow g_1^{\gamma_m}$.

  $\mathcal{B}$ returns $\mathsf{H}_0(m)$ to $\mathcal{A}$ and continues the game. Note that since γ_m is sampled randomly from $\mathbb{Z}_p$, the value $\mathsf{H}_0(m)$ will be indistinguishable from uniformly random for $\mathcal{A}$.
- $\mathsf{H}_1(j)$. If $j \neq j^*$, then $\mathcal{B}$ returns $g_1^{\beta_j}$. Otherwise, $\mathcal{B}$ returns $g_{b,1}$. Note that since all the β_j values are sampled randomly, H_1 will be indistinguishable from random for $\mathcal{A}$.
- $\mathsf{skO}(j)$. If $j = j^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ responds with $g_{a,1}^{\beta_j}$. This is the correct secret key because, for any $j \neq j^*$, $\mathsf{H}_1(j) = g_1^{\beta_j}$, meaning that $\mathsf{sk}_j = \mathsf{H}_1(j)^\alpha = (g_1^{\beta_j})^a = (g_1^a)^{\beta_j} = g_{a,1}^{\beta_j}$.
- $\mathsf{SignO}(j, m)$. We first call $\mathsf{H}_0(m)$. At this point we can assume that $m \neq \delta$, since otherwise $\mathcal{B}$ would have aborted when querying for $\mathsf{H}_0(m)$. Next,
  - For any $j \neq j^*$, $\mathcal{B}$ samples $r \leftarrow\!\$\ \mathbb{Z}_p$, and returns $(g_2^r, g_{a,1}^{\beta_j} \cdot \mathsf{H}_0(m)^r)$ to $\mathcal{A}$. As pointed out earlier, $\mathsf{sk}_j = g_{a,1}^{\beta_j}$, meaning that this signature is distributed identically to a real signature.
  - For $j = j^*$, $\mathcal{B}$ aborts if $R'(m) = 1$. Otherwise, $\mathcal{B}$ samples $r \leftarrow\!\$\ \mathbb{Z}_p$ and generates a signature as follows:

    $$(\sigma_{j^*,m,0} \leftarrow g_2^r \cdot g_{b,2}^{\frac{-1}{m-\delta}}, \sigma_{j^*,m,1} \leftarrow \mathsf{H}_0(m)^r \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}})$$

    Here, $R(m)$ is the value γ_m that was sampled by $\mathcal{B}$ when responding to $\mathsf{H}_0(m)$ query. Note that $R(m)$ cannot be $\perp$ since we always query $\mathsf{H}_0(m)$ implicitly during any signing query on m.

    We claim that this signature is distributed identically to a real signature. To see this, let $\tilde{r} \leftarrow r - \frac{b}{m-\delta}$. Then,

$$\begin{aligned}\sigma_{j^*,m,0} &= g_2^r \cdot g_{b,2}^{\frac{-1}{m-\delta}} \\ &= g_2^r \cdot g_2^{\frac{-b}{m-\delta}} \\ &= g_2^{r-\frac{b}{m-\delta}}\end{aligned}$$

$$\begin{aligned}\sigma_{j^*,m,1} &= \mathsf{H}_0(m)^r \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}} \\ &= \mathsf{H}_0(m)^{\tilde{r}} \mathsf{H}_0(m)^{\frac{b}{m-\delta}} \cdot g_{b,1}^{\frac{-R(m)}{m-\delta}} \\ &= \mathsf{H}_0(m)^{\tilde{r}} (g_1^{\gamma_m} \cdot g_{a,1}^{m-\delta})^{\frac{b}{m-\delta}} \cdot g_{b,1}^{\frac{-\gamma_m}{m-\delta}} \\ &= \mathsf{H}_0(m)^{\tilde{r}} g_1^{\frac{b\gamma_m}{m-\delta}} g_{a,1}^b \cdot g_{b,1}^{\frac{-\gamma_m}{m-\delta}} \\ &= \mathsf{H}_0(m)^{\tilde{r}} g_1^{\frac{b\gamma_m}{m-\delta}} g_1^{ab} \cdot g_1^{\frac{-b\gamma_m}{m-\delta}} \\ &= \mathsf{H}_0(m)^{\tilde{r}} \cdot g_1^{ab}\end{aligned}$$

