One-Time and Interactive Aggregate Signatures from Lattices

Dan Boneh
Stanford University
dabo@cs.stanford.edu

Sam Kim
Stanford University
skim13@cs.stanford.edu

Abstract

An aggregate signature scheme enables one to aggregate multiple signatures generated by different people on different messages into a short aggregate signature. We construct two signature aggregation schemes whose security is based on the standard SIS problem on lattices in the random oracle model. Our first scheme supports public aggregation of signatures, but for a one-time signature scheme. Our second scheme supports aggregation of signatures in a many-time signature scheme, but the aggregation process requires interaction among the signers. In both schemes, the size of the aggregate signature is at most logarithmic in the number of signatures being aggregated.

1 Introduction

A signature scheme is a tuple of three algorithms: KeyGen generates a key pair $(\mathsf{sk}, \mathsf{pk})$, $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$ signs a message m, and $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma) \to 0/1$ verifies a signature σ on message m. A one-time signature is a signature scheme that is existentially unforgeable against an adversary that is given the public key and the signature on a *single* message of its choice.

One-time signatures (OTS) have found many applications in cryptography. They are used to transform any existentially unforgeable signature into a strongly unforgeable signature [BS08], in several chosen-ciphertext secure public-key encryption constructions [DDN03, CHK04], in offline/online signatures [EGM96], and for authenticating streaming data [GR01]. The classic OTS scheme of Lamport [Lam79] and its Winternitz variant [BDE+13] shows that one-time signature schemes can be constructed from any one-way function. Shorter one-time signatures can be constructed from algebraic assumptions, as we discuss below.

Aggregate signatures [BGLS03] are signature schemes with two additional algorithms: $\mathsf{SigAgg}(\mathsf{PK}, \mathsf{M}, \Sigma) \to \sigma_{\mathsf{ag}}$ and $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \to 0/1$. The signature aggregation algorithm SigAgg takes as input a vector of n public keys $\mathsf{PK} = (\mathsf{pk}_1, \dots, \mathsf{pk}_n)$, a vector of n messages $\mathsf{M} = (\mathsf{m}_1, \dots, \mathsf{m}_n)$, and a vector of n signatures $\Sigma = (\sigma_1, \dots, \sigma_n)$, and outputs a short aggregate signature σ_{ag}. This short aggregate signature convinces a verifier that all the signatures aggregated into σ_{ag} are valid. Specifically, if $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) = 1$ then the verifier should be convinced that sk_i was used to sign m_i, for all $i = 1, \dots, n$. Note that the aggregation algorithm can be run by anyone at any time after the signatures have been generated.

Aggregate signatures are used in blockchain systems, where aggregation can compress many signatures in a block into a short aggregate, thereby shrinking the size of the blockchain. They are also used in consensus protocols where many signatures from different block validators are compressed into a short aggregate.

A simple and efficient aggregate signature scheme is built from BLS signatures [BLS01] using pairings [BGLS03]. The Schnorr signature scheme supports a weaker form of aggregation, where the signers must interact with one another while generating the signatures to produce a short aggregate [BN06, MPSW18a, DEF+19]. We refer to this as *interactive aggregation*. If the signers need not be involved in aggregation then we say that aggregation is *non-interactive*. Recently, Doröz et al. proposed a lattice-based aggregate signature scheme, but where the aggregate signature size grows *linearly* with the number of signatures being aggregated [DHSS20].

It is currently an open problem to construct a lattice-based non-interactive aggregation scheme where the aggregate signature size is (almost) independent of the number of signatures being aggregated.

Our contributions. In this paper, we construct two lattice-based aggregate signature schemes whose security is based on the standard SIS assumption, and where the aggregate size grows at most logarithmically in the number of signatures begin aggregated.

Construction 1. Our first result (Section 4) is an aggregate *one-time* signature scheme, where the aggregation process is non-interactive. The starting point for this scheme is a discrete-log based one-time signature scheme due to Mihir Bellare and Sarah Shoup [BS08]. Their scheme uses a group $\mathbb{G}$ of order q with generator g and works as follows: the secret key is a random pair $(x, y) \in \mathbb{Z}_q^2$ and the public key is $(h_1 = g^x,\ h_2 = g^y) \in \mathbb{G}^2$. To sign a message m compute $c \leftarrow H(m)$ using a hash function $H : \mathcal{M} \to \mathbb{Z}_q$ and output $\sigma \leftarrow cx + y \in \mathbb{Z}_q$. A signature σ is valid if $g^\sigma = h_1^c h_2$. Bellare and Shoup prove that this scheme is a secure one-time signature assuming that the one-more discrete-log assumption [BNPS03] holds in $\mathbb{G}$.

We first observe in Section 3.2 that this scheme is aggregatable: it gives a very simple aggregate one-time signature scheme based on the standard discrete-log assumption in the random oracle model. It is tempting to aggregate signatures by simply adding them up, however, that is insecure due to a rogue public key attack discussed in Section 3.2. Instead, we use an aggregation mechanism due to Boneh, Drijvers, and Neven [BDN18b], and prove security of the resulting scheme based on the discrete-log assumption in the random oracle model.

We then adapt the scheme to the lattice settings. First, we point out that the one-time signature scheme of Lyubashevsky and Micciancio [LM08] has a similar structure to the Bellare-Shoup scheme, and can be similarly aggregated. However, proving security of the resulting lattice aggregate scheme is more difficult. We do so in Section 4, and obtain an aggregate one-time signature scheme based on the standard SIS problem, where aggregation does not require any help from the signers.

Both of these one-time aggregate signature schemes can be generalized to obtain an aggregate t-time secure scheme (supporting up to t signatures for each public key). The cost is a larger public key whose size grows linearly in t. The aggregate signature size does not change. However, the security reduction to the underlying hardness assumption becomes quite inefficient for a non-constant t.

We briefly note that even a one-time aggregate signature scheme can be used to shrink the Bitcoin blockchain. Recall that funds in the Bitcoin system [Nak08] are held in records called UTXOs (unspent transaction outputs). A UTXO has an associated value and contains a commitment to one or more public keys. To spend a UTXO one provides one or more digital signatures that verify under the public keys committed in the UTXO. Once spent, the UTXO cannot be spent again, and the funds are transferred to a different UTXO. Currently Bitcoin uses a many-time digital signature such as ECDSA. However, because every UTXO is only spent once, it is possible, in principle, replace ECDSA with a one-time signature. The enduser would need to ensure that one-time public keys are never reused across UTXOs, but there is already a strong privacy incentive to do so. In situations where there is a need to issue multiple signatures for a single public key (e.g., for payment channels), users can revert to using a many-time signature scheme. However, all the signatures in transactions signed by the one-time scheme can be aggregated to shrink the overall blockchain.

Construction 2. Our second result (Section 6) is a many-time aggregate signature scheme, but where the aggregation process is interactive. Here our starting point is the Schnorr signature scheme, and in particular, the interactive aggregation process of Bellare and Neven [BN06]. In their system, the signers engage in an interactive protocol to produce a short aggregate signature.

Lyubashevsky's lattice-based signature scheme [Lyu08, Lyu12] has a similar structure to Schnorr signatures. However, a *rejection sampling* process during signing is necessary to protect the signing key, and this makes the scheme more complicated than Schnorr's.

In Section 6 we show how to adapt the Bellare-Neven interactive aggregation technique to Lyubashevsky's signature scheme. Rejection sampling poses a technical challenge. To prove security we must incorporate rejection sampling into our interactive aggregation protocol. Specifically, at one point in the protocol, every signer flips a coin and either continues as normal, or notifies the other signers that it rejected the signing process. At the end of the protocol, if one or more of the signers rejected the signature, signature generation fails. We set the parameters of the scheme so that correctness (completeness) of the protocol can still be achieved using either sequential or parallel repetition of the signing protocol.

Additional related work. While the focus of this paper is on aggregate signatures, a related concept called *multi-signatures* is a restricted form of signature aggregation, where one can only aggregate signatures from multiple parties on the same message. This concept was introduced in [IN83] and further developed in [OO99, MOR01, Bol03, BN06, MPSW18b, DGNW19] in both the interactive and non-interactive settings.

Pairing-based aggregate signatures were studied extensively in [BGLS03, LOS+06, RY07, BNN07, BDN18b]. Universal signature aggregation, namely an aggregation procedure that works for all signature schemes, were constructed using SNARKs [BCCT13] and obfuscation [HKW15].

Currently, there is little work on aggregate signatures from lattice problems. El Bansarkhani and Jan Sturm [BS16] gave a construction where **describe their construction**. Doröz et al. give a lattice-based aggregate signature scheme, but where the aggregate signature size grows *linearly* with the number of signatures being aggregated [DHSS20].

2 Preliminaries

Basic notation. For two integers $n < m$, we write $[n, m]$ to denote the set $\{n, n+1, \ldots, m\}$. When $n = 1$, we simply write $[m]$ to denote the set $\{1, \ldots, m\}$. Unless specified otherwise, we use λ to denote the security parameter. We say that an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We use $\mathsf{poly}(\lambda)$ to denote a quantity whose value is bounded by a fixed polynomial in λ.

Vectors and matrices. We use bold lowercase letters (*e.g.*, $\mathbf{v}, \mathbf{w}$) to denote vectors and bold uppercase letters (*e.g.*, $\mathbf{A}, \mathbf{B}$) to denote matrices. Throughout this work, we always use the infinity norm for vectors and matrices. Namely, for a vector $\mathbf{x} \in \mathbb{Z}^n$, we write $\|\mathbf{x}\|$ to denote $\max_i |x_i|$. Similarly, for a matrix $\mathbf{A} = (A_{i,j}) \in \mathbb{Z}^{n \times m}$, we write $\|\mathbf{A}\|$ to denote $\max_{i,j} |A_{i,j}|$.

Statistical distance. For any two distributions D_0, D_1 over a finite domain Ω, the *statistical distance* between D_0 and D_1 is defined by $\Delta(X, Y) = \frac{1}{2} \sum_{\omega \in \Omega} |D_0(\omega) - D_1(\omega)|$. For a distribution D, we write $x \leftarrow D$ to denote the procedure of sampling x according to distribution D. For a finite domain Ω, we write $\omega \xleftarrow{\$} \Omega$ to denote the procedure of sampling ω uniformly from Ω.

2.1 Discrete Log Problem

We recall the discrete log problem.

Definition 2.1 (Discrete Log)**.** Fix a security parameter λ and let $\mathbb{G} = \mathbb{G}(\lambda)$ be a group of prime order p. Then, the Discrete Log problem ($\mathsf{DLog}_{\mathbb{G}}$) is defined as follows: given two uniformly random group elements $g, h \xleftarrow{\$} \mathbb{G}$, find $x \in \mathbb{Z}_p$ such that $g^x = h$. More precisely, for an adversary $\mathcal{A}$, we define its $\mathsf{DLog}_{\mathbb{G}}$ advantage as follows:

$$\mathsf{Adv}_{\mathsf{DLog}}(\lambda, \mathcal{A}) = \Pr_{g,h \leftarrow_{\mathrm{R}} \mathbb{G}} \big[\mathcal{A}(g, h) \to x \wedge g^x = h\big].$$

The Discrete Log assumption states that for any efficient adversary $\mathcal{A}$, we have $\mathsf{Adv}_{\mathsf{DLog}}(\lambda, \mathcal{A}) = \mathsf{negl}(\lambda)$.

2.2 Short Integer Solution Problem

In this work, we work with the Short Integer Solution (SIS) problem and its polynomial ring variant. Our constructions that are based on these two problems are similar and therefore, we unify their presentation by working with the *general* SIS problem that is defined over an arbitrary commutative ring, which we denote by $\mathcal{R}$. This way of presentation have been used in prior works such as [BGV14, CC17]. For a ring $\mathcal{R}$ with a multiplicative identity, the ideal $q\mathcal{R}$ for any $q \in \mathbb{N}$ is well-defined. We use $\mathcal{R}_q$ to denote the quotient ring $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ throughout the paper.

For the general SIS problem over a ring $\mathcal{R}$, we need a suitable measure of a *norm* in the ring. For both the integer lattice and polynomial ring instantiations of the general SIS problem, we can define natural norm functions that we define below.

Definition 2.2 (General Short Integer Solutions)**.** Fix a security parameter λ, parameters $\ell = \ell(\lambda)$, $q = q(\lambda)$, $\beta = \beta(\lambda)$, and a ring $\mathcal{R} = \mathcal{R}_\lambda$ with an associated norm function $\|\cdot\| : \mathcal{R} \to \mathbb{N}$. The Short Integer Solutions problem $\mathsf{SIS}_{\mathcal{R},\ell,q,\beta}$ is defined as follows: given a uniformly random vector $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$, find a non-zero vector $\mathbf{x} \in \mathcal{R}^\ell$ such that $\mathbf{a}^\intercal \cdot \mathbf{x} = \mathbf{0}$ and $\|\mathbf{x}\| \leq \beta$. More precisely, for an adversary $\mathcal{A}$, we define its $\mathsf{SIS}_{\mathcal{R},\ell,q,\beta}$ advantage as follows:

$$\mathsf{Adv}_{\mathsf{SIS}_{\mathcal{R},\ell,q,\beta}}(\lambda, \mathcal{A}) = \Pr_{\mathbf{a} \leftarrow_{\mathsf{R}} \mathcal{R}} \big[\mathcal{A}(\mathbf{a}) \to \mathbf{x} \wedge \mathbf{a}^\intercal \mathbf{x} = \mathbf{0} \wedge \|\mathbf{x}\| \leq \beta \wedge \mathbf{x} \neq \mathbf{0}\big].$$

The Short Integer Solution assumption states that for any efficient adversary $\mathcal{A}$, we have $\mathsf{Adv}_{\mathsf{SIS}}(\lambda, \mathcal{A}) = \mathsf{negl}(\lambda)$.

SIS over Integer Lattices. To capture the SIS problem over integer lattices [Ajt96], we can set the ring $\mathcal{R} = \mathbb{Z}^{n \times n}$ for a dimension n determined as a function of the security parameter $n = n(\lambda)$.[1] Then we have $\mathcal{R}_q = \mathbb{Z}_q^{n \times n}$, and we can use the infinity norm over matrices $\|\mathbf{A}\| = \max_{i,j \in [n]} |a_{i,j}|$ for any $\mathbf{A} = (a_{i,j})_{i,j \in [n]} \in \mathbb{Z}_q^{n \times n}$. It can be readily checked that for any two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}^{n \times n}$, we have $\|\mathbf{A} \cdot \mathbf{B}\| \leq n \|\mathbf{A}\| \|\mathbf{B}\|$.

SIS over Ideal Lattices. To capture the SIS problem over polynomial rings (ideal lattices) [HPS98, Mic07, PR06, LM06], we can set the ring $\mathcal{R}$ to be the cyclotomic polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a power-of-two integer $n = n(\lambda)$ and set $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$. For any polynomial $a \in \mathcal{R}$, we can represent its coefficients as entries in a vector over $\mathbb{Z}^n$. We define the norm of an element in $\mathcal{R}$ as the infinity norm of its vector representation in $\mathbb{Z}^n$: $\|a\| = \max_{i \in [n]} |a_{i-1}|$ for any $a = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1} \in \mathbb{Z}[X]/(X^n + 1)$. It can be readily checked that for any two polynomials $a, b \in \mathbb{Z}[X]$, we have $\|a \cdot b\| \leq n \|a\| \|b\|$.

2.3 Forking Lemma

In this work, we rely on the forking lemma [PS00, Sch91, BN06, BCJ08] to prove the security of our constructions. For our aggregate one-time signatures in Sections 3.2 and 4, we do not require the full general form of the forking lemma and therefore, we use a simple variant of the lemma that we formulate below.

Lemma 2.3 (Rewinding Lemma [BDN18a])**.** *Let S, R, and T be finite, non-empty sets, and let $f : S \times R \times T \to \{0, 1\}$ be a function. Let X and Y, Y' and Z, Z' be mutually independent random variables where X takes values in the set S, the variables Y and Y' are each uniformly distributed over R, and Z and Z' take values in the set T. Let $\varepsilon = \Pr[f(X, Y, Z) = 1]$ and $N = |R|$. Then*

$$\Pr\Big[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'\Big] \geq \varepsilon^2 - \varepsilon/N.$$

For our interactive aggregate signature construction in Section 5, we use the general forking lemma. We can formulate the forking lemma in multiple ways and in this work, we follow the presentation in [BN06]. We first define the concept of a simulation algorithm. The simulation algorithm takes in an input, generally an instance to a hard problem such as a discrete log challenge or an SIS challenge, and a number of hash outputs, generally used to program a random oracle in a security experiment, and returns an index and an auxiliary output. The auxiliary output generally corresponds to an adversary's forgery in a security experiment and the index is the forking point that generally corresonds to an adversary's random oracle query related to its forgery.

Definition 2.4 (Simulation Algorithm)**.** Fix a positive integer Q and sets $\mathcal{X}$, $\mathcal{H}$ such that $|\mathcal{H}| > 1$. We say that a randomized algorithm $\mathcal{S}$ is a *simulation algorithm* if on input $x \in \mathcal{X}$ and $h_1, \ldots, h_Q \in \mathcal{H}$, it returns an

[1]We note that inserting $\mathcal{R} = \mathbb{Z}^{n \times n}$ directly into Definition 2.2 gives a slight variant of the traditional SIS where the adversary is required to produce a vector of short *matrices* as opposed to *vectors*. This variant of the SIS problem is at least as hard as the original formulation of the SIS problem by Ajtai in [Ajt96].

index and an auxiliary output $(i, \mathsf{aux}) \in [0, Q] \times \{0,1\}^*$. For a distribution IG over $\mathcal{X}$, we define the *advantage* of the simulation algorithm $\mathcal{S}$ as follows:

$$\mathsf{Adv}^{\mathsf{Sim}}_{\mathsf{IG}}(\lambda, \mathcal{A}) = \Pr\left[(i, \sigma) \xleftarrow{\$} \mathcal{S}(x, h_1, \ldots, h_Q) \wedge i \neq 0\right],$$

where $x \leftarrow \mathsf{IG}$ and $h_1, \ldots, h_Q \xleftarrow{\$} \mathcal{H}$.

Next, we define the forking algorithm that runs two executions of the simulation algorithm with a fork at an index.

Definition 2.5 (Forking Algorithm)**.** Fix a positive integer Q and sets $\mathcal{X}$, $\mathcal{H}$ such that $|\mathcal{H}| > 1$. Then, for a simulation algorithm $\mathcal{S}$, we define the *forking algorithm* $\mathcal{F}_{\mathcal{S}}$ as follows:

```
Forking Algorithm $\mathcal{F}_{\mathcal{S}}(x)$:
  1. Sample random coins $\rho$ for $\mathcal{S}$
  2. $h_1, \ldots, h_Q \xleftarrow{\$} \mathcal{H}$
  3. $(i, \sigma) \leftarrow \mathcal{S}(x, h_1, \ldots, h_Q)$
  4. If $i = 0$, then return $(0, \perp, \perp)$
  5. $h'_i, \ldots, h'_Q \xleftarrow{\$} \mathcal{H}$
  6. $(i', \sigma') \leftarrow \mathcal{S}(x, h_1, \ldots, h_Q)$
  7. If $i = i'$ and $h_i \neq h'_i$, return $(1, \sigma, \sigma')$.
  8. Otherwise, return $(0, \perp, \perp)$.
```

For a distribution IG over $\mathcal{X}$, we define the *advantage* of a forking algorithm $\mathcal{F}_{\mathcal{S}}$ as follows:

$$\mathsf{Adv}^{\mathsf{Fork}}_{\mathsf{IG}}(\lambda, \mathcal{F}, \mathcal{S}) = \Pr\left[(b, \sigma, \sigma') \leftarrow \mathcal{F}_{\mathcal{S}}(x) \wedge b = 1\right],$$

where $x \leftarrow \mathsf{IG}$.

The general forking lemma bounds the success probability of the forking algorithm with respect to the success probability of a simulation algorithm.

Lemma 2.6 (General Forking Lemma)**.** *Fix a positive integer Q, sets $\mathcal{X}$, $\mathcal{H}$ such that $|\mathcal{H}| > 1$. Let IG be a distribution over $\mathcal{X}$, let $\mathcal{S}$ be a simulation algorithm with advantage, and let $\mathcal{F}_{\mathcal{S}}$ be a forking algorithm for $\mathcal{S}$. Then, we have*

$$\mathsf{Adv}^{\mathsf{Fork}}_{\mathsf{IG}}(\lambda, \mathcal{F}, \mathcal{S}) \geq \mathsf{Adv}^{\mathsf{Sim}}_{\mathsf{IG}}(\lambda, \mathcal{S}) \cdot \left(\frac{\mathsf{Adv}^{\mathsf{Sim}}_{\mathsf{IG}}(\lambda, \mathcal{S})}{Q} - \frac{1}{|\mathcal{H}|} \right),$$

or alternatively,

$$\mathsf{Adv}^{\mathsf{Sim}}_{\mathsf{IG}}(\lambda, \mathcal{S}) \leq \frac{Q}{|\mathcal{H}|} + \sqrt{Q \cdot \mathsf{Adv}^{\mathsf{Fork}}_{\mathsf{IG}}(\lambda, \mathcal{F}, \mathcal{S})}.$$

3 Aggregatable OTS

In this section, we recall the notion of aggregatable signatures. We present the formal algorithms with the compactness, correctness, and security requirements in Section 3.1. In Section 3.2, we show how to aggregate Bellare-Shoup [BS08] signatures.

3.1 Definitions

The notion of an *aggregatable* signature scheme was first formalized by Boneh et al. [BGLS03], which generalizes multi-signatures [IN83, MOR01]. The algorithms for an aggregate signature scheme is identical to that of a standard signature scheme, but with an additional signature aggregation algorithm SigAgg. The algorithm SigAgg takes in as input an arbitrary set of signatures $\Sigma = \{\sigma_1, \ldots, \sigma_N\}$ where each signature σ_i certifies a message m_i under a public key pk_i. It then returns a succinct signature σ_{ag} that certifies each of the messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ under each of the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ respectively.

Definition 3.1 (Aggregate Signatures)**.** An aggregate signature scheme Π_{AS} for a message space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{AS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{PrmsGen}(1^\lambda) \to \mathsf{pp}$: On input a security parameter λ, the parameter generation algorithm returns a set of public parameters pp.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: On input a set of public parameters pp, the key generation algorithm returns a signing key sk and a public key pk.
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$: On input a signing key sk and a message $\mathsf{m} \in \mathcal{M}$, the signing algorithm returns a signature σ.
- $\mathsf{SigAgg}(\mathsf{PK}, \mathsf{M}, \Sigma) \to \sigma_{\mathsf{ag}}$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and a set of signatures $\Sigma = (\sigma_i)_{i\in[N]}$, the signature aggregation algorithm returns an aggregated signature σ_{ag}.
- $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma) \to 0/1$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, a set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and a signature σ, the verification algorithm either accepts (outputs 1) or rejects (outputs 0).

  When the set of public keys $\mathsf{PK} = (\mathsf{pk})$ and messages $\mathsf{M} = (\mathsf{m})$ are singleton sets, then we simply write $\mathsf{Verify}(\mathsf{pk}, \mathsf{m}, \sigma)$.

An aggregatable signature scheme must satisfy compactness, correctness, and unforgeability properties.

The compactness property of an aggregatable signature scheme requires that any aggregate signature σ_{ag} that is output by the signature aggregation algorithm SigAgg has size that is independent or at most poly-logarithmic in the number of signatures that it is aggregating. Without the compactness requirement, an aggregatable signature scheme can be constructed trivially as any set of signatures $\sigma_1, \ldots, \sigma_N$ can be concatenated into an aggregate signature $\sigma_{\mathsf{ag}} \leftarrow \sigma_1 \| \ldots \| \sigma_N$.

Definition 3.2 (Compactness)**.** Let $\Pi_{\mathsf{AS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Verify})$ be an aggregate signature scheme for a message space $\mathcal{M}$. We say that Π_{AS} is compact if there exists a polynomial $\mathsf{poly}(\cdot)$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all security parameter λ and set of messages $\mathsf{m}_1, \ldots, \mathsf{m}_N \in \mathcal{M}$, if we set

1. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
2. $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$,
3. $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \mathsf{m}_i)$ for $i \in [N]$,
4. $\sigma_{\mathsf{ag}} \leftarrow \mathsf{SigAgg}((\mathsf{pk}_i)_{i\in[N]}, (\mathsf{m}_i)_{i\in[N]}, (\sigma_i)_{i\in[N]})$,

then we have

$$\Pr\left[|\sigma_{\mathsf{ag}}| \le \mathsf{poly}(\lambda, \log N)\right] = 1 - \mathsf{negl}(\lambda),$$

where $|\sigma_{\mathsf{ag}}|$ is the bit length of σ_{ag}.

For correctness, we require that an aggregated signature of any set of properly generated signatures must verify under the corresponding set of public keys and messages.

Definition 3.3 (Correctness)**.** Let $\Pi_{\mathsf{AS}} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{SigAgg}, \mathsf{Verify})$ be an aggregate signature scheme for a message space $\mathcal{M}$. We say that Π_{AS} is correct if for all security parameter $\lambda \in \mathbb{N}$ and number of signers $N \in \mathbb{N}$, we have

$$\Pr\left[\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \mathsf{SigAgg}(\mathsf{PK}, \mathsf{M}, \Sigma)) = 1\right] = 1,$$

where $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{m}_i)$ for $i \in [N]$, $\mathsf{PK} \leftarrow (\mathsf{pk}_i)_{i\in[N]}$, $\mathsf{M} \leftarrow (\mathsf{m}_i)_{i\in[N]}$, and $\Sigma \leftarrow (\sigma_i)_{i\in[N]}$.

We note that the correctness requirement above implies that an aggregate signature is also correct as a standard signature scheme since the set of public keys PK and messages M can be singleton sets.

Finally, we define the unforgeability security requirement. The unforgeability experiment for an aggregate signature scheme is defined identically to that of a standard signature scheme where the adversary is provided with a public key pk^* and an oracle access to the signing algorithm $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$. The only difference between the unforgeability experiment of an aggregate signature scheme and that of a standard signature scheme is the winning condition of an adversary. Since the verification algorithm for an aggregate signature scheme takes in a *set* of public keys and messages, we allow an adversary to also output a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$ and messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ as part of its forgery. Specifically, we say that an adversary wins the unforgeability experiment if it can produce a verifying set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and a signature σ^* such that $\mathsf{pk}^* = \mathsf{pk}_{i^*}$ for some $i^* \in [N]$ and m_{i^*} is a message that was never queried to the signing oracle.

