Ringtail: Practical Two-Round Threshold Signatures from Learning with Errors

Cecilia Boschini
ETH Zürich, Switzerland

Darya Kaviani
UC Berkeley, USA

Russell W. F. Lai
Aalto University, Finland

Giulio Malavolta
Bocconi University, Italy

Akira Takahashi
JPMorgan AI Research & AlgoCRYPT CoE, USA

Mehdi Tibouchi
NTT Social Informatics Laboratories, Japan

***Abstract*—A threshold signature scheme splits the signing key among ℓ parties, such that any t-subset of parties can jointly generate signatures on a given message. Designing concretely efficient post-quantum threshold signatures is a pressing question, as evidenced by NIST's recent call.**

In this work, we propose, implement, and evaluate a lattice-based threshold signature scheme, Ringtail, which is the first to achieve a combination of desirable properties: (i) The signing protocol consists of only two rounds, where the first round is message-independent and can thus be preprocessed offline. (ii) The scheme is concretely efficient and scalable to $t \leq 1024$ parties. For 128-bit security and $t = 1024$ parties, we achieve 13.4 KB signature size and 10.5 KB of online communication. (iii) The security is based on the standard learning with errors (LWE) assumption in the random oracle model. This improves upon the state-of-the-art (with comparable efficiency) which either has a three-round signing protocol [Eurocrypt'24] or relies on a new non-standard assumption [Crypto'24].

To substantiate the practicality of our scheme, we conduct the first WAN experiment deploying a lattice-based threshold signature, across 8 countries in 5 continents. We observe that an overwhelming majority of the end-to-end latency is consumed by network latency, underscoring the need for round-optimized schemes.

1. Introduction

In a t-out-of-ℓ threshold signature [Des88], [DF90], secret-shares of the signing key are distributed among ℓ parties, such that any t-sized subset can jointly compute a signature on any message of their choice. Furthermore, an adversary corrupting up to $t-1$ users should not be able to forge a new signature. Threshold signatures are a versatile cryptographic primitive widely used to delegate signing rights while tolerating a fraction of corrupted parties. Because of their wide applicability, threshold signatures have been extensively studied in pre-quantum settings, with many efficient constructions known satisfying increasingly stronger security guarantees, see, e.g. [BCK+22] and references therein.

In contrast, post-quantum and, in particular, *lattice-based* threshold signatures are far less studied. Indeed, designing a scheme that is concretely efficient, conceptually simple, and based on well-studied computational assumptions is still very much an open research question. Finding such a scheme is increasingly pressing in the context of the ongoing transition to post-quantum cryptographic systems, particularly the preliminary call for multi-party threshold schemes by the U.S. agency NIST [BP23]. We now describe a set of properties that a threshold signature scheme suitable for large-scale usage should ideally satisfy.

Concrete Efficiency: The signing and verification algorithms must be fast on commodity hardware. The signature size must be comparable to that of ordinary (non-threshold) post-quantum signatures, e.g. Dilithium [LDK+20]. Overall, all relevant efficiency metrics must scale sublinearly (e.g. logarithmically) with both t (the threshold) and ℓ (the total number of parties).

2*-Round Offline/Online Signing:* To avoid delays caused by communication roundtrips, the scheme should minimize round complexity, and should ideally consist of two rounds of simultaneous communication between the users involved in the signing protocol. Furthermore, an important property is the so-called offline/online interaction, which requires that the first round of interaction to be message-independent. This allows parties to precompute many instances of the first round in advance, effectively reducing the interaction needed to compute a signature to a single broadcast.

Established Theoretical Foundations: Finally, it is important that the scheme can be proven secure in a well-established computational model against *standard* computational assumptions. Besides reducing the risk of future cryptanalytic breakthroughs, this makes it easier to conservatively estimate the parameters of the scheme.

1.1. Our Results

In this work, we introduce Ringtail, a lattice-based threshold signature scheme satisfying all the aforementioned desiderata. In particular, Ringtail is concretely efficient, has a 2-round signing protocol with only a single online round, and is proven secure under the standard (module) LWE assumption in the random oracle model, against a bounded number of signature queries. In Section 3.3, we suggest parameters for instantiating Ringtail with 128, 192, and 256 bits of security respectively, assuming that $t \leq 1024$ and that the adversary makes at most 2^{60} signature queries, following NIST's recommendations [BP23].

The parameters are independent of the total number of parties ℓ. We also provide a prototype implementation of Ringtail and benchmark its performance in Section 4. For 128-bit security and $t = 1024$, the signature size is 13.4 KB. We conduct experiments in a WAN setting, with 8 servers across 5 continents as the signing parties. Our online communication is 10.5 KB, which is 81% and 25% lower than the state-of-the-art protocols, respectively [PKM+24], [EKT24]. We find that the efficiency bottleneck of running a full signing protocol is the network latency, whereas the local computation accounts for a relatively insignificant fraction of the total runtime. This highlights the importance of minimizing the round complexity and online communication bandwidth of the threshold signature scheme.

On a technical level, Ringtail is based on the state-of-the-art Raccoon threshold signature family [PEK+23], [PKM+24], [EKT24].[1] It can be seen as a variant of the 2-round threshold signature scheme of Espitau et al. [EKT24] with a few important technical twists partly inspired by MuSig-L [BTT22], a 2-round lattice-based multi-signature scheme based on the standard (module) short integer solution (SIS) assumption. These modifications allow us to argue that Ringtail satisfies the standard security notion for threshold signatures, assuming the hardness of the standard (module) LWE problem, in the random oracle model. In contrast, Threshold Raccoon [PKM+24] requires 3 online rounds for signing, and the scheme of Espitau et al. [EKT24] is only proven secure under a new non-standard assumption called algebraic one-more LWE. A more detailed discussion can be found in Section 1.3.

1.2. Applications

Threshold signatures are naturally applicable in settings where centralized signing keys pose a great threat (e.g. financial or infrastructural) of being compromised. Distributing signing keys removes the central point of attack, helping to mitigate these risks. We highlight a few settings of particular importance. *Digital asset custody* offered by popular MPC wallets secure billions of dollars in funds, which risk being stolen if signing keys are compromised [AB23]. Turning to wallets with threshold signing helps mitigate this financial risk. *Certificate authorities* routinely sign certificates that bind digital identities to cryptographic keys. Breached signing keys have led to fraudulent certificates that green-light malware and impersonate trusted websites [Wol16]. Threshold signatures allow a consortium of certificate authorities (or their administrators) to jointly sign certificates without a single central point of attack. *Well-known code signing services* allow organizations who provide critical software to sign code updates. With a threshold signature, an attacker cannot endorse malicious software unless t signing parties are breached.

1. The ringtail is a small mammal of the raccoon family.

1.3. Related Work

Generic Approaches. The question of efficient distributed signing has received a lot of attention in recent years, due to its application to blockchain-based technologies. Intuitively, this should be a textbook use case for secure multi-party computation (MPC), e.g. [BKP13], [CS19], or fully homomorphic encryption (FHE), e.g. [BGG+18], [ASY22]. In particular, FHE-based solutions allow signing in a single round. Unfortunately, these general methods do not yield concretely efficient schemes, so tailored designs are preferable in practice.

Fiat-Shamir-based Constructions. Several recent constructions rely on the blueprint of efficient Schnorr-based threshold (e.g., FROST family [KG20], [BCK+22], [CKM23], SimpleTSig [CKM23]) and multisignatures (e.g., MuSig2 [NRS21], DWMS [AB21], BN scheme [BN06], mBCJ scheme [BCJ08], [DEF+19]), exploiting the similarity between Schnorr and its lattice-based counterparts in the "Fiat-Shamir with abort [Lyu12]" (FSwA) paradigm.

Damgård et al. [DOTT22] designed a 2-round (t, t)-distributed signature and a multisignature[2] by combining the trapdoor commitment-based simulation from mBCJ with FSwA. Subsequent works improved these constructions by reducing the number of aborts [ADP24], improving the communication complexity [Che23], or turning it into a (t, ℓ)-threshold scheme [GKS24] with threshold homomorphic encryption. The two-round scheme proposed in [GKS24] obtains signatures of size 46.6 KB and public keys of size 13.6 KB for $(t, \ell) = (3, 5)$, while the first round cannot be preprocessed.

Threshold Raccoon (tRaccoon) [PKM+24] is a 3-round (t, ℓ)-threshold signature scheme with a similar design to the SimpleTSig. Crucially, to overcome the technical issue of generalizing the (t, t)-threshold schemes using Shamir secret sharing [Sha79], tRaccoon introduced a masking technique. Roughly, the idea is to use one-time masks to hide the partial signatures generated by individual parties, which may leak non-trivial information about the secret key if revealed in plain. To realize a 2-round scheme, the well-known "random linear combination trick" originated from FROST, MuSig2, DWMS has been adapted to the lattice setting. Boschini et al. [BTT22] obtained a 2-round multi-signature MuSig-L from LWE and SIS assumptions by plugging in a random linear combination into FSwA. The construction by Chairattana-Apirom, Tessaro and Zhu [CATZ24] also follows the structure of FROST but relies on Benaloh-Leichter secret sharing [BL90] with very large

2. The difference is in the key generation algorithm: in multisignatures, the set of signers is not fixed, but each signer generates their key pair when they join, and verification of a signature requires all the public keys of the signers (unless a key aggregation algorithm exists).

(in dimensions) shares. This allows them to obtain a 2-round (t, ℓ)-threshold signature with offline/online signing from standard SIS but at the cost of concrete efficiency. Concretely, the signature size of [CATZ24] is over 200 KB for $\ell = 5$ and arbitrary $0 \leq t \leq \ell$ and the communication cost per signer is over 1MB. Similarly, Espitau, Katsumata, and Takemure [EKT24] (henceforth EKT) obtained a 2-round (t, ℓ)-threshold signature scheme, but by carefully integrating the random linear combination method into tRaccoon [PKM+24], making it compatible with standard Shamir secret sharing and achieving modest signature sizes. The security of these constructions and Ringtail is guaranteed in the static corruption model, in which the adversary commits to a corruption set at the beginning of the unforgeability game. Very recently, [KRT24] presented an adaptively secure variant of tRaccoon that requires 5 rounds of interaction (of which the first round can be preprocessed).

Comparison with EKT**.** The scheme Ringtail introduced in this work can be seen as a variant of EKT, and thus is also based on tRaccoon and FROST, with some crucial differences: 1) *Computational assumptions:* EKT is proven secure against a new non-standard computational assumption called *algebraic one-more LWE (AOM-LWE)*, adapting the one-more discrete log-based proof for FROST and MuSig2, while we follow a MuSig-L-like approach to obtain a security proof from the standard LWE and SIS assumptions. 2) *Distribution of coefficients and randomness:* The difference in proof strategy leads us to sample random coefficients of the linear combination from a discrete *Gaussian distribution*, whereas EKT opts for elements with ℓ_1- and ℓ_∞-norms being 1. Since MuSig-L-style proof crucially relies on preimage sampling of Gaussian vectors, there seems to be no obvious way to adapt our proof strategy to EKT as-is. Moreover, we generate LWE commitments in the offline phase from imbalanced Gaussian samples, to invoke the Hint-LWE assumption [KLSS23]. On the other hand, EKT does not seem to require this. 3) *Full-rank check:* To prove the security of our protocol, it is crucial that each party checks the sum of offline matrices being full-rank. This is a unique step which does not appear in protocols based on "one-more" assumptions including FROST and EKT, as their honest party simulator can respond to signing queries with the help of an oracle even if the sum is malformed. 4) *Security model:* It is worth mentioning that Ringtail has a few drawbacks compared to EKT. As in tRaccoon, we also rely on MACs to implement authenticated channels, while EKT overcame this limitation. Our security proof also assumes that offline signing oracles are aware of the signing coalition (denoted by $\mathcal{T} \subseteq [\ell]$), while in EKT, the offline phase can be agnostic of the coalition, and $\mathcal{T}$ can be determined online upon receiving a message. Removing these constraints while retaining a proof from standard assumptions is an interesting direction for future work.