Next, observe that $\mathsf{sk}_{j^*} = \mathsf{H}_1(j^*)^a$, and since $\mathsf{H}_1(j^*) = g_{b,1}$, we get that $\mathsf{sk}_{j^*} = g_{b,1}^a = g_1^{ab}$. This means that, the above signature is of the form $(g_2^{\tilde{r}}, \mathsf{H}_0(m^*)^{\tilde{r}} \cdot \mathsf{sk}_{j^*})$, and $\tilde{r}$ is uniform in $\mathbb{Z}_p$ as required. Hence, it is a valid response to the signing query. Lastly, $\mathcal{B}$ adds m to the list $\mathcal{Q}_{\mathsf{sig}}^{j^*}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, (\mathcal{J}^*, \sigma_0^*, \sigma_1^*))$. $\mathcal{B}$ aborts if $j^* \notin \mathcal{J}^*$, or if $m^* \in \mathcal{Q}_{\mathsf{sig}}^{j^*}$. $\mathcal{B}$ also aborts if $R'(m^*) = 0$ or if $\mathcal{A}$ does not return a valid forgery, i.e. if $\mathsf{Vf}(\mathsf{vk}, m^*, \sigma^*) = 0$ or $\mathsf{Trace}(m^*, \sigma^*) \subseteq \mathcal{Q}^{\mathsf{sk}} \cup \mathcal{Q}^{\mathsf{sig}}(m^*)$. Otherwise, $\mathcal{B}$ responds with the following expression:

$$\frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{R(m^*)}, \sigma_0^*) \cdot e(g_{c,1}^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, h)}$$

We claim that if $\mathcal{B}$ does not abort, then $\mathcal{B}$ wins the co-bdh game. To see this, we note that the forgery being valid implies the following:

$$\begin{aligned}e(\sigma_1^*, g_2) &= e(\mathsf{H}_0(m^*), \sigma_0^*) \cdot e(\prod_{j \in \mathcal{J}^*} \mathsf{H}_1(j), h) && (7)\\ &= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(\prod_{j \in \mathcal{J}^*} \mathsf{H}_1(j), g_2^a) && (8)\\ &= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1} \prod_{j \in \mathcal{J}^* \setminus \{j^*\}} \mathsf{H}_1(j), g_2^a) && (9)\\ &= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1}, g_{a,2}) \cdot e(\prod_{j \in \mathcal{J}^* \setminus \{j^*\}} \mathsf{H}_1(j), g_{a,2}) && (10)\\ &= e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_{b,1}, g_{a,2}) \cdot e(g_1^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2}) && (11)\end{aligned}$$

Equation 7 follows from the validity of the forgery. Equation 8 uses the fact that $h = g_{a,2}$ and $\mathsf{H}_0(m^*) = g_1^{R(m^*)}$, since $R'(m^*) = 1$. Next, since $j^* \in \mathcal{J}^*$ and $\mathsf{H}_1(j^*) = g_{b,1}$, we get Equation 9. Equation 11 follows from the fact that $\mathcal{B}$ programmed $\mathsf{H}_1(j)$ to be $g_1^{\beta_j}$ for all $j \neq j^*$.

The above equation implies that

$$e(g_{b,1}, g_{a,2}) = \frac{e(\sigma_1^*, g_2)}{e(g_1^{R(m^*)}, \sigma_0^*) \cdot e(g_1^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})}$$

Raising both sides to the power c, we get,

$$\begin{aligned}
e(g_{b,1}, g_{a,2})^c &= \frac{e(\sigma_1^*, g_2)^c}{e(g_1^{R(m^*)}, \sigma_0^*)^c \cdot e(g_1^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})^c} \\
e(g_{b,1}, g_{a,2})^c &= \frac{e(\sigma_1^*, g_2^c)}{e(g_1^{cR(m^*)}, \sigma_0^*) \cdot e(g_1^{c \cdot \Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})} \\
e(g_{b,1}, g_{a,2})^c &= \frac{e(\sigma_1^*, g_2^c)}{e(g_{c,1}^{R(m^*)}, \sigma_0^*) \cdot e(g_{c,1}^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})} \\
e(g_1^b, g_2^a)^c &= \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{R(m^*)}, \sigma_0^*) \cdot e(g_{c,1}^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})} \\
e(g_1, g_2)^{abc} &= \frac{e(\sigma_1^*, g_{c,2})}{e(g_{c,1}^{R(m^*)}, \sigma_0^*) \cdot e(g_{c,1}^{\Sigma_{j \in \mathcal{J}^* \setminus \{j^*\}} \beta_j}, g_{a,2})}
\end{aligned}$$

This proves that if $\mathcal{B}$ does not abort, then it responds correctly to its challenger.