Definition 3.4 (Unforgeability)**.** Let Π_{AS} be an aggregate signature scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A}$, we define the signature unforgeability experiment $\mathsf{EXP}_{\mathsf{AS}}[\lambda, \mathcal{A}]$ as follows:

1. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
2. $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$,
3. $(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}^*,\cdot)}(\mathsf{pp}, \mathsf{pk}^*)$,
4. Output $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}})$.

We say that $\mathcal{A}$ is an admissible adversary for the unforgeability experiment if for any execution of the experiment, the following property holds: for any forgery $(\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M} = (\mathsf{m}_i)_{i\in[N]}, \sigma_{\mathsf{ag}})$ that $\mathcal{A}$ returns at the end of the experiment, we have $\mathsf{pk}_{i^*} = \mathsf{pk}^*$ for some $i^* \in [N]$, and the message m_{i^*} is never queried to the signing oracle $\mathsf{Sign}(\mathsf{sk}^*, \cdot)$. We say that an aggregate signature scheme Π_{AS} is unforgeable if for all efficient and admissible adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathsf{EXP}_{\mathsf{AS}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

In this work, we work with the notion of a *one-time signature* scheme where unforgeability is guaranteed under the condition that a signing key is used at most once to generate a signature. We define the unforgeability experiment for aggregatable one-time signature schemes analogously to Definition 3.5.

Definition 3.5 (Unforgeability for OTS)**.** Let Π_{AS} be an aggregate signature scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we define the one-time signature unforgeability experiment $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ as follows:

1. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
2. $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$,
3. $(\hat{\mathsf{m}}, \mathsf{state}) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{pk}^*)$,
4. $\hat{\sigma} \leftarrow \mathsf{Sign}(\mathsf{sk}^*, \hat{\mathsf{m}})$,
5. $(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \leftarrow \mathcal{A}_2(\mathsf{state}, \hat{\sigma})$,
6. Output $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}})$.

We say that $\mathcal{A}$ is an admissible adversary for the unforgeability experiment if for any execution of the experiment, the following holds: for any forgery $(\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}, \mathsf{M} = (\mathsf{m}_i)_{i\in[N}, \sigma_{\mathsf{ag}})$ that it returns at the end of the experiment, we have $\mathsf{pk}_{i^*} = \mathsf{pk}^*$ for some $i^* \in [N]$, and $\hat{\mathsf{m}} \neq \mathsf{m}_{i^*}$. We say that an aggregate signature scheme Π_{AS} is unforgeable if for all efficient and admissible adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

3.2 Warm-up: Aggregating Bellare-Shoup OTS

In this section, we show how to aggregate the Bellare-Shoup one-time signature scheme [BS08]. The construction is straightforward, but it conveys the main intuitions for the lattice constructions in Sections 4 and 6.

The Bellare-Shoup signature scheme works over a group $\mathbb{G}$ of prime order p with a generator $g \in \mathbb{G}$. A public key in the signature scheme consists of two group elements $h_0, h_1 \in \mathbb{G}$ and the corresponding secret key consists of the discrete log of these elements x_0, x_1 with respect to g: $h_0 = g^{x_0}$ and $h_1 = g^{x_1}$. A signature σ for a message m is defined as $\sigma \leftarrow c \cdot x_0 + x_1 \in \mathbb{Z}_p$ where $c \leftarrow H(\mathsf{m})$ and H is a hash function that is modeled as a random oracle that maps messages to elements in $\mathbb{Z}_p$. The signature can be verified by the relation $g^\sigma = h_0^c \cdot h_1$.

Let $\sigma_1, \sigma_2 \in \mathbb{Z}_p$ be two signatures under two public keys $(h_{1,0}, h_{1,1})$, $(h_{2,0}, h_{2,1})$ and two messages $\mathsf{m}_1, \mathsf{m}_2$ in the Bellare-Shoup signature scheme. Then, a natural first attempt at aggregating these signatures is to simply add them together: $\sigma_{\mathsf{ag}} \leftarrow \sigma_1 + \sigma_2 \in \mathbb{Z}_p$. The sum of the two signatures can still be verified by the group relation

$$g^{\sigma_{\mathsf{ag}}} = \big(h_{1,0}^{c_1} \cdot h_1\big) \cdot \big(h_{2,0}^{c_2} \cdot h_{2,1}\big),$$

where $c_1 \leftarrow H(\mathsf{m}_1)$ and $c_2 \leftarrow H(\mathsf{m}_2)$. This way of aggregating signatures, however, results in *rogue attacks*. Namely, given a signature σ_1 under a public key $(h_{1,0}, h_{1,1})$ and a message m_1, an adversary can produce a another signature σ_2, public key $(h_{2,0}, h_{2,1})$, and two messages m_1', m_2 for $\mathsf{m}_1' \neq \mathsf{m}_1$ such that σ_{ag} verifies a message tuple $(\mathsf{m}_1', \mathsf{m}_2)$. Specifically, the adversary can sample any arbitrary signature $\sigma_2 \in \mathbb{Z}_p$, messages $\mathsf{m}_1', \mathsf{m}_2 \in \mathsf{M}$, public key component $h_{2,0}$, and then solve for $h_{2,1}$ in the group relation

$$g^{\sigma_1 + \sigma_2} = \big(h_{1,0}^{c_1'} \cdot h_1\big) \cdot \big(h_{2,0}^{c_2} \cdot h_{2,1}\big),$$

where $c_1' \leftarrow H(\mathsf{m}_1')$ and $c_2 \leftarrow H(\mathsf{m}_2)$.

One way of preventing such type of attacks is to require that any public key in the scheme includes a Proof-of-Knowledge (PoK) of discrete log on each of the two group elements [Bol03, LOS$^+$06, RY07]. However, this inevitably increases the size of the public keys. In this work, we prevent rogue attacks without increasing the size of the public keys by relying on the power of random oracles. Given two signatures $\sigma_1, \sigma_2 \in \mathbb{Z}_p$ under public keys $(h_{1,0}, h_{1,1})$, $(h_{2,0}, h_{2,1})$ and messages $\mathsf{m}_1, \mathsf{m}_2$, we define the aggregate signature as the weighted sum $\sigma_{\mathsf{ag}} \leftarrow t_1 \cdot \sigma_1 + t_2 \cdot \sigma_2 \in \mathbb{Z}_p$ where

$$(t_1, t_2) \leftarrow H'\big((h_{1,0}, h_{1,1}), (h_{2,0}, h_{2,1}), \mathsf{m}_1, \mathsf{m}_2\big) \in \mathbb{Z}_p^2,$$

and H' is an additional hash function that is modeled as a random oracle. Since σ_{ag} is a linear combination of the signatures σ_1, σ_2, it can still be verified using group operations as before. At the same time, unless an adversary can predict the output of H' that is modeled as a random oracle, it cannot carry out a rogue attack as before.

We formally define an aggregate one-time signature scheme based on the Bellare-Shoup signature scheme [BS08] below. In the construction description, we use a hash function that has a dynamic output $H_1 : \{0,1\}^* \to (\mathbb{Z}_p)^*$. Such a hash function can be constructed from any hash function with a static output $\tilde{H}_1 : \{0,1\}^* \to \mathbb{Z}_p$ by iteratively applying it on an input. For instance, to map an input $x \in \{0,1\}^*$ to N elements in $\mathbb{Z}_p$, we can set

$$H_1(x) = \big(\tilde{H}_1(1, x, N), \tilde{H}_1(2, x, N), \ldots, \tilde{H}_1(N, x, N)\big).$$

Construction 3.6. Fix a message space $\mathcal{M}$ and let $H_0 : \mathcal{M} \to \mathbb{Z}_p$, $H_1 : \{0,1\}^* \to (\mathbb{Z}_p)^*$ be two hash functions. We construct an aggregate signature scheme as follows:

- $\mathsf{PrmsGen}(1^\lambda) \to \mathsf{pp}$: On input a security parameter λ, the parameter generation algorithm defines a group $\mathbb{G} = \mathbb{G}(\lambda)$ of prime order p. It samples a group element $g \xleftarrow{\$} \mathbb{G}$ and returns $\mathsf{pp} = (\mathbb{G}, g)$.

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: On input the public parameter pp, the key generation algorithm samples two exponents $x_0, x_1 \xleftarrow{\$} \mathbb{Z}_p$ and computes $h_0 = g^{x_0}, h_1 = g^{x_1} \in \mathbb{G}$. It sets
$$\mathsf{sk} = (x_0, x_1), \qquad \mathsf{pk} = (h_0, h_1).$$
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$: On input a signing key $\mathsf{sk} = (x_0, x_1)$ and a message $\mathsf{m} \in \mathcal{M}$, the signing algorithm computes the hash $c \leftarrow H_0(\mathsf{m})$. It sets $\sigma \leftarrow c \cdot x_0 + x_1$ and returns σ.
- $\mathsf{SigAgg}(\mathsf{PK}, \mathsf{M}, \Sigma) \to \sigma_{\mathsf{ag}}$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and signatures $(\sigma_i)_{i \in [N]}$, the aggregation algorithm computes the hash $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$. It then defines the aggregate signature to be
$$\sigma_{\mathsf{ag}} \leftarrow \sum_{i \in [N]} t_i \cdot \sigma_i,$$
and returns σ_{ag}.
- $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma) \to 0/1$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and a signature σ, the verification algorithm first parses the public keys as $\mathsf{pk}_i = (h_{i,0}, h_{i,1})$ for $i \in [N]$. It then computes the hash values
  - $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$,
  - $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$,

  and verifies the following relation:
$$g^{\sigma} = \prod_{i \in [N]} \left(h_{i,0}^{c_i} \cdot h_{i,1}\right)^{t_i}.$$
  If the relation holds, then the algorithm accepts the signature (outputs 1) and otherwise, it rejects (outputs 0).

We now state the compactness, correctness, and security properties of Construction 3.6.

Theorem 3.7 (Compactness)**.** *The aggregate signature scheme in Construction 3.6 is compact (Definition 3.2).*

Theorem 3.8 (Correctness)**.** *The aggregate signature scheme in Construction 3.6 is correct (Definition 3.3).*

Theorem 3.9 (Security)**.** *Suppose that the discrete log problem $\mathsf{DLog}_{\mathbb{G}}$ is hard. Then the aggregate one-time signature scheme in Construction 3.6 satisfies unforgeability (Definition 3.5).*

More precisely, suppose that there exists an adversary $\mathcal{A}$ in the unforgeability experiment $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that makes at most Q_0 random-oracle queries to H_0 and Q_1 random-oracle queries to H_1 such that
$$\varepsilon = \Pr\left[\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1\right].$$
Then, there exists a discrete log algorithm $\mathcal{B}$ such that
$$\left(\varepsilon - \frac{1}{p}\right) \cdot \frac{1}{Q_0 \cdot Q_1} \cdot \frac{1}{|\mathcal{M}|} \leq \sqrt{\mathsf{Adv}_{\mathsf{DLog}}[\lambda, \mathcal{A}] + \frac{1}{p}}.$$

We provide the proof of the theorems above in Appendix A. We provide a high level overview of the security proof below.

Security proof overview. For the main intuition behind the security proof, let us first consider the security of the Bellare-Shoup signature scheme without signature aggregation. When aggregation is not involved, we can construct a simple reduction algorithm $\mathcal{B}$ that simulates the view of the standard (one-time) unforgeability experiment for an adversary $\mathcal{A}$ as follows:

1. Upon receiving a discrete log challenge $\hat{h} \in \mathbb{G}$, algorithm $\mathcal{B}$ generates a random exponent $c \xleftarrow{\$} \mathbb{Z}_p$, signature $\sigma \xleftarrow{\$} \mathbb{G}$, and sets the public key as $\mathsf{pk}^* = (h_0 = \hat{h}, h_1 = g^\sigma / \hat{h}^c)$. The public key is defined precisely to satisfy the verification relation $g^\sigma = h_0^c \cdot h_1$.

2. Suppose that adversary $\mathcal{A}$ makes Q_0 random oracle queries to H_0: $\mathsf{m}_1, \ldots, \mathsf{m}_{Q_0}$. $\mathcal{B}$ guesses one of these messages m_{i^*} for some $i^* \in [Q_0]$, programs the random oracle $H_0(\mathsf{m}_{i^*}) \leftarrow c$, and hopes that $\mathcal{A}$ makes the single signing query on the message m_{i^*}. It can be readily checked that if $\mathcal{A}$ does indeed make the single signing query on m_{i^*}, then the public key components h_0, h_1, random exponent c, and the signature σ of m_{i^*} are correctly distributed.

3. Suppose that $\mathcal{A}$ does indeed make the signing query on m_{i^*} and at the end of the experiment, produces a valid forgery (m', σ') such that $g^{\sigma'} = h_0^{c'} \cdot h_1$ where $c' \leftarrow H_0(\mathsf{m}')$. Algorithm $\mathcal{B}$ solves for the variable x_0 in the two linear relations

$$\begin{aligned} \sigma &= c \cdot x_0 + x_1, \\ \sigma' &= c' \cdot x_0 + x_1. \end{aligned}$$

Assuming that $\mathsf{m}_{i^*} \neq \mathsf{m}'$, these two relations are linearly independent with high probability. The variable x_0 is the solution to the discrete log challenge h^*.

The main idea of the reduction above is that algorithm $\mathcal{B}$ "prepares" a single signature to provide to $\mathcal{A}$ without knowing a valid signing key, but only by programming the random oracle H_0. When adversary $\mathcal{A}$ generates a forgery, algorithm $\mathcal{B}$ knows two valid signatures under a single signing key and can immediately deduce the signing key, which contains the discrete log solutions to the public key components.

The unforgeability experiment for aggregate signatures in Definition 3.5 is identical to the standard unforgeability experiment for one-time signatures except for the structure of an adversary's forgery. Instead of outputting a message-signature pair, an adversary now produces a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and a signature σ_{ag} such that

$$g^{\sigma_{\mathsf{ag}}} = \prod_{i \in [N]} (h_{i,0}^{c_i} \cdot h_{i,1})^{t_i}, \tag{3.1}$$

where $\mathsf{pk}_i = (h_{i,0}, h_{i,1})$, $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$, and $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$. Now, simply applying the same reduction algorithm $\mathcal{B}$ above does not work since the additional public-key components prevent $\mathcal{B}$ from deriving a linear relation to solve. Therefore, to emulate the proof strategy above, we must simplify (3.1) by rewinding (Lemma 2.3).

Since the hash function H_1 is modeled as a random oracle, a reduction algorithm $\mathcal{B}$ can program its output $H_1(\mathsf{PK}, \mathsf{M})$ for any query $(\mathsf{PK}, \mathsf{M})$ that an adversary $\mathcal{A}$ makes. Hence, we define algorithm $\mathcal{B}$ to simulate two executions of the aggregate unforgeability experiment in exactly the same way except for when the adversary $\mathcal{A}$ makes a random oracle query on the set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$ and messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$ that it will eventually forge on. For simplicity, suppose that $\mathsf{pk}_1 = \mathsf{pk}^*$ corresponds to $\mathcal{B}$'s (the challenger's) public key that it generates at the start of the experiment. Then during the first execution of the experiment, algorithm $\mathcal{B}$ samples $(t_1, t_2, \ldots, t_N) \xleftarrow{\$} \mathbb{Z}_p^N$ and programs $H_1(\mathsf{PK}, \mathsf{M}) \leftarrow (t_1, t_2, \ldots, t_N)$. During the second experiment, it samples $t_1' \xleftarrow{\$} \mathbb{Z}_p$ and programs $H_1(\mathsf{PK}, \mathsf{M}) \leftarrow (t_1', t_2, \ldots, t_N)$. Now suppose that adversary $\mathcal{A}$ produces a valid forgery corresponding to the exact set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$ in both executions of the experiment to produce signatures σ_{ag} and σ'_{ag} respectively. Then, these two signatures must satisfy the verification relation

$$g^{\sigma_{\mathsf{ag}}} = (h_{1,0}^{c_1} \cdot h_{i,1})^{t_1} \prod_{i \in [N] \setminus \{1\}} (h_{i,0}^{c_i} \cdot h_{i,1})^{t_i},$$

$$g^{\sigma'_{\mathsf{ag}}} = (h_{1,0}^{c_1} \cdot h_{i,1})^{t_1'} \prod_{i \in [N] \setminus \{1\}} (h_{i,0}^{c_i} \cdot h_{i,1})^{t_i}.$$

Dividing the second relation onto the first relation, algorithm $\mathcal{B}$ can now remove the additional public key components from the group relations:

$$g^{\sigma_{\mathsf{ag}}-\sigma'_{\mathsf{ag}}} = (h_{1,0}^{c_1} \cdot h_{i,1})^{t_1-t'_1}.$$

Converting the group relation above as a linear relation over the exponents, the reduction algorithm $\mathcal{B}$ can now solve for the discrete log challenge with high probability as before.

We note that the reduction algorithm $\mathcal{B}$ above succeeds only if adversary $\mathcal{A}$ produces a valid forgery on the *same* set of public keys and messages. There are other set of bad events that may force the reduction algorithm to fail as well. For instance, if $t_1 = t'_1$, then the reduction algorithm $\mathcal{B}$ above fails. The bulk of the proof constitutes carefully bounding the probability that any of these bad events occur.

Remark 3.10 (Optimization)**.** For practical implementations of group-based constructions, we generally work over elliptic curve groups $\mathbb{G}$ of order p where p is a 256-bit prime. For Construction 3.6, we can optimize the performance of the verification algorithm by restricting the output of the hash function H_1 to a subset $R = \{1, \ldots, 2^{128}\}$ or $R = \{-2^{64}, \ldots, 2^{64}\}$ in $\mathbb{Z}_p$:

$$H_1 : \{0,1\}^* \to R^*.$$

The reduction loss in Theorem 3.9 remains the same except for the additive factors $1/p$, which becomes $1/|R| = 1/2^{128}$. This optimization still provides a very reasonable security guarantee for practice. Since the exponents t_i's now live in a smaller set, the verification algorithm is guaranteed to execute smaller exponentiation operations.

Remark 3.11 (Extension to stateless k-time signatures)**.** We note that by allowing the signing algorithm to be stateful, any one-time signature scheme can be easily converted into a k-time signature scheme (for a priori bounded k) by invoking k parallel instances of the one-time signature scheme. A signing key can consist of k independent signing keys that are produced from the key-generation algorithm. To sign a message, the signing algorithm uses one of the k signing keys to generate a signature. However, to prevent double signing on any of the k signing keys, the signing algorithm must remember all the keys that were previously used, making it a stateful algorithm.

For the case of Construction 3.6, we can extend the scheme to a k-time signature scheme with a *stateless* signing algorithm. Instead of defining the public key to be a pair of group elements $h_0, h_1 \in \mathbb{G}$, we define the public key to consist of $k+1$ group elements $h_0, \ldots, h_k \in \mathbb{G}$. A signing key consists of the discrete log of these elements $x_0, \ldots, x_k \in \mathbb{Z}_p$. To generate a signature on a message m, the signing algorithm sets $\sigma \leftarrow x_0 + c_1 \cdot x_1 + \ldots + c_k \cdot x_k$ where $(c_1, \ldots, c_k) \leftarrow H_0(\mathsf{m})$ and $H_0 : \mathcal{M} \to \mathbb{Z}_p^k$ is a hash function that maps messages to vectors in $\mathbb{Z}_p^k$. This signature can be verified by the group relation

$$g^\sigma = h_0 \cdot \prod_{i \in [N]} h_i^{c_i}.$$

It can be easily checked that the signatures that are generated in this way are also aggregatable in the same way as in Construction 3.6.

The proof of Theorem 3.9 can be extended to show that this extension of the Bellare-Shoup signature scheme is k-time secure. As described in the proof overview above, we prove that Construction 3.6 is one-time secure by constructing a reduction algorithm $\mathcal{B}$ that can "prepare" a single signature to provide to an adversary $\mathcal{A}$ by programming the hash function H_0. When $\mathcal{A}$ produces a forgery, $\mathcal{B}$ combines the linear relation that is induced by the signature that it generated itself and the signature generated by $\mathcal{A}$ to solve a discrete log challenge. For the extension to a k-time secure scheme as above, we can modify $\mathcal{B}$ such that it prepares k signatures to provide to $\mathcal{A}$ by programming the random oracle H_0 on k different messages. At this point, $\mathcal{B}$ knows exactly k linear relations over $k+1$ unknown variables. When $\mathcal{A}$ produces a forgery, $\mathcal{B}$ can now solve for $k+1$ variables over $k+1$ linear relations and solve a discrete log challenge.

Although this extension of the construction and the proof is natural, it is ideal only for small values of k. The reason is that in order to use the rewinding lemma (Lemma 2.3), the actual proof of Theorem 3.9 require guessing arguments whose success probability degrades expoentially as k grows.

Remark 3.12 (Schnorr Signatures)**.** The Bellare-Shoup one-time signature scheme is fundamentally related to the Schnorr identification scheme [Sch91]. Specifically, a public key in the Bellare-Shoup signature scheme can be viewed as consisting of a prover's public key in the Schnorr identification protocol along with its first message in the protocol. A Bellare-Shoup signature then consists of the prover's second message in the protocol.

The Schnorr identification protocol can be alternatively compiled into a full-fledged *Schnorr signature* scheme using the Fiat-Shamir transform [FS86]. Instead of including the first message of the identification protocol into the public key, the Schnorr signature scheme includes both the prover's first and the second protocol messages in the signature itself. With this interpretation, a Bellare-Shoup signature can be viewed as consisting of the "half" of Schnorr's signature.

Therefore, Construction 3.6 and Theorems 3.7, 3.8, and 3.9 show that Schnorr signatures are also partly aggregatable. Although this property of the Schnorr signature scheme is folklore, we are unaware of any previous work that rigorously proves this fact. It is also useful to note that by our discussion preceding Construction 3.6, aggregating Schnorr signatures by simply "adding" the relevant components together is susceptible to rogue attacks. Our construction shows how to securely aggregate the relevant components of Schnorr signatures in a secure way.

Remark 3.13 (Okamoto Protocol)**.** The main property of the Bellare-Shoup signature scheme that we used to construct an aggregate signature scheme is the linearity of the decryption algorithm. The linear decryption algorithm allowed multiple signatures to be added together, and we used the unpredictability of hash functions to prevent rogue attacks. This technique can be applied to existing signature schemes that have similarly linear decryption algorithms over groups. For instance, the Okamoto identification protocol [Oka92] can be extended into an aggregatable one-time signature scheme as in Construction 3.6.

4 Aggregatable OTS from Lattices

In this section, we show how to aggregate the Lyubashevsky-Micciancio one-time signature scheme [LM08]. The Lyubashevsky-Micciancio signature scheme has a similar structure to the Bellare-Shoup signature scheme, but over the ring $\mathcal{R}$ that can be instantiated over either integer or ideal lattices (see Section 2.2). Technically, the Lyubashevsky-Micciancio one-time signature scheme works in the standard model. In this section, we present the random-oracle variant of the construction as it simplifies the analysis and we must work in the random-oracle model to prevent rogue attacks regardless.

In the Lyubashevsky-Micciancio one-time signature scheme, the public parameters include a vector of ring elements $\mathbf{a} \in \mathcal{R}_q^\ell$, which is analogous to a group generator in the Bellare-Shoup scheme. A public key consists of two ring elements $v_0, v_1 \in \mathcal{R}_q$, and the corresponding secret key consists of a vector of short ring elements $\mathbf{s}_0, \mathbf{s}_1 \in \mathcal{R}^\ell$ such that $\mathbf{a}^\intercal \mathbf{s}_0 = v_0$ and $\mathbf{a}^\intercal \mathbf{s}_1 = v_1$. A signature σ for a message m is defined as $\sigma \leftarrow \mathbf{s}_0 \cdot c + \mathbf{s}_1$ where $c \leftarrow H(\mathsf{m})$ and H is a hash function that maps messages to short elements in $\mathcal{R}$.[2] The signature can be verified by the relation $\mathbf{a}^\intercal \sigma = v_0 \cdot c + v_1$.

As in the case of Bellare-Shoup, a natural way to aggregate these signatures is to add them together over the ring $\mathcal{R}$. However, like before, we use an additional hash function H' that maps public keys and messages to short elements in $\mathcal{R}$ to prevent rogue attacks. We define the aggregate signature of a set of signatures to be the weighted sum of the signatures with respect to the hash of the corresponding public keys and messages as in Construction 3.6.

We note that since signatures in the Lyubashevsky-Micciancio scheme consist of short vectors in $\mathcal{R}^\ell$, their (weighted) sum does increase the *norm* of the final signature. For instance, when aggregating N signatures, the final signature will have a greater norm compared to the individual signatures by a factor of N. When considering the actual *bit-length* of an aggregate signature, an increase in the norm by a factor of N only increases the size of the aggregate signature by a logarithmic factor. Hence, this way of aggregation

[2]We note that since $\mathcal{R}$ can be a non-commutative ring, whether we multiply c to $\mathbf{s}_0$ on the left or right impacts the correctness condition of the scheme.

does satisfy our compactness requirement in Definition 3.2. This logarithmic increase in size for signature aggregation appears to be inherent for any SIS-based signatures in general.

We formally define an aggregate one-time signature scheme based on the Lyubashevsky-Micciancio scheme below. For the presentation of the construction, we use $\mathcal{B}_\beta$ for $\beta \in \mathbb{N}$ to denote the set of ring elements in $\mathcal{R}$ with norm at most β: $\mathcal{B}_\beta = \{r \in \mathcal{R} : \|r\| \leq \beta\} \subseteq \mathcal{R}$. As in Construction 3.6, we rely on hash functions with dynamic output.