2. Preliminaries

Let $\lambda \in \mathbb{N}$ be the security parameter. For $m, n \in \mathbb{N}$, write $[m] := \{1, 2, \ldots, m\}$, $[n, m] := \{n, n+1, \ldots, m\}$, $\mathbb{Z}_m$ for the ring of integers modulo m with representatives taken from $[-m/2, m/2) \cap \mathbb{Z}$, and $\mathbb{Z}_m^\times$ its unit group. For $x \in \mathbb{Z}_m$, $|x|$ is defined by the absolute value of its representative in $[-m/2, m/2) \cap \mathbb{Z}$. For a vector $\mathbf{x} \in \mathbb{R}^m$, we write $\|\mathbf{x}\| := \|\mathbf{x}\|_\infty$ for the ℓ_∞-norm, and $\|\mathbf{x}\|_2$ for the ℓ_2-norm.

For a distribution $\mathcal{D}$, we write $x \leftarrow \mathcal{D}$ to denote that x was sampled from $\mathcal{D}$. When sampling uniformly at random from a set S, we use the shorthand $x \overset{\$}{\leftarrow} S$. We write $\mathcal{D} \approx_c \mathcal{D}'$ (resp. $\mathcal{D} \approx_s \mathcal{D}'$) to denote the two distributions are computationally (resp. statistically) indistinguishable.

For every matrix $\mathbf{M} \in \mathbb{R}^{k \times \ell}$ we can find a singular value decomposition $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{k \times k}$ and $\mathbf{V} \in \mathbb{R}^{\ell \times \ell}$ are orthogonal matrices, i.e., $\mathbf{U}\mathbf{U}^T = \mathbf{I}_k$ and $\mathbf{V}\mathbf{V}^T = \mathbf{I}_\ell$, and $\mathbf{D} \in \mathbb{R}^{k \times \ell}$ is an upper diagonal matrix. The entries on the diagonal of $\mathbf{D}$ are called the singular values of $\mathbf{M}$. We denote by $s_{\max}(\mathbf{M})$ the largest singular value of $\mathbf{M}$, and by $s_{\min}(\mathbf{M})$ its smallest singular value.

2.1. Lattices and Discrete Gaussians

For $\mathbf{x} \in \mathbb{R}^m$ and $\sigma > 0$, define the Gaussian function with parameter σ as $\rho_\sigma(\mathbf{x}) := \exp(-\pi \|\mathbf{x}\|_2^2 / \sigma^2)$. For a lattice $\Lambda \subseteq \mathbb{R}^m$ and offset $\mathbf{d} \in \mathbb{R}^m$, define $\rho_\sigma(\Lambda + \mathbf{d}) := \sum_{\mathbf{x} \in \Lambda} \rho_\sigma(\mathbf{x} + \mathbf{d})$. The discrete Gaussian distribution with parameter σ over the lattice coset $\Lambda + \mathbf{d}$ is defined as $\mathcal{D}_{\Lambda+\mathbf{d},\sigma}(\mathbf{x}) := \rho_\sigma(\mathbf{x}) / \rho_\sigma(\Lambda + \mathbf{d})$.

Lemma 2.1 ([Lyu12, Lemma 4.4(1,3)])**.** *Let $\Lambda \subseteq \mathbb{R}^m$ be a lattice, $\sigma > 0$. It holds that*

1) *For any $k > 0$, $\Pr\left[\|z\|_\infty > k\sigma : z \overset{\$}{\leftarrow} \mathcal{D}_{\mathbb{Z},\sigma}\right] \leq 2\exp(-k^2/2)$.*
2) *For any $k > 1$, $\Pr\left[\|\mathcal{D}_{\Lambda,\sigma}\|_2 > k\sigma\sqrt{m}\right] \leq k^m \exp(m/2(1-k^2))$.*

The ε-smoothing parameter of a lattice $\Lambda \subset \mathbb{R}^m$, denoted by $\eta_\varepsilon(\Lambda)$, is defined as

$$\eta_\varepsilon(\Lambda) := \inf s > 0 : \rho_{1/s}(\Lambda^* \setminus \{0\}) \leq \varepsilon$$

where Λ^* denotes the dual lattice of Λ.

Lemma 2.2 ([GPV08, Corollary 2.8 of [GPV07]])**.** *Let $\Psi \subset \Lambda \subset \mathbb{R}^m$ be lattices with $\mathsf{Span}(\Psi) = \mathsf{Span}(\Lambda) = \mathbb{R}^m$. For any $\varepsilon \in (0, 1/2)$, any $s \geq \eta_\varepsilon(\Psi)$, and any centre $\mathbf{c} \in \mathbb{R}^m$, the distribution of $(\mathcal{D}_{\Lambda,s,\mathbf{c}} \bmod \Psi)$ is within statistical distance 2ε from the uniform distribution over $(\Lambda \bmod \Psi)$.*

2.2. Module Lattices

Let $\zeta = \zeta_\mathfrak{f} \in \mathbb{C}$ denote any fixed primitive $\mathfrak{f}$-th root of unity where $\mathfrak{f}$ is a power of 2, $\mathcal{K} = \mathbb{Q}(\zeta)$ the cyclotomic field of conductor $\mathfrak{f}$ and degree $\varphi = \varphi(\mathfrak{f}) = \mathfrak{f}/2$, and $\mathcal{R} = \mathbb{Z}[\zeta] \cong \mathbb{Z}[X]/\langle \Phi_\mathfrak{f}(X) \rangle$ its ring of integers, also called a cyclotomic ring, where $\Phi_\mathfrak{f}(X) = X^\varphi + 1$ is the $\mathfrak{f}$-th cyclotomic polynomial.

An element $x \in \mathcal{K}$ (resp. $\mathcal{R}$) is represented as a linear combination of the power basis, i.e. $x = \sum_{j=0}^{\varphi-1} x_j \zeta^i$ where $x_j \in \mathbb{Q}$ (resp. $\mathbb{Z}$). The vector $\mathsf{coeff}(x) := (x_j)_{j=0}^{\varphi-1}$ is called the coefficient embedding of x. We write $\|x\|_p :=$

$\|\mathsf{coeff}(x)\|$ for the ℓ_p norm of the coefficient embedding of x. The notation extends naturally to vectors $\mathbf{x} = (x_i)_{i=1}^m \in \mathcal{K}^m$, where we write $\mathsf{coeff}(\mathbf{x})$ for the concatenation of $\mathsf{coeff}(x_i)$ for $i \in [m]$, and $\|\mathbf{x}\|_p := \|\mathsf{coeff}(\mathbf{x})\|_p$.

The ring expansion factor $\gamma = \gamma_{\mathcal{R}} := \max_{a,b\in\mathcal{R}} \|a \cdot b\| / (\|a\| \cdot \|b\|)$ measures the norm growth (with respect to a choice of norm $\|\cdot\|$) when multiplying $\mathcal{R}$ elements. For power-2 cyclotomic rings $\mathcal{R}$ of degree φ, it is known (e.g. [AL21, Prop. 2]) that the expansion factor for the ℓ_∞ norm is at most φ.

When $\mathfrak{f}$ is a power of 2, the spaces $\mathcal{K}^m$ and $\mathbb{R}^{\varphi m}$ are isomorphic as inner-product spaces via the coefficient embedding $\mathsf{coeff}\cdot$. The module $\mathcal{R}^m$ can thus be viewed as a lattice. With $\mathcal{D}_\sigma^m$, we refer to the discrete Gaussian distribution with parameter σ over $\mathsf{coeff}(\mathcal{R}^m)$. For $\mathbf{A} \in \mathcal{R}_q^{n\times m}$ and $\mathbf{w} \in \mathcal{R}_q^n$, we define the m-dimensional q-ary module lattice $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{x} \in \mathcal{R}_q^m : \mathbf{A}\mathbf{x} = \mathbf{0} \bmod q\}$ and its coset $\Lambda_q^{\mathbf{w}}(\mathbf{A}) = \{\mathbf{x} \in \mathcal{R}_q^m : \mathbf{A}\mathbf{x} = \mathbf{w} \bmod q\}$.

2.3. Shamir Secret Sharing

For prime q, we define the (t, ℓ)-Shamir secret sharing scheme over $\mathcal{R}_q^n$ with evaluation points given by an arbitrary subset $\{\alpha_1, \ldots, \alpha_\ell\} \subseteq \{1, \ldots, q-1\}$. On input a secret $\mathbf{s} \in \mathcal{R}_q^n$, the $\mathsf{Share} = \mathsf{Share}_{\mathcal{R}_q,n,t,\ell}$ algorithm samples $\mathbf{r}_1, \ldots, \mathbf{r}_{t-1} \xleftarrow{\$} \mathcal{R}_q^n$ and outputs the shares $(\mathbf{s}_i)_{i=1}^\ell$, where

$$\mathbf{s}_i = \mathbf{s} + \mathbf{r}_1 \cdot \alpha_i + \ldots + \mathbf{r}_{t-1} \cdot \alpha_i^{t-1} \bmod q.$$

Given any t-subset $\mathcal{T} \subseteq [\ell]$, and shares $(\mathbf{s}_i)_{i\in\mathcal{T}}$, one can recover the secret $\mathbf{s}$ via the linear combination

$$\mathbf{s} = \sum_{i\in\mathcal{T}} \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \bmod q$$

where $\lambda_{\mathcal{T},i} \in \mathcal{R}_q$ are Lagrange coefficients $\lambda_{\mathcal{T},i} := \prod_{j\in\mathcal{T}:j\neq i} \frac{\alpha_j}{\alpha_j - \alpha_i} \bmod q$. The Shamir secret sharing scheme is secure in the following sense: Fix any distribution $\mathcal{D}$ over $\mathcal{R}_q$ and let $\mathbf{s} \leftarrow \mathcal{D}$ and $(\mathbf{s}_i)_{i\in[\ell]} \leftarrow \mathsf{Share}(\mathbf{s}, q, t, \ell)$. Then, for any subset $I \subset [\ell]$ with $|I| < t$, $(\mathbf{s}_i)_{i\in I}$ contains no information about $\mathbf{s}$.

2.4. Rounding

The following material is taken almost verbatim from Raccoon [PEK+23] and tRaccoon [PKM+24]. Let $\nu \in \mathbb{N} \setminus \{0\}$. Any integer $x \in \mathbb{Z}$ can be uniquely decomposed as

$$x = 2^\nu \cdot x_{\mathsf{hi}} + x_{\mathsf{lo}}, (x_{\mathsf{hi}}, x_{\mathsf{lo}}) \in \mathbb{Z} \times ([-2^{\nu-1}, 2^{\nu-1} - 1] \cap \mathbb{Z})$$

which consists essentially in separating the lower-order bits from the higher-order ones, that is, it drops ν lower bits. We define the function

$$\lfloor\cdot\rceil_\nu : \mathbb{Z} \to \mathbb{Z} \text{ s.t. } \lfloor x\rceil_\nu = \lfloor x/2^\nu\rceil = x_{\mathsf{hi}}$$

where $\lfloor\cdot\rceil : \mathbb{R} \to \mathbb{Z}$ denotes the rounding operator. More precisely, the "rounding half-up" method $\lfloor x\rceil = \lfloor x + 1/2\rfloor$ half-way values are rounded up: e.g. $\lfloor 2.5\rceil = 3$ and $\lfloor -2.5\rceil = -2$. With a slight overload of notation, when $q > 2^\nu$, we extend $\lfloor\cdot\rceil_\nu$ to take inputs in $\mathbb{Z}_q$, in which case, we assume the output is an element in $\mathbb{Z}_{q_\nu}$ where $q_\nu = \lfloor q/2^\nu\rfloor$. Formally, we define:

$$\lfloor\cdot\rceil_\nu : \mathbb{Z}_q \to \mathbb{Z}_{q_\nu} \text{ s.t. } \lfloor x\rceil_\nu = \lfloor x/2^\nu\rceil \bmod q_\nu = x_{\mathsf{hi}} \bmod q_\nu$$

where $x \in \mathbb{Z}_q$ is assumed to have the unsigned representative, i.e. $x \in \{0, 1, \ldots, q-1\}$.

The function $\lfloor\cdot\rceil_\nu$ naturally extends to vectors coefficient-wise. We recall the useful lemma that bounds the norm of difference of rounded vectors [PKM+24], [EKT24].