Let Abort denote the event in which $\mathcal{B}$ aborts prematurely. This means that,

$$\Pr\left[\mathbf{G}^{\text{co-bdh}}_{\mathcal{G},\mathcal{B}}(\lambda) = 1\right] = \Pr\left[\overline{\mathsf{Abort}}\right]$$

We now analyse the abort probability. Let E_1 be the event where $\mathcal{B}$ aborts due to a H_0 query on δ. Let E_2 be the event where $\mathcal{B}$ guessed j^* correctly, i.e. (a) $j^* \in \mathcal{J}$ and (b) $\mathcal{A}$ never queries $\mathsf{skO}(j^*)$ and (c) $m^* \notin \mathcal{Q}_{\mathsf{sig}}^{j^*}$. Let E_3 be the event where $\mathcal{B}$ aborts during a signing query because $R'(m) = 1$. Let E_4 be the event where $\mathcal{B}$ aborts due to $R'(m^*) = 0$. Let E_5 be the event where $\mathcal{B}$ aborts because $\mathcal{A}$ outputs an invalid forgery.

Next, observe that $\Pr\left[\overline{\mathsf{E}_1}\right] = \left(1 - \frac{1}{p}\right)^{q_S + q_H} \geq 1 - \frac{(q_S + q_H)}{p}$, since $\mathcal{B}$ implicitly queries $\mathsf{H}_0(m)$ whenever $\mathcal{A}$ queries $\mathsf{SignO}(\cdot, m)$. For E_2, we note that, for $\mathcal{A}$ to produce a valid forgery, there has to be at least one party $\hat{j}$, such that $\hat{j} \in \mathcal{J}^*$ but $\mathcal{A}$ never queries $\mathsf{skO}(\hat{j})$ or $\mathsf{SignO}(\hat{j}, m^*)$. Hence, $\Pr\left[\mathsf{E}_2 \mid \overline{\mathsf{E}_5}\right] \geq \frac{1}{n}$. $\Pr\left[\overline{\mathsf{E}_3} \mid \overline{\mathsf{E}_1}\right] = \left(1 - \frac{1}{q_S + 1}\right)^{q_S} \geq \frac{1}{e}$, since $R'(m)$ is set to 1 with a probability $\frac{1}{q_S+1}$ for all m. Similarly, we get $\Pr\left[\overline{\mathsf{E}_4} \mid \overline{\mathsf{E}_1}\right] = \frac{1}{q_S+1}$.

Combining the above, we get that,

$$\begin{aligned}
\Pr\left[\overline{\mathsf{Abort}}\right] &= \Pr\left[\overline{\mathsf{E}_1} \wedge \mathsf{E}_2 \wedge \overline{\mathsf{E}_3} \wedge \overline{\mathsf{E}_4} \wedge \overline{\mathsf{E}_5}\right] \\
&= \Pr\left[\mathsf{E}_2 \mid \overline{\mathsf{E}_5}\right] \cdot \Pr\left[\overline{\mathsf{E}_5}\right] \cdot \Pr\left[\overline{\mathsf{E}_1} \wedge \overline{\mathsf{E}_3} \wedge \overline{\mathsf{E}_4}\right] \\
&= \Pr\left[\mathsf{E}_2 \mid \overline{\mathsf{E}_5}\right] \cdot \Pr\left[\overline{\mathsf{E}_5}\right] \cdot \Pr\left[\overline{\mathsf{E}_1}\right] \cdot \Pr\left[\overline{\mathsf{E}_3} \mid \overline{\mathsf{E}_1}\right] \cdot \Pr\left[\overline{\mathsf{E}_4} \mid \overline{\mathsf{E}_1}\right] \\
&\geq \frac{1}{n} \cdot \mathsf{Adv}^{\text{uf}}_{\mathsf{SIG}_3,\mathcal{A}}(\lambda) \cdot \left(1 - \frac{(q_S + q_H)}{p}\right) \cdot \frac{1}{e} \cdot \frac{1}{q_S + 1}
\end{aligned}$$

Lastly, since we assume that $2(q_S + q_H) < p$, we get that

$$\left(1 - \frac{(q_S + q_H)}{p}\right) \geq \frac{1}{2}$$

. This combined with the above equation completes the proof.

C.3 Proof of Theorem 4

General Forking Lemma We restate the general forking lemma from [BN06], since it is used in the unforgeability proof of the scheme LSIG_3.