Construction 4.1. Let $\ell, q, \beta_{\mathbf{s}}, \beta_0, \beta_1$ and β_{ver} be positive integers and let $\mathcal{R}$ be a ring. Fix a message $\mathcal{M}$ and let $H_0 : \mathcal{M} \to \mathcal{B}_{\beta_0}$, $H_1 : \{0,1\}^* \to (\mathcal{B}_{\beta_1})^*$ be two hash functions. We construct an aggregate signature scheme as follows:

- $\mathsf{PrmsGen}(1^\lambda) \to \mathsf{pp}$: On input a security parameter λ, the parameter generation algorithm generates a uniformly random vector of ring elements $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$ and sets $\mathsf{pp} = \mathbf{a}$.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: On input the public parameters pp, the key generation algorithm samples two vectors of ring elements $\mathbf{s}_0, \mathbf{s}_1 \xleftarrow{\$} \mathcal{B}_{\beta_{\mathbf{s}}}^\ell$. It then defines $v_0 \leftarrow \mathbf{a}^\intercal \mathbf{s}_0$, $v_1 \leftarrow \mathbf{a}^\intercal \mathbf{s}_1$ and sets
$$\mathsf{sk} = (\mathbf{s}_0, \mathbf{s}_1), \qquad \mathsf{pk} = (v_0, v_1).$$
- $\mathsf{Sign}(\mathsf{sk}, \mathsf{m}) \to \sigma$: On input a signing key $\mathsf{sk} = (\mathbf{s}_0, \mathbf{s}_1)$ and a message $\mathsf{m} \in \mathcal{M}$, the signing algorithm computes the hash $c \leftarrow H_0(\mathsf{m})$. It sets $\sigma \leftarrow \mathbf{s}_0 \cdot c + \mathbf{s}_1$ and returns σ.
- $\mathsf{SigAgg}(\mathsf{PK}, \mathsf{M}, \Sigma) \to \sigma_{\mathsf{ag}}$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and signatures $(\sigma_i)_{i \in [N]}$, the aggregation algorithm computes the hash $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$ in $(\mathcal{B}_{\beta_1})^N$. It then defines the aggregate signature as
$$\sigma_{\mathsf{ag}} \leftarrow \sum_{i \in [N]} \sigma_i \cdot t_i,$$
and returns $\sigma_{\mathsf{ag}} \in \mathcal{R}^\ell$.
- $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma) \to \{0,1\}$: On input a set of public keys as $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and a signature σ, the verification algorithm first parses the public keys $\mathsf{pk}_i = (v_{i,0}, v_{i,1})$ for $i \in [N]$. It then computes the hash values
  - $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$,
  - $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$,

  and then verifies the following conditions:
  - $\mathbf{a}^\intercal \cdot \sigma = \sum_{i \in [N]} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i$,
  - $\|\sigma\| \leq \beta_{\mathsf{ver}}$.

  If both of these conditions are true, then the algorithm accepts the signature (outputs 1) and otherwise, it rejects (outputs 0).

We now state the compactness, correctness, and security properties of Construction 4.1.

Theorem 4.2 (Compactness)**.** *Let $n = \mathsf{poly}(\lambda)$ be a positive integer, $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, and suppose that the parameters $\ell, q, \beta_{\mathbf{s}}, \beta_0, \beta_1 = \mathsf{poly}(\lambda)$ are set as any polynomial functions over λ. Then, the aggregate signature scheme in Construction 4.1 is compact (Definition 3.2).*

Theorem 4.3 (Correctness)**.** *Suppose that the parameters of Construction 4.1 are set such that the ring $\mathcal{R}$ is instantiated as either $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and that the inequality $\beta_{\mathsf{ver}} \geq N n \beta_{\mathbf{s}} \beta_1 (n\beta_0 + 1)$ holds. Then, Construction 4.1 is correct (Definition 3.3).*

Theorem 4.4 (Security)**.** *Suppose that the parameters of Construction 4.1 are instantiated such that the ring $\mathcal{R}$ is instantiated as either $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and that $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$. Additionally, suppose that*

- *when $\mathcal{R} = \mathbb{Z}^{n \times n}$, the inequality $2\beta_{\mathbf{s}} + 1 \geq q^{1/\ell} \cdot 2^{\lambda/n\ell}$ holds.*
- *when $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, the inequality $2\beta_{\mathbf{s}} + 1 \geq q^{1/\ell} \cdot 2^{\lambda/n\ell}$ holds.*

With these parameter settings, assume that the SIS problem $\mathsf{SIS}_{\mathcal{R},\ell,q,\beta^\star}$ for $\beta^\star = 3 \cdot \beta_{\mathsf{ver}} + (n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}}) \cdot 2n\beta_1 + (n\beta_0 + 1)\beta_{\mathbf{s}}$ is hard. Then Construction 4.1 satisfies unforgeability (Definition 3.5).

More precisely, suppose that there exists an adversary $\mathcal{A}$ in the unforgeability experiment $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$ that makes at most Q_0 random-oracle queries to H_0 and Q_1 random-oracle queries to H_1, such that

$$\varepsilon = \Pr\left[\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]\right].$$

Then, there exists an SIS algorithm $\mathcal{B}$ such that

$$\left(\varepsilon - \frac{1}{|\mathcal{B}_{\beta_1}|}\right) \cdot \frac{1}{Q_0 \cdot Q_1} \cdot \frac{1}{|\mathcal{M}|} \leq \sqrt{2 \cdot \mathsf{Adv}_{\mathsf{SIS}_{\mathcal{R},\ell,q,\beta^\star}}[\lambda, \mathcal{B}]}.$$

We provide formal proofs of the theorems above in Definition B. We provide an overview of the security proof below.

Security proof overview. Although the Bellare-Shoup signature scheme and the Lyubashevsky-Micciancio signature scheme have similar algebraic structures, the approaches that are needed to prove their security are different. Let us first ignore signature aggregation and just consider these constructions as one-time signature schemes. To prove that the Bellare-Shoup scheme is secure, we construct a reduction algorithm that takes a discrete log challenge and simulates the unforgeability experiment without knowing a valid signing key for the experiment. To prove that the Lyubashevsky-Micciancio scheme is secure, we instead define a reduction algorithm $\mathcal{B}$ that knows a valid signing key for the entire duration of the unforgeability experiment that it simulates. Specifically, interacting with an adversary $\mathcal{A}$, algorithm $\mathcal{B}$ simulates the unforgeability experiment as follows:

1. Given an SIS challenge $\mathbf{a} \in \mathcal{R}_q^\ell$, algorithm $\mathcal{B}$ generates the signing and public keys exactly as in the key generation algorithm
$$\mathsf{sk} = (\mathbf{s}_0, \mathbf{s}_1), \qquad \mathsf{pk} = (v_0, v_1).$$
2. Using the signing key sk, $\mathcal{B}$ simulates the rest of the unforgeability experiment exactly as in the unforgeability experiment.
3. At the end of the experiment, suppose that adversary $\mathcal{A}$ produces a valid forgery (m, σ) such that $\mathbf{a}^\intercal\sigma = v_0 \cdot c + v_1$ where $c \leftarrow H_0(\mathsf{m})$. At this point, $\mathcal{B}$ generates its own signature σ' on the same message m using its own signing key. If $\sigma = \sigma'$, then $\mathcal{B}$ fails to solve the SIS challenge. However, if $\sigma \neq \sigma'$, then by the property of the verification condition, $\sigma - \sigma' \in \mathcal{R}$ is a short solution to the SIS challenge.

The main technical part of the proof is on bounding the probability that $\sigma \neq \sigma'$.

Now consider the unforgeability experiment for aggregatable signatures (Definition 3.5). In this experiment, an adversary can produce a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and a signature σ_{ag} such that

$$\mathbf{a}^\intercal\sigma_{\mathsf{ag}} = \sum_{i \in [N]} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i,$$

where $\mathsf{pk}_i = (v_{i,0}, v_{i,1})$, $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$, and $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$. The additional public-key components prevent the reduction algorithm $\mathcal{B}$ above from using σ_{ag} to generate a solution to an SIS challenge. As in the proof of security for Construction 3.6, we use rewinding so that $\mathcal{B}$ can remove these additional public-key components. Specifically, algorithm $\mathcal{B}$ simulates two executions of the aggregate unforgeability experiment identically except for the way it programs the random oracle H_1 on the set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$ and messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ that adversary $\mathcal{A}$ will forge on. Suppose that pk_1 in PK corresponds to the public key that $\mathcal{B}$ (the challenger) generated at the start of the experiment. Then, in the first execution, $\mathcal{B}$ samples $(t_1, t_2, \ldots, t_N) \xleftarrow{\$} \mathcal{B}_{\beta_1}^N$ and programs $H_1(\mathsf{PK}, \mathsf{M}) \leftarrow (t_1, t_2, \ldots, t_N)$. In the second execution, $\mathcal{B}$ samples $t_1' \xleftarrow{\$} \mathcal{B}_{\beta_1}$ and programs $H_1(\mathsf{PK}, \mathsf{M}) \leftarrow (t_1', t_2, \ldots, t_N)$.

Now suppose that adversary $\mathcal{A}$ produces two valid forgeries corresponding to PK and M in both executions of the experiment to produce σ_{ag} and σ_{ag}'. These signatures must satisfy the verification relation

$$\mathbf{a}^{\mathsf{T}}\sigma_{\mathsf{ag}} = (v_{1,0} \cdot c_1 + v_{1,1}) \cdot t_1 + \sum_{i\in[N]\setminus\{1\}} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i,$$

$$\mathbf{a}^{\mathsf{T}}\sigma_{\mathsf{ag}}' = (v_{1,0} \cdot c_1 + v_{1,1}) \cdot t_1' + \sum_{i\in[N]\setminus\{1\}} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i.$$

By subtracting the two relations, algorithm $\mathcal{B}$ can remove the extra public key components

$$\mathbf{a}^{\mathsf{T}}(\sigma_{\mathsf{ag}} - \sigma_{\mathsf{ag}}') = (v_{i,0} \cdot c_1 + v_{1,1}) \cdot (t_1 - t_1').$$

Since $\mathcal{B}$ generated the public key components $v_{i,0}$ and $v_{i,1}$ itself, it knows the corresponding signing keys $\mathbf{s}_0, \mathbf{s}_1$ for $v_{i,0}, v_{i,1}$ such that $\mathbf{a}^{\mathsf{T}}\mathbf{s}_0 = v_{1,0}$ and $\mathbf{a}^{\mathsf{T}}\mathbf{s}_1 = v_{1,1}$. Hence, algorithm $\mathcal{B}$ can deduce the relation

$$\mathbf{a}^{\mathsf{T}}(\sigma_{\mathsf{ag}} - \sigma_{\mathsf{ag}}') = \mathbf{a}^{\mathsf{T}}(\mathbf{s}_0 \cdot c_1 + \mathbf{s}_1) \cdot (t_1 - t_1'),$$

which implies that $(\sigma_{\mathsf{ag}} - \sigma_{\mathsf{ag}}') - (\mathbf{s}_0 \cdot c_1 + \mathbf{s}_1) \cdot (t_1 - t_1')$ is a potential short solution to the SIS challenge as long as it is not an identically zero ring element. The bulk of the proof is in showing that this is indeed the case with constant probability. In the proof, we also bound the probability of a series of bad events that can cause the reduction algorithm to fail as in the proof of Definition 3.9.

Remark 4.5 (Optimization for $\mathcal{R} = \mathbb{Z}^{n\times n}$)**.** One can naturally optimize Construction 4.1 over integer lattices ($\mathcal{R} = \mathbb{Z}^{n\times n}$) by working with *vectors* as opposed to only *matrices*. By modifying the hash function H_1 to output short vectors in $\mathbb{Z}^n$ as opposed to short matrices in $\mathbb{Z}^{n\times n}$, the final aggregated signature becomes a short vector in $\mathbb{Z}^n$ as opposed to matrices $\mathbb{Z}^{n\times n}$. The proofs of correctness (Definition 4.3) and security (Definition 4.4) can be extended to this setting in a straightforward way.

One can naturally consider further shrinking the parameters of Construction 4.1 by working with vectors. For instance, by setting $\mathbf{s}_1$ to be vectors in $(\mathbb{Z}^n)^\ell$ as opposed to matrices in $(\mathbb{Z}^{n\times n})^\ell$, and then working with message hash functions H_0 that map messages to short vectors in $(\mathbb{Z}^n)^\ell$ as opposed to short matrices in $(\mathbb{Z}^{n\times n})^n$, we can greatly shrink the size of the public/secret keys as well as the signatures. When signature aggregation is not a concern, then this optimized construction can be shown to satisfy both correctness and security as a one-time signature scheme. However, now that the signatures themselves become vectors, the output of the hash function H_1 must be short scalar values in $\mathbb{Z}$, which causes the scheme to be insecure. As shown in Theorem 4.4, it is important that the output space of the hash function H_1 have have size at least 2^λ.

4.1 Setting the parameters

Theorems 4.2, 4.3 and 4.4 give some flexibility into how we can set the concrete parameters for applications.

- We can first set n to be large enough to prevent existing attacks on the SIS or Ring-SIS problem. For the case SIS, $n = 640$ is a reasonable choice to provide sufficient security against classical attacks and $n = 768$ is a reasonable choice for sufficient quantum security. For the Ring-SIS problem, $n = 1048$ can be reasonable for both classical and quantum security.

- Next, the modulus q can be set according to implementation considerations and the maximum number of signatures N that require aggregation for a particular application. A reasonable choice for q can be either a 24-bit prime, in which case elements in $\mathbb{Z}_q$ fit into a 3-byte unsigned integer, or a 32-bit prime, in which case elements fit into a 4-byte unsigned integer. The former case can safely support a small number of aggregation up to $N = 32$, and the latter case can safely support aggregation of up to $N = 1024$ signatures.

- Next, there is a trade-off between the vector length ℓ and the norm bounds $\beta_{\mathbf{s}}$, β_0, and β_1 for the secret key and the hash functions. One way to set these parameters is to first set $\beta_0 = \beta_1 = 1$ to minimize type conversions on binary outputs of hash functions. Then, we can set $\ell = 2\lceil \log q \rceil$ and $\beta_{\mathbf{s}}$ to be the smallest positive integer such that the security conditions of Theorem 4.4 are satisfied.

- Finally, once all the parameters above are set, the signature norm bound β_{ver} can be set to be large enough such that the correctness condition in Theorem 4.4 are satisfied, but still smaller than the modulus q.

5 Aggregate Signatures with Interaction

Although Constructions 3.6 and 4.1 are applicable to many practical scenarios such as blockchains, the fact that a single signing key can be used to sign only a single message can be a limiting factor for other applications. In this section, we define aggregate signatures with an interactive signing procedure. Instead of requiring that the signing and signature aggregation algorithms to be completely non-interactive algorithms as defined in Section 3, we allow signers to jointly generate compact signatures in an interactive protocol. This relaxation of the model can still be reasonable for many applications and it allows us to construct an aggregate signature scheme from lattices that allow unbounded re-use of signing keys (see Section 6).

In our interactive aggregate signature definition, the syntax of the parameter generation, key generation, and signature verification algorithm remains unchanged from Definition 3.1. However, the separate signature generation and signature aggregation algorithms are replaced with a joint protocol among a pre-determined set of signers. Suppose that N number of signers wish to generate a joint signature on a set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$. At the start of the protocol, each signer $i \in [N]$ has access to its own signing key sk_i and a message m_i to sign as well as the public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$ of other participants (co-signers) of the protocol. At the end of the protocol, each signer can generate a short signature σ_{ag} that certifies each messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ under each of the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$.

Definition 5.1 (Interactive Aggregate Signatures)**.** An interactive aggregate signature scheme $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ for a message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{AS}}^{\leftrightarrow} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}^{\leftrightarrow}, \mathsf{Verify})$ with the following properties:

- $\mathsf{PrmsGen}(1^\lambda)$: On input a security parameter λ, the parameter generation algorithm returns a set of public parameters pp.

- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: On input a set of public parameters pp, the key generation algorithm returns a signing key sk and a public key pk.

- $\mathsf{Sign}^{\leftrightarrow}\big\langle (\mathsf{PK}, \mathsf{sk}_i, \mathsf{m}_i)_{i\in[N]} \big\rangle \to \sigma_{\mathsf{ag}}/\bot$: The signing protocol is run by a set of signers in the system.
  - **Start of the protocol**: Each signer $i \in [N]$ holds a set of public keys $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$, its own signing key sk_i, and a message to sign $\mathsf{m}_i \in \mathcal{M}$.
  - **End of the protocol**: Each signer $i \in [N]$ holds an aggregate signature σ_{ag} that authenticates all messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ under the public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$.

- $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \to \{0, 1\}$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, a set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and a signature σ_{ag}, the verification algorithm either accepts (outputs 1) or rejects (outputs 0).

Similarly to a non-interactive aggregate signature scheme, we enforce a compactness requirement on the final signatures of the signing protocol in order to rule out trivial constructions.

Definition 5.2 (Compactness)**.** Let $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ be an interactive aggregate signature scheme for a message space $\mathcal{M}$. We say that $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ is compact if there exists a polynomial $\mathsf{poly}(\cdot)$ and a negligible function $\mathsf{negl}(\cdot)$ such that for all security parameter λ, number of signers $N \in \mathbb{N}$, and messages $\mathsf{m}_1, \ldots, \mathsf{m}_N \in \mathcal{M}$, if we set

1. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
2. $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$,
3. $\sigma_{\mathsf{ag}} \leftarrow \mathsf{Sign}^{\leftrightarrow}\big\langle (\mathsf{PK}, \mathsf{sk}_i, \mathsf{m}_i)_{i\in[N]} \big\rangle$,

then we have

$$\Pr\big[|\sigma_{\mathsf{ag}}| \leq \mathsf{poly}(\lambda, \log N)\big] = 1 - \mathsf{negl}(\lambda),$$

where $|\sigma_{\mathsf{ag}}|$ is the bit-length of σ_{ag}.

For the correctness condition, we can define an analogous definition to Definition 3.3 where the signing protocol produces a verifying signature with overwhelming probability. However, since the signing procedure is a protocol as oppose to a single signing algorithm, it is useful to relax this condition and allow the signing protocol to successfully produce a signature with only a constant probability. With sequential or parallel repetition, such a protocol can be upgraded into a protocol that succeeds to produce a valid signature with overwhelming probability.

Definition 5.3 (Correctness)**.** Let $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ be an interactive aggregate signature scheme for a message space $\mathcal{M}$. We say that $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ is correct if for all security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and messages $\mathsf{m}_1, \ldots, \mathsf{m}_N \in \mathcal{M}$, there exists a constant $C \in \mathbb{N}$ such that

$$\Pr\big[\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) = 1\big] = 1/C,$$

where $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, $\sigma_{\mathsf{ag}} \leftarrow \mathsf{Sign}_{\mathsf{pp}}^{\leftrightarrow}\big\langle (\mathsf{PK}, \mathsf{sk}_i, \mathsf{m}_i)_{i\in[N]} \big\rangle$, $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, and $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$.

Finally, we define the unforgeability experiment for an interactive aggregate signature scheme. The unforgeability security experiment is largely similar to Definition 3.4 except for the way an adversary's signing query is defined. Since there exists no single signing algorithm that an adversary can get access to, we allow the adversary to be able to instantiate the signing protocol with the challenger. An adversary does this by first submitting a set of public keys $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$ and a set of messages $\mathsf{M} = (\mathsf{m}_j)_{j\in[N]}$ to the challenger. For a valid initiation of the protocol, there must exist an index $i \in [N]$ for which pk_i corresponds to the challenger's public key that it provides to the adversary at the start of the experiment. Once an initialization of the signing protocol is made, the challenger plays the role of the i^{th} signer in the protocol while the adversary plays the role of the rest of the signers in the protocol. We allow the adversary to pause a protocol and instantiate a new signing protocol at any point in the experiment. Hence, an adversary can participate in multiple parallel executions of the signing protocol throughout the experiment.

Definition 5.4 (Unforgeability)**.** Let Π_{AS} be an interactive aggregate signature scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A}$, we define the signature unforgeability experiment $\mathsf{Expt}_{\mathsf{AS}^{\leftrightarrow}}[\lambda, \mathcal{A}]$ as follows:

1. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
2. $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$,
3. $(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \leftarrow \mathcal{A}^{\mathsf{Sign}^{\leftrightarrow}\langle\,\cdot\,\rangle}(\mathsf{pp}, \mathsf{pk}^*)$,
4. Output $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}})$.

where the oracle $\mathsf{Sign}^{\leftrightarrow}\langle\,\cdot\,\rangle$ is defined as follows:

- A valid signing query consists of a set of public keys $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$ and a set of messages $\mathsf{M} = (\mathsf{m}_j)_{j\in[N]}$ where $\mathsf{pk}_i = \mathsf{pk}^*$ for some $i \in [N]$.

  Each query $(\mathsf{PK}, \mathsf{M})$ instantiates a new signing protocol between a challenger and the adversary $\mathcal{A}$. The challenger plays the role of the signer with input $(\mathsf{PK}, \mathsf{sk}^*, \mathsf{m}_i)$ and the adversary plays the role of the other signers.

  For each protocol, the adversary $\mathcal{A}$ can submit new signing queries at any point in the experiment such that multiple protocols run concurrently.

We say that $\mathcal{A}$ is an admissible adversary for the unforgeability experiment if the following holds: for any $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$ that it returns at the end of the experiment, we have $\mathsf{pk}_i = \mathsf{pk}^*$ for some $i \in [N]$, and $\mathcal{A}$ never initiated a protocol with $(\mathsf{PK}, \mathsf{M})$ as an input. We say that an interactive aggregate signature scheme $\Pi_{\mathsf{AS}}^{\leftrightarrow}$ is unforgeable if for all efficient and admissible adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathsf{Expt}_{\mathsf{AS}^{\leftrightarrow}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

6 Aggregate Signatures via Interaction from Lattices

In this section, we present our interactive aggregate signature scheme that allow unbounded key-reuse. We first provide an overview of our construction in Section 6.1. Then, we establish basic notations and recall basic facts on the discrete Gaussian distribution in Section 6.2. We provide the formal construction description in Section 6.3 and its proofs in Appendix C.

6.1 Construction Overview

Our interactive aggregate signature scheme is inspired by the Bellare-Neven [BN06] aggregate signature scheme, which is also an extension of Schnorr signatures [Sch91]. We briefly recall these schemes to provide more context for our lattice-based scheme.

Schnorr and Bellare-Neven signatures. The Schnorr signature scheme is based on the Schnorr identification protocol [Sch91] that works over a group $\mathbb{G}$ of prime order p with a generator $g \in \mathbb{G}$. The public key of a user in the system consists of a group element $h \in \mathbb{G}$ and the corresponding secret key is the discrete log of the public key: $x \in \mathbb{Z}_p$ such that $g^x = h$. To prove knowledge of the secret exponent x, the prover (user) and a verifier proceeds as follows:

1. The prover generates a random exponent $y \xleftarrow{\$} \mathbb{Z}_p$ and sends its commitment $w \leftarrow g^y$ to the verifier.
2. The verifier sends a random challenge $c \xleftarrow{\$} \mathbb{Z}_p$ to the prover.
3. The prover provides $z \leftarrow c \cdot x + y \in \mathbb{Z}_p$ to the verifier.

Using standard arguments, one can show that the protocol is sound and honest-verifier zero-knowledge. The protocol can be compiled into a non-interactive signature scheme using the Fiat-Shamir transform [FS86]. A signature σ then consists of the prover's first and second messages $\sigma = (w, z)$ where c is derived from both the hash of the prover's public key and the message to be signed.

It is natural to ask whether the Schnorr signatures can be aggregated. Indeed, it is possible to aggregate the second components of Schnorr signatures as demonstrated in Section 3.2 (see Remark 3.12). However, one can quickly realize that any algebraic approach to aggregating the first components is difficult as they must be fed into a hash function that is modeled as a random oracle in order to verify the signature.

The Bellare-Neven overcomes this issue by allowing the signing algorithm to be interactive. Suppose that there are N number of signers indexed by $i = 1, \ldots, N$ that wish to sign messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ respectively. Each signer $i \in [N]$ holds a signing key $x_i \in \mathbb{Z}_p$ corresponding to its public key $h_i = g^{x_i}$. Then the signers can jointly generate a short signature that authenticates the entire set of messages as follows:

1. Each user $i \in [N]$ generates a random exponent $y_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts its commitment $w_i \leftarrow g^{y_i}$ to other signers.

2. Upon receiving the commitments from other signers, each signer i aggregates the commitments into a group product $w^\star \leftarrow \prod_{i\in[N]} w_i$ and uses it to proceed with the rest of the Schnorr protocol. Namely, it computes $z_i \leftarrow c_i \cdot x_i + y_i$ where $c_i \in \mathbb{Z}_p$ is the hash of the entire set of public keys, messages, the signer index, and the aggregated commitments. It broadcasts the components z_i to other signers.

3. Finally, upon receiving the components $z_1, \ldots, z_N$, each signer aggregates these components $z^\star \leftarrow \sum_{i\in[N]} z_i$ and defines the final signature to be $\big(w^\star, z^\star\big)$.

Since all the exponents $c_1, \ldots, c_N$ are now derived from the aggregated commitment $w^\star$, correctness of the interactive signature scheme can be easily verified. Bellare-Neven [BN06] showed that with an additional round of hash commitments to w_i's at the start of the protocol, the protocol can be shown to be secure.

Lyubashevsky signatures and our construction. Given the relation between the Schnorr and Bellare-Neven signature schemes as discussed above, there is a natural starting point to constructing an interactive aggregate signature scheme for lattices: the Lyubashevsky's identification protocol [Lyu08, Lyu12]. Lyubashevsky's protocol is a natural lattice analogue to the Schnorr identification protocol that works over a ring $\mathcal{R}$ with a public vector $\mathbf{a} \in \mathcal{R}_q^\ell$. A user's public key consists of a ring element $v \in \mathcal{R}_q$ and the corresponding secret key is a short vector $\mathbf{s} \in \mathcal{R}^\ell$ such that $\mathbf{a}^\intercal \mathbf{s} = v$. To prove knowledge of the secret key $\mathbf{s}$, the prover and a verifier proceeds in the protocol as follows:

1. The prover generates a short vector $\mathbf{y} \in \mathcal{R}^\ell$ and sends its commitment $w \leftarrow \mathbf{a}^\intercal \mathbf{y} \in \mathcal{R}_q$ to the verifier.

2. The verifier sends a random challenge c to the prover. The challenge c is a short ring element in $\mathcal{R}$.

3. With certain probability, the prover sends $\mathbf{z} \leftarrow \mathbf{s} \cdot c + \mathbf{y} \in \mathcal{R}^\ell$ to the verifier (see *rejection sampling* below).

At the end of the protocol, the verifier can check whether $\mathbf{a}^\intercal \mathbf{z} = v \cdot c + w$ and whether $\mathbf{z}$ has a small norm. With a proper *rejection* sampling mechanism that is incorporated into the final round, the protocol can be shown to be secure as an identification protocol [Lyu08, Lyu12].

Our interactive aggregate signature scheme then works similarly to the Bellare-Neven construction in the context of Lyubashevsky's protocol. As above, suppose that there are N number of signers indexed by $i = 1, \ldots, N$ that wish to sign messages $\mathsf{m}_1, \ldots, \mathsf{m}_N$ respectively. Each signer $i \in [N]$ holds a signing key $\mathbf{s}_i \in \mathcal{R}^\ell$ corresponding to its public key $v_i = \mathbf{a}^\intercal \mathbf{s}_i$. Then, the signers can jointly generate a short signature that authenticates the messages as follows:

1. Each user $i \in [N]$ generates a short vector $\mathbf{y}_i \in \mathcal{R}^\ell$ and broadcasts its commitment $w_i \leftarrow \mathbf{a}^\intercal \mathbf{y}_i \in \mathcal{R}_q$ to other signers.