Lemma 2.3. *Let $\nu, q \in \mathbb{N}$ such that $q > 2^\nu$, $\nu \geq 4$, and set $q_\nu = \lfloor q/2^\nu\rfloor$. Moreover, assume q and ν satisfy $q_\nu = \lfloor q/2^\nu\rceil$, that is, q can be decomposed as $q = 2^\nu \cdot q_\nu + q_{\mathsf{lo}}$ for $q_{\mathsf{lo}} \in [0, 2^{\nu-1} - 1]$. Then, for any $x \in \mathbb{Z}_q$, we have*

$$|x - 2^\nu \cdot \lfloor x\rceil_\nu| \leq 2^\nu - 1.$$

Moreover, for any $\mathbf{x}, \boldsymbol{\delta} \in \mathbb{Z}_q^m$, we have

$$\begin{aligned}&\|2^\nu \cdot (\lfloor \mathbf{x} + \boldsymbol{\delta}\rceil_\nu - \lfloor \mathbf{x}\rceil_\nu \bmod q_\nu) \bmod q\| \\ &\leq \|2^\nu \cdot \lfloor\boldsymbol{\delta}\rceil_\nu \bmod q\| + 2^\nu \cdot \|\mathbf{1}\|.\end{aligned}$$

2.5. Computational Assumptions

We recall a set of computational assumptions required by the security proof for our construction. All of the lattice assumptions are stated for modules, so we drop the prefix "module" from the name of each assumption for brevity's sake. We recall the (ring/module) LWE assumption defined over $\mathcal{R}$ with noise distribution χ.

Definition 2.4 (LWE). *Let $\mathcal{R}, m, n, q, \chi$ be parametrised by λ, where χ is a distribution over $\mathcal{R}$. The $\mathsf{dLWE}_{\mathcal{R},q,m,n,\chi}$ assumption states that for all PPT algorithm $\mathcal{A}$, the following probability is negligible in λ:*

$$\begin{aligned}&\mathbf{Adv}^{\mathsf{dLWE}}_{\mathcal{R},q,m,n,\chi}(\lambda) \\ :=&\left|\Pr\left[b = 1 : \begin{array}{l}\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n}; \mathbf{s} \xleftarrow{\$} \mathcal{R}_q^n; \mathbf{e} \leftarrow \chi^m; \\ \mathbf{y} := \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y})\end{array}\right]\right. \\ &\left. - \Pr\left[b = 1 : \begin{array}{l}\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n}; \\ \mathbf{y} \xleftarrow{\$} \mathcal{R}_q^m \bmod q; b \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{y})\end{array}\right]\right|.\end{aligned}$$

If $\chi = \mathcal{D}_\sigma$ for a Gaussian parameter σ, we use the notation $\mathsf{dLWE}_{\mathcal{R},q,m,n,\sigma}$. Moreover, if in the above game each entry of the secret $\mathbf{s}$ is also sampled from $\chi = \mathcal{D}_\sigma$ instead of uniform distribution, then we call this variant the $\mathsf{dLWE}'_{\mathcal{R},q,m,n,\sigma}$ assumption.

We recall a variant of the LWE problem, which allows an adversary to learn partial leakages of LWE secret and error vectors. This problem was introduced in [DKL+23], [KLSS23].

Definition 2.5 (Hint-LWE). *Let $\mathcal{R}, m, n, k, q, \chi, \bar{\chi}, \mathcal{L}$ be parametrised by λ, where χ, $\bar{\chi}$, $\mathcal{L}$ are distributions over $\mathcal{R}$. The $\mathsf{hLWE}_{\mathcal{R},q,m,n,k,\chi,\bar{\chi},\mathcal{L}}$ assumption states that for all*

PPT adversaries $\mathcal{A}$, the following probability is negligible in λ: $\mathbf{Adv}^{\mathsf{hLWE}}_{\mathcal{R},q,m,n,k,\chi,\bar{\chi},\mathcal{L}}(\lambda) :=$

$$\left| \Pr\left[b = 1 : \begin{array}{c} \mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n};\ \mathbf{s} \xleftarrow{\$} \chi^n;\ \mathbf{e} \leftarrow \chi^m; \\ \forall\ i \in [1,k] : \bar{\mathbf{s}}_i \leftarrow \bar{\chi}^n; \bar{\mathbf{e}}_i \leftarrow \bar{\chi}^m; c_i \leftarrow \mathcal{L} \\ \mathbf{l} := \left\{ c_i \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix} + \begin{pmatrix} \bar{\mathbf{r}}_i \\ \bar{\mathbf{e}}_i \end{pmatrix} \right\}_{i=1}^{k}; \\ \mathbf{c} := \{c_i\}_{i=1}^k \\ \mathbf{y} := \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q;\ b \leftarrow \mathcal{A}(\mathbf{A},\mathbf{y},\mathbf{l},\mathbf{c}) \end{array} \right] - \Pr\left[b = 1 : \begin{array}{c} \mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n};\ \mathbf{s} \xleftarrow{\$} \chi^n;\ \mathbf{e} \leftarrow \chi^m; \\ \forall\ i \in [1,k] : \bar{\mathbf{s}}_i \leftarrow \bar{\chi}^n; \bar{\mathbf{e}}_i \leftarrow \bar{\chi}^m; c_i \leftarrow \mathcal{L} \\ \mathbf{l} := \left\{ c_i \cdot \begin{pmatrix} \mathbf{s} \\ \mathbf{e} \end{pmatrix} + \begin{pmatrix} \bar{\mathbf{s}}_i \\ \bar{\mathbf{e}}_i \end{pmatrix} \right\}_{i=1}^{k}; \\ \mathbf{c} := \{c_i\}_{i=1}^k \\ \mathbf{y} \xleftarrow{\$} \mathcal{R}_q^m;\ b \leftarrow \mathcal{A}(\mathbf{A},\mathbf{y},\mathbf{l},\mathbf{c}) \end{array} \right] \right|.$$

It is shown in [KLSS23] that, for the parameters that we use in this work, this problem is as hard as the standard LWE.

Theorem 2.6 (Hardness of Hint LWE [KLSS23, Theorem 1])**.** *Let m, n, q, k be positive integers and $\mathcal{L}$ be a distribution over $\mathcal{R}$. Let $B > 0$ be a real number which satisfies $\sum_{i=1}^{k} \|c_i\|_1^2 \leq B$ for any possible $(c_1, \ldots, c_k)$ sampled from $\mathcal{L}$. For $\sigma_1, \sigma_2 > 0$, let $\sigma > 0$ be a real number defined as $\frac{1}{\sigma^2} = 2(\frac{1}{\sigma_1^2} + \frac{B}{\sigma_2^2})$. If $\sigma \geq \sqrt{2} \cdot \eta_\varepsilon(\mathbb{Z}^\varphi)$ for $0 < \varepsilon \leq 1/2$, then there exists an efficient reduction from $\mathsf{dLWE}'_{\mathcal{R},q,m,n,\sigma}$ to $\mathsf{hLWE}_{\mathcal{R},q,m,n,k,\sigma_1,\sigma_2,\mathcal{L}}$ that reduces the advantage by at most $(m+n) \cdot 2\varepsilon$.*

We recall the definition of the self-target SIS (stSIS) problem, which is equivalent to SIS (which itself is implied by LWE) in the classical random oracle model (ROM) via the forking lemma [KLS18], [PKM+24, Lemma B.1]. Recently, Jackson et al. analyzed the hardness of stSIS in the quantum ROM and showed a reduction from LWE [JMW24]. The concrete hardness of SelfTargetSIS is analyzed in [PKM+24, Section 8.1].

Definition 2.7 (SelfTargetSIS)**.** *Let m, n, q be integers and $\beta > 0$ be a real number. Let C be a subset of $\mathcal{R}_q$ and let $\mathsf{G} : \mathcal{R}_q^m \times \{0,1\}^* \to C$ be a cryptographic hash function modeled as a random oracle. The $\mathsf{stSIS}_{\mathcal{R},q,m,n,C,\beta}$ problem states that for all PPT adversaries $\mathcal{A}$, the following probability is negligible in λ*

$$\mathbf{Adv}^{\mathsf{stSIS}}_{\mathcal{R},q,m,n,C,\beta}(\lambda) := \Pr\left[\begin{array}{l} \|\mathbf{y}\|_2 \leq \beta \wedge \\ \mathsf{G}([\mathbf{A} \mid \mathbf{I}_m] \cdot \mathbf{y}, \mathsf{msg}) = c \end{array} : \begin{array}{c} \mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n}; \\ (\mathsf{msg}, \mathbf{y}) \leftarrow \mathcal{A}^{\mathsf{G}}(\mathbf{A}); \\ \begin{pmatrix} c \\ \mathbf{z} \end{pmatrix} := \mathbf{y} \end{array} \right].$$

2.6. Trapdoors for Module Lattices

For a positive integer q, b, m, $k = \lceil \log_b q \rceil$, let $\mathbf{g}^t = [1, b, b^2, \ldots, b^{k-1}]$ a gadget vector and $\mathbf{G} = \mathbf{I} \otimes \mathbf{g}^t \in \mathcal{R}^{m \times mk}$ a gadget matrix. A module-based and computational instantiation of [MP12] gives rise to the following trapdoor generation and preimage sampling algorithms. The parameter constraints can be found e.g. in [BEP+21].

Theorem 2.8. *Let b, m be positive integers, φ a power of two, q a prime modulus, $k = \lceil \log_b q \rceil$ and $\bar{d} = 2m + mk$. Let $\sigma_g > \sqrt{2b} \cdot (2b+1) \cdot \sqrt{\log(2k(1+1/\varepsilon))/\pi}$, $\sigma_{\mathsf{td}} > \eta_\varepsilon(\mathbb{Z}^{m\varphi})$ Gaussian parameters. There exist PPT algorithms* $(\mathsf{GenTrap}, \mathsf{SamplePre})$ *satisfying the following:*

$\underline{\mathsf{GenTrap}(\mathcal{R}_q, m)}$ *takes as input parameters $\mathcal{R}_q$, m and outputs $(\mathbf{D}, \mathbf{T}) \in \mathcal{R}_q^{m \times \bar{d}} \times \mathcal{R}^{2m \times mk}$ such that*

$$\mathbf{D} \begin{pmatrix} \mathbf{T} \\ \mathbf{I} \end{pmatrix} = \mathbf{G} \bmod q.$$

where $\mathbf{T}$ is sampled from $\mathcal{D}_{\mathcal{R}^{2m\times mk}, \sigma_{\mathsf{td}}}$

$\underline{\mathsf{SamplePre}(\mathbf{D}, \mathbf{T}, \mathbf{w}, \sigma_u)}$ *takes as input $(\mathbf{D}, \mathbf{T}) \in \mathcal{R}_q^{m \times \bar{d}} \times \mathcal{R}^{2m \times mk}$ generated by* $\mathsf{GenTrap}$*, a target vector $\mathbf{w} \in \mathcal{R}^m$ and Gaussian parameter*

$$\sigma_u > \sqrt{(\sigma_g^2 + 1) s^2_{\mathsf{max}}(\mathbf{T}) + \eta^2_\varepsilon(\mathbb{Z}^{\bar{d}\varphi})},$$

and outputs $\mathbf{u} \in \mathcal{R}^{\bar{d}}$ such that $\mathbf{D}\mathbf{u} = \mathbf{w} \bmod q$.

The distribution $\{\mathbf{D} : (\mathbf{D}, \mathbf{T}) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)\}$ is indistinguishable from the uniform distribution over full-rank matrices in $\mathcal{R}_q^{m \times \bar{d}}$ under the $\mathsf{dLWE}'_{\mathcal{R},m,m,q,\sigma_{\mathsf{td}}}$ assumption (Definition 2.4).