Lemma 5. *Let Q be a number of queries and C be a set of size > 2. Let $\mathcal{B}$ be a randomized algorithm that on input $x, h_1, \ldots, h_Q$ returns an index $i \in [0, Q]$ and a side output out. Let IGen be an input generator. Let $\mathcal{F}_{\mathcal{B}}$ be a forking algorithm that works as in Fig. 10 given x as input and given black-box access to $\mathcal{B}$. Suppose the following probabilities:*

$$\mathrm{acc} := \Pr[i \neq 0 : x \leftarrow\!\$\, \mathsf{IGen}(1^\lambda); h_1, \ldots, h_Q \leftarrow\!\$\, C; (i, out) \leftarrow\!\$\, \mathcal{B}(x, h_1, \ldots, h_Q)]$$

$$\mathrm{frk} := \Pr[b = 1 : x \leftarrow\!\$\, \mathsf{IGen}(1^\lambda); (b, out, out') \leftarrow\!\$\, \mathcal{F}_{\mathcal{B}}(x)]$$

Then,

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

The forking algorithm $\mathcal{F}_{\mathcal{B}}$

On input x,

1. Pick a random coin ρ for $\mathcal{B}$ and sample $h_1, \ldots, h_Q \leftarrow\!\$\, C$.
2. $(i, out) \leftarrow\!\$\, \mathcal{B}(x, h_1, \ldots, h_Q; \rho)$. If $i = 0$ then output $(0, \perp, \perp)$.
3. Sample $h'_i, \ldots, h'_Q \leftarrow\!\$\, C$.
4. Then run $(i', out') \leftarrow\!\$\, \mathcal{B}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h'_Q; \rho)$.
5. If $i = i'$ and $h_i \neq h'_i$ then output $(1, out, out')$, otherwise output $(0, \perp, \perp)$.

Fig. 10. The forking algorithm $\mathcal{F}_{\mathcal{B}}$

Proof (of Theorem 4). We first construct an algorithm $\mathcal{B}$ around $\mathcal{A}$ that simulates the behavior of the challenger in the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{LSIG}_3}$. Then, we invoke the forking algorithm $\mathcal{F}_{\mathcal{B}}$ from Lemma 5 to obtain two forgeries with distinct challenges, which allow to construct a solution to MSIS or break binding of the commitment scheme Com. We now discuss how to realize this via several intermediate hybrids, as in [DOTT21].

Let q_H, q_S denote an upper bound on the number of random oracle and signing queries by $\mathcal{A}$ respectively.

$\mathbf{G}_0$. $\mathcal{B}$ gets as input $\{h_i \in C\}_{i \in [q_H + q_S + 1]}$ along with $(ck \in S_{ck}, \boldsymbol{A} \in R^{k \times \ell}, \boldsymbol{t})$. We will discuss where these inputs come from later in the proof.

$\mathcal{B}$ receives n from $\mathcal{A}$, and runs $(n, cpp) = \mathsf{pp} \leftarrow\$ \mathsf{Setup}(1^\lambda, n)$. $\mathcal{B}$ samples $i^* \leftarrow\$ [n]$, as its guess for the party that $\mathcal{A}$ will try to blame in its forgery.

Next, for all $i \in [n]$, it samples $\boldsymbol{t}_i \leftarrow\$ R^k$ and sets $\mathsf{H}_1(i) \leftarrow \boldsymbol{t}_i$. It then runs $(\mathsf{pk}, \mathsf{pkc}, \mathsf{vk}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow\$ \mathsf{KeyGen}()$, and sends $(\mathsf{pp}, \mathsf{pk}, \mathsf{pkc}, \mathsf{vk})$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries. $\mathcal{B}$ maintains hash tables to store auxiliary information about responses to these queries. Specifically, it maintains $T_{\mathsf{chal}} : \{0,1\}^* \to C$, $T_{\mathsf{td}} : \{0,1\}^* \to S_{td}$ and $T_{\mathsf{ck}} : \{0,1\}^* \to S_{ck}$. These are initialized with $\perp$ for all input values. $\mathcal{B}$ also maintains a counter ctr which is initialized with 0. We now show how $\mathcal{B}$ responds to $\mathcal{A}$'s queries:

- $\mathsf{H}_1(i)$: $\mathcal{B}$ responds with $\boldsymbol{t}_i$.
- $\mathsf{H}_{\mathsf{ck}}(x)$: If $T_{\mathsf{ck}}(x) = \perp$, it samples $T_{\mathsf{ck}}(x) \leftarrow\$ S_{ck}$, and then outputs $T_{\mathsf{ck}}(x)$.
- $\mathsf{H}_{\mathsf{chal}}(x)$: Parse x as $(m, \mathsf{pk}, \mathcal{J}, c, i)$. $\mathcal{B}$ first queries $\mathsf{H}_{\mathsf{ck}}(m, \mathsf{pk})$. If $T_{\mathsf{chal}}(x) = \perp$, then,
  - If $i \in \mathcal{J}$ and $i^* \in \mathcal{J}$, then, for all $i \in \mathcal{J} \setminus \{i^*\}$, set $T_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, c, i) \leftarrow\$ C$. Next, set $ctr \leftarrow ctr + 1$ and $T_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, c, i^*) \leftarrow h_{ctr}$.
  - Otherwise, set $T_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, c, i) \leftarrow\$ C$.