2. Upon receiving the commitments from other signers, each signer i aggregates the commitments into a sum $w^\star \leftarrow \sum_{i\in[N]} w_i$ and uses it to proceed with the rest of Lyubashevsky's protocol. Namely, it computes $\mathbf{z}_i \leftarrow \mathbf{s}_i \cdot c_i + \mathbf{y}_i$ where $c_i \in \mathcal{R}$ is derived from $w^\star$. It broadcasts the component $\mathbf{z}_i$ to other signers.

3. Finally, upon receiving the components $\mathbf{z}_1, \ldots, \mathbf{z}_N$, each signer aggregates these components $\mathbf{z}^* \leftarrow \sum_{i\in[N]} \mathbf{z}_i$ and defines the final signature to be $(w^\star, \mathbf{z}^*)$.

Taking advantage of the linear verification condition, we can check the validity of an aggregated signature as before. In our formal protocol, we incorporate the additional round of hash commitments to w_i's at the start of the protocol as in the Bellare-Neven protocol.

Rejection sampling. Although our construction is analogous to the Bellare-Neven protocol, there is an additional technical challenge that we have to address in our protocol related to rejection sampling. In the

Lyubashevsky's scheme, the prover at the last step of the protocol (step 3 above) must reject the proof and output $\perp$ with certain probability. This step is called "rejection sampling" and it ensures that no information about the secret key $\mathbf{s}$ is leaked from the last message $\mathbf{z} = \mathbf{s} \cdot c_i + \mathbf{y}$. This step is needed because the vector $\mathbf{y}$ is only a short vector in $\mathcal{R}^\ell$ that cannot completely randomized $\mathbf{z}$ in the entire space of $\mathcal{R}^\ell$, unlike in the discrete log setting. The correctness (completeness) of the protocol can still be achieved with either sequential or parallel repetition of the protocol.

To prove the security of our protocol, we must incorporate rejection sampling in our protocol as well. Specifically, when a signer computes the aggregated commitments $w^\star$ and computes $\mathbf{z}_i \leftarrow \mathbf{s} \cdot c_i + \mathbf{y}_i$, it flips a coin and either broadcasts $\mathbf{z}_i$ to the other signers or notifies them that it rejected the signature. At the end of the protocol, if at least one of the signers rejected the signature, the final signature also becomes $\perp$.

In Section 6.4, we set the parameter of our protocol such that each signer rejects with a low enough probability so that the entire protocol rejects with at most a constant probability. If we want the protocol to produce a valid signature except with negligible probability, we can apply either sequential or parallel repetition of the protocol. Such repetition does increase the size of the total communication in the protocol, but the resulting aggregate signature still remains the same size.

6.2 Background on Gaussian Distribution and Rejection Sampling

We define the Gaussian function on $\mathbb{R}^n$ with Gaussian parameter $s \in \mathbb{R}^+$ that is centered at $\mathbf{c} \in \mathbb{R}^n$ as follows:

$$\forall\ \mathbf{x} \in \mathbb{R}^n, \quad \rho_{s,\mathbf{c}}(\mathbf{x}) = \exp(-\pi \left\| \mathbf{x} - \mathbf{c} \right\|^2 / s^2).$$

For any $\mathbf{c} \in \mathbb{R}^n$ and Gaussian parameter $s \in \mathbb{R}^+$, we define the *discrete Gaussian distribution* over $\mathbb{Z}^n$ denoted $\mathcal{D}_{s,\mathbf{c}} : \mathbb{Z}^n \to \mathbb{R}$ as follows:

$$\forall\ \mathbf{x} \in \mathbb{Z}^n, \quad \mathcal{D}_{s,\mathbf{c}}(\mathbf{x}) = \frac{\rho_{s,\mathbf{c}}(\mathbf{x})}{\rho_{s,\mathbf{c}}(\mathbb{Z}^n)}.$$

When $\mathbf{c} = \mathbf{0} \in \mathbb{Z}^n$, then we simply write $\mathcal{D}_s$ in place of $\mathcal{D}_{s,\mathbf{c}}$.

We can naturally extend the discrete Gaussian distribution over $\mathbb{Z}^n$ to the rings $\mathcal{R} = \mathbb{Z}^{n \times n}$ and $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$. When $\mathcal{R} = \mathbb{Z}^{n \times n}$, we define the discrete Gaussian distribution $\mathcal{D}_{\mathcal{R},s,c}$ for $c \in \mathcal{R}$ to be the distribution where each column of the matrix in $\mathbb{Z}^{n \times n}$ is distributed as in the discrete Gaussian distribution over $\mathbb{Z}^n$. When $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, we define the discrete Gaussian distribution $\mathcal{D}_{\mathcal{R},s,c}$ for $c \in \mathcal{R}$ to be the distribution where the n coefficients of the polynomial in $\mathbb{Z}[X]/(X^n + 1)$ are distributed as a discrete Gaussian vector over $\mathbb{Z}^n$. The discrete Gaussian distribution can then be extended to vectors of ring elements $\mathcal{R}^\ell$ for any $\ell \in \mathbb{N}$ in a natural way. We denote this discrete Gaussian distribution over $\mathcal{R}^\ell$ as $\mathcal{D}^\ell_{\mathcal{R},s,\mathbf{c}}$ where $s \in \mathbb{R}^+$ and $\mathbf{c} \in \mathcal{R}^\ell$.

For the proofs for Construction 6.7, we make use of a number of standard facts on the discrete Gaussian distribution. We recall these facts in this subsection. The following lemma bounds the norm of a discrete Gaussian sample in $\mathcal{R}$.

Lemma 6.1 ([Lyu12, Lemma 4.4])**.** *Let n and ℓ be positive integers, $s \in \mathbb{R}^+$ a Gaussian parameter, and let $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$. Then for any positive integer $k > 1$, we have*

$$\Pr_{\mathbf{z} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}} \left[\|\mathbf{z}\| > k \cdot s \right] \leq 2e^{\frac{-k^2}{2}}.$$

The following lemma bounds the ratio of two discrete Gaussian distributions with different centers. The lemma applies to both settings of $\mathcal{R} = \mathbb{Z}^{n \times n}$ and $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$.

Lemma 6.2 ([Lyu12, Lemma 4.5 adapted])**.** *Let n, ℓ, and N be positive integers and let $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$. Then, for any $\mathbf{v} \in \mathcal{R}^\ell$ and $s = \omega\big(\|\mathbf{v}\| \sqrt{\log n} \cdot \log \big(N/(N-1)\big)\big)$, we have*

$$\Pr_{\mathbf{z} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}} \left[\frac{\mathcal{D}^\ell_{\mathcal{R},s}(\mathbf{z})}{\mathcal{D}^\ell_{\mathcal{R},s,\mathbf{v}}(\mathbf{z})} = O\Big(\frac{N}{N-1}\Big) \right] = 1 - \ell \cdot 2^{-\omega(\log n)}.$$

Next, we recall the standard rejection sampling lemma of Lyubashevsky [Lyu12].

Lemma 6.3 (Rejection Sampling Lemma [Lyu12])**.** *Let n and ℓ be positive integers, $\mathcal{R}$ be a ring, V be any set, and $h : V \to \mathbb{R}$, $f : \mathcal{R}^\ell \to \mathbb{R}$ be probability distributions. Suppose that $g_v : \mathcal{R}^\ell \to \mathbb{R}$ is a family of probability distributions indexed by $v \in V$ with the following property: there exist $M, \varepsilon \in \mathbb{R}$ such that for all $v \in V$ and $\mathbf{z} \in \mathcal{R}^\ell$,*

$$\Pr_{z \leftarrow f}\big[M \cdot g_v(\mathbf{z}) \geq f(\mathbf{z})\big] \geq 1 - \varepsilon.$$

Then, the output distributions of the following two algorithms have statistical distance at most ε/M:

- *Sample $v \leftarrow h$, $\mathbf{z} \leftarrow g_v$, and return $(\mathbf{z}, v)$ with probability $f(z)/(M \cdot g_v(z))$.*
- *Sample $v \leftarrow h$, $\mathbf{z} \leftarrow f$, and return $(\mathbf{z}, v)$ with probability $1/M$.*

Finally, we recall the leftover hash lemma [HILL99]. For concreteness, we present the leftover hash lemma applied to each specific rings of $\mathcal{R} = \mathbb{Z}^{n \times n}$ and $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$.

Lemma 6.4 (Leftover Hash Lemma over $\mathcal{R} = \mathbb{Z}^{n\times n}$)**.** *Let n, ℓ, q be positive integers such that $\ell \geq 2 \log q$, let $\mathcal{R} = \mathbb{Z}^{n\times n}$ be a ring of integer matrices, and let $s = \omega(\sqrt{\log n})$ be a Gaussian parameter. Then, for $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$, $\mathbf{y} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}$, and $u \xleftarrow{\$} \mathcal{R}_q$, we have*

$$\Delta\big((\mathbf{a}, \mathbf{a}^\intercal \mathbf{y}), (\mathbf{a}, u)\big) \leq n \cdot q^{-n}.$$

Lemma 6.5 (Leftover Hash Lemma over $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$)**.** *Let n, ℓ, q be positive integers, $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a ring of polynomials, and let $s = \omega(\sqrt{\log n})$ be a Gaussian parameter. Furthermore, suppose that n is power-of-two integer, $\ell \geq 2 \log q$, and q is set such that $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ is a field. Then, for $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$, $\mathbf{y} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}$, and $u \xleftarrow{\$} \mathcal{R}_q$, we have*

$$\Delta\big((\mathbf{a}, \mathbf{a}^\intercal \mathbf{y}), (\mathbf{a}, u)\big) \leq q^{-n}.$$

We now state a core lemma that we use in the security proof for Construction 6.7 (Theorem 6.10).

Lemma 6.6. *Let n, ℓ, q, $\beta_{\mathbf{s}}$, β_1, and N be positive integers, $\mathcal{R} = \mathbb{Z}^{n\times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ be a ring, and $s = \omega\big(n \cdot \beta_{\mathbf{s}}\beta_1\sqrt{\log n} \cdot \log\big(N/(N-1)\big)\big)$ be a Gaussian parameter such that the conditions of Lemma 6.4 or Lemma 6.5 are satisfied. Then, there exists a constant $M \in \mathbb{R}$ such that the following two distributions are within statistical distance $\frac{N-1}{N} \cdot 2^{-\omega(\log n)}$.*

- Real distribution*:*
  1. *Sample $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$, $\mathbf{s} \xleftarrow{\$} \mathcal{B}^\ell_{\beta_{\mathbf{s}}}$, and set $v \leftarrow \mathbf{a}^\intercal\mathbf{s} \in \mathcal{R}_q$.*
  2. *Sample $\mathbf{y} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}$, $c \xleftarrow{\$} \mathcal{B}_{\beta_1}$, and set $\mathbf{z} \leftarrow \mathbf{s} \cdot c + \mathbf{y} \in \mathcal{R}$, $w \leftarrow \mathbf{a}^\intercal\mathbf{y} \in \mathcal{R}_q$.*
  3. *Let $p_{\mathsf{acc}} = \min\left(\frac{N-1}{N}\frac{\mathcal{D}^\ell_{\mathcal{R},s}(\mathbf{z})}{\mathcal{D}^\ell_{\mathcal{R},s,\mathbf{s}\cdot c}(\mathbf{z})}, 1\right)$. Then:*
     - *With probability p_{acc}, return $(\mathbf{a}, \mathbf{s}, w, \mathbf{z})$.   (accept)*
     - *With probability $1 - p_{\mathsf{acc}}$, return $(\mathbf{a}, \mathbf{s}, w, \perp)$.   (reject)*
- Ideal distribution*:*
  1. *Sample $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$, $\mathbf{s} \xleftarrow{\$} \mathcal{B}^\ell_{\beta_{\mathbf{s}}}$, and set $v \leftarrow \mathbf{a}^\intercal\mathbf{s} \in \mathcal{R}_q$.*
  2. *Sample $\mathbf{z} \leftarrow \mathcal{D}^\ell_{\mathcal{R},s}$, $c \xleftarrow{\$} \mathcal{B}_{\beta_1}$, and set $w \leftarrow \mathbf{a}^\intercal\mathbf{z} - \mathbf{a}^\intercal\mathbf{s} \cdot c$.*
  3. *Let $p_{\mathsf{acc}} = \frac{N-1}{N}$. Then:*
     - *With probability p_{acc}, return $(\mathbf{a}, \mathbf{s}, w, \mathbf{z})$.   (accept)*
     - *With probability $1 - p_{\mathsf{acc}}$, return $(\mathbf{a}, \mathbf{s}, w, \perp)$.   (reject)*

Proof. To prove the lemma, we show the following:

1. Let $p_{\mathsf{acc}}^{\mathsf{real}} = \min\left(\frac{N-1}{N} \cdot \frac{\mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z})}{\mathcal{D}^{\ell}_{\mathcal{R},s,\mathbf{s}\cdot c}(\mathbf{z})}, 1\right)$ and $p_{\mathsf{acc}}^{\mathsf{ideal}} = \frac{N-1}{N}$. Then $\left|p_{\mathsf{acc}}^{\mathsf{real}} - p_{\mathsf{acc}}^{\mathsf{ideal}}\right| = \frac{N-1}{N} \cdot 2^{-\omega(\log n)}$.
2. Given that the two distributions output accepting tuples, the two distributions have statistical distance $\frac{N-1}{N} \cdot 2^{-\omega(\log n)}$.
3. Given that the two distributions output rejecting tuples, the two distributions have statistical distance $2n \cdot q^{-n}$.

Conditions 1 and 2 follow from Lemmas 6.2 and 6.3. Namely, using Lemma 6.2 and the bound $s = \omega\big(\|\mathbf{s}\cdot c\| \sqrt{\log n} \cdot \log\big(N/(N-1)\big)\big) = \omega\big(n \cdot \beta_{\mathbf{s}}\beta_1 \sqrt{\log n} \log\big(N/(N-1)\big)\big)$, there exists a constant $M \in \mathbb{R}$ such that

$$\Pr_{\mathbf{z}\leftarrow\mathcal{D}^{\ell}_{\mathcal{R},s}}\left[M \cdot \mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z}) \geq \mathcal{D}^{\ell}_{\mathcal{R},s,\mathbf{s}\cdot c}(\mathbf{z})\right] = 1 - \ell \cdot 2^{-\omega(\log n)}.$$

Then, letting the set V to be $\mathcal{B}^{\ell}_{\beta_1}$, distribution f to be $\mathcal{D}^{\ell}_{\mathcal{R},s}$, and the function g_v to be $\mathcal{D}^{\ell}_{\mathcal{R},s,\mathbf{s}\cdot c}$ in Lemma 6.3, conditions 1 and 2 follow.

To show condition 3, we use the leftover hash lemma (Lemmas 6.4 and 6.5). Given that the real and ideal distributions output rejecting tuple, the two distributions are identical except for the way the element w is generated.

- In the real distribution, the element w is set to be $w = \mathbf{a}^{\intercal}\mathbf{y}$ for $\mathbf{y} \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$. Then, by the leftover hash lemma, the element w is within $n \cdot q^{-n}$ distance from a uniformly random element in $\mathcal{R}_q$.
- In the ideal distribution, the element w is set to be $w = \mathbf{a}^{\intercal}\mathbf{z} - \mathbf{a}^{\intercal}\mathbf{s} \cdot c$ where $\mathbf{z} \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$ and the components s and c are independent of $\mathbf{z}$. Therefore, by the leftover hash lemma, the vector w is within $n \cdot q^{-n}$ distance from a uniformly random element in $\mathcal{R}_q$.

Combining conditions 1, 2, and 3, the lemma follows. □

6.3 Formal Construction Description

In this section, we provide the formal description of our interactive aggregate signature construction. We provide the main intuition behind the construction in Section 6.1.

Construction 6.7. Let ℓ, q, r, $\beta_{\mathbf{s}}$, β_1, and β_{ver} be positive integers and $s \in \mathbb{R}^+$ be a Gaussian parameter. Let $\mathcal{M}$ be a message space and let $H_0 : \mathcal{R}_q^{\ell} \to \{0,1\}^r$, $H_1 : \{0,1\}^* \to \mathcal{B}_{\beta_1}$ be hash functions that are modeled as random oracles. We construct an interactive aggregate signature scheme as follows:

- $\mathsf{PrmsGen}(1^{\lambda})$: On input a security parameter λ, the parameter generation algorithm samples a random vector $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^{\ell}$ and sets $\mathsf{pp} = \mathbf{a}$.
- $\mathsf{KeyGen}(\mathsf{pp})$: On input the public parameters $\mathsf{pp} = \mathbf{a}$, the key generation algorithm samples a short vector $\mathbf{s} \xleftarrow{\$} \mathcal{B}^{\ell}_{\beta_{\mathbf{s}}}$ and defines $v \leftarrow \mathbf{a}^{\intercal}\mathbf{s} \in \mathcal{R}_q$. It sets
$$\mathsf{sk} = \mathbf{s}, \quad \mathsf{pk} = v.$$
- $\mathsf{Sign}^{\leftrightarrow}\langle(\mathsf{PK}, \mathsf{sk}_i, \mathsf{m}_i)_{i\in[N]}\rangle$: Each signer $i \in [N]$ has access to a list of public keys $\mathsf{PK} = (v_j)_{j\in[N]}$, its own signing key $\mathsf{sk}_i = \mathbf{s}_i$, and a message $\mathsf{m}_i \in \mathcal{M}$ that it seeks to sign. It proceeds in each round of the protocol as follows:
  - **Round 1**: Signer i samples a vector $\mathbf{y}_i \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathcal{R},s}$ and computes $w_i \leftarrow \mathbf{a}^{\intercal}\mathbf{y}_i \in \mathcal{R}_q$. It computes a hash commitment $h_i \leftarrow H_0(w_i)$ and sends $h_i \in \{0,1\}^r$ to each cosigners.
  - **Round 2**: Signer i receives the hash commitments $(h_j)_{j\in[N]\setminus\{i\}}$ from the cosigners. It sends the committed value w_i to each cosigners.

  - **Round 3**: Signer i receives $(w_j)_{j\in[N]\setminus\{i\}}$ from the cosigners. It proceeds as follows:
    1. It verifies that $h_j = H_0(w_j)$ for all $j \in [N]$. It aborts the protocol and returns $\perp$ if any of these checks fail.
    2. It computes $w_{\mathsf{ag}} \leftarrow \sum_{j\in[N]} w_j$ and computes the hash $c_i \leftarrow H_1(\mathsf{pk}_i, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$.
    3. It sets $\mathbf{z}_i \leftarrow \mathbf{s}_i \cdot c_i + \mathbf{y}_i$. It sends $\mathbf{z}_i$ to each cosigners with probability $\left(\frac{N-1}{N}\frac{\mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z}_i)}{\mathcal{D}^{\ell}_{\mathcal{R},s,\mathbf{s}\cdot c_i}(\mathbf{z}_i)}, 1\right)$. Otherwise, it sends abort to the signers to signal that it aborted.

  If there exists at least one signer that aborted during the protocol, each signer returns $\perp$. Otherwise, at the end of the protocol, each signer $i \in [N]$ holds the element $w_{\mathsf{ag}} \in \mathcal{R}_q$ as well as the set of vectors $\{\mathbf{z}_j\}_{j\in[N]}$. Each signer sets $\mathbf{z}_{\mathsf{ag}} \leftarrow \sum_{j\in[N]} \mathbf{z}_j$ and returns

$$\sigma_{\mathsf{ag}} = \big(w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}\big).$$

- $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma)$: On input a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and a signature $\sigma = (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$, the verification algorithm computes $c_i \leftarrow H_1(\mathsf{pk}_i, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$ for $i \in [N]$, and defines $v_{\mathsf{ag}} \leftarrow \sum_{i\in[N]} v_i \cdot c_i$. It accepts the signature if both of the following conditions hold:
  - $\mathbf{a}^{\mathsf{T}}\mathbf{z}_{\mathsf{ag}} = v_{\mathsf{ag}} + w_{\mathsf{ag}}$,
  - $\|\mathbf{z}_{\mathsf{ag}}\| \le \beta_{\mathsf{ver}}$.

We now state the compactness, correctness, and security theorems for Construction 6.7.

Theorem 6.8 (Compactness)**.** *Let $n = \mathsf{poly}(\lambda)$ be a positive integer, $\mathcal{R} = \mathbb{Z}^{n\times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, and suppose that the parameters $\ell, q, \beta_{\mathbf{s}}, \beta_1 = \mathsf{poly}(\lambda)$ are set as polynomial functions over λ. Then, the aggregate signature scheme in Construction 6.7 is compact (Definition 5.2).*

Theorem 6.9 (Correctness)**.** *Suppose that the parameters of Construction 6.7 are set such that the ring $\mathcal{R}$ is instantiated as either $\mathcal{R} = \mathbb{Z}^{n\times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and that the inequality $N \cdot (n\beta_{\mathbf{s}}\beta_1 + \lambda \cdot s) \le \beta_{\mathsf{ver}}$ holds. Then, Construction 6.7 is correct (Definition 5.3).*

Theorem 6.10 (Security)**.** *Suppose that the parameters of Construction 6.7 are instantiated such that the ring $\mathcal{R}$ is instantiated with either $\mathcal{R} = \mathbb{Z}^{n\times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, $s = \omega(n \cdot \beta_{\mathbf{s}}\beta_1\sqrt{\log n})$ such that Lemma 6.6, and that the parameters satisfy the conditions of the leftover hash lemma (Lemma 6.4 and 6.5). With these parameter settings, assume that the SIS problem $\mathsf{SIS}_{\mathcal{R},\ell,q,\beta^\star}$ for $\beta^\star = 2\beta_{\mathsf{ver}} + 2n\beta_{\mathbf{s}}\beta_1$ is hard. Then, the interactive aggregate signature scheme in Construction 6.7 satisfies unforgeability (Definition 5.4).*

More precisely, suppose that there exists an adversary $\mathcal{A}$ in the unforgeability experiment $\mathsf{Expt}_{\mathsf{AS}^{\leftrightarrow}}[\lambda, \mathcal{A}]$ that makes at most Q_0 random-oracle queries to H_0, Q_1 random-oracle queries to H_1, and Q_{sign} signing queries such that

$$\varepsilon = \Pr[\mathsf{Expt}_{\mathsf{AS}^{\leftrightarrow}}[\lambda, \mathcal{A}] = 1].$$

Then, there exists an SIS algorithm $\mathcal{B}$ such that

$$\varepsilon \le \gamma + \frac{Q_1 + Q_{\mathsf{sign}}}{|\mathcal{B}_{\beta_1}|} + \sqrt{(Q_1 + Q_{\mathsf{sign}}) \cdot \mathsf{Adv}_{\mathsf{SIS}_{\mathcal{R},\ell,q,\beta^\star}}[\lambda, \mathcal{B}] + \frac{1}{\mathcal{B}_{\beta_{\mathbf{s}}}}},$$

where

$$\gamma = \frac{N \cdot Q_{\mathsf{sign}} + (Q_0 + N \cdot Q_{\mathsf{sign}})^2}{2^r} + \frac{2n \cdot Q_{\mathsf{sign}}}{q^n} + \frac{1}{|\mathcal{B}_{\beta_1}|} + \frac{Q_{\mathsf{sign}} \cdot 2^{-\omega(\log n)}}{M}.$$

We provide the formal proofs of the statements above in Appendix C.

Security proof overview. The proof of Theorem 6.10 uses the similar ideas that are used in the proof of Construction 4.1. We construct a reduction algorithm $\mathcal{B}$ that given an SIS challenge $\mathbf{a} \in \mathcal{R}_q^\ell$ runs two executions of the unforgeability experiment for an adversary $\mathcal{A}$. Algorithm $\mathcal{B}$ simulates $\mathcal{A}$'s views of the two experiments as follows:

- During the first execution, algorithm $\mathcal{B}$ generates a secret-public key pair $(\mathbf{s}^*, v^*)$ following the real key-generation algorithm. It simulates the unforgeability experiment exactly as in Definition 5.4 using its signing key.
- Suppose that at the end of the experiment, adversary $\mathcal{A}$ produces a valid forgery that consists of a set of public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$, and signature $\sigma_{\mathsf{ag}} = (w_{\mathsf{ag}}, v_{\mathsf{ag}})$. For simplicity, suppose that $\mathsf{pk}_1 = \mathsf{pk}^*$.
- Since the hash function H_1 is modeled as a random-oracle, $\mathcal{A}$ must have submitted a random-oracle query on input $(\mathsf{pk}_1, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_1)$ to H_1 during the first execution of the experiment to have any non-negligible probability of succeeding in a forgery. Algorithm $\mathcal{B}$ rewinds the experiment to the point when $\mathcal{A}$ makes this query. Starting from this point, algorithm $\mathcal{B}$ re-runs the unforgeability experiment with fresh randomness.
- At the end of the second execution of the protocol, algorithm $\mathcal{B}$ receives another valid forgery that consists of a set of public keys $\mathsf{PK}' = (\mathsf{pk}'_i)_{i\in[N]}$, messages $\mathsf{M}' = (\mathsf{m}'_i)_{i\in[N]}$, and signature $\sigma'_{\mathsf{ag}} = (w'_{\mathsf{ag}}, v'_{\mathsf{ag}})$.