The following distributions are statistically close:

$$\left\{ (\mathbf{D}, \mathbf{u}, \mathbf{w}) : \begin{array}{c} (\mathbf{D}, \mathbf{T}) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m) \\ \mathbf{u} \leftarrow \mathcal{D}_{\mathcal{R}^{\bar{d}}, \sigma_u}; \mathbf{w} = \mathbf{D}\mathbf{u} \bmod q \end{array} \right\},$$

$$\left\{ (\mathbf{D}, \mathbf{u}, \mathbf{w}) : \begin{array}{c} (\mathbf{D}, \mathbf{T}) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m) \\ \mathbf{w} \xleftarrow{\$} \mathcal{R}_q^m; \mathbf{u} \leftarrow \mathsf{SamplePre}(\mathbf{D}, \mathbf{T}, \mathbf{w}, \sigma_u) \end{array} \right\}.$$

Remark 1. *To obtain a concrete lower bound on σ_u, one needs to determine an upper bound on $s^2_{\mathsf{max}}(\mathbf{T})$. In [BEP+21, A.4], the authors experimentally find $s^2_{\mathsf{max}}(\mathbf{T}) < 1.1\sigma_{\mathsf{td}}(\sqrt{2m\varphi} + \sqrt{mk\varphi} + 4.7)$ with high probability.*

2.7. Pseudorandom Functions

Definition 2.9 (Pseudorandom Function ensemble (PRF), from [Gol01])**.** *An (efficiently computable) PRF ensemble is an ensemble of polynomial-time computable functions $\{\mathsf{PRF}_\lambda : \{0,1\}^l \times \{0,1\}^x \to \{0,1\}^y\}_{\lambda \in \mathbb{N}}$, each takes as input a seed* sd *of length $l = l(\lambda)$ and a bit string of length $x = x(\lambda)$, and returns a bit string of length $y = y(\lambda)$ such that the following holds:*

Pseudorandomness: *For all* PPT *adversaries* $\mathcal{A}$, *let* $\mathbf{Adv}^{\mathsf{PRF}}(\lambda)$ *be its advantage in distinguishing* PRF *from any random functions* $\mathsf{F}_\lambda : \{0,1\}^x \to \{0,1\}^y$, *that is*

$$\mathbf{Adv}^{\mathsf{PRF}}(\lambda) := \mid \Pr[1 \leftarrow \mathcal{A}^{\mathsf{PRF}_\lambda(\mathsf{sd},\cdot)}(1^\lambda) \ : \ \mathsf{sd} \xleftarrow{\$} \{0,1\}^l] - \Pr[1 \leftarrow \mathcal{A}^{\mathsf{F}_\lambda(\cdot)}(1^\lambda) \ : \ \mathsf{F}_\lambda \xleftarrow{\$} \mathcal{F}_\lambda] \mid$$

where $\mathcal{F}_\lambda := \{\mathsf{F}_\lambda\}$. *We say that the ensemble* $\{\mathsf{PRF}_\lambda\}$ *is pseudorandom if* $\mathbf{Adv}^{\mathsf{PRF}}(\lambda)$ *is negligible in* λ.

To keep the notation lean, in the paper we will drop the subscript and denote a PRF simply as PRF.

2.8. Raccoon Signature Scheme

We recall the signature scheme Raccoon = (Setup, Gen, Sign, Ver), since it is going to be the base of our threshold scheme. Raccoon follows the standard Fiat-Shamir paradigm applied to a three-move identification protocol. On a very high-level, the underlying identification can be viewed as "LWE-based Schnorr" where prover and verifer hold the public key $\mathbf{b} = \mathbf{As} + \mathbf{e}$ and prover has the corresponding secret key $\mathbf{s}$, respectively. An identification prover first commits to ephemeral randomness $\mathbf{r}^*$ by sending $\mathbf{h} = \mathbf{Ar}^* + \mathbf{e}^* \bmod q$ to verifier. Upon receiving challenge c with small coefficients, prover responds with $\mathbf{z} = \mathbf{s}c + \mathbf{r}^*$, which is accepted by verifier if 1) $\mathbf{Az} - c\mathbf{b} \approx \mathbf{h} \bmod q$ and 2) $\mathbf{z}$ has small norm. Here, the verification condition 1) only holds approximately due to small error terms. To implement the approximate check, one can discard low-order bits using the rounding operation introduced in Section 2.4 and have prover additionally send "hint" information $\mathbf{\Delta}$ fixing approximation error. To further reduce the public key size, Raccoon also applies the same rounding operation to $\mathbf{b}$.

In more detail, the scheme is parametrised by a ring $\mathcal{R}$, dimensions $n, m \in \mathbb{N}$, modulus q, Gaussian parameters $\sigma_e, \sigma^* > 0$, a hash function $\mathsf{H}_c : \{0,1\}^* \to C$, which is modelled as a random oracle with outputs sampled from the *challenge set* $C = \left\{c \in \mathcal{R} : \|c\|_\infty = 1 \wedge \|c\|_1 = \kappa\right\}$, and rounding parameters $\nu, \xi > 0$, all dependent on λ. A public key consists of a rounded LWE sample

$$\tilde{\mathbf{b}} = \lfloor \mathbf{As} + \mathbf{e} \bmod q \rceil_\xi$$

and to sign a message, one computes another LWE sample $\mathbf{h} = \mathbf{Ar}^* + \mathbf{e}^* \bmod q$ and hashes a rounded version to compute the challenge c as

$$c = \mathsf{H}_c\left(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu\right).$$

The signature is then computed as $\mathbf{z} = \mathbf{s}c + \mathbf{r}^*$. In addition, the signer also includes a hint $\mathbf{\Delta}$ that allows the verifier to check the validity of the signature, by checking that

$$c = \mathsf{H}_c\left(\mathsf{pp}, \mathsf{pk}, \left\lfloor \mathbf{Az} - 2^\xi \cdot \tilde{\mathbf{b}} \cdot c \right\rceil_\nu + \mathbf{\Delta} \bmod q_\nu, \mu\right)$$

and that the norm of $\mathbf{z}$ and $\mathbf{\Delta}$ is small. We present a formal description of the scheme below.

Raccoon.Setup(λ): Generate a uniformly random matrix $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m \times n}$. Output $\mathsf{pp} = \mathbf{A}$.

Raccoon.Gen(pp): Sample $\mathbf{s} \leftarrow \mathcal{D}_{\sigma_e}^n$ and $\mathbf{e} \leftarrow \mathcal{D}_{\sigma_e}^m$. Compute $\tilde{\mathbf{b}} = \lfloor \mathbf{As} + \mathbf{e} \bmod q \rceil_\xi$ and output $(\mathsf{pk}, \mathsf{sk}) = (\tilde{\mathbf{b}}, \mathbf{s})$.

Raccoon.Sign(sk, μ): Perform the following steps.
- $\mathbf{r}^* \leftarrow \mathcal{D}_{\sigma^*}^n$
- $\mathbf{e}^* \leftarrow \mathcal{D}_{\sigma^*}^m$
- $\mathbf{h} = \mathbf{Ar}^* + \mathbf{e}^* \bmod q$.
- $c = \mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$.
- $\mathbf{z} = \mathbf{s}c + \mathbf{r}^*$.
- $\mathbf{\Delta} = \lfloor \mathbf{h} \rceil_\nu - \left\lfloor \mathbf{Az} - 2^\xi \cdot \tilde{\mathbf{b}} \cdot c \right\rceil_\nu \bmod q_\nu$.
- Output $\sigma = (c, \mathbf{z}, \mathbf{\Delta})$.

Raccoon.Ver(pk, μ, σ): Check that

$$c = \mathsf{H}_c\left(\mathsf{pp}, \mathsf{pk}, \left\lfloor \mathbf{Az} - 2^\xi \cdot \tilde{\mathbf{b}} \cdot c \right\rceil_\nu + \mathbf{\Delta} \bmod q_\nu, \mu\right)$$

and $\|(\mathbf{z}, 2^\nu \cdot \mathbf{\Delta})\|_2 \le B_2$.

2.9. Threshold Signatures

We provide the syntax for (t, ℓ)-threshold signatures. Our syntax is tailored to *two-round* interactive threshold signatures, whose first round (i.e., offline phase) is independent of the message to be signed. We use the convention that t is the number of parties required to sign, and allow the adversary to corrupt up to $t-1$ parties.

Definition 2.10 (Threshold Signatures). *A* (t,ℓ)*-threshold signature scheme consists of a tuple of PPT (interactive) algorithms* $\mathsf{TS} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Combine}, \mathsf{Ver})$ *with the following syntax:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$: *on input the security parameter* λ, *it returns the public parameters* pp *(which are given implicitly as an input to all the other algorithms).*

$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[\ell]}) \leftarrow \mathsf{Gen}(1^\ell, 1^t)$: *on input the number of parties* ℓ *and the threshold* t, *return the verification key* pk *(which is given implicitly as an input to all the other algorithms except* Setup*), and the secret keys of all parties.*

$\mathsf{Sign}_1, \mathsf{Sign}_2$: *These are the algorithm run by each party in a signing set* $\mathcal{T} \subseteq [\ell]$, *where* $|\mathcal{T}| \ge t$, *and each constitutes a stage of the 2-round interactive signing protocol, of which the first can be seen as a preprocessing phase:*

$$(\rho_i, \mathsf{st}_i) \leftarrow \mathsf{Sign}_1(\mathsf{sk}_i),$$
$$\sigma_i \leftarrow \mathsf{Sign}_2(\{\rho_j\}_{j\in\mathcal{T}\setminus\{i\}}, \mu, \mathsf{st}_i).$$

All parties are assumed to take as common input a message μ *(for the second round). The output* ρ_i *of the first round is broadcast to all parties in* $\mathcal{T}$.

$\sigma \leftarrow \mathsf{Combine}(\mu, \mathcal{T}, \{\rho_i, \sigma_i\}_{i\in\mathcal{T}})$: *On input a message* μ, *the set of signers* $\mathcal{T}$, *and partial signatures* ρ_i, σ_i, *it returns the aggregated signature* σ.

$b \leftarrow \mathsf{Ver}(\mathsf{pk}, \mu, \sigma)$: *on input the verification key* pk, *a message* μ *and a signature* σ *it returns a bit* $b \in \{0,1\}$.

Correctness of a threshold signature is trivially defined.

Definition 2.11 (Correctness of TS)**.** *A threshold signature scheme* TS *is correct if for all* $\lambda \in \mathbb{N}$ *there exists a negligible function* negl *such that for all allowable* $1 \le t \le \ell$*, for all* $\mathcal{T} \subseteq [\ell]$ *with* $|\mathcal{T}| \ge t$*, and for all messages* $\mu \in \{0,1\}^*$*, the following probability is overwhelming in* λ

$$\Pr\left[b = 1 : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[\ell]}) \leftarrow \mathsf{Gen}(1^\ell, 1^t) \\ \forall i \in \mathcal{T}\ (\rho_i, \mathsf{st}_i) \leftarrow \mathsf{Sign}_1(\mathsf{sk}_i) \\ \forall i \in \mathcal{T}\ \sigma_i \leftarrow \mathsf{Sign}_2(\{\rho_j\}_{j\in\mathcal{T}\setminus\{i\}}, \mu, \mathsf{st}_i) \\ \sigma \leftarrow \mathsf{Combine}(\mu, \mathcal{T}, \{\rho_i, \sigma_i\}_{i\in T}) \\ b \leftarrow \mathsf{Ver}(\mathsf{pk}, \mu, \sigma) \end{array} \right]$$

We require the threshold signature to be unforgeable against a *malicious* (active) adversary doing *static* corruptions and triggering *concurrent signing* in the *trusted key generation* model with *public aggregation* (i.e., the adversary gets to see the partial signatures of honest parties even when the set of signers $\mathcal{T}$ only includes honest parties) in the *ROM*.

Definition 2.12 (TUF: Unforgeability of TS in the ROM)**.** *A threshold signature* TS *is unforgeable if for all* $\lambda \in \mathbb{N}$*, for all* PPT *two-stage adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *there exists a negligible function* negl *such that*

$$\mathbf{Adv}^{\mathsf{TUF}}_{\mathcal{A},\mathsf{TS}}(\lambda) := \Pr\left[\mathsf{Exp}^{\mathsf{TUF}}_{\mathsf{TS},\mathcal{A}}(1^\lambda) = 1\right] \le \mathsf{negl}(\lambda)$$

where $\mathsf{Exp}^{\mathsf{TUF}}_{\mathsf{TS},\mathcal{A}}$ *is defined in Game 1 and* H *denotes the random oracle.*

Remark 2. *Following the stateless security notion of [CKM23], we introduce an execution ID (*eid*) to keep track of signing sessions. Note that* eid *is local to each honest party and thus is different from a global session ID that all parties must agree upon before the first round. We remark that our definition of unforgeability slightly deviates from the game for "partially non-interactive" threshold signatures such as [BCK+22]. First, the offline signing oracle* $\mathcal{O}_{\mathsf{Sign}_1}$ *takes a signing coalition* $\mathcal{T}$ *as input and* $\mathcal{O}_{\mathsf{Sign}_2}$ *demands consistent* $\mathcal{T}$ *is used for the same* eid*.Second, the online signing oracle refuses to proceed in case there exists some honest party* j *that has not generated its offline token* ρ_j *for the same signing coalition* $\mathcal{T}$*. That is, the oracle is guaranteed to use legitimately generated* ρ_j *instead of adversarially picked ones. Although this seems to be a weak security guarantee, in practice one can implement such checks assuming the existence of* authenticated channels *between honest parties, i.e., if both sender and receiver are honest, an adversary cannot tamper with transcripts between them. Following* tRaccoon *[PKM+24], an authenticated channel can be easily implemented by having the (trusted) key generation output a key pair of a (non-threshold) signature scheme for each signer or pair-wise MAC keys, and by having every party* i *sign the tuple* $(\mathcal{T}, i, \rho_i)$ *after executing the offline algorithm* Sign_1*. In this way, one can assume that in the unforgeability game the adversary cannot send to the signing oracles inputs on behalf of honest parties, as otherwise that would violate the unforgeability of the (single-signer) signature scheme. For simplicity of exposition we do not include this routine in the syntax of our construction, but our program code contains an implementation of MACs.*

Table 1: Parameters for our threshold signature.