  $\mathcal{B}$ then returns $T_{\mathsf{chal}}(x)$.
- $\mathsf{skO}(i)$: If $i = i^*$ then $\mathcal{B}$ returns $(0, \perp)$ (and ends the simulation). Otherwise, $\mathcal{B}$ outputs sk_i.
- $\mathsf{Sign}_1\mathsf{O}(\cdot)$: $\mathcal{B}$ behaves exactly like the honest protocol.
- $\mathsf{Sign}_2\mathsf{O}(\cdot)$: $\mathcal{B}$ behaves exactly like the honest protocol.

When $\mathcal{A}$ outputs its forgery $m^*, \sigma^* = (\boldsymbol{z}^*, r^*, \hat{c}^*, \mathcal{J}^*)$ in the end, if $i^* \notin \mathcal{J}^*$ or if $i^* \in \mathcal{Q}^{\mathsf{sig}}(m^*)$, then $\mathcal{B}$ outputs $(0, \perp)$ and ends the simulation. Let $ck^* \leftarrow \mathsf{H}_{\mathsf{ck}}(m^*, \mathsf{pk})$, and $\{d_j^* \leftarrow \mathsf{H}_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, \hat{c}^*, j)\}_{j \in \mathcal{J}^*}$. Compute $\boldsymbol{w}^* \leftarrow \bar{\boldsymbol{A}}\boldsymbol{z}^* - \Sigma_{j \in \mathcal{J}^*} d_j^* \mathsf{H}_1(j)$. If $||\boldsymbol{z}^*||_2 > B$ or if $\mathsf{Com.Open}(ck^*, \hat{c}^*, \boldsymbol{w}^*; r^*) = 0$, $\mathcal{B}$ outputs $(0, \perp)$ and ends the game.

Otherwise, $\mathcal{B}$ finds index i_f such that $d_{i^*}^* = h_{i_f}$, and then outputs $(i_f, out = (\hat{c}^*, \{d_j^*\}_{j \in \mathcal{J}^* \setminus \{i^*\}}, d_{i^*}^*, \boldsymbol{z}^*, r^*, m^*, \mathcal{J}^*,$

For any $i \in \mathbb{N}$, let $\Pr[\mathbf{G}_i]$ be the probability that $\mathcal{B}$ does not output $(0, \perp)$ at the end of game $\mathbf{G}_i$.

Observe that, for the forgery to be valid, there must be at least one index $\hat{i} \in \mathcal{J}^*$ which is not in $\mathcal{Q}^{\mathsf{sk}} \cup \mathcal{Q}^{\mathsf{sig}}(m^*)$. Hence $\mathcal{B}$ will correctly guess i^* with probability at least $1/n$, which gives us:

$$\Pr[\mathbf{G}_0] = \frac{1}{n}\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{LSIG}_3, \mathcal{A}}(\lambda)$$

$\mathbf{G}_1$. This game is identical to $\mathbf{G}_0$ except at the following points.

- $\mathsf{H}_{\mathsf{ck}}(x)$: Parse x as (m, pk). If $T_{\mathsf{ck}}(x) = \perp$, then with probability ω, $\mathcal{B}$ computes $(tck, td) \leftarrow\$ \mathsf{Com.TCGen}(cpp)$, sets $T_{\mathsf{td}}(x) \leftarrow td$, $T_{\mathsf{ck}}(x) \leftarrow tck$. With probability $1 - \omega$, $\mathcal{B}$ sets $T_{\mathsf{ck}}(x) \leftarrow ck$ (ck is one of the inputs to $\mathcal{B}$). Finally, $\mathcal{B}$ sends $T_{\mathsf{ck}}(x)$ to $\mathcal{A}$.
- $\mathsf{Sign}_1\mathsf{O}(m, \mathcal{J}, i)$: If $i \neq i^*$, then, $\mathcal{B}$ simply executes the signing procedure as in the protocol. Otherwise, it first queries $\mathsf{H}_{\mathsf{ck}}(m, \mathsf{pk})$. If $T_{\mathsf{td}}(m, \mathsf{pk}) = \perp$ (i.e. TCGen was not called), $\mathcal{B}$ sets a flag bad_4 and halts with output $(0, \perp)$. Otherwise, let $td \leftarrow T_{\mathsf{td}}(m, \mathsf{pk})$. Then, instead of committing to $\boldsymbol{w}_{i^*}$, $\mathcal{B}$ does $c_{i^*} \leftarrow\$ \mathsf{Com.TCommit}(T_{\mathsf{ck}}(m, \mathsf{pk}), td)$. The rest of the Sign_1 protocol remains the same, i.e. $\mathcal{B}$ responds with c_{i^*}.
- $\mathsf{Sign}_2\mathsf{O}(\cdot)$: If $i \neq i^*$, then, $\mathcal{B}$ simply executes the signing procedure as in the protocol. Otherwise, after computing $\boldsymbol{z}_{i^*} \leftarrow d_{i^*}\mathsf{sk}_{i^*} + \boldsymbol{y}_{i^*}$, $\mathcal{B}$ derives the randomness for the trapdoor commitment: $r_{i^*} \leftarrow \mathsf{Com.Eqv}(T_{\mathsf{ck}}(m, \mathsf{pk}), T_{\mathsf{td}}(m, \mathsf{pk}), c_{i^*}, \boldsymbol{w}_{i^*})$.