Now, suppose that the two forgeries that $\mathcal{B}$ receives from $\mathcal{A}$ in the two executions are on the *same* set of public keys and messages: $\mathsf{PK} = \mathsf{PK}'$ and $\mathsf{M} = \mathsf{M}'$. Then similarly to the proof of Construction 4.1, algorithm $\mathcal{B}$ can deduce a potential solution to the SIS challenge $\mathbf{a}$. Assuming that $\mathcal{A}$'s two forgeries are valid, we have

$$\mathbf{a}^\intercal \mathbf{z}_{\mathsf{ag}} = v_1 \cdot c_1 + \sum_{i\in[N]\setminus\{1\}} v_i \cdot c_i + w_{\mathsf{ag}},$$

$$\mathbf{a}^\intercal \mathbf{z}'_{\mathsf{ag}} = v_1 \cdot c'_1 + \sum_{i\in[N]\setminus\{1\}} v_i \cdot c'_i + w'_{\mathsf{ag}},$$

where $v_i = \mathsf{pk}_i$ for $i \in [N]$ are the public key components, $c_i \leftarrow H_1(\mathsf{pk}_i, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$ for $i \in [N]$ are the hash outputs of H_1 during the first execution of the experiment, and $c'_i \leftarrow H_1(\mathsf{pk}_i, w'_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$ for $i \in [N]$ are the outputs of H_2 during the second execution. Then as long as $c_i = c'_i$ for all $i \in [N]\setminus\{1\}$ and $w_{\mathsf{ag}} = w'_{\mathsf{ag}}$, the verification relation above can be combined into a simple relation

$$\mathbf{a}^\intercal (\mathbf{z}_{\mathsf{ag}} - \mathbf{z}'_{\mathsf{ag}}) = v_1 \cdot (c_1 - c'_1).$$

Since $v_1 = v^* = \mathbf{a}^\intercal \mathbf{s}^*$, algorithm $\mathcal{B}$ can now deduce a potential solution to the SIS challenge:

$$\mathbf{a}^\intercal \left(\mathbf{z}_{\mathsf{ag}} - \mathbf{z}'_{\mathsf{ag}} - \mathbf{s}^*(c_1 - c'_1)\right) = 0.$$

Using similar arguments as in the proof of Theorem 4.4 in combination with the rejection sampling lemma (Lemma 6.6), we can bound the probability that the vector $\mathbf{z}_{\mathsf{ag}} - \mathbf{z}'_{\mathsf{ag}} - \mathbf{s}^*(c_1 - c'_1)$ is a short non-zero vector.

The technical part of the proof is in the way algorithm $\mathcal{B}$ manipulates the two executions of the experiment such that $c_i = c'_i$ for $i \in [N]$ and $w_{\mathsf{ag}} = w'_{\mathsf{ag}}$. For this, we rely on the properties of the random-oracle in combination with rewinding. Due to the hash commitments in the first round of each protocol, adversary $\mathcal{A}$ is forced reveal any of its round 1 and round 2 messages of a protocol instance to $\mathcal{B}$ before the protocol actually starts. This allows algorithm $\mathcal{B}$ to program the random-oracle H_1 such that $c_i = c'_i$ for $i \in [N]\setminus\{1\}$ and $w_{\mathsf{ag}} = w'_{\mathsf{ag}}$. We refer to Appendix C for the full details.

6.4 Setting the parameters

We can set the parameters similarly to Section 4.1 with the precise requirements on Theorems 6.8, 6.9 and 6.10.

- We can first set n to be large enough to prevent existing attack on the SIS or Ring-SIS problem. As was the case for Construction 4.1, $n = 640$ (classical security) or $n = 768$ (quantum security) are reasonable choices for the SIS instantiation of the construction. For the Ring-SIS instantiation, $n = 1048$ can be reasonable for both classical and quantum security.
- Next, we can set the modulus q according implementation considerations and the maximum number of signers N for an instance of the signing protocol. Compared to Construction 4.1, the signing protocol in Construction 6.7 does not involve weighted sums of signatures and therefore, the norm of the final signature can be smaller. However, the requirements for the rejection sampling lemma enforces additional lower bound on the modulus q. As in Section 4.1, we can reasonably set q to be either a 24-bit prime or a 32-bit prime according to an upper bound on the number of signers in a particular application.
- As in Section 4.1, we can set $\beta_1 = 1$ for efficiency. The parameter r that determines the output of the hash function H_0 can be set to be 256. Then, it is reasonable to set $\ell = 2\lceil \log q \rceil$, and then set β_{s} and σ to be the smallest positive integers such that Theorem 6.10 is satisfied.
- Finally, once all the parameters above are set, the signature norm bound β_{ver} can be set to be large enough such that the correctness condition in Theorem 6.9 is satisfied, but still smaller than the modulus q.

References

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC'13*, pages 111–120, 2013.

[BCJ08] Ali Bagherzandi, Jung-Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *CCS*, 2008.

[BDE$^+$13] Johannes Buchmann, Erik Dahmen, Sarah Ereth, Andreas Hülsing, and Markus Rückert. On the security of the winternitz one-time signature scheme. *International Journal of Applied Cryptography*, 3(1):84–96, 2013.

[BDN18a] Dan Boneh, Manu Drijvers, and Gregory Neven. Bls multi-signatures with public-key aggregation. `https://crypto.stanford.edu/~dabo/pubs/papers/BLSmultisig.html`, 2018.

[BDN18b] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, 2018.

[BGLS03] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, 2003.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, 2001.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS*, 2006.

[BNN07] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Unrestricted aggregate signatures. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, pages 411–422, 2007.

[BNPS03] Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-rsa-inversion problems and the security of chaum's blind signature scheme. *Journal of Cryptology*, 16(3), 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *PKC*, 2003.

[BS08] Mihir Bellare and Sarah Shoup. Two-tier signatures from the fiat-shamir transform, with applications to strongly unforgeable and one-time signatures. *IET Information Security*, 2(2):47–63, 2008.

[BS16] Rachid El Bansarkhani and Jan Sturm. An efficient lattice-based multisignature scheme with applications to bitcoins. In *CANS 2016*, pages 140–155, 2016.

[CC17] Ran Canetti and Yilei Chen. Constraint-hiding constrained prfs for nc1 from lwe. In *EUROCRYPT*, 2017.

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT 2004*, pages 207–222, 2004.

[DDN03] Danny Dolev, Cynthia Dwork, and Moni Naor. Nonmalleable cryptography. *SIAM Review*, 45(4):727–784, 2003.

[DEF$^+$19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE S&P*, pages 1084–1101. IEEE, 2019.

[DGNW19] Manu Drijvers, Sergey Gorbunov, Gregory Neven, and Hoeteck Wee. Pixel: Multi-signatures for consensus. *IACR Cryptol. ePrint Arch.*, 2019:514, 2019.

[DHSS20] Yarkın Doröz, Jeffrey Hoffstein, Joseph H. Silverman, and Berk Sunar. Mmsat: A scheme for multimessage multiuser signature aggregation. Crypto ePrint Archive, Report 2020/520, 2020. eprint.iacr.org/2020/520.

[EGM96] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *J. Cryptology*, 9(1):35–67, 1996.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986.

[GR01] Rosario Gennaro and Pankaj Rohatgi. How to sign digital streams. *Inf. Comput.*, 165(1):100–116, 2001.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HKW15] Susan Hohenberger, Venkata Koppula, and Brent Waters. Universal signature aggregators. In *EUROCRYPT*, 2015.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *ANTS*, 1998.

[IN83] Kazuharu Itakura and Katsuhiro Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71:1–8, 1983.

[Lam79] Leslie Lamport. Constructing digital signatures from a one-way function. Technical report, Technical Report CSL-98, SRI International Palo Alto, 1979.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP*, 2006.

[LM08] Vadim Lyubashevsky and Daniele Micciancio. Asymptotically efficient lattice-based digital signatures. In *TCC*, 2008.

[LOS$^+$06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, 2006.

[Lyu08] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. In *International Workshop on Public Key Cryptography*, pages 162–179. Springer, 2008.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT*, 2012.

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *computational complexity*, 16(4):365–411, 2007.

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures. In *CCS*, pages 245–254. ACM, 2001.

[MPSW18a] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018. `https://eprint.iacr.org/2018/068`.

[MPSW18b] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple schnorr multi-signatures with applications to bitcoin. Cryptology ePrint Archive, Report 2018/068, 2018. `https://eprint.iacr.org/2018/068`.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In *CRYPTO*, 1992.

[OO99] Kazuo Ohta and Tatsuaki Okamoto. Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 82(1):21–31, 1999.

[PR06] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC*, 2006.

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

A Proofs for Bellare-Shoup Signatures

A.1 Proof of Theorem 3.7

By construction, any signature that is output by the signing algorithm Sign or the signature aggregation algorithm SigAgg consists of a single field element in $\mathbb{Z}_p$. Therefore, the size of an aggregate signature depends only on the security parameter and is independent of the number of signatures that were aggregated.

A.2 Proof of Theorem 3.8

Fix a security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and any set of messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$. Let $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, and $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{m}_i)$ for $i \in [N]$. By definition, each signing key consists of two field elements $\mathsf{sk}_i = (x_{i,0}, x_{i,1}) \in \mathbb{Z}_p^2$, each public key consists of two group elements $\mathsf{pk}_i = (h_{i,0}, h_{i,1}) \in \mathbb{G}^2$, and each signature has the form $\sigma_i = c_i \cdot x_{i,0} + x_{i,1}$ for $i \in [N]$ and $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$. The signature aggregation algorithm computes $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$ and sets $\sigma_{\mathsf{ag}} = \sum_{i \in [N]} t_i \cdot \sigma_i$. Therefore, by linearity, the aggregated signature satisfies the relation

$$g^{\sigma_{\mathsf{ag}}} = g^{\sum_{i \in [N]} t_i \cdot \sigma_i} = \prod_{i \in [N]} \left(g^{\sigma_i}\right)^{t_i} = \prod_{i \in [N]} \left(h_{i,0}^{c_i} \cdot h_{i,1}\right)^{t_i},$$

and the verification algorithm always accepts σ_{ag}. The correctness follows.

A.3 Proof of Theorem 3.9

To prove Theorem 3.9, we first define a *selective* variant of the one-time unforgeability experiment of Definition 3.5 where an adversary is forced to commit to its signing query *at the start* of the experiment. We show that an adversary that wins in original experiment of Definition 3.5 can be converted into an adversary that wins in the selective experiment with only a small loss in its winning advantage. Finally, we show that an adversary that wins in the selective unforgeability experiment can be converted into an algorithm for discrete log (Definition 2.1).

We first define the selective one-time unforgeability experiment. This security experiment is identical to Definition 3.5 except that the adversary must commit to its signing query $\hat{\mathsf{m}}$ before the start of the experiment.

Definition A.1 (Selective Unforgeability for OTS)**.** Let Π_{AS} be an aggregate signature scheme for a message space $\mathcal{M}$. For a security parameter $\lambda \in \mathbb{N}$ and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we define the *selective* one-time signature unforgeability experiment $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$ as follows:

1. $(\hat{\mathsf{m}}, \mathsf{state}) \leftarrow \mathcal{A}_1(1^\lambda)$,
2. $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$,
3. $(\mathsf{sk}^*, \mathsf{pk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$,
4. $\hat{\sigma} \leftarrow \mathsf{Sign}(\mathsf{sk}^*, \hat{\mathsf{m}})$,
5. $(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}}) \leftarrow \mathcal{A}_2(\mathsf{state}, \mathsf{pk}^*, \hat{\sigma})$,
6. Output $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}})$.

We say that $\mathcal{A}$ is an admissible adversary for the selective unforgeability experiment if for any execution of the experiment, the following holds: for any forgery $(\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}, \mathsf{M} = (\mathsf{m}_i)_{i \in [N]}, \sigma_{\mathsf{ag}})$ that it returns at the end of the experiment, we have $\mathsf{pk}_{i^*} = \mathsf{pk}^*$ for some $i^* \in [N]$, and $\mathsf{m}_{i^*} \neq \hat{\mathsf{m}}$. We say that an aggregate signature scheme Π_{AS} satisfies *selective* unforgeability if for all efficient and admissible adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}] = 1\right] = \mathsf{negl}(\lambda).$$

We show that an adversary that wins in Definition 3.5 also wins in Definition A.1 without too much loss in its winning probability. The following lemma follows by a simple guessing argument.

Lemma A.2. *Let $\mathcal{A}$ be an adversary in $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ (Definition 3.5) that makes at most Q_0 random-oracle queries to H_0 and Q_1 random-oracle queries to H_1. Then, there exists an adversary $\mathcal{B}$ in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$ (Definition A.1) that makes at most a single random-oracle query to H_1 such that*

$$\Pr\left[\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1\right] \leq Q_0 \cdot Q_1 \cdot |\mathcal{M}| \cdot \Pr\left[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}] = 1\right] + \frac{1}{p}.$$

Proof. We construct an algorithm $\mathcal{B}$, which uses $\mathcal{A}$ to break $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$. Algorithm $\mathcal{B}$ simulates the experiment $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ as follows:

- At the start of the experiment, $\mathcal{B}$ sample a random message $\hat{\mathsf{m}}' \xleftarrow{\$} \mathcal{M}$ and commits to $\hat{\mathsf{m}}'$ in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$. It receives a public key $\mathsf{pk}^* = (h_0^*, h_1^*)$ and a signature $\hat{\sigma}$ from the challenger. It provides pk^* to $\mathcal{A}$. Then, it initializes two tables
  - $\mathsf{T}_0 : \mathcal{M} \to \mathbb{Z}_p$ - used to answer $\mathcal{A}$'s random oracle queries to H_0 consistently.
  - $\mathsf{T}_1 : \{0,1\}^* \to (\mathbb{Z}_p)^*$ - used to answer $\mathcal{A}$'s random oracle queries to H_1 consistently.

  Finally, $\mathcal{B}$ samples random indices $j_0 \xleftarrow{\$} [Q_0]$, $j_1 \xleftarrow{\$} [Q_1]$.
- Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s queries as follows:
  - *Query to H_0*: For $\mathcal{A}$'s j_0^{th} query m_{j_0} to H_0, algorithm $\mathcal{B}$ uses its challenger to simulate $H(\mathsf{m}_{j_0})$. Namely, it submits $\hat{\mathsf{m}}'$ (which it generated at the start of the experiment) as an oracle query to H_0 in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$, receives a response $\hat{c} \in \mathbb{Z}_p$, sets $\mathsf{T}_0[\mathsf{m}_{j_0}] \leftarrow \hat{c}$, and provides $\hat{c}$ to $\mathcal{A}$.
    For the rest of $\mathcal{A}$'s random oracle queries m to H_0, algorithm $\mathcal{B}$ first checks if $\mathsf{T}_0[\mathsf{m}] = \perp$. If this is the case, it samples a random element $c \xleftarrow{\$} \mathbb{Z}_p$ itself, sets $\mathsf{T}_0[\mathsf{m}] \leftarrow c$, and provides c to $\mathcal{A}$. If $\mathsf{T}_0[\mathsf{m}] \neq \perp$, it simply provides $c \leftarrow \mathsf{T}_0[\mathsf{m}]$ to $\mathcal{A}$.
  - *Query to H_1*: For $\mathcal{A}$'s j_1^{th} query $(\tilde{\mathsf{PK}} = (\mathsf{pk}_i)_{i \in [N]}, \tilde{\mathsf{M}} = (\mathsf{m}_i)_{i \in [N]})$ to H_1, algorithm $\mathcal{B}$ verifies whether there exists an index i^* such that $\mathsf{pk}_{i^*} = \mathsf{pk}^*$. If no such index exists, then $\mathcal{B}$ aborts and returns $\perp$. Otherwise, it submits $(\tilde{\mathsf{PK}}, \tilde{\mathsf{M}})$ to its challenger, receives a response $(t_1, \ldots, t_N) \in \mathbb{Z}_p^N$, sets $\mathsf{T}_1[\tilde{\mathsf{PK}}, \tilde{\mathsf{M}}] \leftarrow (t_1, \ldots, t_N) \in \mathbb{Z}_p^N$, and provides $(t_1, \ldots, t_N)$ to $\mathcal{A}$.
    For the rest of $\mathcal{A}$'s random oracle queries $(\mathsf{PK}, \mathsf{M})$ to H_1, algorithm $\mathcal{B}$ first checks if $\mathsf{T}_1[\mathsf{PK}, \mathsf{M}] = \perp$. If this is the case, then it samples a random vector $(t_1, \ldots, t_N) \xleftarrow{\$} \mathbb{Z}_p^N$, sets $\mathsf{T}_1[\mathsf{PK}, \mathsf{M}] \leftarrow (t_1, \ldots, t_N)$, and provides $(t_1, \ldots, t_N)$ to $\mathcal{A}$. If $\mathsf{T}_1[\mathsf{PK}, \mathsf{M}] \neq \perp$, it simply provides $(t_1, \ldots, t_N) \leftarrow \mathsf{T}_1[\mathsf{PK}, \mathsf{M}]$ to $\mathcal{A}$.
  - *Signing query*: For $\mathcal{A}$'s single signing query $\hat{\mathsf{m}}$, algorithm $\mathcal{B}$ provides $\hat{\sigma}$ to $\mathcal{A}$.
- At the end of the experiment, adversary $\mathcal{A}$ returns a forgery $(\mathsf{PK}, \mathsf{M}, \sigma)$. Algorithm $\mathcal{B}$ first parses the public keys $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$ and messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and then verifies the following conditions:
  - The message that $\mathcal{A}$ submitted to the signing oracle is equal to its j_0^{th} query to H_0: $\hat{\mathsf{m}} = \mathsf{m}_{j_0}$.
  - The set of public keys and messages that $\mathcal{A}$ submitted as part of its forgery is equal to its j_1^{th} query to H_1: $(\mathsf{PK}, \mathsf{M}) = (\tilde{\mathsf{PK}}, \tilde{\mathsf{M}})$.
  - Let $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$. Then, there exists $i^* \in [N]$ such that $\mathsf{pk}_{i^*} = \mathsf{pk}^*$. Furthermore, $\hat{\mathsf{m}} \neq \mathsf{m}_{i^*}$.

  If any of these conditions are not satisfied, then it outputs fail. Otherwise, algorithm $\mathcal{B}$ returns $(\mathsf{PK}, \mathsf{M}, \sigma)$ as its own forgery in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$.

To prove the lemma, we show that as long as $\mathcal{B}$ does not output fail at the end of the experiment:

1. Algorithm $\mathcal{B}$ correctly simulates the experiment $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ for $\mathcal{A}$.
2. If $\mathcal{A}$ successfully forges a signature in $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, then $\mathcal{B}$ successfully forges a signature in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$.

Then, we bound the probability that $\mathcal{B}$ does not output fail and that $\mathcal{A}$ successfully returns a forgery to complete the proof.

Correctness of simulation. Suppose that $\mathcal{B}$ does not output fail at the end of the experiment. We note that the way a challenger generates the public keys in $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ and $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$ are identical. Furthermore, the way a challenger answers an adversary's queries in the two experiments are also identical. Therefore, the only component of $\mathcal{B}$'s simulation that we must verify correctness of is $\mathcal{B}$'s simulation of $\mathcal{A}$'s queries to H_0 and H_1 excluding its j_0^{th} and j_1^{th} queries to these hash functions respectively. By specification, algorithm $\mathcal{B}$ provides $\mathcal{A}$ with uniformly sampled outputs $c \xleftarrow{\$} \mathbb{Z}_p$ for H_0 and $(t_1, \ldots, t_N) \xleftarrow{\$} \mathbb{Z}_p^N$ for H_0. Since the hash functions H_0 and H_1 are modeled as random oracles and the outputs of these hash functions are independent of the rest of $\mathcal{A}$'s queries in the experiment assuming that $\hat{\mathsf{m}} = \mathsf{m}_{j_0}$ and $(\mathsf{PK}, \mathsf{M}) = (\tilde{\mathsf{PK}}, \tilde{\mathsf{M}})$, this is a correct simulation.

Correctness of forgery. Suppose that $\mathcal{B}$ does not output fail at the end of the experiment and that $\mathcal{A}$ returns a valid forgery $(\mathsf{PK}, \mathsf{M}, \sigma)$ for $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$. Then by definition of a forgery in $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, it must be the case that

1. $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma) = 1$,
2. $\mathsf{pk}_{i^*} = \mathsf{pk}^*$ for some $i^* \in [N]$,
3. $\hat{\mathsf{m}} \neq \mathsf{m}_{i^*}$,

where pk^* is the public key that $\mathcal{B}$ provided to $\mathcal{A}$ during the query phase and $\hat{\mathsf{m}}$ is the single signing query that $\mathcal{A}$ submitted during the signing query. To prove that $(\mathsf{PK}, \mathsf{M}, \sigma)$ is also a valid forgery in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$, we must show that

1. $\mathsf{Verify}(\mathsf{PK}, \mathsf{M}, \sigma) = 1$
2. $\mathsf{pk}_{i^*} = \mathsf{pk}^*$ for some $i^* \in [N]$,
3. $\hat{\mathsf{m}}' \neq \mathsf{m}_{i^*}$,

where pk^* is the public key that $\mathcal{B}$ receives from its challenger and $\hat{\mathsf{m}}'$ is the message that it commits at the start of the experiment. Conditions 1 and 2 follows simply by specification of $\mathcal{B}$. Furthermore, assuming that $\mathcal{B}$ does not output fail, we have $\hat{\mathsf{m}}' \neq \hat{\mathsf{m}}$ and therefore, condition 3 is also satisfied. Hence, assuming that $\mathcal{B}$ does not output fail and that $\mathcal{A}$ returns a valid forgery for $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, algorithm $\mathcal{B}$ also returns a valid forgery for $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$.

Probability of success. We bound the probability that $\mathcal{B}$ does not output fail and $\mathcal{A}$ succeeds in forging a signature. Let $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and σ be a forgery that $\mathcal{A}$ submits at the end of the experiment. Let us define the following random variables:

- Let X_0 denote the event that the message that $\mathcal{A}$ submits as its signing oracle, is equal to its j_0^{th} query to H_0: $\hat{\mathsf{m}} = \mathsf{m}_{j_0}$.
- Let X_1 denote the event that the set of public keys and messages, which $\mathcal{A}$ submits as part of its forgery, is equal to its j_1^{th} query to H_1: $(\mathsf{PK}, \mathsf{M}) = (\tilde{\mathsf{PK}}, \tilde{\mathsf{M}})$.
- Let X_2 denote the event that $\hat{\mathsf{m}}' \neq \mathsf{m}_{i^*}$.
- Let Y denote the event that the adversary $\mathcal{A}$ outputs a valid forgery $(\mathsf{PK}, \mathsf{M}, \sigma)$ for $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$.
- Let Z be the event that each message m_i for $i \in [N]$ is queried to H_0 and $(\mathsf{PK}, \mathsf{M})$ is queried to H_1 before the end of the experiment.

Then, we must bound the following probability:

$$
\begin{aligned}
\Pr\left[X_0 \wedge X_1 \wedge X_2 \wedge Y\right] &= \Pr\left[X_0 \wedge X_1 \wedge X_2 \wedge Y \wedge Z\right] + \Pr\left[X_0 \wedge X_1 \wedge X_2 \wedge Y \wedge \bar{Z}\right] \\
&= \Pr\left[X_0 \wedge X_1 \wedge X_2 \wedge Y \wedge Z\right] \\
&\geq \Pr\left[X_0 \wedge X_1 \wedge X_2 \mid Y \wedge Z\right] \cdot \Pr\left[Y \wedge Z\right] \\
&= \underbrace{\Pr\left[X_0 \wedge X_1 \wedge X_2 \mid Y \wedge Z\right]}_{q_0} \cdot \big(\Pr\left[Y\right] - \underbrace{\Pr\left[Y \wedge \bar{Z}\right]}_{q_1}\big)
\end{aligned}
$$

We bound each of the probabilities q_0 and q_1 as follows:

- q_0: Algorithm $\mathcal{B}$ samples the indices j_0 and j_1 uniformly at random from $[Q_0]$ and $[Q_1]$ respectively. Furthermore, the message $\hat{\mathsf{m}}'$ that $\mathcal{B}$ commits to in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$ is sampled uniformly at random from $\mathcal{M}$. Since the hash functions H_0 and H_1 are modeled as random oracles, the indices j_0, j_1, and message $\hat{\mathsf{m}}$ that $\mathcal{B}$ samples during the setup phase of the experiment is independent of the views of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that $\mathcal{B}$ provides. Therefore, given the fact that m_i for $i \in [N]$ is queries to H_0 and $(\mathsf{PK}, \mathsf{M})$ is queried to H_1, we have
$$q_0 = \frac{1}{Q_0 \cdot Q_1} \cdot \frac{1}{|\mathcal{M}|}.$$
- q_1: Since H_0 and H_1 are modeled as random oracles it is impossible for $\mathcal{A}$ to generate a valid forgery with probability greater than $1/|\mathbb{Z}_p| = 1/p$. Hence, $q_1 \leq 1/p$.

Combining the probabilities q_0 and q_1, we have

$$\Pr\left[X_0 \wedge X_1 \wedge Y\right] \geq \frac{1}{Q_0 \cdot Q_1} \cdot \left(\Pr\left[Y\right] - \frac{1}{p}\right),$$

and the lemma follows. $\square$

Lemma A.3. *Let $\mathcal{A}$ be an adversary in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that makes at most a single random-oracle query to H_1 and let*
$$\varepsilon_{\mathcal{A}} = \Pr[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1].$$
Then, there exists an algorithm $\mathcal{B}$ with $\mathsf{DLog}_{\mathbb{G}}$ advantage
$$\varepsilon_{\mathcal{A}}^2 - \varepsilon_{\mathcal{A}}/p \leq \mathsf{Adv}_{\mathsf{DLog}}[\lambda, \mathcal{A}].$$

Proof. Let $\mathcal{A}$ be an adversary in the $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that makes at most a single random-oracle query to H_1. To simplify the analysis of the proof, we make the following assumption on the adversary $\mathcal{A}$:

- $\mathcal{A}$ makes the single oracle query to H_1 (as opposed to making no oracle query to H_1).
- Let $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ be $\mathcal{A}$'s single oracle query to H_1. Then, there exists some index i^* such that pk_{i^*} corresponds to the challenge public key that was generated by the challenger.
- $\mathcal{A}$ forges on PK and M.

If $\mathcal{A}$ violates any of these conditions, then it must inevitably have at most $1/p$ probability of forging a signature as it cannot predict the output of the random oracle H_1.