λ	Security parameter
$\mathcal{R}$	$\mathfrak{f}$-th cyclotomic ring, $\mathcal{R} = \mathbb{Z}[\zeta] \cong \mathbb{Z}[X]/\langle\Phi_{\mathfrak{f}}(X)\rangle$
φ	degree of $\mathcal{R}$, $\varphi = \varphi(\mathfrak{f})$
q	Ring modulus
κ	ℓ_1-norm of challenge c
ν	Number of low-order bits discarded from $\mathbf{h}$
ξ	Number of low-order bits discarded from $\mathbf{b}$
q_ν	$q_\nu = \lfloor q/2^\nu \rceil$, $\mathbb{Z}_{q_\nu}$ defines a space $\tilde{\mathbf{h}} = \lfloor \mathbf{h} \rceil_\nu$ and $\mathbf{\Delta}$ live in
q_ξ	$q_\xi = \lfloor q/2^\xi \rceil$, $\mathbb{Z}_{q_\xi}$ defines a space $\tilde{\mathbf{b}} = \lfloor \mathbf{b} \rceil_\xi$ lives in
n	Length of secret key
m	Length of the public key
t	Number of parties needed to sign
ℓ	Total number of parties holding key shares
l	Length of a PRF seed
d	Width of $\mathbf{D}_i$
$\bar{d}$	$\bar{d} = d - 1$, Length of $\mathbf{u}$
σ_e	Gaussian parameter of the LWE public key $\mathbf{b}$
σ^*	Gaussian parameter of the LWE commitment $\mathbf{d}_{i,0}$
σ_E	Gaussian parameter of the LWE commitment $\mathbf{D}_{i,1}$
σ_u	Gaussian parameter of the $\mathbf{u}$ sampled during signing
σ_{td}	Gaussian parameter of the trapdoor matrix $\mathbf{T}$ (only used in the security proof)
B_2	ℓ_2-norm bound of a valid signature vector $(\mathbf{z}, 2^\nu \cdot \mathbf{\Delta})$
Q	Max number of signing queries supported by our scheme
Q_c	Max number of hash queries to H_c supported by our scheme
Q_u	Max number of hash queries to H_u supported by our scheme

3. Ringtail Threshold Signature Scheme

In the following, we give an informal and intuitive description of our t-out-of-ℓ threshold-signature scheme Ringtail, whereas we refer the reader to Algorithm 1 for a formal presentation. We present the parameters for the scheme in Table 1. Additionally, the scheme uses two hash functions $\mathsf{H}_u : \{0,1\}^* \to \mathcal{D}^{\bar{d}}_{\sigma_u}$ and $\mathsf{H}_c : \{0,1\}^* \to C$, modeled as random oracles. Here $C \subseteq \mathcal{R}$ is a challenge set consisting of ternary polynomials with Hamming weight κ (as defined in Section 2.8). On the other hand, outputs of H_u are distributed as Gaussians. To implement H_u using a standard hash function, one may first hash the input string into a string ρ, and then use ρ as a randomness source to run an appropriate Gaussian sampling algorithm that samples a module element from $\mathcal{D}^{\bar{d}}_{\sigma_u}$ coefficient wise.

In a threshold signature scheme, the key generation is performed by a trusted party. In the case of Ringtail, this means sampling an LWE key $\mathbf{b} := \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$, where both the secret $\mathbf{s}$ and the noise $\mathbf{e}$ are sampled from discrete Gaussian distributions. The trusted party then shares the secret $\mathbf{s}$ using the standard Shamir secret sharing and distributes the resulting shares $(\mathbf{s}_1, \ldots, \mathbf{s}_\ell)$ to the participants. We shall think of $\mathbf{b}$ as the *joint public key* of the participants and, jumping ahead, a valid signature will be identical to a Raccoon signature under $\mathbf{b}$. The crux of the protocol will

Game 1: Unforgeability experiment with static corruptions for threshold signature

```
$\underline{\mathsf{Exp}^{\mathsf{TUF}}_{\mathsf{TS},\mathcal{A}}(1^\lambda)}$
 1: $S \leftarrow \varnothing$ // open signing sessions
 2: $S' \leftarrow \varnothing$ // closed signing sessions
 3: $\mathcal{M} \leftarrow \varnothing$ // set of signed messages
 4: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
 5: $\mathcal{C} \leftarrow \mathcal{A}_1^{\mathsf{H}}(\mathsf{pp})$
 6: if $(|\mathcal{C}| \geq t) \lor (\mathcal{C} \nsubseteq [\ell])$ then
 7:     return 0
 8: $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[\ell]}) \leftarrow \mathsf{Gen}(1^\ell, 1^t)$
 9: $\mathcal{H} \leftarrow \{i\}_{i\in[\ell]\setminus\mathcal{C}}$ // honest party indices
10: $(\mu^*, \sigma^*) \leftarrow \mathcal{A}_2^{\mathcal{O}_{\mathsf{Sign}_1}, \mathcal{O}_{\mathsf{Sign}_2}, \mathsf{H}}(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in\mathcal{C}})$
11: $b \leftarrow \mathsf{Ver}(\mathsf{pk}, \mu^*, \sigma^*)$
12: if $(b = 1) \land (\mu^* \notin \mathcal{M})$ then
13:     return 1
14: else
15:     return 0
```

```
$\underline{\mathcal{O}_{\mathsf{Sign}_1}(i, \mathcal{T})}$
 1: if $(i \notin \mathcal{H}) \lor (i \notin \mathcal{T}) \lor (|\mathcal{T}| < t) \lor (\mathcal{T} \nsubseteq [\ell])$ then
 2:     return $\perp$
 3: $\mathsf{eid} \xleftarrow{\$} \{0,1\}^*$ // local execution ID
 4: $(\rho_i, \mathsf{st}_i) \leftarrow \mathsf{Sign}_1(\mathsf{sk}_i)$
 5: $S \leftarrow S \cup \{(\mathsf{eid}, i, \mathcal{T}, \rho_i, \mathsf{st}_i)\}$
 6: return $(\mathsf{eid}, \rho_i)$
```

```
$\underline{\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{eid}, i, \{\rho_j\}_{j\in\mathcal{T}\setminus\{i\}}, \mu)}$
 1: if $((\mathsf{eid}, i, \mathcal{T}, \cdot, \cdot) \notin S) \lor ((\mathsf{eid}, i) \in S') \lor (\exists j \in (\mathcal{T} \cap \mathcal{H}) \setminus \{i\} : (\cdot, j, \mathcal{T}, \rho_j, \cdot) \notin S)$ then
 2:     return $\perp$
 3: Load $\mathsf{st}_i$ s.t. $(\mathsf{eid}, i, \mathcal{T}, \cdot, \mathsf{st}_i) \in S$
 4: $\sigma_i \leftarrow \mathsf{Sign}_2(\{\rho_j\}_{j\in\mathcal{T}\setminus\{i\}}, \mu, \mathsf{st}_i)$
 5: $S' \leftarrow S' \cup \{(\mathsf{eid}, i)\}$
 6: $\mathcal{M} \leftarrow \mathcal{M} \cup \{\mu\}$
 7: return $\sigma_i$
```

Algorithm 1: Ringtail

```
$\underline{\mathsf{Setup}(1^\lambda)}$
 1: $\mathsf{pp} := \mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{m\times n}$
 2: return $\mathsf{pp}$
```

```
$\underline{\mathsf{Gen}(1^\ell, 1^t)}$
 1: $\mathbf{s} \xleftarrow{\$} \mathcal{D}_{\sigma_e}^n$
 2: $\mathbf{e} \leftarrow \mathcal{D}_{\sigma_e}^m$
 3: $\mathbf{b} := \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$
 4: $\mathsf{pk} := \tilde{\mathbf{b}} = \lfloor \mathbf{b} \rceil_\xi$
 5: $(\mathbf{s}_1, \ldots, \mathbf{s}_\ell) \leftarrow \mathsf{Share}(\mathbf{s}, q, t, \ell)$
 6: for $i \in [\ell]$ do
 7:     for $j \in [\ell]$ do
 8:         $\mathsf{sd}_{i,j} \xleftarrow{\$} \{0,1\}^l$,
 9: for $i \in [\ell]$ do
10:     $\mathsf{sk}_i := (\mathbf{s}_i, (\mathsf{sd}_{i,j}, \mathsf{sd}_{j,i})_{j\in[\ell]})$
11: return $(\mathsf{pk}, (\mathsf{sk}_i)_{i\in[\ell]})$
```

```
$\underline{\mathsf{Combine}(\mu, \mathcal{T}, (\rho_i, \sigma_j)_{j\in\mathcal{T}})}$
 1: $(\mathbf{D}_j, \mathbf{z}_j)_{j\in\mathcal{T}} := (\rho_j, \sigma_j)_{j\in\mathcal{T}}$
 2: $\mathbf{u} := \mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
 3: $\mathbf{D} := \sum_{j\in\mathcal{T}} \mathbf{D}_j$
 4: $\mathbf{h} := \mathbf{D}\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$
 5: $\tilde{\mathbf{h}} := \lfloor \mathbf{h} \rceil_\nu$
 6: $c := \mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \tilde{\mathbf{h}}, \mu)$
 7: $\mathbf{z} := \sum_{j\in\mathcal{T}} \mathbf{z}_j \bmod q$
 8: $\mathbf{\Delta} := \tilde{\mathbf{h}} - \left\lfloor \mathbf{A}\mathbf{z} - 2^\xi \cdot \tilde{\mathbf{b}} \cdot c \bmod q \right\rceil_\nu \bmod q_\nu$
 9: $\sigma := (c, \mathbf{z}, \mathbf{\Delta})$
10: return $\sigma$
```

```
$\underline{\mathsf{Sign}_1(\mathsf{sk}_i)}$
 1: $(\mathbf{s}_i, (\mathsf{sd}_{i,j}, \mathsf{sd}_{j,i})_{j\in[\ell]}) := \mathsf{sk}_i$
 2: $\mathbf{r}_i^* \leftarrow \mathcal{D}_{\sigma^*}^n$, $\mathbf{e}_i^* \leftarrow \mathcal{D}_{\sigma^*}^m$
 3: $\mathbf{R}_i \leftarrow \mathcal{D}_{\sigma_E}^{n\times\bar{d}}$, $\mathbf{E}_i \leftarrow \mathcal{D}_{\sigma_E}^{m\times\bar{d}}$
 4: $\mathbf{D}_i := [\mathbf{d}_{i,0} \mid \mathbf{D}_{i,1}] = \mathbf{A}[\mathbf{r}_i^* \mid \mathbf{R}_i] + [\mathbf{e}_i^* \mid \mathbf{E}_i] \bmod q \in \mathcal{R}_q^{m\times d}$
 5: $\rho_i := \mathbf{D}_i$
 6: $\mathsf{st}_i := (\mathcal{T}, \mathsf{sk}_i, \mathbf{r}_i^*, \mathbf{R}_i, \mathbf{D}_i)$
 7: return $(\rho_i, \mathsf{st}_i)$
```

```
$\underline{\mathsf{Sign}_2((\rho_j)_{j\in\mathcal{T}\setminus\{i\}}, \mu, \mathsf{st}_i)}$
 1: $(\mathcal{T}, \mathsf{sk}_i, \mathbf{r}_i^*, \mathbf{R}_i^*, \mathbf{D}_i) := \mathsf{st}_i$
 2: for $j \in \mathcal{T} \setminus \{i\}$ do
 3:     $\mathbf{D}_j := \rho_j$
 4: $\mathbf{u} := \mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu) \in \mathcal{R}^{\bar{d}}$
 5: $\mathbf{D} := \sum_{j\in\mathcal{T}} \mathbf{D}_j \in \mathcal{R}_q^{m\times d}$
 6: $[\mathbf{d} \mid \bar{\mathbf{D}}] := \mathbf{D}$
 7: if $\bar{\mathbf{D}} \in \mathcal{R}_q^{m\times\bar{d}}$ is not full-rank then
 8:     abort
 9: $\mathbf{h} := \mathbf{D}\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$
10: $\tilde{\mathbf{h}} := \lfloor \mathbf{h} \rceil_\nu$
11: $c := \mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \tilde{\mathbf{h}}, \mu)$
12: $\mathbf{m}_i \leftarrow \sum_{j\in\mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
13: $\mathbf{m}_i' \leftarrow \sum_{j\in\mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
14: $\mathbf{z}_i := \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \mid \mathbf{R}_i] \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} + \mathbf{m}_i' - \mathbf{m}_i \bmod q$
15: return $\sigma_i = \mathbf{z}_i$
```

$\underline{\mathsf{Ver}(\mathsf{pk}, \mu, \sigma)}$ Identical to Raccoon (see Section 2.8) therefore be in how an authorized set of t parties can jointly compute such signature, which we describe next.