When $\mathcal{A}$ outputs a forgery at the end of the game, then, similar to $\mathbf{G}_0$, $\mathcal{B}$ computes $ck^*, \{d_j^*\}_{j\in\mathcal{J}^*}$. If $\mathsf{Com.Open}(ck^*, \hat{c}^*, \boldsymbol{w}^*; r^*) = 0$ or $||\boldsymbol{z}^*||_2 > B$ then $\mathcal{B}$ halts and outputs $(0, \perp)$. If $T_{\mathsf{td}}(m^*, \mathsf{pk}) \neq \perp$, i.e. ($\mathsf{TCGen}$ was called for (m^*, pk)), then $\mathcal{B}$ sets a flag bad_5 and halts with output $(0, \perp)$. This means that, if $\mathcal{B}$ does not halt, then, $ck^* = ck = \mathsf{H}_{\mathsf{ck}}(m^*, \mathsf{pk})$.

Observe that the simulation is only successful if the random oracle H_{ck} internally uses a trapdoor commitment key for all but one query, and it uses a predefined key ck for the forgery query: (m^*, pk). In other words, it is successful if neither of bad_4 or bad_5 flags are set. Combining with the fact that the statistical distance between each commitment key in $\mathbf{G}_0$ and the corresponding trapdoor-based commitment key in $\mathbf{G}_1$ is bounded by ϵ_{td}, we get that,

$$\Pr[\mathbf{G}_1] \geq \omega^{q_H+q_S} \cdot (1-\omega) \cdot \Pr[\mathbf{G}_0] - (q_H + q_S) \cdot \epsilon_{td}$$

We set $\omega = \frac{q_H+q_S}{q_H+q_S+1}$, so we get:

$$\Pr[\mathbf{G}_1] \geq \frac{\Pr[\mathbf{G}_0]}{e(q_H + q_S + 1)} - (q_H + q_S) \cdot \epsilon_{td}$$

$\mathbf{G}_2$. The game is identical to $\mathbf{G}_1$ except for how $\mathcal{B}$ responds to signing queries for i^*:

$\mathcal{B}$ does not generate $\boldsymbol{z}_{i^*}$ honestly. Instead, in $\mathsf{Sign_1O}$, it simply samples a trapdoor commitment c_{i^*} (like in game $\mathbf{G}_1$) , and in $\mathsf{Sign_2O}$, it samples $\boldsymbol{z}_{i^*} \leftarrow\!\!\$\ D_s^{k+\ell}$ and derives randomness $r_{i^*} \leftarrow \mathsf{Com.Eqv}(T_{\mathsf{ck}}(m, \mathsf{pk}), T_{\mathsf{td}}(m, \mathsf{pk}), c_{i^*}, \bar{\boldsymbol{A}}\boldsymbol{z}_{i^*} - d_{i^*}\mathsf{H}_1(i^*))$. Here, d_{i^*} is the $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, c, i^*)$ value with $\mathcal{J}, c$ corresponding to this particular signing query. Then, with probability $1 - 1/M$, it aborts, and otherwise, it outputs s_{i^*} as defined in the protocol.