We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve discrete log. Algorithm $\mathcal{B}$ simulates two instances of the experiment $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. After one execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, it rewinds $\mathcal{A}$ to the point it makes the single oracle query to H_1 and executes the experiment once again. From the two forgeries that it receives from $\mathcal{A}$ in the two executions of the experiment, $\mathcal{B}$ solves its discrete log challenge. Formally, algorithm $\mathcal{B}$ simulates the two executions of the experiment $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ as follows:

- At the start of the experiment, algorithm $\mathcal{B}$ receives a discrete log challenge $\hat{h} \in \mathbb{G}$ as well as $\mathcal{A}$'s committed message $\hat{\mathsf{m}} \in \mathcal{M}$. It first samples random elements $\hat{\sigma}, \hat{c} \xleftarrow{\$} \mathbb{Z}_p$. Then, it sets the public key $\mathsf{pk}^* = (h_0^*, h_1^*)$ as follows:
  - $h_0^* \leftarrow \hat{h}$,
  - $h_1^* \leftarrow g^{\hat{\sigma}}/(h_0^*)^{\hat{c}}$,

  and provides pk^* to $\mathcal{A}$. Finally, $\mathcal{B}$ initializes a table $\mathsf{T} : \mathcal{M} \to \mathbb{Z}_p$ that will be used to answer $\mathcal{A}$'s random oracle queries to H_0 consistently.
- For each of $\mathcal{A}$'s queries, algorithm $\mathcal{B}$ responds as follows. By assumption, adversary $\mathcal{A}$ makes a single random-oracle query to H_1.
  - *Queries to H_0*: When $\mathcal{A}$ submits $\hat{\mathsf{m}}$, algorithm $\mathcal{B}$ returns c_0. For all other messages, it responds with a random element in $\mathbb{Z}_p$. Namely, on a query m, algorithm $\mathcal{B}$ checks if $\mathsf{T}[\mathsf{m}] \neq \perp$. If this is the case, then it returns $\mathsf{T}[\mathsf{m}]$. Otherwise, it samples $c \xleftarrow{\$} \mathbb{Z}_p$, sets $\mathsf{T}[\mathsf{m}] \leftarrow c$, and returns c to $\mathcal{A}$.
  - *Query to H_1*: Let $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ be $\mathcal{A}$'s single query to H_1. Let $i^* \in [N]$ be the index for which $\mathsf{pk}_{i^*} = \mathsf{pk}^*$. Algorithm $\mathcal{B}$ samples elements $t_1, \ldots, t_N \xleftarrow{\$} \mathbb{Z}_p$ and $t'_{i^*} \xleftarrow{\$} \mathbb{Z}_p$. On the first execution of the experiment, it provides $(t_1, \ldots, t_{i^*-1}, t_{i^*}, t_{i^*+1}, \ldots, t_N)$ to $\mathcal{A}$. On the second execution of the experiment, it provides $(t_1, \ldots, t_{i^*-1}, t'_{i^*}, t_{i^*+1}, \ldots, t_N)$ to $\mathcal{A}$.
  - *Signing query*: For $\mathcal{A}$'s single signing query $\hat{\mathsf{m}}$, algorithm $\mathcal{B}$ provides $\hat{\sigma}$ for both executions of the experiment.
- At the end of the two executions of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, algorithm $\mathcal{B}$ holds two forgeries that are output by $\mathcal{A}$: $(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0)$, $(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1)$. It checks the following conditions:
  - $\mathsf{PK}_0 = \mathsf{PK}_1$ and $\mathsf{M}_0 = \mathsf{M}_1$,
  - $\mathsf{Verify}(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0) = 1$ and $\mathsf{Verify}(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1) = 1$,
  - $t_{i^*} \neq t'_{i^*}$.

  If any of the conditions above are not satisfied, then $\mathcal{B}$ outputs Fail. Otherwise, it looks up $c_{i^*} \leftarrow \mathsf{T}[\mathsf{m}_{i^*}]$ and returns
  $$\big((\sigma_0 - \sigma_1)(t_{i^*} - t'_{i^*})^{-1} - \hat{\sigma}\big)(c_{i^*} - \hat{c})^{-1} \in \mathbb{Z}_p,$$
  as the solution to the discrete log challenge $\hat{h}$.

To prove the lemma, we first show that as long as $\mathcal{B}$ does not output Fail at the end of the experiment:

1. Algorithm $\mathcal{B}$ correctly simulates an execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ for $\mathcal{A}$.
2. If $\mathcal{A}$ successfully forges signatures in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$, then $\mathcal{B}$ successfully solves the discrete log of $\hat{h}$.

Then, we bound the probability that $\mathcal{B}$ does not output Fail and that $\mathcal{A}$ successfully forges the signatures in the two executions of the experiment to complete the proof.

Correctness of simulation. The correctness of the simulation follows straightforwardly from the specification of algorithm $\mathcal{B}$ and experiment $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. Each of the public key components h_0^*, h_1^* in pk^* are distributed uniformly in $\mathbb{G}$. Furthermore, the signature σ, and the hash value $c_0 \leftarrow H_0(\hat{\mathsf{m}})$ are distributed uniformly in $\mathbb{Z}_p$ under the condition that $g^\sigma = (h_0^*)^{c_0} \cdot h_1^*$ exactly as in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. Therefore, $\mathcal{B}$ simulates $\mathcal{A}$'s queries to H_0 and the signing oracle for the message $\hat{\mathsf{m}}$ correctly. For the rest of $\mathcal{A}$'s queries to H_0 and H_1, the correctness of $\mathcal{B}$'s simulation follows immediately from its specification.

Correctness of forgery. Suppose that $\mathcal{A}$ successfully generates a forgery in the two executions of the experiment,$(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0)$, $(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1)$, and that $\mathcal{B}$ does not return Fail at the end of its simulation. Then, by definition, we have

- $\mathsf{PK}_0 = \mathsf{PK}_1$ and $\mathsf{M}_0 = \mathsf{M}_1$,
- $\mathsf{Verify}(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0) = \mathsf{Verify}(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1) = 1$.

Let $\mathsf{PK}_0 = \mathsf{PK}_1 = (h_{i,0}, h_{i,1})_{i \in [N]}$ and $\mathsf{M}_0 = \mathsf{M}_1 = (\mathsf{m}_i)_{i \in [N]}$. Let $i^* \in [N]$ be the index for which $(h_{i^*,0}, h_{i^*,1}) = (h_0^*, h_1^*)$. Then, by the definition of the verification algorithm Verify, we have

$$g^{\sigma_0} = (h_{i^*,0}^{c_{i^*}} \cdot h_{i^*,1})^{t_{i^*}} \prod_{i \in [N] \setminus \{i^*\}} (h_{i,0}^{c_i} \cdot h_{i,1})^{t_i},$$

$$g^{\sigma_1} = (h_{i^*,0}^{c_{i^*}} \cdot h_{i^*,1})^{t'_{i^*}} \prod_{i \in [N] \setminus \{i^*\}} (h_{i,0}^{c_i} \cdot h_{i,1})^{t_i},$$

where $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$ and the values $(t_1, \ldots, t_{i^*-1}, t_{i^*}, t_{i^*+1}, \ldots, t_N)$ and $(t_1, \ldots, t_{i^*-1}, t'_{i^*}, t_{i^*+1}, \ldots, t_N)$ are the outputs of $H_1(\mathsf{PK}, \mathsf{M})$ in each of the two executions of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$. Combining these two relations, we have

$$g^{\sigma_0 - \sigma_1} = \left((h_0^*)^{c_{i^*}} \cdot h_1^*\right)^{t_{i^*} - t'_{i^*}},$$

and hence,

$$g^{(\sigma_0 - \sigma_1) \cdot (t_{i^*} - t'_{i^*})^{-1}} = (h_0^*)^{c_{i^*}} \cdot h_1^*.$$

Now, using the fact that $g^{\hat{\sigma}} = (h_0^*)^{\hat{c}} \cdot h_1^*$, we have

$$g^{(\sigma_0 - \sigma_1) \cdot (t_{i^*} - t'_{i^*})^{-1} - \hat{\sigma}} = (h_0^*)^{c_{i^*} - \hat{c}}.$$

Therefore, $\left((\sigma_0 - \sigma_1) \cdot (t_{i^*} - t'_{i^*})^{-1} - \hat{\sigma}\right) \cdot (c_{i^*} - \hat{c})^{-1}$ is a valid solution to the discrete log challenge $\hat{h} = h_0^*$.

Probability of success. To bound the abort probability, we use the rewinding lemma (Lemma 2.3). We first define the following random variables that models an execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$:

- Let X be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ up to and including the query to H_1 and its response, but excluding t_{i^*} and t'_{i^*}.
- Let Y and Y' be t_{i^*} and t'_{i^*} respectively.
- Let Z be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ in the first execution of the experiment following $\mathcal{A}$'s query to H_1.
- Let Z' be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ in the second execution o the experiment following $\mathcal{A}$'s query to H_1.

The random variables (X, Y, Z) corresponds to the view that $\mathcal{B}$ provides to $\mathcal{A}$ in the first execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$, and (X, Y', Z') corresponds to $\mathcal{A}$'s view in the second execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$. To bound $\mathcal{B}$'s abort probability at the end of its simulation, let $f(X, Y, Z)$ be the function that outputs 1 if and only if a forgery $(\mathsf{PK}, \mathsf{M}, \sigma)$ that $\mathcal{A}$ outputs during an execution of $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$ is valid. Then, letting $\varepsilon = \Pr[f(X, Y, Z) = 1]$, the probability that $\mathcal{B}$ aborts the experiment is bounded by

$$\Pr[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'] \geq \varepsilon^2 - \varepsilon/p,$$

by Lemma 2.3. By definition, we have

$$\Pr[f(X, Y, Z) = 1] = \Pr[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}] = 1],$$

and the lemma now follows. □

B Proofs for Lyubashevsky-Micciancio Signatures

B.1 Proof of Theorem 4.2

Fix a security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and any set of message $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$. Let $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, and $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{m}_i)$ for $i \in [N]$. By definition, each signing key consists of two vectors of ring elements $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}) \in (\mathcal{B}^\ell_{\beta_\mathbf{s}})^2$, and each signature has the form $\sigma_i = \mathbf{s}_{i,0} \cdot c_i + \mathbf{s}_{i,1}$ for $i \in [N]$ and $c_i \leftarrow H_0(\mathsf{m}_i)$ in $\mathcal{B}_{\beta_0}$. The signature aggregation algorithm computes $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$ in $\mathcal{B}^N_{\beta_1}$ and sets $\sigma_\mathsf{ag} = \sum_{i\in[N]} \sigma_i \cdot t_i$. Therefore, we can bound the norm of σ_ag using the triangle inequality:

$$\begin{aligned}
\|\sigma_\mathsf{ag}\| &= \left\| \sum_{i\in[N]} \sigma_i \cdot t_i \right\| \\
&\le N \cdot n \|t_i\| \|\sigma_i\| \\
&\le Nn\beta_1 \|\mathbf{s}_{i,0} \cdot c_i + \mathbf{s}_{i,1}\| \\
&\le N\beta_1 \big(n \|c_i\| \|\mathbf{s}_{i,0}\| + \|\mathbf{s}_{i,1}\| \big) \\
&\le Nn\beta_1(n\beta_0\beta_\mathbf{s} + \beta_\mathbf{s}) \\
&= Nn(n\beta_0 + 1)\beta_1\beta_\mathbf{s}.
\end{aligned}$$

Therefore, the bit size of σ_ag is $\mathsf{poly}(\lambda, \log N)$.

B.2 Proof of Theorem 4.3

Fix a security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and any set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$. Let $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, and $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mathsf{m}_i)$ for $i \in [N]$. By definition, each signing key consists of two vectors of ring elements $(\mathbf{s}_{i,0}, \mathbf{s}_{i,1}) \in (\mathcal{B}^m_{\beta_\mathbf{s}})^2$, each public key consists of two ring elements $(v_{i,0}, v_{i,1}) \in \mathcal{R}^2$, and each signature has the form $\sigma_i = \mathbf{s}_{i,0} \cdot c_i + \mathbf{s}_{i,1}$ for $i \in [N]$ and $c_i \leftarrow H_0(\mathsf{m}_i)$ in $\mathcal{B}_{\beta_0}$. The signature aggregation algorithm algorithm computes $(t_1, \ldots, t_N) \leftarrow H_1(\mathsf{PK}, \mathsf{M})$ in $\mathcal{B}^N_{\beta_1}$ and sets $\sigma_\mathsf{ag} = \sum_{i\in[N]} \sigma_i \cdot t_i$. Therefore, by linearity, the aggregated signature satisfies the relation

$$\mathbf{a}^\intercal \sigma_\mathsf{ag} = \mathbf{a}^\intercal \left(\sum_{i\in[N]} \sigma_i \cdot t_i \right) = \mathbf{a}^\intercal \left(\sum_{i\in[N]} (\mathbf{s}_{i,0} \cdot c_i + \mathbf{s}_{i,1}) \cdot t_i \right) = \sum_{i\in[N]} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i.$$

Therefore, it is sufficient to show that $\|\sigma_\mathsf{ag}\| \le Nn(n\beta_0 + 1)\beta_1\beta_\mathbf{s}$. This follows by the same triangle inequality used in the proof of Theorem 4.2 above. The correctness now follows.

B.3 Proof of Theorem 4.4

As in the proof of Theorem 3.9, we use the selective variant of the one-time unforgeability experiment (Definition 3.5) where the adversary is forced to commit to its signing query at the start of the experiment. We can show that an adversary that wins in the original experiment of Definition 3.5 can be converted into an adversary that wins in the selective experiment with only a small loss in its winning advantage. The proof of the following lemma is identical to that of Lemma A.2.

Lemma B.1. *Let $\mathcal{A}$ be an adversary in $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ (Definition 3.5) that makes at most Q_0 random-oracle queries to H_0 and Q_1 random-oracle queries to H_1. Then, there exists an adversary $\mathcal{B}$ in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}]$ (Definition A.1) that makes at most a single random-oracle query to H_1 such that*

$$\Pr\left[\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1\right] \le Q_0 \cdot Q_1 \cdot |\mathcal{M}| \cdot \Pr\left[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{B}] = 1\right] + \frac{1}{|\mathcal{B}_{\beta_1}|}.$$

Proof. Identical to the proof of Lemma A.2. □

As in the proof of Theorem 3.9, we show that an adversary that wins in the selective experiment can be converted into an algorithm for the SIS problem (Definition 2.2). To do this, we crucially rely on the fact that when we use Construction 4.1 to instantiate Definition 3.5, there exists at least two signing keys that can potentially *explain* the view of the adversary. Before presenting our main security lemma, we state the properties that we need in the following lemmas.

Integer Lattices. We first state the properties that we need when we instantiate the ring $\mathcal{R}$ to be the ring of integer matrices. The first lemma states that when we use Construction 4.1 to instantiate the one-time unforgeability experiment, there exist at least two signing keys that can explain an adversary's view.

Lemma B.2 ([LM08, Lemma 4.4]). *Let n, ℓ, q and $\beta_{\mathbf{s}}$ be positive integers, $\mathcal{R} = \mathbb{Z}_q^{n\times n}$ be the ring of integer matrices, and suppose that $2\beta_{\mathbf{s}} + 1 \geq q^{1/\ell} \cdot 2^{\lambda/n\ell}$. Then, for any $\mathbf{a} \in \mathcal{R}_q^\ell$ and $c \in \mathcal{B}_{\beta_0}$, we have*

$$\Pr_{\mathbf{s}_0,\mathbf{s}_1 \leftarrow_r \mathcal{B}_{\beta_{\mathbf{s}}}} \left[\ \exists\ \mathbf{s}_0', \mathbf{s}_1' \in \mathcal{B}_{\beta_{\mathbf{s}}}^\ell\ :\ \mathbf{a}^\intercal \mathbf{s}_0 = \mathbf{a}^\intercal \mathbf{s}_0' \ \wedge\ \mathbf{a}^\intercal \mathbf{s}_1 = \mathbf{a}^\intercal \mathbf{s}_1' \ \wedge\ \mathbf{s}_0 \cdot c + \mathbf{s}_1 = \cdot \mathbf{s}_0' \cdot c + \mathbf{s}_1'\ \right] = 1 - 2^{-\lambda}.$$

The next lemma states that once an adversary comes up with a forgery, there exists only a unique signing key that can explain both the adversary's view in the unforgeability experiment and its forgery.

Lemma B.3. *Let $n, \ell, q, \beta_{\mathbf{s}}, \beta_0$, and β_1 be positive integers such that $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$ and $\mathcal{R} = \mathbb{Z}^{n\times n}$. Let $c, \hat{c} \in \mathcal{B}_{\beta_0}$, $\sigma, \hat{\sigma} \in \mathcal{R}$, $t \in \mathcal{B}_{\beta_1}$ be any set of ring elements such that $\det(c - \hat{c})$ and $\det(t) \neq 0$. Then, there is at most a single set of vectors $\mathbf{s}_0, \mathbf{s}_1 \in \mathcal{B}_{\beta_{\mathbf{s}}}^\ell$ such that*

- $\mathbf{s}_0 \cdot c + \mathbf{s}_1 = \sigma$,
- $(\mathbf{s}_0 \cdot \hat{c} + \mathbf{s}_1) \cdot t = \hat{\sigma}$.

Proof. Let $(\mathbf{s}_0, \mathbf{s}_1)$, and $(\mathbf{s}_0', \mathbf{s}_1')$ be any two pairs of vectors in $\mathcal{B}_{\beta_{\mathbf{s}}}^\ell$ such that

- $\mathbf{s}_0 \cdot c + \mathbf{s}_1 = \mathbf{s}_0' \cdot c + \mathbf{s}_1' = \sigma$,
- $(\mathbf{s}_0 \cdot \hat{c} + \mathbf{s}_1) \cdot t = (\mathbf{s}_0' \cdot \hat{c} + \mathbf{s}_1') \cdot t = \hat{\sigma}$.

These relations can be re-written as

- $(\mathbf{s}_0 - \mathbf{s}_0') \cdot c + (\mathbf{s}_1 - \mathbf{s}_1') = 0$,
- $(\mathbf{s}_0 - \mathbf{s}_0') \cdot \hat{c} \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t = 0$.

We note that since $t \in \mathcal{B}_{\beta_1}$, $c \in \mathcal{B}_{\beta_0}$, and $\mathbf{s}_0, \mathbf{s}_0', \mathbf{s}_1, \mathbf{s}_1' \in \mathcal{B}_{\beta_1}$, we have $\|(\mathbf{s}_0 - \mathbf{s}_0') \cdot c + (\mathbf{s}_1 - \mathbf{s}_1')\| \leq 2\beta_{\mathbf{s}}(n\beta_0 + 1)$ and $\|(\mathbf{s}_0 - \mathbf{s}_0') \cdot c \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t\| \leq 2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1)$. Now, we can rewrite first relation

$$(\mathbf{s}_0 - \mathbf{s}_0') + (\mathbf{s}_1 - \mathbf{s}_1') \cdot c = 0 \quad \Longrightarrow \quad (\mathbf{s}_0 - \mathbf{s}_0') \cdot c \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t = 0.$$

The two relations then imply that

$$(\mathbf{s}_0 - \mathbf{s}_0') \cdot (c - \hat{c}) \cdot t = 0.$$

Since $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$ and since $\det(c - \hat{c}) \neq 0$ and $\det(t) \neq 0$, we have $\mathbf{s}_0 = \mathbf{s}_0'$. This also implies that $\mathbf{s}_1 = \mathbf{s}_1'$ by the first relation and the lemma follows. □

Ideal Lattices. We state the analogous properties as in Lemma B.2 and Lemma B.3 in the following two lemmas.

Lemma B.4 ([LM08, Lemma 4.9]). *Let n, ℓ, q and $\beta_{\mathbf{s}}$ be positive integers, $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ polynomial ring, and suppose that ${\beta_{\mathbf{s}}}^{\ell n} \geq 2^\lambda {\beta_0}^n q^n$. Then, for any $\mathbf{a} \in \mathcal{R}_q^\ell$ and $c \in \mathcal{B}_{\beta_0}$, we have*

$$\Pr_{\mathbf{s}_0,\mathbf{s}_1 \leftarrow_r \mathcal{B}_{\beta_{\mathbf{s}}}} \left[\ \exists\ \mathbf{s}_0', \mathbf{s}_1' \in \mathcal{B}_{\beta_{\mathbf{s}}}^\ell\ :\ \mathbf{a}^\intercal \mathbf{s}_0 = \mathbf{a}^\intercal \mathbf{s}_0' \ \wedge\ \mathbf{a}^\intercal \mathbf{s}_1 = \mathbf{a}^\intercal \mathbf{s}_1' \ \wedge\ \mathbf{s}_0 \cdot c + \mathbf{s}_1 = \mathbf{s}_0' \cdot c + \mathbf{s}_1'\ \right] = 1 - 2^{-\lambda}.$$

Lemma B.5. *Let $\ell, q, \beta_{\mathbf{s}}, \beta_0$, and β_1 be positive integers, n a power-of-two positive integer, $\mathcal{R} = \mathbb{Z}[X]/(X^n+1)$ polynomial ring, and suppose that q is a prime such that $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$. Let $c, \hat{c} \in \mathcal{B}_{\beta_0}$, $\sigma, \hat{\sigma} \in \mathcal{R}$, $t \in \mathcal{B}_{\beta_1}$ be any set of ring elements such that $c \neq \hat{c}$ and $t \neq 0$. Then, there is at most a single set of vectors $\mathbf{s}_0, \mathbf{s}_1 \in \mathcal{B}_{\beta_{\mathbf{s}}}$ such that*

- $c \cdot \mathbf{s}_0 + \mathbf{s}_1 = \sigma$,
- $t \cdot (\hat{c} \cdot \mathbf{s}_0 + \mathbf{s}_1) = \hat{\sigma}$.

Proof. The proof is identical to Lemma B.3. Let $(\mathbf{s}_0, \mathbf{s}_1)$, and $(\mathbf{s}_0', \mathbf{s}_1')$ be any two pairs of vectors in $\mathcal{B}_{\beta_{\mathbf{s}}}^{\ell}$ such that

- $\mathbf{s}_0 \cdot c + \mathbf{s}_1 = \mathbf{s}_0' \cdot c + \mathbf{s}_1' = \sigma$,
- $(\mathbf{s}_0 \cdot \hat{c} + \mathbf{s}_1) \cdot t = (\mathbf{s}_0' \cdot \hat{c} + \mathbf{s}_1') \cdot t = \hat{\sigma}$.

These relations can be re-written as

- $(\mathbf{s}_0 - \mathbf{s}_0') \cdot c + (\mathbf{s}_1 - \mathbf{s}_1') = 0$,
- $(\mathbf{s}_0 - \mathbf{s}_0') \cdot \hat{c} \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t = 0$.

We note that since $t \in \mathcal{B}_{\beta_1}$, $c \in \mathcal{B}_{\beta_0}$, and $\mathbf{s}_0, \mathbf{s}_0', \mathbf{s}_1, \mathbf{s}_1' \in \mathcal{B}_{\beta_1}$, we have $\|(\mathbf{s}_0 - \mathbf{s}_0') \cdot c + (\mathbf{s}_1 - \mathbf{s}_1')\| \leq 2\beta_{\mathbf{s}}(n\beta_0 + 1)$ and $\|(\mathbf{s}_0 - \mathbf{s}_0') \cdot c \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t\| \leq 2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1)$. Now, we can rewrite first relation

$$(\mathbf{s}_0 - \mathbf{s}_0') + (\mathbf{s}_1 - \mathbf{s}_1') \cdot c = 0 \quad \Longrightarrow \quad (\mathbf{s}_0 - \mathbf{s}_0') \cdot c \cdot t + (\mathbf{s}_1 - \mathbf{s}_1') \cdot t = 0.$$

The two relations then imply that

$$(\mathbf{s}_0 - \mathbf{s}_0') \cdot (c - \hat{c}) \cdot t = 0.$$

Since $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$ and since $\mathcal{R}$ is an integral domain, we have $\mathbf{s}_0 = \mathbf{s}_0'$. This also implies that $\mathbf{s}_1 = \mathbf{s}_1'$ by the first relation. The lemma follows. □

Main lemma. We now prove the main technical lemma, which states that an adversary that wins in the selective one-time unforgeability experiment can be converted into an algorithm for the SIS problem.

Lemma B.6. *Suppose that the parameters of Construction 4.1 is set such that the ring $\mathcal{R}$ is instantiated as either $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and that $2\beta_{\mathbf{s}}\beta_1(n\beta_0 + 1) < q$. Additionally, suppose that*

- *when $\mathcal{R} = \mathbb{Z}^{n \times n}$, the inequality $2\beta_{\mathbf{s}} + 1 \geq q^{1/\ell} \cdot 2^{\lambda/n\ell}$ holds.*
- *when $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$, the inequality $2\beta_{\mathbf{s}} + 1 \geq q^{1/\ell} \cdot 2^{\lambda/n\ell}$ holds.*

With these parameter settings, let $\mathcal{A}$ be an adversary in $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that makes at most a single random-oracle query to H_1 with advantage

$$\varepsilon_{\mathcal{A}} = \Pr\left[\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}] = 1\right].$$

Then, there exists an algorithm $\mathcal{B}$ with $\mathsf{SIS}_{n,m,q,\beta^\star}$ advantage

$$\varepsilon_{\mathcal{A}}^2 - \varepsilon_{\mathcal{A}}/|\mathcal{B}_{\beta_1}| \leq 2 \cdot \mathsf{SIS}_{n,m,q,\beta^\star}[\lambda, \mathcal{A}],$$

for $\beta^\star = 3 \cdot \beta_{\mathsf{ver}} + (n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}}) \cdot 2n\beta_1 + (n\beta_0 + 1)\beta_{\mathbf{s}}$.

Proof. Let $\mathcal{A}$ be an adversary in the $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ that makes at most a single random-oracle query to H_1. As in the proof of Lemma A.3, we make the following assumption on the adversary $\mathcal{A}$:

- $\mathcal{A}$ makes the single oracle query to H_1.
- Let $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$ be $\mathcal{A}$'s single oracle query to H_1. Then, there exists some index i^* such that pk_{i^*} corresponds to the challenge public key that was generated by the challenger.

- $\mathcal{A}$ forges on PK and M.

If $\mathcal{A}$ violates any of these conditions, then it must inevitably have at most $1/|\mathcal{B}_{\beta_1}|$ probability of forging a signature as it cannot predict the output of the random oracle H_1.

We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the $\mathsf{SIS}_{n,m,q,\beta^\star}$ problem. Algorithm $\mathcal{B}$ simulates two instances of the experiment $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$. After one execution of $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$, it rewinds $\mathcal{A}$ to the point it makes the single oracle query to H_1 and executes the experiment once again. From the two forgeries that it receives from $\mathcal{A}$ in the two executions of the experiment, $\mathcal{B}$ solves its $\mathsf{SIS}_{n,m,q,\beta^\star}$ challenge. Formally, algorithm $\mathcal{B}$ simulates the two executions of the experiment $\mathsf{EXP}^{\mathsf{sel}}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$ as follows:

- At the start of the experiment, algorithm $\mathcal{B}$ receives an $\mathsf{SIS}_{n,m,q,\beta^\star}$ challenge $\mathbf{a} \in \mathcal{R}_q^\ell$. It samples two vectors $\mathbf{s}_0^*, \mathbf{s}_1^* \xleftarrow{\$} \mathcal{B}_{\beta_\mathbf{s}}$ and sets the public key $\mathsf{pk}^* = (v_0^*, v_1^*)$ as follows:
  - $v_0^* \leftarrow \mathbf{a}^\mathsf{T}\mathbf{s}_0^*$,
  - $v_1^* \leftarrow \mathbf{a}^\mathsf{T}\mathbf{s}_1^*$.