As a straw-man protocol, one may first describe the following simple method to aggregate t Raccoon signatures (Section 2.8) with a shared challenge c: for each party $i \in \mathcal{T}$ with $|\mathcal{T}| = t$, 1) announce $\mathbf{d}_{i,0} = \mathbf{A}\mathbf{r}_i^* + \mathbf{e}_i^*$, 2) compute challenge $c = \mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$ where $\mathbf{h} = \sum_{i\in\mathcal{T}} \mathbf{d}_{i,0}$, 3) announce response $\mathbf{z}_i = \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + \mathbf{r}_i^*$, and 4) output $c, \mathbf{z} = \sum_{i\in\mathcal{T}} \mathbf{z}_i$ and the corresponding hint vector as a finalized signature. As observed in numerous works e.g. [DEF+19], [BLL+21], [DOTT22], [PKM+24], such a straw-man protocol is susceptible to concurrent attacks, because an adversary launching parallel signing sessions is able to craft a forgery by carefully combining partial signatures from different sessions. Taking inspiration from FROST [KG20] and MuSig-L [BTT22], to protect against concurrent attacks, our Ringtail introduces an additional commitment matrix $\mathbf{D}_{i,1} = \mathbf{A}\mathbf{R}_i + \mathbf{E}_i$. Now, we modify the straw-man protocol by hashing $\mathbf{h} = \sum_i (\mathbf{d}_{i,0} + \mathbf{D}_{i,1}\mathbf{u})$, where $\mathbf{u}$ itself is derived by hashing all the commitments and a message μ. Intuitively, transformation by $\mathbf{u}$ makes it harder for an adversary to control the challenge c by adaptively choosing $\mathbf{D}_{i,1}$ and μ after seeing honest parties's contribution.

We now elaborate on each step of the modified protocol. The protocol proceeds with a single broadcast round followed by a single aggregation round. In the first (offline, message-independent) round, each party samples an *imbalanced* LWE sample

$$\mathbf{D}_i \coloneqq \mathbf{A}[\mathbf{r}_i^* \,|\, \mathbf{R}_i] + [\mathbf{e}_i^* \,|\, \mathbf{E}_i] \bmod q$$

where $\mathbf{r}_i^*$ and $\mathbf{e}_i^*$ are sampled from a Gaussian distribution with a much larger standard deviation than $\mathbf{R}_i$ and $\mathbf{E}_i$ (whose purpose will become clear in the proof). Each party then broadcasts $\mathbf{D}_i$. In the second round, the parties from the authorized set $\mathcal{T}$ compute the combined matrix

$$\mathbf{D} \coloneqq \sum_{j\in\mathcal{T}} \mathbf{D}_j \quad \text{and} \quad \mathbf{h} \coloneqq \mathbf{D}\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$$

where $\mathbf{u}$ is computed by hashing the transcript so far. Then each party can derive the sigma protocol challenge c by hashing a *rounded version of* $\mathbf{h}$ (where the rounding is done to discard the noisy low-order bits). Finally, the share of the signature can be computed as

$$\mathbf{z}_i \coloneqq \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \,|\, \mathbf{R}_i] \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$$

and it can be verified that summing up all $\{\mathbf{z}_j\}_{j\in\mathcal{T}}$ one obtains a valid signature under $\mathbf{b}$, as desired. In the actual protocol, each party will add a *masking term* $\mathbf{m}_i' - \mathbf{m}_i$ to the signature share $\mathbf{z}_i$, which will serve the purpose of hiding the partial contribution of each party. Note that the pairwise PRF keys $\mathsf{sd}_{i,j}, \mathsf{sd}_{j,i}$ are generated by Gen in such a way that the sum of corresponding shares $\sum_{j\in\mathcal{T}} \mathbf{m}_j' - \mathbf{m}_j$ will always be $\mathbf{0}$. We take this masking technique verbatim from tRaccoon [PKM+24], [KRT24].

We further require that an aggregated $\mathbf{D} = [\mathbf{d} \,|\, \bar{\mathbf{D}}] \in \mathcal{R}_q^m \times \mathcal{R}_q^{m\times\bar{d}}$ is well-formed (Line 7). This is essentially to guarantee sufficiently high min-entropy of (rounded) $\mathbf{d} + \bar{\mathbf{D}}\mathbf{u} \bmod q$ even if the adversary contributes to $\mathbf{D}$ adaptively after viewing honest parties' contribution.

The verification operation is identical to a non-threshold version of Raccoon (Section 2.8), although the ℓ_2-norm bound B_2 must be set carefully to retain correctness while accommodating a bounded number of active signers t as in tRaccoon. Following [PKM+24], we set $t \leq 1024$ and estimate parameters based on this bound.

Remark 3 (Precomputable operations in Sign_2)**.** *For an optimized implementation, one can preprocess several message-independent operations in* Sign_2 *of Algorithm 1. As soon as* $\mathbf{D}_j$*'s are received from other parties in the offline phase, each signer can pre-compute a hash of* $(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ *and reuse it as input to both* H_u *and* PRF *once a message* μ *to be signed is obtained. It is also possible to preprocess the full rank-ness test of* $\bar{\mathbf{D}}$ *(Line 7).*

3.1. Correctness Analysis

In this section, we describe the choice of the verification bound B_2 ensuring that the scheme of Algorithm 1 is correct with correctness error $\lesssim 2^{-\lambda}$. We also show that the probability that one execution of the protocol fails due to the aborting condition (Line 7 of Algorithm 1) being triggered is very small ($< 2^{-1000}$ for all our parameters).

Correctness. Signature verification passes if (though not necessarily only if) $\lfloor \mathbf{A}\mathbf{z} - \mathbf{b}c \bmod q \rceil_\nu + \mathbf{\Delta}$ is equal to the value $\lfloor \mathbf{h} \rceil_\nu$ computed in the Combine algorithm, and $\left\| (z, 2^\nu \mathbf{\Delta}) \right\|_2 \leq B_2$. The first condition is satisfied by construction for all signatures output by the Combine algorithm, so it suffices to check the second one.

Due to the cancellation of the masking values $\mathbf{m}_j, \mathbf{m}_j'$ and correctness of Shamir secret sharing, the $\mathbf{z}$ element in a signature can be expressed as:

$$\mathbf{z} = \sum_{j\in\mathcal{T}} \mathbf{z}_j \equiv \mathbf{s} \cdot c + \sum_{j\in\mathcal{T}} (\mathbf{r}_j^*, \mathbf{R}_j) \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \pmod q.$$

where $\mathcal{T} \subseteq [\ell]$ is a signing coalition such that $|\mathcal{T}| = t$. In this expression, the dominant term by far is the sum $\bar{\mathbf{r}} \coloneqq \sum_j \mathbf{r}_j^*$, which is distributed as a discrete Gaussian of parameter $\sigma^*\sqrt{t}$. The fact that this term is of much larger magnitude than all the others is necessary for the security of the scheme with respect to the hLWE problem. Furthermore:

$$\begin{aligned}\mathbf{A}\mathbf{z} - \mathbf{b}c &\equiv \mathbf{A}\mathbf{s} \cdot c + \mathbf{A}\sum_{j\in\mathcal{T}} (\mathbf{r}_j^*, \mathbf{R}_j) \cdot \begin{pmatrix}1\\ \mathbf{u}_j\end{pmatrix} - \mathbf{A}\mathbf{s}c - \mathbf{e}c \\ &\equiv \sum_{j\in\mathcal{T}} \left(\mathbf{D}_j - (\mathbf{e}_j^*, \mathbf{E}_j)\right) \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} - \mathbf{e}c \\ &\equiv \mathbf{h} - \left(\sum_{j\in\mathcal{T}} (\mathbf{e}_j^*, \mathbf{E}_j)\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} + \mathbf{e}c\right) \pmod q.\end{aligned}$$

Thus, if we write $\mathbf{b}_\perp = \mathbf{b} - 2^\xi \tilde{\mathbf{b}}$, we get:

$$\mathbf{A}\mathbf{z} - 2^\xi \tilde{\mathbf{b}} c \equiv \mathbf{h} - \left(\sum_{j \in \mathcal{T}} (\mathbf{e}_j^*, \mathbf{E}_j) \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix} + \mathbf{e}c + \mathbf{b}_\perp c \right) \pmod q.$$

Denote the term in braces on the R.H.S. by $\tilde{\mathbf{e}}$. We have:

$$\boldsymbol{\Delta} = \lfloor \mathbf{h} \rceil_\nu - \lfloor \mathbf{h} - \tilde{\mathbf{e}} \rceil_\nu$$

and hence, by Lemma 2.3:

$$\|2^\nu \boldsymbol{\Delta}\|_2 \leq \|2^\nu \lfloor \tilde{\mathbf{e}} \rceil_\nu\|_2 + 2^\nu \|\mathbf{1}\|_2 \leq \|\tilde{\mathbf{e}}\|_2 + 2^{\nu+1} \|\mathbf{1}\|_2 .$$

Now, in the error term $\tilde{\mathbf{e}}$, there is again a dominant term by far, namely $\bar{\mathbf{e}} := \sum_{j \in \mathcal{T}} \mathbf{e}_j^*$. Indeed, it is much larger than the terms $\mathbf{E}_j \mathbf{u}$ and $\mathbf{e}c$ due to the security constraints with respect to hLWE, and much larger than $\mathbf{b}_\perp c$ because ξ is chosen such that $\sigma^* \gg 2^\xi$. Similarly, ν is also chosen in such a way that $\sigma^* \gg 2^\nu$, and therefore, the previous inequality reduces to $\|2^\nu \boldsymbol{\Delta}\|_2 \lesssim \|\bar{\mathbf{e}}\|_2$. Overall, it follows that:

$$\left\| (z, 2^\nu \boldsymbol{\Delta}) \right\|_2 \lesssim \left\| (\bar{\mathbf{r}}, \bar{\mathbf{e}}) \right\|_2 ,$$

and the vector on the right-hand side is distributed as a discrete Gaussian of parameter $\sigma^* \sqrt{t}$. Thus, to ensure that the scheme is correct with correctness error $\lesssim 2^{-\lambda}$, it suffices to choose B_2 as an upper bound on the Euclidean norm of a Gaussian vector of that parameter, except with probability $\leq 2^{-\lambda}$. It suffices to take:

$$B_2 = k\sigma^* \sqrt{t(n+m)\varphi}$$

where k satisfies $k^2 - \log(k^2) \geq 1 + \frac{2\lambda \log 2}{(n+m)\varphi}$. Such a k can be expressed directly in terms of Lambert's W-function:

$$k := \sqrt{W_{-1}\left(1 + \frac{2\lambda \log 2}{(n+m)\varphi}\right)}.$$

Failure probability. The abort condition (Line 7 of Sign_2 in Algorithm 1) is triggered if the matrix $\bar{\mathbf{D}} \in \mathcal{R}_q^{m \times \bar{d}}$ is not full-rank. In an honest verification of the protocol, the matrix $\bar{\mathbf{D}}$ is uniformly random over $\mathcal{R}_q$; therefore, the abort happens with probability at most $\varphi/q^{\bar{d}-m}$ by Corollary 3.2 below. This abort probability will be completely negligible (less than 2^{-1000}) in all of our parameter settings.