The signature simulated this way is statistically indistinguishable from that generated honestly, because rejection sampling ensures that the real signature is distributed identically to the gaussian distribution $D_s^{k+\ell}$. By Lemmas 3 and 4 from [DOTT21] (which extend Lemma 2), we get that

$$|\Pr[\mathbf{G}_2] - \Pr[\mathbf{G}_1]| \leq q_S \cdot \frac{2e^{-t^2}}{M}$$

$\mathbf{G}_3$. Observe that, signature queries on i^* for any message do not actually use the secret key of this party. We now make the following changes to $\mathcal{B}$:

Key Generation. Our overall goal is to embed a challenge commitment key ck and an instance of $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ which is denoted as $[\boldsymbol{A}'|\boldsymbol{I}]$ with $\boldsymbol{A}' \leftarrow\!\!\$\ R_q^{k\times(\ell+1)}$. In $\mathbf{G}_2$, the tuple $(\boldsymbol{A}, \mathsf{H}_1(i^*))$ (where $[\boldsymbol{A}|\boldsymbol{I}_k]$ is the public key) is uniformly distributed in $R_q^{k\times\ell} \times R_q^k$, so we can simply replace it with $\boldsymbol{A}'$, which can be regarded as $[\boldsymbol{A}|\mathsf{H}_1(i^*)]$. Additionally, the simulation of H_{ck} guarantees that ck follows the uniform distribution over S_{ck} which is perfectly indistinguishable from honestly generated commitment keys, since the keys are uniform.

More formally, given an MSIS instance $\boldsymbol{A}' = [\hat{\boldsymbol{A}}|\boldsymbol{t}]$ and a challenge commitment key ck, the inputs to $\mathcal{B}$ are $(ck, \hat{\boldsymbol{A}}, \boldsymbol{t})$. It then sets $\mathsf{pk} = [\hat{\boldsymbol{A}}|\boldsymbol{I}_k]$ (instead of running the KeyGen algorithm). Next, it sets $\mathsf{H}_1(i^*) \leftarrow \boldsymbol{t}$. For all $i \in [n] \setminus \{i^*\}$, it samples a short vector $\mathsf{sk}_i \leftarrow\!\!\$\ S_\eta^{k+\ell}$, and programs $\mathsf{H}_1(i) \leftarrow ([\hat{\boldsymbol{A}}|\boldsymbol{I}_k] \cdot \mathsf{sk}_i) \bmod q$. We set $\hat{\eta} = \eta$, so that by the tail bound in Lemma 1, the secret keys in $\mathbf{G}_2$ and this game have L_2 norm bounded by $\sqrt{\ell + k} \cdot \eta$ with high probability. Observe that if an adversary can distinguish between this game and $\mathbf{G}_2$, then we can use it to build an adversary $\mathcal{B}_1$ that can break MLWE.

The rest of the simulation remains the same (except for H_1 queries which are now answered based on the programming described above).

So we get that there exists an adversary $\mathcal{B}_1$ with the following advantage:

$$|\Pr[\mathbf{G}_3] - \Pr[\mathbf{G}_2]| \leq (n-1) \cdot \mathsf{Adv}_{\mathsf{MLWE}_{q,k,\ell,\eta},\mathcal{B}_1}(\lambda)$$

The proof is by a hybrid argument. Consider a series of games: $\mathbf{G}_{2,i}$ for $i \in [0, n-1]$, where in $\mathbf{G}_{2,i}$, the first i secret keys in $[n] \setminus \{i^*\}$ are sampled like in $\mathbf{G}_3$, and the remaining $(n-i-1)$ secret keys are sampled as in the game $\mathbf{G}_2$. Then, for each i, there exists an adversary $\mathcal{B}'_i$ that can break $\mathsf{MLWE}_{q,k,\ell,\eta}$ when given an adversary $\mathcal{B}_i$ that can distinguish between $\mathbf{G}_{2,i}$ and $\mathbf{G}_{2,i+1}$. Note that $\mathbf{G}_{2,0} = \mathbf{G}_2$ and $\mathbf{G}_{2,n-1} = \mathbf{G}_3$. So the adversary $\mathcal{B}_1$ simply samples $i \leftarrow\!\!\$\ [n-1]$ and invokes $\mathcal{B}'_i$ to decide on its output. Hence, the advantage of $\mathcal{B}_1$ is at least $1/(n-1)$ times the advantage of an adversary in distinguishing between $\mathbf{G}_2$ and $\mathbf{G}_3$.