  It provides pk^* to $\mathcal{A}$. Algorithm $\mathcal{B}$ then initializes a table $\mathsf{T} : \mathcal{M} \to \mathcal{B}_{\beta_0}$ that will be used to answer $\mathcal{A}$'s random oracle queries to H_0 consistently.

- For each of $\mathcal{A}$'s queries, algorithm $\mathcal{B}$ responds as follows. By assumption, adversary $\mathcal{A}$ makes a single random-oracle query to H_1.
  - *Queries to* H_0: For each of $\mathcal{A}$'s queries m, algorithm $\mathcal{B}$ checks if $\mathsf{T}[\mathsf{m}] = \perp$. If this is the case, then it samples $c \xleftarrow{\$} \mathcal{B}_{\beta_0}$, sets $\mathsf{T}[\mathsf{m}] = c$, and returns c to $\mathcal{A}$. If $\mathsf{T}[\mathsf{m}] \neq \perp$, then it returns $\mathsf{T}[\mathsf{m}]$ to $\mathcal{A}$.
  - *Query to* H_1: Let $\mathsf{PK} = (\mathsf{pk}_i)_{i\in[N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$ be $\mathcal{A}$'s single query to H_1. Let $i^* \in [N]$ be the index for which $\mathsf{pk}_{i^*} = \mathsf{pk}^*$. Algorithm $\mathcal{B}$ samples elements $t_1, \ldots, t_N \leftarrow \mathcal{B}_{\beta_1}$ and $t'_{i^*} \leftarrow \mathcal{B}_{\beta_1}$. On the first execution of the experiment, it provides $(t_1, \ldots, t_{i^*-1}, t_{i^*}, t_{i^*+1}, \ldots, t_N)$ to $\mathcal{A}$. On the second execution of the experiment, it provides $(t_1, \ldots, t_{i^*-1}, t'_{i^*}, t_{i^*+1}, \ldots, t_N)$ to $\mathcal{A}$.
  - *Signing query*: For $\mathcal{A}$'s single signing query $\hat{\mathsf{m}}$, algorithm $\mathcal{B}$ queries $\hat{c} \leftarrow H_0(\hat{\mathsf{m}})$ and returns $\mathbf{s}_0^* \cdot \hat{c} + \mathbf{s}_1^*$ to $\mathcal{A}$.

- At the end of the two executions of $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$, algorithm $\mathcal{B}$ holds two forgeries that are output by $\mathcal{A}$: $(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0)$, $(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1)$. It checks the following conditions:
  - $\mathsf{PK}_0 = \mathsf{PK}_1$ and $\mathsf{M}_0 = \mathsf{M}_1$,
  - $\mathsf{Verify}(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0) = 1$ and $\mathsf{Verify}(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1) = 1$,
  - $t_{i^*} \neq t'_{i^*}$.
  - $\sigma^\star = (\sigma_0 - \sigma_1) - (t_{i^*} - t'_{i^*})(c_{i^*}\mathbf{s}_0^* + \mathbf{s}_1^*) + (\hat{\sigma} - \hat{c} \cdot \mathbf{s}_0^* - \mathbf{s}_1^*) \neq \mathbf{0}$,

  If any of the conditions above are not satisfied, then $\mathcal{B}$ outputs Fail. Otherwise, it returns $\sigma^\star$ as its solution to the $\mathsf{SIS}_{n,m,q,\beta^\star}$ challenge $\mathbf{a}$.

To prove the lemma, we first show that as long as $\mathcal{B}$ does not output Fail at the end of the experiment:

1. Algorithm $\mathcal{B}$ correctly simulates an execution of $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$ for $\mathcal{A}$.
2. If $\mathcal{A}$ successfully forges signatures in $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$, then $\mathcal{B}$ successfully solves the SIS challenge $\mathcal{A}$.

Then, we bound the probability that $\mathcal{B}$ does not output Fail and that $\mathcal{A}$ successfully forges the signatures in the two executions of the experiment to complete the proof.

Correctness of simulation. The correctness of simulation follows immediately from the specification of $\mathcal{B}$. Algorithm $\mathcal{B}$ generates the signing key $\mathsf{sk} = (\mathbf{s}_0^*, \mathbf{s}_1^*)$ and public key $\mathsf{pk} = (v_0^*, v_1^*)$ exactly as in the specification of KeyGen, and it simulates the single signing query according to Sign in Construction 4.1. For each of $\mathcal{A}$'s random-oracle queries, $\mathcal{B}$ replies with uniformly random samples from the range of H_0 and H_1.

Correctness of forgery. Suppose that $\mathcal{A}$ successfully generates a forgery in the two executions of the experiment, $(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0)$, $(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1)$, and that $\mathcal{B}$ does not return Fail at the end of its simulation. Then, we have

- $\mathsf{PK}_0 = \mathsf{PK}_1$ and $\mathsf{M}_0 = \mathsf{M}_1$,
- $\mathsf{Verify}(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0) = \mathsf{Verify}(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1) = 1$,
- $\sigma^\star = (\sigma_0 - \sigma_1) - (\mathbf{s}_0^* \cdot c_{i^*} + \mathbf{s}_1^*)(t_{i^*} - t'_{i^*}) + (\hat{\sigma} - \mathbf{s}_0^* \cdot \hat{c} - \mathbf{s}_1^*) \neq \mathbf{0}$.

Let $\mathsf{PK}_0 = \mathsf{PK}_1 = (v_{i,0}, v_{i,1})_{i \in [N]}$ and $\mathsf{M}_0 = \mathsf{M}_1 = (\mathsf{m}_i)_{i \in [N]}$. Let $i^* \in [N]$ be the index for which $(v_{i^*,0}, v_{i^*,1}) = (v_0^*, v_1^*)$. Then, by the definition of the verification algorithm Verify, we have

$$\mathbf{a}^\mathsf{T}\sigma_0 = (v_0^* \cdot c_{i^*} + v_1^*) \cdot t_{i^*} + \sum_{i \in [N] \setminus \{i^*\}} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i,$$

$$\mathbf{a}^\mathsf{T}\sigma_1 = (v_0^* \cdot c_{i^*} + v_1^*) \cdot t'_{i^*} + \sum_{i \in [N] \setminus \{i^*\}} (v_{i,0} \cdot c_i + v_{i,1}) \cdot t_i,$$

where $c_i \leftarrow H_0(\mathsf{m}_i)$ for $i \in [N]$ and the values $(t_1, \ldots, t_{i^*-1}, t_{i^*}, t_{i^*+1}, \ldots, t_N)$ and $(t_1, \ldots, t_{i^*-1}, t'_{i^*}, t_{i^*+1}, \ldots, t_N)$ are the outputs of $H_1(\mathsf{PK}, \mathsf{M})$ in each of the two executions of $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$. Combining these relations, we have

$$\mathbf{a}^\mathsf{T}(\sigma_0 - \sigma_1) = (v_0^* \cdot c_{i^*} + v_1^*)(t_{i^*} - t'_{i^*}),$$

and hence, using the fact that $v_0^* = \mathbf{a}^\mathsf{T}\mathbf{s}_0^*$ and $v_1^* = \mathbf{a}^\mathsf{T}\mathbf{s}_1^*$, we have

$$\mathbf{a}^\mathsf{T}\big((\sigma_0 - \sigma_1) - (\mathbf{s}_0^* \cdot c_{i^*} + \mathbf{s}_1^*)(t_{i^*} - t'_{i^*})\big) = \mathbf{0}.$$

Now, given that $\mathbf{a}^\mathsf{T}\hat{\sigma} = v_0^* \cdot \hat{c} + v_1^* = \mathbf{a}^\mathsf{T}(\mathbf{s}_0^* \cdot \hat{c} + \mathbf{s}_1^*)$, we have

$$\mathbf{a}^\mathsf{T}\big((\sigma_0 - \sigma_1) - (\mathbf{s}_0^* \cdot c_{i^*} + \mathbf{s}_1^*)(t_{i^*} - t'_{i^*}) + (\hat{\sigma} - \mathbf{s}_0^* \cdot \hat{c} - \mathbf{s}_1^*)\big) = \mathbf{0}.$$

Therefore, as long as $\|\sigma^\star\| = \|(\sigma_0 - \sigma_1) - (\mathbf{s}_0^* \cdot c_{i^*} + \mathbf{s}_1^*)(t_{i^*} - t'_{i^*}) + (\hat{\sigma} - \mathbf{s}_0^* \cdot \hat{c} - \mathbf{s}_1^*)\| \leq \beta^\star$, the vector $\sigma^\star$ is a valid solution to the $\mathsf{SIS}_{n,m,q,\beta^\star}$ challenge. Since $\mathsf{Verify}(\mathsf{PK}_0, \mathsf{M}_0, \sigma_0) = \mathsf{Verify}(\mathsf{PK}_1, \mathsf{M}_1, \sigma_1) = 1$, we have $\|\sigma_0\|, \|\sigma_1\| \leq Nn(n\beta_0 + 1)\beta_1\beta_{\mathbf{s}} = \beta_{\mathsf{ver}}$. Therefore, we can bound the norm of $\sigma^\star$ as follows:

$$\begin{aligned}\|\sigma^\star\| &\leq \|\sigma_0 - \sigma_1\| + \|(\mathbf{s}_0^* \cdot c_{i^*} + \mathbf{s}_1^*)(t_{i^*} - t'_{i^*})\| + \|\hat{\sigma} - \mathbf{s}_0^* \cdot \hat{c} - \mathbf{s}_1^*\| \\ &\leq 2 \cdot \beta_{\mathsf{ver}} + (n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}}) \cdot 2n\beta_1 + \beta_{\mathsf{ver}} - n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}} \\ &= 3 \cdot \beta_{\mathsf{ver}} + (n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}}) \cdot 2n\beta_1 + (n\beta_0 + 1)\beta_{\mathbf{s}}.\end{aligned}$$

Hence, $\sigma^\star$ is a valid solution to $\mathsf{SIS}_{n,m,q,\beta^\star}$ for $\beta^\star = 3 \cdot \beta_{\mathsf{ver}} + (n\beta_{\mathbf{s}}\beta_0 + \beta_{\mathbf{s}}) \cdot 2n\beta_1 + (n\beta_0 + 1)\beta_{\mathbf{s}}$.

Probability of success. To bound the abort probability, we use the rewinding lemma (Lemma 2.3). We first bound the probability that $t_{i^*} \neq t'_{i^*}$ with respect to the probability that $\mathcal{A}$ successfully generates a forgery in $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$. For this, we define the following random variables that models an execution of $\mathsf{EXP}_{\mathsf{AS},\mathsf{OT}}[\lambda, \mathcal{A}]$:

- Let X be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ up to and including the query to h_1 and its response but excluding t_{i^*} and t'_{i^*}.
- Let Y and Y' be t_{i^*} and t'_{i^*} respectively.
- Let Z be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ in the first execution of the experiment following $\mathcal{A}$'s query to H_1.
- Let Z' be all the quantities that $\mathcal{B}$ provides to $\mathcal{A}$ in the second execution of the experiment following $\mathcal{A}$'s query to H_1.

The random variables (X, Y, Z) corresponds to the view that $\mathcal{B}$ provides to $\mathcal{A}$ in the first execution of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ and (X, Y', Z') corresponds to $\mathcal{A}$'s view in the second execution of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. Let $f(X, Y, Z)$ be the function that outputs 1 if and only if the forgery $(\mathsf{PK}, \mathsf{M}, \sigma)$ that $\mathcal{A}$ outputs during an execution of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ is valid. Then, letting $\varepsilon = \Pr[f(X, Y, Z) = 1]$, the probability that $t_{i^*} \neq t'_{i^*}$ is bounded by

$$\Pr[f(X, Y, Z) = 1 \wedge f(X, Y', Z') = 1 \wedge Y \neq Y'] \geq \varepsilon^2 - \varepsilon/|\mathcal{B}_{\beta_1}|,$$

by Lemma 2.3. Now, to bound the final abort probability of $\mathcal{B}$, we define the following random variables:

- Let S represent the event that $\sigma^\star = (\sigma_0 - \sigma_1) - (t_{i^*} - t'_{i^*})(c_{i^*}\mathbf{s}_0^* + \mathbf{s}_1^*) + (\hat{\sigma} - \hat{c} \cdot \mathbf{s}_0^* - \mathbf{s}_1^*) \neq \mathbf{0}$ after two executions of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$.
- Let T represent the event that $\mathcal{A}$ generates valid forgeries in the two executions of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ and that $t_{i^*} \neq t'_{i^*}$. This corresponds to the event that $f(X, Y, Z) = 1$, $f(X, Y', Z') = 1$ and $Y \neq Y'$ as formulated above.

The probability that $\mathcal{B}$'s does not abort at the end of its simulation can then be bounded by the probability

$$\Pr[\ S \wedge T\ ] = \Pr[\ S \mid T\ ] \cdot \Pr[\ T\ ] \geq \Pr[\ S \mid T\ ] \cdot \left(\varepsilon^2 - \varepsilon/|\mathcal{B}_{\beta_1}|\right).$$

To bound the probability $\Pr[\ S \mid T\ ]$, suppose that $\mathcal{A}$ successfully produces a forgery σ_0, σ_1 in the two executions of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$ and that $t_{i^*} \neq t'_{i^*}$. Then, by Lemma B.2 and B.4, there exists at least two possible keys $(\mathbf{s}_0, \mathbf{s}_1)$, $(\mathbf{s}'_0, \mathbf{s}'_1)$ that can explain $\mathcal{A}$'s view of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. By the specification, algorithm $\mathcal{B}$ generates the signing key $(\mathbf{s}_0^*, \mathbf{s}_1^*)$ uniformly at random from $\mathcal{B}_{\beta_\mathbf{s}}$. Therefore, the signing key $(\mathbf{s}_0^*, \mathbf{s}_1^*)$ is information-theoretically hidden from $\mathcal{A}$ in the two executions of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. By Lemma B.3 and B.5, there exists a unique key $(\tilde{\mathbf{s}}_0, \tilde{\mathbf{s}}_1)$ that is determined by the signature $\sigma_0 - \sigma_1$ and the signature $\hat{\sigma}$ that $\mathcal{B}$ provides to $\mathcal{A}$ in response to the single signing query in the two executions of $\mathsf{EXP}_{\mathsf{AS,OT}}[\lambda, \mathcal{A}]$. Hence, $\mathcal{A}$ has at most 1/2 probability of producing two forgeries σ_0, σ_1 such that $\sigma^\star = (\sigma_0 - \sigma_1) - (t_{i^*} - t'_{i^*})(c_{i^*}\mathbf{s}_0^* + \mathbf{s}_1^*) + (\hat{\sigma} - \hat{c} \cdot \mathbf{s}_0^* - \mathbf{s}_1^*) \neq \mathbf{0}$. Therefore, we have

$$\Pr[\ S \mid T\ ] \geq 1/2,$$

and the theorem statement now follows.

□

Combining Lemmas B.1 and B.6, Theorem 4.4 now follows.

C Proofs for Construction 6.7

C.1 Proof of Theorem 6.8

Fix a security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and any set of messages $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$. Let $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, and let $\sigma_{\mathsf{ag}} = (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$ be a signature that is output by the signing protocol on these messages and secret-public keys. By the specification of the protocol, the component w_{ag} is a single ring element in $\mathcal{R}_q$. Therefore, its size is dependent only in the security parameter and independent of N.

Consider the component of the signature $\mathbf{z}_{\mathsf{ag}} \in \mathcal{R}^\ell$. By the specification of the protocol, we have $\mathbf{z}_{\mathsf{ag}} = \sum_{i \in [N]} \mathbf{z}_i$ for $\mathbf{z}_i \leftarrow \mathbf{s}_i \cdot c_i + \mathbf{y}_i$ where

- The components $\mathbf{s}_i$ for $i \in [N]$ are the signing keys of each user. By definition of the key generation algorithm, we have $\|\mathbf{s}_i\| \leq \beta_\mathbf{s}$.
- The components c_i for $i \in [N]$ are the output of the hash function H_1. By definition of H_1, we have $\|c_i\| \leq \beta_1$.

- The components $\mathbf{y}_i$ for $i \in [N]$ are vectors that each signer generates at round 1 of the protocol from $\mathcal{D}^{\ell}_{\mathcal{R},s}$. By the tail bound on the discrete Gaussian distribution (Lemma 6.1), we have $\|\mathbf{y}\| \le \lambda \cdot s$ except with probability $2e^{-\lambda^2/2}$, which is negligible.

Hence, the norm of each vectors $\|\mathbf{z}_i\|$ for $i \in [N]$ is independent of N. Since $\|\mathbf{z}_{\mathsf{ag}}\| \le N \max_{i\in[N]} \|\mathbf{z}_i\|$, the bit-size of $\mathbf{z}_{\mathsf{ag}}$ is logarithmic in N. The compactness now follows.

C.2 Proof of Theorem 6.9

To prove correctness, we must show the following two properties:

- If an instance of a signing protocol successfully produces a signature, then the signature verifies under the corresponding public keys and messages.
- An instance of a signing protocol successfully produces a signature with at least a constant probability.

We prove each of these properties below.

Signature verification. Fix a security parameter $\lambda \in \mathbb{N}$, number of signers $N \in \mathbb{N}$, and any set of messages $\mathsf{M} = (\mathsf{m}_i)_{i\in[N]}$. Let $\mathsf{pp} \leftarrow \mathsf{PrmsGen}(1^\lambda)$, $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [N]$, and let $\sigma_{\mathsf{ag}} = (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$ be a signature that is output by the signing protocol on these messages and secret-public keys. We must show that $\mathbf{a}^\intercal \mathbf{z}_{\mathsf{ag}} = v_{\mathsf{ag}} + w_{\mathsf{ag}}$ where $v_{\mathsf{ag}} \leftarrow \sum_{i\in[N]} v_i \cdot c_i$ for $\mathsf{pk}_i = v_i$, $c_i \leftarrow H_1(\mathsf{pk}_i, w_{\mathsf{ag}}, (\mathsf{pk}_i)_{i\in[N]}, \mathsf{m}_i)$ for $i \in [N]$, and that $\|\mathbf{z}_{\mathsf{ag}}\| \le \beta_{\mathsf{ver}}$.

By the specification of the protocol, we have $w_{\mathsf{ag}} = \sum_{i\in[N]} w_i$ for $w_i \leftarrow \mathbf{a}^\intercal \mathbf{y}_i$, and $\mathbf{z}_{\mathsf{ag}} = \sum_{i\in[N]} \mathbf{z}_i$ for $\mathbf{z}_i \leftarrow \mathbf{s}_i \cdot c_i + \mathbf{y}_i$. We can verify the algebraic verification condition as follows:

$$\begin{aligned}
\mathbf{a}^\intercal \mathbf{z}_{\mathsf{ag}} &= v_{\mathsf{ag}} + w_{\mathsf{ag}} \\
&= \sum_{i\in[N]} v_i \cdot c_i + \sum_{i\in[N]} w_i \\
&= \sum_{i\in[N]} \mathbf{a}^\intercal \mathbf{s}_i \cdot c_i + \sum_{i\in[N]} \mathbf{a}^\intercal \mathbf{y}_i \\
&= \sum_{i\in[N]} \mathbf{a}^\intercal (\mathbf{s}_i \cdot c_i + \mathbf{y}_i) \\
&= \mathbf{a}^\intercal \sum_{i\in[N]} \mathbf{z}_i.
\end{aligned}$$

To show that the norm check on $\mathbf{z}_{\mathsf{ag}}$ succeeds, we can first bound the following individual components:

- The components $\mathbf{s}_i$ for $i \in [N]$ are the signing keys of each user. By definition of the key generation algorithm, we have $\|\mathbf{s}_i\| \le \beta_{\mathbf{s}}$.
- The components c_i for $i \in [N]$ are the output of the hash function H_1. By definition of H_1, we have $\|c_i\| \le \beta_1$.
- The components $\mathbf{y}_i$ for $i \in [N]$ are vectors that each signer generates at round 1 of the protocol from $\mathcal{D}^{\ell}_{\mathcal{R},s}$. By the tail bound on the discrete Gaussian distribution (Lemma 6.1), we have $\|\mathbf{y}\| \le \lambda \cdot s$ except with probability $2e^{-\lambda^2/2}$, which is negligible.

Hence, we can bound the norm $\|\mathbf{z}_{\mathsf{ag}}\|$ as follows:

$$\begin{aligned}
\|\mathbf{z}_{\mathsf{ag}}\| &\le N \cdot \max_{i\in[N]} \|\mathbf{z}_i\| \\
&= N \cdot \max_{i\in[N]} \|\mathbf{s}_i \cdot c_i + \mathbf{y}_i\| \\
&\le N \cdot (n\beta_{\mathbf{s}}\beta_1 + \lambda \cdot s) \\
&= \beta_{\mathsf{ver}}.
\end{aligned}$$

Success probability. Consider an instance of a protocol between N signers. Each signer $i \in [N]$ computes its individual signature $\mathbf{z}_i \in \mathcal{R}^\ell$ and sends this component to the other signers with probability $\left(\frac{N-1}{N} \frac{\mathcal{D}^\ell_{\mathcal{R},s}(\mathbf{z}_i)}{\mathcal{D}^\ell_{\mathcal{R},s,\mathbf{s}\cdot c_i}(\mathbf{z}_i)}, 1\right)$. By Lemma 6.6, this probability is $\frac{N-1}{N} \cdot 2^{-\omega(\log n)}$ close to the probability $\frac{N-1}{N}$. Hence, the probability that all signers in the protocol does not abort is bounded by $\left(\frac{N-1}{N}\right)^N = O(1)$ and the correctness follows.

C.3 Proof of Theorem 6.10

We first define a set of hybrid arguments that we use for the proof. The hybrid Hyb_0 corresponds to the real unforgeability security experiment. We use Hyb_1 to capture a set of rare bad events that can cause our final reduction algorithm to fail. We use Hyb_2 to analyze the rejection distribution of the real protocol using Lemma 6.6.

- Hyb_0: This is the original unforgeability security experiment of Definition 5.4 instantiated with Construction 6.7.
- Hyb_1: In this experiment, we introduce a set of abort conditions to bound a set of rare events that are related to the random oracles H_0 and H_1.

  Let $\mathcal{A}$ be an adversary that makes at most Q_0 queries to H_0, Q_1 queries to H_1, and Q_{sign} queries to the signing oracle. The challenger simulates the experiment for $\mathcal{A}$ as follows:

  – Before the start of the experiment, the challenger samples $\nu = Q_1 + Q_{\mathsf{sign}}$ vectors $c'_1, \dots, c'_\nu \xleftarrow{\$} \mathcal{B}_{\beta_1}$. It also instantiates a set of look-up tables:

    * Table $T_0 : \mathcal{R}_q \to \{0,1\}^r$ is used by the challenger to answer $\mathcal{A}$'s random-oracle queries to H_0 consistently.
    * Table $T_1 : [Q_1 + NQ_{\mathsf{sign}}] \times \{0,1\}^* \to \mathcal{B}_{\beta_1}$ is used by the challenger to answer $\mathcal{A}$'s random-oracle queries to H_1 consistently.
    * Table $T_2 : \mathcal{R}_q^\ell \to [Q_1 + NQ_{\mathsf{sign}}]$ keeps track of the public keys that $\mathcal{A}$ submits as a random oracle query to H_1.

    Then, the challenger generates the vector $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$ and a secret-public key pair $\mathsf{sk}^* = \mathbf{s}^*$ and $\mathsf{pk}^* = v^*$ exactly as in Hyb_1. It sets $T_2[\mathsf{pk}^*] \leftarrow 0$.

  – Before $\mathcal{A}$ makes any queries, the challenger initializes flag variables $\mathsf{Bad}_1, \mathsf{Bad}_2, \mathsf{Bad}_3, \mathsf{Bad}_4$ initially set to be `false` and counters $\mathsf{ctr}_1, \mathsf{ctr}_2$ initially set to be 0:

    * ctr_1: This counter is used to assign a unique index to each input that $\mathcal{A}$ submits as a random oracle query to H_1.
    * ctr_2: This counter is used to assign a unique index to each new public keys that $\mathcal{A}$ submits as a random oracle query to H_1.

    The challenger then answers each of $\mathcal{A}$'s queries as follows:

    * $H_0(w)$: If $T_0[w]$ is empty, the challenger sets $T_0[w] \leftarrow \{0,1\}^r$. It returns $T_0[w]$ to $\mathcal{A}$.
    * $H_1(\mathsf{pk} \| \mathsf{str})$: If $T_2[\mathsf{pk}]$ is undefined, then the challenger sets $\mathsf{ctr}_2 \leftarrow \mathsf{ctr}_2 + 1$ and $T_2[\mathsf{pk}] \leftarrow \mathsf{ctr}_2$. If $T_1[\mathsf{ctr}_2, \mathsf{str}]$ is undefined, then it sets $T_1[j, \mathsf{str}] \xleftarrow{\$} \mathcal{B}^\ell_{\beta_1}$ for all $j \in [Q_1 + NQ_{\mathsf{sign}}]$. Finally, it sets $\mathsf{ctr}_1 \leftarrow \mathsf{ctr}_1 + 1$ and $T_1[0, \mathsf{str}] \leftarrow c_{\mathsf{ctr}_1}$.
    * *Signing queries*: Let $\mathsf{PK} = (\mathsf{pk}_j)_{j \in [N]}$ and $\mathsf{M} = (\mathsf{m}_j)_{j \in [N]}$ be a valid signing query where $\mathsf{pk}_i = \mathsf{pk}^*$ for some $i \in [N]$. The challenger answers the query as follows:

      Round 1:

      1. For each index $j \in [N] \backslash \{i\}$ for which $T_2[\mathsf{pk}_j]$ is undefined, the challenger increases $\mathsf{ctr}_2 \leftarrow \mathsf{ctr}_2 + 1$ and sets $T_2[\mathsf{pk}_j] \leftarrow \mathsf{ctr}_2$.
      2. Then, it increases $\mathsf{ctr}_1 \leftarrow \mathsf{ctr}_1 + 1$ and sets $c_i \leftarrow c'_{\mathsf{ctr}_1}$.