Lemma 3.1. *A uniformly random $m \times \bar{d}$ matrix over the field $\mathbb{F}_{q_0}$ with q_0 elements is full-rank except with probability at most $1/q_0^{\bar{d}-m}$.*

Proof. Let $\mathbf{v} \in \mathbb{F}_{q_0}^m$ be an arbitrary non-zero vector. For a uniformly random $\mathbf{M} \in \mathbb{F}_{q_0}^{m \times \bar{d}}$, the vector $\mathbf{v}^t \mathbf{M}$ is uniformly distributed in $\mathbb{F}_{q_0}^{\bar{d}}$, and hence $\Pr[\mathbf{v}^t \mathbf{M} = \mathbf{0}] = 1/q_0^{\bar{d}}$. Now, $\mathbf{M}$ is *not* full-rank if and only if there exists a non-zero $\mathbf{v}$ such that $\mathbf{v}^t \mathbf{M} = \mathbf{0}$. By the union bound, it follows that $\Pr[\mathbf{M} \text{ not full-rank}] \leq (q_0^m - 1)/q_0^{\bar{d}} < 1/q_0^{\bar{d}-m}$ as required. □

Corollary 3.2. *A uniformly random $m \times \bar{d}$ matrix over $\mathcal{R}_q$ is full-rank except with probability at most $\varphi/q^{\bar{d}-m}$.*

Proof. Let r be the inertia degree of the prime q in $\mathcal{R}$, so that $\mathcal{R}_q$ is isomorphic to the product $\mathbb{F}_{q^r} \times \cdots \times \mathbb{F}_{q^r}$ of φ/r copies of $\mathbb{F}_{q^r}$ (we have $r = 1$ in the case of an NTT-friendly q). A matrix $\mathbf{M}$ over $\mathcal{R}_q$ is full-rank if and only if all of its components over this decomposition are full-rank. If $\mathbf{M} \in \mathcal{R}_q^{m \times \bar{d}}$ is uniformly random, the components are uniformly random as well, and by the previous lemma, each of them is thus full-rank except with probability at most $1/q^{r(\bar{d}-m)}$. Hence, by the union bound, the probability that $\mathbf{M}$ is not full-rank over $\mathcal{R}_q$ is at most $(\varphi/r)/q^{r(\bar{d}-m)} \leq \varphi/q^{\bar{d}-m}$ as required. □

3.2. Security Analysis

Theorem 3.3. *Let $q, m, \bar{d}$ such that $1/q^{\bar{d}-m} \leq \mathsf{negl}(\lambda)$. Suppose furthermore that $(2^\nu \cdot (1+\delta)/(\sigma_u \sqrt{2\pi e}))^{m\varphi} \leq \mathsf{negl}(\lambda)$, where $\delta = q/2^\nu - q_\nu$. For the parameters satisfying constraints from Lemma 2.3 and Theorem 2.8, the scheme* Ringtail *is unforgeable as per Definition 2.12 under the following assumptions*

- $\mathsf{stSIS}_{\mathcal{R},q,m,n+1,C,\beta}$ *with* $\beta = B_2 + \sqrt{\kappa} + (\kappa \cdot 2^\xi + 2^{\nu+1}) \cdot \sqrt{\varphi m}$ *(Definition 2.7)*
- $\mathsf{hLWE}_{\mathcal{R},q,m,n,k,\sigma,\bar{\sigma},\mathcal{L}}$ *for* $(k, \sigma, \bar{\sigma}, \mathcal{L}) \in \{(1, \sigma_E, \sigma^*, \mathcal{D}_{\sigma_u}), (Q_u, \sigma_e, \sigma^*, C)\}$ *(Definition 2.5)*
- $\mathsf{dLWE}'_{\mathcal{R},q,m,k,\sigma}$ *for* $(k, \chi) \in \{(n, \sigma^*), (m, \sigma_{\mathsf{td}})\}$ *with* $\sigma_{\mathsf{td}} > \eta_\varepsilon(\mathbb{Z}^{m\varphi})$ *(Definition 2.4)*
- *Security of PRF (Definition 2.9)*

Concretely, for any PPT adversary $\mathcal{A}$ against the unforgeability game for Ringtail*,*

$$\begin{aligned} &\mathbf{Adv}_{\mathcal{A}}^{\mathsf{TUF}}(\lambda) \leq (Q_c + Q_u) \cdot \mathbf{Adv}_{\mathcal{R},q,m,n+1,C,\beta}^{\mathsf{stSIS}}(\lambda) \\ &+ \mathbf{Adv}_{\mathcal{R},q,m,n,Q_u,\sigma_e,\sigma^*,C}^{\mathsf{hLWE}}(\lambda) \\ &+ QQ_u \Big(\bar{d} \cdot \mathbf{Adv}_{\mathcal{R},q,m,n,1,\sigma_E,\sigma^*,\mathcal{D}_{\sigma_u}}^{\mathsf{hLWE}}(\lambda) \\ &\quad + (\bar{d} - 2m) \cdot \mathbf{Adv}_{\mathcal{R},q,m,m,\sigma_{\mathsf{td}}}^{\mathsf{dLWE}'}(\lambda) \\ &\quad + Q_u \cdot \mathbf{Adv}_{\mathcal{R},q,m,n,\sigma^*}^{\mathsf{dLWE}'}(\lambda) \Big) + \mathbf{Adv}^{\mathsf{PRF}}(\lambda) + \mathsf{negl}(\lambda) \end{aligned}$$

where $\mathsf{negl}(\lambda)$ *denotes a statistical loss which is negligible in λ, Q is the number of signing queries to $\mathcal{O}_{\mathsf{Sign}}$, Q_u is the number of hash queries to H_u, Q_c is the number hash queries to H_c, respectively. (We assume Q_u and Q_c include hash queries made through queries to $\mathcal{O}_{\mathsf{Sign}_2}$.)*

Proof Sketch. We sketch a proof of security for the construction in Algorithm 1 and refer to Section A for a formal proof. To illustrate the intuition, we focus on the full threshold case i.e. $\ell = t$ where we already require the same proof technique. Its generalization to a general t-out-of-ℓ case mostly follows the idea of one-time masking from [PKM+24], although additional care is required for protecting against adversaries making queries to multiple honest party oracles with inconsistent inputs. Since we assume all-but-one corruption in this special case, in the description below, we fix the index of an honest party to $h \in [t]$ and a set $\mathcal{T} = [t]$ of participants in every session.

To understand our proof strategy, it is useful to recall how one would ideally prove the unforgeability, for the simplified case of a single honest party and a single signing query. Loosely speaking, our simulation needs to ensure that the verification relation is satisfied w.r.t. a conceptual public key for the honest party i.e. a shifted public key determined by the shares of corrupt parties $\mathbf{b}_h = \mathbf{b} - (\sum_{j\in\mathcal{T}\setminus\{h\}} \mathbf{A}\mathbf{s}_j \lambda_{\mathcal{T},j}) = \mathbf{A}\mathbf{s}_h\lambda_{\mathcal{T},h} + \mathbf{e}$:

$$\mathbf{D}_{h,1}\mathbf{u} \approx \mathbf{A}\mathbf{z}_h - \mathbf{b}_h c - \mathbf{d}_{h,0} \tag{1}$$

where $[\mathbf{d}_{h,0} \,|\, \mathbf{D}_{h,1}] := \mathbf{D}_h$ and the equality ignores noise terms. The natural strategy would be to simulate the signing query (without actually using the secret key) by sampling c, $\mathbf{u}$, and $\mathbf{z}_h$ from the appropriate distribution, where the distribution of $\mathbf{z}_h$ is roughly $\mathcal{D}^n_{\sigma^*}$ assuming $\sigma^* \gg \sigma_e, \sigma_E$ but still depends on the secret. Then one may sample all but one column of $\mathbf{D}_h$ uniformly, namely $\mathbf{D}_{h,1}$, and finally solve for $\mathbf{d}_{h,0}$ by linear algebra. One can show that this results in a distribution that is indistinguishable from the original one, if one appropriately programs c and $\mathbf{u}$ as the outputs of the respective random oracles.

When executing this strategy, one would encounter a fundamental problem: One needs to program $\mathbf{u}$ as the output of H_u, but at this point the matrix $\mathbf{D}_h$ is already fixed (since it is taken as an input). Thus the simulator needs to know ahead of time the query to the random oracle that corresponds to the actual transcript. For the case of a single signing query, the simulator can always guess such query, which results into a loss of Q_u in the advantage. However, for k queries, this guessing translates into a loss of Q_u^k, which does not lead to any meaningful bound.

To overcome the above problem, we instead use a different simulation strategy inspired by [BTT22]. Our simulator programs a lattice trapdoor directly into the matrix $\mathbf{D}_{h,1}$, which allows us to simulate signatures by pre-image sampling the vector $\mathbf{u}$, instead of the column $\mathbf{d}_{h,0}$. The advantage is that no guessing is needed in order for the simulation to succeed, and we can furthermore prove that the distribution induced by this alternative signing algorithm is computationally close to the original one. To be slightly more precise, we first argue that $\mathbf{D}_{h,1}$ is indistinguishable from uniformly random matrix with a reduction against the *Hint*-LWE problem (Definition 2.5), viewing $\mathbf{D}_{h,1} = \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$ as an LWE instance, and $\mathbf{R}_h\mathbf{u} + \mathbf{r}_h^*$ and $\mathbf{E}_h\mathbf{u} + \mathbf{e}_h^*$ as leakage vectors for the secret $\mathbf{R}_h$ and an error $\mathbf{E}_h$, respectively. Here, obtaining leakage vectors is crucial for the reduction to simulate the view of an adversary in the unforgeability game. Once $\mathbf{D}_{h,1}$ is regarded uniform, we can now invoke an appropriate trapdoor generation algorithm to turn $\mathbf{D}_{h,1}$ into pseudorandom matrix with a trapdoor, which can be used to sample $\mathbf{u}$ satisfying (1).

Our proof then proceeds with a careful hybrid argument, where we iteratively remove the need to use secret key for each signing query, programming instead the matrix $\mathbf{D}_h$ that is defined in the first round. Interestingly, the "guessing strategy" defined above will be used as an intermediate simulation, but *only to prove indistinguishability* between the real and the simulated queries. Crucially, our simulator will never have to guess more than a single query at a time.

After replacing all signing queries with simulated ones, we are essentially left to 1) remove $\mathbf{z}_h$'s dependency on the shared secret $\mathbf{s}$, and 2) bound the probability that the adversary outputs a valid forgery. These two steps are essentially equivalent to those of $\mathsf{tRaccoon}$. For 1), we rely on Hint-LWE again to argue the public key $\mathbf{b}$ is pseudorandom, this time viewing $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$ as an LWE instance, and $c\mathbf{s} + \mathbf{r}_h^*$ and $c\mathbf{e} + \mathbf{e}_h^*$ as leakage vectors, respectively. Again, leakage vectors are necessary for simulating the view of an adversary: $c\mathbf{s} + \mathbf{r}_h^*$ is programmed into $\mathbf{z}_h = (c\mathbf{s} + \mathbf{r}_h^*) + \mathbf{R}_h\mathbf{u} - \sum_{j\in\mathcal{T}\setminus\{h\}} c\lambda_{\mathcal{T},j}\mathbf{s}_j$, and $c\mathbf{e} + \mathbf{e}_h^*$ is programmed into (1), both of which are needed for signing oracle simulation. Since at this stage the secret key is not used anywhere, we can now view $[\mathbf{A} \,|\, \mathbf{b}]$ as an instance of (SelfTarget)SIS (Definition 2.7) and turn the forgery tuple $(c^*, \mathbf{z}^*, \mathbf{\Delta}^*)$ with bounded ℓ_2-norm into a solution vector for SIS.

This description oversimplifies many aspects of the proof. For instance, we also need to ensure that, whenever we program c as the output of H_c, the oracle is not yet defined at that point. We can show this with an information theoretic argument on the min-entropy of $\mathbf{h}$, which is established by analyzing the distribution of $\bar{\mathbf{D}}\mathbf{u}$. Furthermore, we need to ensure that the simulation hides the contribution of individual parties, which we do with (by now standard) masking techniques. We refer to Section A for more details.

3.3. Concrete Parameter Selection

We now derive concrete parameters for the scheme, targeting the $\lambda = 128$, 192 and 256-bit security levels. We aim to support up to $t = 1024$ signers, and a large bound $Q = 2^{60}$ on the number of signing queries per signing key. As usual when deriving concrete parameters, we aim for security with respect to the underlying problems, and ignore the polynomial loss factors in security proofs.