We now prove the theorem by constructing $\mathcal{B}'$ around $\mathcal{B}$ that either (1) breaks binding of commitment with respect to ck, or (2) finds a solution to the $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ on input $\boldsymbol{A}' = [\hat{\boldsymbol{A}}|\boldsymbol{t}]$. $\mathcal{B}'$ invokes $\mathcal{F}_{\mathcal{B}}$ on input $(ck, \hat{\boldsymbol{A}}, \boldsymbol{t})$ from Lemma 5. With probability frk we get two forgeries, $out = (c^*, \{d_j^*\}_{j \in \mathcal{J}^* \setminus \{i^*\}}, d_{i^*}^*, \boldsymbol{z}^*, r^*, m^*, \mathcal{J}^*, ck^*)$ and $\hat{out} = (\hat{c}^*, \{\hat{d}_j^*\}_{j \in \hat{\mathcal{J}}^* \setminus \{\hat{i}^*\}}, \hat{d}_{\hat{i}^*}^*, \hat{\boldsymbol{z}}^*, \hat{r}^*, \hat{m}^*, \hat{\mathcal{J}}^*, \hat{ck}^*)$, where frk satisfies:

$$\Pr[\mathbf{G}_3] = \text{acc} \leq \frac{q_H + q_S + 1}{|C|} + \sqrt{(q_H + q_S + 1) \cdot \text{frk}}$$

By construction of $\mathcal{B}$ and $\mathcal{F}_{\mathcal{B}}$, we have that $\mathcal{A}$'s view is identical in the two executions until the point of forking i_f. Additionally, $\mathcal{B}$ samples i^*, samples sk_i for all $i \in [n] \setminus \{i^*\}$ and sets $\mathsf{H}_1(i)\ \forall\ i \in [n]$ before the forking point. Hence, we have $c^* = \hat{c}^*$, $\mathcal{J}^* = \hat{\mathcal{J}}^*$, $i^* = \hat{i}^*$, $d_j^* = \hat{d}_j^*$ for all $j \in \mathcal{J}^* \setminus \{i^*\}$, $m^* = \hat{m}^*$, and $ck^* = \hat{ck}^*$ because $\mathsf{H}_{\mathsf{ck}}(m^*, \mathsf{pk})$ is invoked right before $\mathsf{H}_{\mathsf{chal}}$ is programmed.

Since both forgeries are verified under the same commitment key ck, we have that, $||\boldsymbol{z}^*||_2 \leq B$ and $||\hat{\boldsymbol{z}}^*||_2 \leq B$. Moreover,

$$\mathsf{Com.Open}(ck, c^*, r^*, \bar{\boldsymbol{A}}\boldsymbol{z}^* - \Sigma_{j \in \mathcal{J}^* \setminus \{i^*\}} d_j^* \mathsf{H}_1(j) - d_{i^*}^* \boldsymbol{t}) = 1$$

$$\mathsf{Com.Open}(ck, c^*, \hat{r}^*, \bar{\boldsymbol{A}}\hat{\boldsymbol{z}}^* - \Sigma_{j \in \mathcal{J}^* \setminus \{i^*\}} d_j^* \mathsf{H}_1(j) - \hat{d}_{i^*}^* \boldsymbol{t}) = 1$$

There are two cases. If $\bar{\boldsymbol{A}}\boldsymbol{z}^* - d_{i^*}^* \boldsymbol{t} \neq \bar{\boldsymbol{A}}\hat{\boldsymbol{z}}^* - \hat{d}_{i^*}^* \boldsymbol{t}$ then $\mathcal{B}'$ breaks computational binding with respect to key ck, and can succeed only with probability $\leq \epsilon_{bind}$. If $\bar{\boldsymbol{A}}\boldsymbol{z}^* - d_{i^*}^* \boldsymbol{t} = \bar{\boldsymbol{A}}\hat{\boldsymbol{z}}^* - \hat{d}_{i^*}^* \boldsymbol{t}$, rearranging the terms gives us,

$$[\hat{\boldsymbol{A}}|\boldsymbol{I}_k|\boldsymbol{t}] \begin{bmatrix} \boldsymbol{z}^* - \hat{\boldsymbol{z}}^* \\ \hat{d}_{i^*}^* - d_{i^*}^* \end{bmatrix} = 0$$

Recall that $[\boldsymbol{A}'|\boldsymbol{I}_k] = [\hat{\boldsymbol{A}}|\boldsymbol{t}|\boldsymbol{I}_k]$ is an instance of the $\mathsf{MSIS}_{q,k,\ell+1,\beta}$ problem, we have found a valid solution if $\beta = \sqrt{2B^2 + 4\kappa}$. Putting the two cases together, we get that there exists an adversary $\mathcal{B}_2$ that uses $\mathcal{B}'$ as a sub-procedure such that,

$$\text{frk} \leq \epsilon_{bind} + \mathsf{Adv}_{\mathsf{MSIS}_{q,k,\ell+1,\beta},\mathcal{B}_2}(\lambda)$$

This proves the theorem.