3. The challenger samples $\mathbf{y}_i \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$, sets $w_i \leftarrow \mathbf{a}^{\intercal}\mathbf{y}_i$, and checks if $T_0[w_i]$ is undefined. If so, it samples $h_i \leftarrow \{0,1\}^r$ and sets $T_0[w_i] \leftarrow h_i$. It sends h_i to the cosigners.[3]

Round 2:

1. After receiving $(h_j)_{j\in[N]\backslash\{i\}}$ from the cosigners, the challenger checks the following abort conditions.
   · If there exists an index $j \in [N]\backslash\{i\}$ for which h_j is not in T_0 (there does not exist any vector $w \in \mathcal{R}_q$ such that $T_0[w] = h_j$), then it sets $\mathsf{Bad}_1 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully predicts the output of the random oracle H_0.
   · If there exists an index $j \in [N]\backslash\{i\}$ for which there exist two distinct vectors $w, w' \in \mathcal{R}_q$ such that $T_0[w] = T_0[w'] = h_j$, then it sets $\mathsf{Bad}_2 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully finds a collision on the random oracle H_0.
2. If it does not abort, then there exists a unique vector w_j such that $T_0[w_j] = h_j$ for all $j \in [N]$. It defines $w_{\mathsf{ag}} \leftarrow \sum_{j\in[N]} w_j$ and checks the following abort condition.
   · If the entry $T_1[0, \mathsf{str}]$ for $\mathsf{str} = (w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$ is undefined, then it sets $\mathsf{Bad}_3 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully predicts the vector w_{ag}.
3. Finally, if the challenger does not abort the experiment, then it sets $T_1[0, \mathsf{str}] \leftarrow c_i$, samples $T_1[j, \mathsf{str}] \xleftarrow{\$} \mathcal{B}^{\ell}_{\beta_1}$ for all $j \in [N]\backslash\{i\}$, and sends w_i to the cosigners.

Round 3:

1. After receiving $(w_j)_{j\in[N]\backslash\{i\}}$ from the cosigners, the challenger checks if $T_0[w_j] = h_j$ for all $j \in [N]$. If this is not the case, then it aborts and returns $\perp$.[4]
2. If it does not abort, then it sets $\mathbf{z}_i \leftarrow \mathbf{s}^* \cdot c_i + \mathbf{y}_i$. It sends $\mathbf{z}_i$ to the cosigners with probability $\min\left(\frac{N-1}{N}\frac{\mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z}_i)}{\mathcal{D}^{\ell}_{\mathcal{R},s,\mathbf{s}\cdot c_i}(\mathbf{z}_i)}, 1\right)$ and otherwise, it sends abort to the signers to signal that it aborted.

$*$ Eventually, the adversary $\mathcal{A}$ returns a forgery that consists of a set of public keys $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$, a set of messages $\mathsf{M} = (\mathsf{m}_j)_{j\in[N]}$, and a signature $\sigma_{\mathsf{ag}} = (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$. The challenger first checks if $T_2[\mathsf{pk}_j]$ is defined for each $j \in [N]$. If not, then it sets $\mathsf{Bad}_4 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully predicts the output of the random oracle H_1.

If the challenger does not abort, then it computes $v_{\mathsf{ag}} \leftarrow \sum_{j\in[N]} v_j \cdot c_j$ for $v_j = \mathsf{pk}_j$, $c_j \leftarrow T_1[j, \mathsf{str}_j]$, and $\mathsf{str}_j = (w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_j)$ for $j \in [N]$. It checks the following conditions:

· $\mathbf{a}^{\intercal} \cdot \mathbf{z}_{\mathsf{ag}} = w_{\mathsf{ag}} + v_{\mathsf{ag}}$,

· $\|\mathbf{z}_{\mathsf{ag}}\| \leq \beta_{\mathsf{ver}}$.

If both of these conditions are true, then the challenger returns 1. Otherwise, it returns 0.

- Hyb_2: This experiment is identical to Hyb_1 except for the way the challenger simulates the responses to $\mathcal{A}$'s signing queries. Namely, in Hyb_2, the challenger simulates the responses to $\mathcal{A}$'s queries without relying on the signing key $\mathbf{s}^*$.

  Let $\mathsf{PK} = (\mathsf{pk}_j)_{j\in[N]}$, $\mathsf{M} = (\mathsf{m}_j)_{j\in[N]}$ be a valid signing query made by $\mathcal{A}$. Then there exists an index $i \in [N]$ such that $\mathsf{pk}_i = \mathsf{pk}^*$. The challenger responds to each of these queries as follows. The steps that are underlined differs from those of the previous experiment Hyb_1.

  Round 1:

  1. For each index $j \in [N]\backslash\{i\}$ for which $T_2[\mathsf{pk}_j]$ is undefined, the challenger increases $\mathsf{ctr}_2 \leftarrow \mathsf{ctr}_2 + 1$ and sets $T_2[\mathsf{pk}_j] \leftarrow \mathsf{ctr}_2$.

[3]Note that these cosigners are controlled by the adversary $\mathcal{A}$.

[4]Note that this abort condition is part of the protocol specification and does not correspond to any bad event.

2. Then, it increases $\mathsf{ctr}_1 \leftarrow \mathsf{ctr}_1 + 1$ and sets $c_i \leftarrow c'_{\mathsf{ctr}_1}$.
3. The challenger samples $\mathbf{z} \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$, $c_i \leftarrow c'_{\mathsf{ctr}_1}$, sets $w_i \leftarrow \mathbf{a}^{\intercal}\mathbf{z} - v^* c_i$. It sets $h_i \leftarrow H_0(w_i)$ and sends h_i to the cosigners.

Round 2:

1. After receiving $(h_j)_{j \in [N]\setminus\{i\}}$ from the cosigners, the challenger checks the following abort conditions.
   – If there exists an index $j \in [N]\setminus\{i\}$ for which h_j is not in T_0 (there does not exist any vector $w \in \mathcal{R}_q$ such that $T_0[w] = h_j$), then it sets $\mathsf{Bad}_1 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully predicts the output of the random oracle H_0.
   – If there exists an index $j \in [N]\setminus\{i\}$ for which there exist two distinct vectors $w, w' \in \mathbb{Z}_p^n$ such that $T_0[w] = T_0[w'] = h_j$, then it sets $\mathsf{Bad}_2 \leftarrow \mathtt{true}$ and aborts the experiment. This abort condition occurs if $\mathcal{A}$ successfully finds a collision on the random oracle H_0.
2. The challenger finds unique entries w_j such that $T_0[w_j] = h_j$ for each $j \in [N]\setminus\{i\}$ and sets $w_{\mathsf{ag}} \leftarrow \sum_{j\in[N]} w_j$. The challenger sets $\mathsf{str} \leftarrow (w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$, programs the table $T_1[j, \mathsf{str}] \leftarrow \mathcal{B}^{\ell}_{\beta_1}$ for all $j \in [Q_1 + NQ_{\mathsf{sign}}]$ and $T_1[0, \mathsf{str}] \leftarrow c_i$.
3. Finally, the challenger sets $T_1[0, \mathsf{str}] \leftarrow c_i$, samples $T_1[j, \mathsf{str}] \xleftarrow{\$} \mathcal{B}^{\ell}_{\beta_1}$ for all $j \in [N]\setminus\{i\}$, and sends w_i to the cosigners.

Round 3:

1. After receiving $(w_j)_{j \in [N]\setminus\{i\}}$ from the cosigners, the challenger checks if $T_0[w_j] = h_j$ for all $j \in [N]$. If this is not the case, then it aborts and returns $\perp$.[5]
2. If it does not abort, then it sets $\mathbf{z}_i \leftarrow \mathbf{s}^* \cdot c_i + \mathbf{y}_i$. It sends $\mathbf{z}_i$ to the cosigners with probability $\frac{N-1}{N}$ and otherwise, it sends abort to the signers to signal that it aborted.

The rest of the experiment remains unchanged from Hyb_1.

We now bound the probability that an adversary $\mathcal{A}$ makes a successful forgery in each of the hybrid experiments. For each hybrid experiment Hyb, we use the notation $\mathsf{Hyb}(\mathcal{A})$ to denote the probability that $\mathcal{A}$ successfully forges a signature in experiment Hyb.

Lemma C.1. *Suppose that the ring $\mathcal{R}$ is instantiated as $\mathcal{R} = \mathbb{Z}^{n\times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and let $s = \omega(n \cdot \beta_{\mathbf{s}}\beta_1\sqrt{\log n})$ be a Gaussian parameter such that the conditions of Lemma 6.4 or Lemma 6.5 are satisfied. Then, for any (unbounded) adversary $\mathcal{A}$ that makes Q_0 oracle queries to H_0, Q_1 oracle queries to H_1, and Q_S signing queries, we have*

$$\big|\Pr\big[\mathsf{Hyb}_0(\mathcal{A}) = 1\big] - \Pr\big[\mathsf{Hyb}_1(\mathcal{A}) = 1\big]\big| \le \frac{N \cdot Q_S}{2^r} + \frac{(Q_0 + N\cdot Q_S)^2}{2^r} + \frac{2n\cdot Q_S}{q^n} + \frac{1}{|\mathcal{B}_{\beta_1}|}.$$

Proof. The only difference between the two hybrid experiments Hyb_0 and Hyb_1 is the additional abort condition in Hyb_1 when any of the events $\mathsf{Bad}_1, \mathsf{Bad}_2, \mathsf{Bad}_3$ and Bad_4 occurs. Therefore, the statistical distance between $\mathcal{A}$'s views in Hyb_0 and Hyb_1 is bounded by probability that any of these events occur during an execution of Hyb_1. We bound the probability that each of these bad events occur as follows:

- Bad_1: The event Bad_1 occurs only when $\mathcal{A}$ predicts the output of H_0. The check occurs at most N times during each signing query. Therefore, we have

$$\Pr[\mathsf{Bad}_1] \le \frac{N\cdot Q_S}{2^r}.$$

[5]Note that this abort condition is part of the protocol specification and does not correspond to any bad event.

- Bad_2: The event Bad_1 occurs only when a collision occurs in T_0. An adversary $\mathcal{A}$ makes at most Q_0 random oracle queries to H_0 and the challenger makes at most N random oracle queries to H_0 per $\mathcal{A}$'s signing query. Therefore, a collision in T_0 occurs with probability at most
$$\Pr[\mathsf{Bad}_2] \leq \frac{(Q_0 + N \cdot Q_S)^2}{2^r}.$$
- Bad_3: The event Bad_3 occurs only when $\mathcal{A}$ successfully predicts w_{ag}. By definition, $w_{\mathsf{ag}} = \sum_{j \in [N]} w_j$ and $w_i = \mathbf{a}^\intercal \cdot \mathbf{y}$ for $\mathbf{y} \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$. By the leftover hash lemma (Lemmas 6.4 and 6.5), the element w_i is within $n \cdot q^{-n}$ statistical distance of a uniformly random vector in $\mathcal{R}_q$. Since the check occurs for each of $\mathcal{A}$'s signing query, the probability that $\mathcal{A}$ successfully predicts the output of w_{ag} for any of these signing queries is bounded by
$$\Pr[\mathsf{Bad}_3] \leq \frac{2n \cdot Q_S}{q^n}.$$
- Bad_4: The event Bad_4 occurs only when $\mathcal{A}$ successfully predicts the output of the random oracle H_1. This check occurs once when $\mathcal{A}$ submits its final forgery. Therefore, we have
$$\Pr[\mathsf{Bad}_4] \leq \frac{1}{|\mathcal{B}_{\beta_1}|}.$$

Combining each of these probabilities by a union bound, the lemma follows. □

Lemma C.2. *Suppose that the ring $\mathcal{R}$ is instantiated as $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n, and let $s = \omega(n \cdot \beta_{\mathsf{s}} \beta_1 \sqrt{\log n})$ be a Gaussian parameter such that the conditions of Lemma 6.4 or Lemma 6.5 are satisfied. Then, for any (unbounded) adversary $\mathcal{A}$ that makes Q_S signing queries, we have*
$$\big| \Pr\big[\mathsf{Hyb}_1(\mathcal{A}) = 1\big] - \Pr\big[\mathsf{Hyb}_2(\mathcal{A}) = 1\big] \big| \leq Q_S \cdot 2^{-\omega(\log n)}/M.$$

Proof. The lemma follows straightforwardly from Lemma 6.6 and the abort conditions. The only difference between the two hybrid experiments is in the way the challenger responds to $\mathcal{A}$'s signing queries. Let $\mathsf{PK} = (\mathsf{pk}_j)_{j \in [N]}$, $\mathsf{M} = (\mathsf{m}_j)_{j \in [N]}$ be a signing query made by $\mathcal{A}$ such that $\mathsf{pk}_i = \mathsf{pk}^*$. Then, the main differences in the challenger's specifications in the two experiments are as follows:

- **Rounds 1 and 2**: In Hyb_1, the challenger first samples $\mathbf{y}_i \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$ and sets $w_i \leftarrow \mathbf{a}^\intercal \mathbf{y}_i$. In Hyb_2, the challenger first samples $\mathbf{z} \leftarrow \mathcal{D}^{\ell}_{\mathcal{R},s}$ and sets $w_i \leftarrow \mathbf{a}^\intercal \mathbf{z} - v^* c_i$ where $c_i \xleftarrow{\$} \mathcal{B}_{\beta_1}$.
- **Round 3**: In Hyb_1, the challenger aborts the protocol with probability $\min\left(\frac{N-1}{N}\frac{\mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z})}{\mathcal{D}^{\ell}_{\mathcal{R},s}(\mathbf{z})}, 1\right)$ while in Hyb_2, the challenger aborts with probability $\frac{N-1}{N}$.

Therefore, given the public vector $\mathbf{a}$, public key $\mathsf{pk}^* = v^*$, and the outputs of the random oracles H_0 and H_1, the adversary $\mathcal{A}$'s view on a single signing query in Hyb_1 is identical to the real distribution in Lemma 6.6. Furthermore, as long as the random oracle H_1 is consistent with c_i that the challenger samples, $H_1(\mathsf{pk}^*, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i) = c_i$, the adversary $\mathcal{A}$'s view on the signing query in Hyb_2 is identical to the ideal distribution in Lemma 6.6. We have $H_1(\mathsf{pk}^*, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i) \neq c_i$ only when the table $T_1[0, \mathsf{str}]$ for $\mathsf{str} = (w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_i)$ is already defined before the challenger computes w_{ag} in Round 2 of the protocol. This event corresponds to the event $\mathsf{Bad}_3 = \mathtt{true}$ in hybrids Hyb_1 and Hyb_2, and therefore, the challenger returns $\perp$ in both experiments. The lemma now follows by the union bound over Q_S number of $\mathcal{A}$'s signing queries. □

Lemma C.3. *Suppose that the ring $\mathcal{R}$ is instantiated as $\mathcal{R} = \mathbb{Z}^{n \times n}$ or $\mathcal{R} = \mathbb{Z}[X]/(X^n + 1)$ for a positive integer n and let $\mathcal{A}$ be an adversary in Hyb_2 that makes Q_1 random oracle queries to H_1 and Q_{sign} signing queries. Then, there exists an adversary $\mathcal{B}$ such that*
$$\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \leq \frac{Q_1 + Q_{\mathsf{sign}}}{|\mathcal{B}_{\beta_1}|} + \sqrt{(Q_1 + Q_{\mathsf{sign}}) \cdot \mathsf{Adv}_{\mathsf{SIS}_{\mathcal{R},\ell,q,\beta^\star}}[\lambda, \mathcal{B}] + \frac{1}{|\mathcal{B}_{\beta_{\mathsf{s}}}|}},$$
for $\beta^\star = 2\beta_{\mathsf{ver}} + 2n\beta_{\mathsf{s}}\beta_1$.

Proof. We use the forking lemma (Lemma 2.6) to prove the lemma. Specifically, we construct an algorithm $\mathcal{B}$ that uses the forking algorithm on a simulator for the experiment Hyb_2 to solve an SIS challenge. We first specify a suitable simulation algorithm (Definition 2.4) that interacts with a forging adversary in Hyb_2 and outputs an index and auxiliary output (i, aux). For our proof, the auxiliary output will correspond to an adversary's forgery at the end of a successful execution of Hyb_2 and the index i will correspond to the random oracle query associated with the forgery.

Let $\mathcal{A}$ be a forging adversary that makes at most Q_0 queries to H_0, Q_1 queries to H_1, and Q_{sign} queries to the signing oracle. We specify a simulation algorithm $\mathcal{S}$ for $\mathcal{A}$ as follows:

- $\mathcal{S}(\mathbf{a}, c'_1, \ldots, c'_\nu)$: The simulation algorithm takes in an SIS challenge $\mathbf{a} \in \mathcal{R}_q^\ell$ and a set of ring elements $c'_1, \ldots, c'_\nu \in \mathcal{B}_{\beta_1}$ for $\nu = Q_1 + Q_{\mathsf{sign}}$. It follows the exact specification of the challenger in Hyb_2 except that it uses the vector $\mathbf{a}$ as the public parameter vector and the elements $c'_1, \ldots, c'_\nu$ as the outputs to program H_1 as specified in Hyb_2.

  Suppose that $\mathcal{A}$ produces a forgery $(\mathsf{PK}, \mathsf{M}, \sigma_{\mathsf{ag}})$ for $\mathsf{PK} = (\mathsf{pk}_i)_{i \in [N]}$, $\mathsf{M} = (\mathsf{m}_i)_{i \in [N]}$, and $\sigma_{\mathsf{ag}} = (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$. If the output of the experiment is 0 (adversary's forgery is not valid) or it aborted the experiment at any point, then the simulation algorithm returns $(0, \varepsilon)$. If the output of the experiment is 1 (adversary's forgery is valid), then it first finds an index $i^* \in [N]$ for which $\mathsf{pk}^* = \mathsf{pk}_{i^*}$ and an index $j^* \in [N]$ for which $T_1[0, \mathsf{str}] = c'_{j^*}$ where $\mathsf{str} = (w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_{i^*})$. It returns $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})\big)$.

When the SIS challenge $\mathbf{a}$ is generated uniformly at random $\mathbf{a} \xleftarrow{\$} \mathcal{R}_q^\ell$ as in the SIS problem, and the hash outputs $c'_1, \ldots, c'_\nu$ are generaled uniformly at random, then the simulation algorithm $\mathcal{S}$ perfectly simulates Hyb_2 by definition. Therefore, the probability that $\mathcal{S}$ returns $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})\big)$ for $j^* \neq 0$, $(w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}) \neq \varepsilon$ equates to the probability that adversary $\mathcal{A}$'s wins in Hyb_2:

$$\mathsf{Adv}^{\mathsf{Sim}}(\lambda, \mathcal{A}) = \Pr\big[\mathsf{Hyb}_2(\mathcal{A}) = 1\big].$$

Next, we invoke the forking algorithm $\mathcal{F}_\mathcal{S}$ (Definition 2.5) on the simulation algorithm $\mathcal{S}$ above. The forking algorithm takes in an SIS challenge $\mathbf{a} \in \mathcal{R}_q^\ell$ and invokes two instances of $\mathcal{S}$. It returns an index and two forgeries $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}), (\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})\big)$. Using the general forking lemma Lemma 2.6, we can bound the probability that $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}), (\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})\big) \neq (0, \perp, \perp)$:

$$\mathsf{Adv}^{\mathsf{Sim}}(\lambda, \mathcal{S}) \leq \frac{\nu}{|\mathcal{B}_{\beta_1}|} + \sqrt{\nu \cdot \mathsf{Adv}^{\mathsf{Fork}}(\lambda, \mathcal{F}, \mathcal{S})}.$$

Finally, we construct a reduction algorithm $\mathcal{B}$ that invokes the forking algorithm $\mathcal{F}_\mathcal{S}$ and solves an SIS challenge.

- $\mathcal{B}(\mathbf{a})$: The reduction algorithm takes in an SIS challenge $\mathbf{a} \in \mathcal{R}_q^\ell$ and invokes the forking algorithm $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}), (\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})\big) \leftarrow \mathcal{F}_\mathcal{S}(\mathbf{a})$. If $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}), (\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})\big) = (0, \perp, \perp)$, then it return Fail. Otherwise, it returns the element $\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})$ as a solution to the SIS challenge $\mathbf{a}$.

To complete the proof of the lemma, we show that as long as algorithm $\mathcal{B}$ does not output Fail, the solution $\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})$ is a valid SIS challenge except with probability $1/|\mathcal{B}_{\beta_{\mathsf{s}}}|$.

Correctness of algorithm $\mathcal{B}$. Suppose that $\mathcal{F}_\mathcal{S}$ on input $\mathbf{a} \in \mathcal{R}_q$ returns $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}}), (w'_{\mathsf{ag}}, \mathbf{z}'_{\mathsf{ag}})\big) \neq (0, \perp, \perp)$ for some $j^* \in [\nu]$ after two executions of $\mathcal{S}$. Let us consider these two executions with the following inputs and outputs:

- $\mathcal{S}(\mathbf{a}, c'_1, \ldots, c'_{j^*-1}, c'_{j^*}, \ldots, c'_\nu)$ returned $\big(j^*, (w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})\big)$,
- $\mathcal{S}(\mathbf{a}, c'_1, \ldots, c'_{j^*-1}, \tilde{c}'_{j^*}, \ldots, \tilde{c}'_\nu)$ returned $\big(j^*, (\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})\big)$,

where $c'_1, \ldots, c'_\nu \xleftarrow{\$} \mathcal{B}_{\beta_1}$ and $\tilde{c}'_{j^*}, \ldots, \tilde{c}'_N \xleftarrow{\$} \mathcal{B}_{\beta_1}$. Since the forged signatures $(w_{\mathsf{ag}}, \mathbf{z}_{\mathsf{ag}})$ and $(\tilde{w}_{\mathsf{ag}}, \tilde{\mathbf{z}}_{\mathsf{ag}})$ are $\mathcal{A}$'s valid forgeries, we have

$$\mathbf{a}^\intercal \cdot \mathbf{z}_{\mathsf{ag}} = \sum_{j \in [N]} v_j \cdot c_j + w_{\mathsf{ag}}. \tag{C.1}$$

$$\mathbf{a}^\intercal \cdot \tilde{\mathbf{z}}_{\mathsf{ag}} = \sum_{j \in [\tilde{N}]} \tilde{v}_j \cdot \tilde{c}_j + \tilde{w}_{\mathsf{ag}}, \tag{C.2}$$

where

- The elements $v_1, \ldots, v_N$ and $\tilde{v}_1, \ldots, \tilde{v}_{\tilde{N}}$ correspond to the set of public keys $\mathsf{PK} = (\mathsf{pk}_j)_{j \in [N]}$ and $\tilde{\mathsf{PK}} = (\tilde{\mathsf{pk}}_j)_{j \in [\tilde{N}]}$ included in $\mathcal{A}$'s two forgeries of the experiments.
- The vectors $c_1, \ldots, c_N$ and $\tilde{c}_1, \ldots, \tilde{c}_{\tilde{N}}$ correspond to the output of the random oracle H_1 in the two experiment executions: $c_j \leftarrow H_1(\mathsf{pk}_j, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_j)$ for $j \in [N]$ and $\tilde{c}_j \leftarrow H_1(\tilde{\mathsf{pk}}_j, \tilde{w}_{\mathsf{ag}}, \tilde{\mathsf{PK}}, \tilde{\mathsf{m}}_j)$ for $j \in [\tilde{N}]$. By specification of the simulation algorithm (the challenger in Hyb_2), we have $c_{j^*} = c'_{j^*}$ and $\tilde{c}_{j^*} = \tilde{c}'_{j^*}$.

We now note that the assignment $c_{j^*} = c'_{j^*}$ occurs precisely when the adversary $\mathcal{A}$ in the first execution of the experiment makes its first random oracle query to H_1 for $(\mathsf{pk}_j, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_j)$ for some $j \in [N]$. Similarly, the assignment $\tilde{c}_{j^*} = \tilde{c}'_{j^*}$ occurs when the adversary $\mathcal{A}$ in the second execution of the experiment makes its second random oracle query to H_1 for $(\tilde{\mathsf{pk}}_j, \tilde{w}_{\mathsf{ag}}, \tilde{\mathsf{PK}}, \mathsf{m}_j)$ for some $j \in [N]$. Therefore, up to this point of the experiment, the two executions of Hyb_2 by $\mathcal{S}$ are identical, which means that the arguments to the random oracle H_1 are also identical. This implies that $N = \tilde{N}$, $v_j = \tilde{v}_j$ for all $j \in [N]$, $w_{\mathsf{ag}} = \tilde{w}_{\mathsf{ag}}$, and $\mathsf{m}_j = \mathsf{m}_{\tilde{j}}$. Furthermore, since the table entries $\mathsf{T}_1[j, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_j]$ for $j \in [N]$ are assigned before $\mathcal{A}$ sees the output of H_1 on query $(\mathsf{pk}_j, w_{\mathsf{ag}}, \mathsf{PK}, \mathsf{m}_j) = (\tilde{\mathsf{pk}}_j, \tilde{w}_{\mathsf{ag}}, \tilde{\mathsf{PK}}, \mathsf{m}_{\tilde{j}})$ in the two executions, we have $c_j = \tilde{c}_j$ for all $j^* < j \leq N$.

Therefore, combining the two relations in (C.1) and (C.2), we have

$$\mathbf{a}^\intercal \cdot (\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}}) = \sum_{j \in [N]} \big(v_j \cdot c_j - \tilde{v}_j \cdot \tilde{c}_j\big) - (w_{\mathsf{ag}} - \tilde{w}_{\mathsf{ag}}) = v^* \cdot (c_{j^*} - \tilde{c}_{j^*}).$$

Using the fact that $v^* = \mathbf{a}^\intercal \cdot \mathbf{s}^*$, we have $\mathbf{a}^\intercal \big(\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})\big) = 0$ and therefore, $\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})$ is a potential solution to the SIS challenge with norm

$$\|\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})\| \leq 2\beta_{\mathsf{ver}} + 2n\beta_{\mathbf{s}}\beta_1.$$

What remains is to bound the probability that the element $\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*})$ is not the zero element in $\mathcal{R}$.

We note that by the specification of Hyb_2, the view of the experiment that $\mathcal{S}$ simulates to $\mathcal{A}$ is independent of the signing key $\mathbf{s}^*$ that $\mathcal{S}$ generates at the start of the experiment. Therefore, the vectors $\mathbf{z}_{\mathsf{ag}}$ and $\tilde{\mathbf{z}}_{\mathsf{ag}}$ are independent of the vector $\mathbf{s}^*$. Furthermore, since the forking algorithm $\mathcal{F}_{\mathcal{S}}$ does not output $(0, \bot, \bot)$, we have $c_{j^*} \neq \tilde{c}_{j^*}$. Hence, given that $\mathbf{s}^* \xleftarrow{\$} \mathcal{B}_{\beta_{\mathsf{s}}}$, the probability that $\mathbf{z}_{\mathsf{ag}} - \tilde{\mathbf{z}}_{\mathsf{ag}} - \mathbf{s}^*(c_{j^*} - \tilde{c}_{j^*}) = 0$ is at most $1/|\mathcal{B}_{\beta_{\mathsf{s}}}|$. The lemma now follows.

□