The parameter constraints can be summed up as follows. In all cases, we work over the ring $\mathcal{R} \cong \mathbb{Z}[x]/(x^\varphi + 1)$ with φ a power of two. The modulus q is chosen as a prime such that $q \equiv 1 \pmod{2\varphi}$ and $\lfloor q/2^\nu \rceil = \lfloor q/2^\nu \rfloor$, so that $\mathcal{R}_q$ supports a fast NTT-based multiplication and the condition for Lemma 2.3 is satisfied. Other constraints are as follows.

Gaussian widths. For accurate and well-behaved sampling of the discrete Gaussians, we should have $\sigma_e, \sigma_{\mathsf{td}} \geq \eta_\varepsilon(\mathcal{R}^m)$ and $\sigma_E \geq \eta_\varepsilon(\mathcal{R}^{\bar{d}})$.

Correctness. For correctness, §3.1 says that we should have $B_2 \geq k\sigma^*\sqrt{t(n+m)\varphi}$, with:

$$k = \sqrt{W_{-1}\left(1 + \frac{2\lambda \log 2}{(n+m)\varphi}\right)}$$

to ensure a correctness error $\leq 2^{-\lambda}$.

Size of σ_u. By Theorem 2.8, we need to take $\sigma_u^2 \geq (\sigma_g^2 + 1)\sigma_{\max}(\mathbf{T})^2 + \eta_\varepsilon(\mathcal{R}^{\bar{d}})^2$ where $\sigma_g^2 \geq 2b \cdot (2b+1)^2 \eta_\varepsilon(\mathbb{Z}^k)$, for b the chosen trapdoor radix and $k = \lceil \log_b q \rceil$. Moreover, by Bai–Yin's law, $\sigma_{\max}(\mathbf{T}) = (\sigma_{\mathsf{td}} + o(1)) \cdot (\sqrt{\varphi \bar{d}} + \sqrt{2\varphi m})$

with high probability. These constraints together determine the lower bound on σ_u.

Size of $|C|$. For the target collision resistance of H_c and the security of the stSIS problem, we need $|C| \geq 2^\lambda$. Since $|C| = 2^\kappa \binom{\varphi}{\kappa}$, this gives a lower bound on the Hamming weight κ.

Security I. Hardness of $\mathsf{stSIS}_{\mathcal{R},q,m,n+1,C,\beta}$ for $\beta = B_2 + \sqrt{\kappa} + (2^\xi \kappa + 2^{\nu+1}) \cdot \sqrt{\varphi m}$.

Security II. Hardness of $\mathsf{dLWE}'_{\mathcal{R},q,m,k,\sigma}$ for $(k, \sigma) \in \{(n, \sigma^*), (m, \sigma_{\mathsf{td}}), \}$. Among those constraints, the one on σ_{td} is more stringent.

Security III. Hardness of $\mathsf{hLWE}_{\mathcal{R},q,m,n,k,\sigma,\bar{\sigma},\chi}$ for $(k, \sigma, \bar{\sigma}, \chi) = (1, \sigma_E, \sigma^*, \mathcal{D}_{\sigma_u})$ and $(Q_u, \sigma_e, \sigma^*, C)$.

Estimating security with respect to the dLWE' and stSIS problems is standard, and we carry it out using Albrecht et al.'s LWE Estimator [APS15], [A$^+$], which now supports both LWE and SIS problems. Regarding hLWE, we use the following corollary of Theorem 2.6 to express it as a dLWE' instance, again estimated using the LWE Estimator.

Corollary 3.4. *Let $\sigma_0 = \sqrt{2}\eta_\varepsilon(\mathbb{Z}^\varphi)$, $\sigma = \alpha\eta_\varepsilon(\mathbb{Z}^\varphi)$ for some $\alpha > 2$, and χ some distribution on $\mathcal{R}^k$. Let B be some bound such that for $(\gamma_1, \ldots, \gamma_k)$ sampled from χ, $\sum_j \|\gamma_j\|_1^2 \leq B$ with overwhelming probability, and σ^* such that:*

$$\sigma^* \geq \eta_\varepsilon(\mathbb{Z}^\varphi) \cdot \sqrt{B} \cdot \frac{2\alpha}{\sqrt{\alpha^2 - 4}}.$$

Then there exists an efficient reduction from $\mathsf{dLWE}'_{\mathcal{R},q,m,n,\sigma_0}$ to $\mathsf{hLWE}_{\mathcal{R},q,m,n,k,\sigma,\bar{\sigma},\chi}$.

This imposes further constraints. For the first hLWE instance, we need $\sigma_E = \alpha\eta_\varepsilon(\mathbb{Z}^\varphi)$ for some $\alpha > 2$ and

$$\sigma^* \geq \eta_\varepsilon(\mathbb{Z}^\varphi) \cdot \sqrt{B_u} \cdot \frac{2\alpha}{\sqrt{\alpha^2 - 4}},$$

where B_u is such that $\Pr[\|u\|_1^2 > B_u] \leq 2^{-\lambda}$ for $u \sim \mathcal{D}_{\sigma_u}$. Now standard subgaussian bounds show that $\Pr[\|u\|_1 > k\sigma_u] \leq 2^\varphi \exp\big(-k^2/(2\varphi)\big)$, so it suffices to pick $B_u = \sigma_u^2 \cdot 2\varphi(\varphi \log 2 + \lambda)$.

For the second hLWE instance, we need $\sigma_e = \alpha\eta_\varepsilon(\mathbb{Z}^\varphi)$ for some $\alpha > 2$ and

$$\sigma^* \geq \eta_\varepsilon(\mathbb{Z}^\varphi) \cdot \sqrt{B_c} \cdot \frac{2\alpha}{\sqrt{\alpha^2 - 4}},$$

where B_c is a bound on $\sum_{j=1}^{Q_u} \|c_j\|_1^2$ for $c_j \in C$. Since $\|c_j\|^2 = \kappa^2$ for all j, we simply have $B_c = \kappa^2 Q_u$.

Both hLWE instances are at least as hard as $\mathsf{dLWE}'_{\mathcal{R},q,m,n,\sigma_0}$ with $\sigma_0 = \sqrt{2}\eta_\varepsilon(\mathbb{Z}^\varphi)$.

Since the lower bound on σ^* given by the second hLWE instance depends on Q_u, we need to pick a concrete value for this bound on the number of H_u queries as well, and we set $Q_u = Q = 2^{60}$. While it may seem prudent to pick $Q_u \gg Q$ instead due to the fact that hash queries are local computations, we claim that this choice more closely captures the power of a real-world attacker. Indeed, even though the reduction to hLWE (Lemma A.3) needs to receive Q_u leakage vectors from the hLWE game (as it does not know which H_u queries are used by the online signing oracle $\mathcal{O}_{\mathsf{Sign}_2}$), the reduction only reveals at most Q leakage vectors to a forger through $\mathcal{O}_{\mathsf{Sign}_2}$. In fact, if the reduction could correctly guess in advance which H_u queries are used by $\mathcal{O}_{\mathsf{Sign}_2}$, it would be possible to construct a reduction to hLWE parameterized by Q. In that sense, the fact that the hLWE problem is parametrized by Q_u instead of Q seems to be an artifact of the security proof, and hence choosing $Q_u \gg Q$ would highly overestimate the amount of leakage available to a real-world forger.

We propose parameters satisfying those constraints at the 128, 192 and 256-bit security levels in Table 2.

4. Implementation & Evaluation

Our evaluation aims to assess the following efficiency metrics of Ringtail:

1) network bandwidth cost,
2) local per-party latency,
3) end-to-end latency in a realistic, global setting.

In particular, we compare to tRaccoon [PKM$^+$24], which is the current state-of-art implementation for lattice-based threshold signatures. While tRaccoon signing takes 3 rounds, Ringtail takes only 2, where only the last round requires knowledge of the message. Thus, the first round can be preprocessed in an offline phase. The majority of compute *after* the first round can also be locally computed and preprocessed by each party without knowledge of the message. In a live deployment, eliminating message-specific pairwise communication between parties is instrumental since network transmission time often dominates the latency of multi-party protocols, as we show in Section 4.4.

4.1. Implementation

We implement Ringtail in ≈ 1800 lines of Golang using Lattigo [lat23], [MBTPH20], an open-source library with optimized arithmetic for lattice-based cryptography. Our codebase can be found at:

https://github.com/daryakaviani/ringtail

We use Lattigo's power-of-2 cyclotomic ring operations with an NTT-friendly modulus

$$q = 2^{48} + 2^{14} + 2^{11} + 2^9 + 1 = 281474976729601,$$

serialization, as well as uniform, discrete Gaussian, and ternary sampling. We instantiate our implementation with the parameters in Section 3.3.

Hash functions, MACs, & PRFs. We build our hash functions, MACs, and pseudorandom functions from the BLAKE3 hash function. In particular, H_c seeds a Gaussian sampler with the BLAKE3 hash function output, and H_u seeds a ternary sampler (configured with outputs of Hamming weight κ) with the hash function output.

Optimizations. For fast ring element multiplications, we integrated Lattigo's implementations of the number theoretic

Table 2: Parameters for Ringtail at the 128, 192 and 256-bit security levels. Sizes in KB.

Level	Bit security (LWE/SIS)	φ	$\lfloor \log_2 q \rceil$	κ	$(n, m, \bar{d})$	$\sigma_e = \sigma_E$	$\log_2 \sigma_u$	$\log_2 \sigma^*$	$\log_2 B_2$	(ν, ξ)	|vk|	|sig|
128	129 / 138	256	48	23	$(7, 8, 48)$	6.1	27.2	37.3	48.6	(29,30)	4.5	13.4
192	199 / 214	512	46	31	$(5, 6, 42)$	6.2	23.5	36.4	48.0	(25,29)	6.4	19.9
256	276 / 282	512	48	44	$(7, 8, 48)$	9.9	27.8	38.6	50.3	(29,31)	8.5	27.3

Table 3: Bandwidth (in bytes) for sending the verification key and signature across the network in Ringtail, tRaccoon and EKT, as well as the the communication required between parties for signature generation and mask exchange.

Scheme	|vk|	|sig|	Sign (Total)	Sign (Online)	Gen
Ringtail	4,608	13,702	$612{,}864 + 16t$	10,752	$10{,}752 + 32\ell$
tRaccoon [PKM+24]	3,856	12,736	$40{,}800 + 16t$	$40{,}800 + 16t$	$12{,}556 + 32\ell$
EKT [EKT24]	5,632	11,059	282,311	14,400	$14{,}400 + 16\ell$

Table 4: Local per-party latency (ms) for Ringtail and tRaccoon for different thresholds. The tRaccoon entries in the Online column are their total local signing time across all 3 of their online rounds.

t	Scheme	Gen	Sign_1	Sign_2	Sign_2	Finalize	Verify
				Preprocess	*Online*		
4	Ringtail	6.962	18.995	131.944	4.601	0.143	1.361
	tRaccoon	0.282	—	—	10.581	0.537	0.521
16	Ringtail	14.936	29.084	164.646	13.904	0.179	1.331
	tRaccoon	0.199	—	—	12.208	0.576	0.520
64	Ringtail	57.672	73.684	313.938	52.036	0.339	1.627
	tRaccoon	0.389	—	—	14.308	0.752	0.524
256	Ringtail	224.084	254.253	829.033	201.272	0.821	1.515
	tRaccoon	1.351	—	—	57.462	1.517	0.521
1024	Ringtail	1237.583	978.182	2949.853	794.288	2.706	1.434
	tRaccoon	5.472	—	—	194.149	5.510	0.527

Table 5: Global wide area network latency (ms) for Ringtail and for $t = \ell = 8$.

Round	Local Online Compute	Network Latency	Combine	End-to-End
Sign_1	26.248	1888.147	—	1914.395
Sign_2 *Online*	7.497	620.513	0.173	628.183

transform (NTT) and Montgomery coefficient-wise multiplications. We utilize a hash function with AVX2 acceleration[3] for SIMD operations.

Networking. We design a networking stack for the WAN setting, which allows signers to form peer-to-peer network connections with other parties. Each party concurrently communicates with every other party by serializing and sending its messages through outgoing TCP sockets, while simultaneously receiving and processing incoming messages.

Preprocessing & Full-Rankness Check. The Sign_2 Preprocessing phase involves MAC verification, hashing $\mathbf{D}$, and the full rank-ness test on $\bar{\mathbf{D}}$. These are steps after Sign_1 which are message-independent and can be preprocessed in the offline phase. To create a performant implementation for the full-rankness check, we compute the element-wise NTT of $\bar{\mathbf{D}}$, resulting in φ matrices over $\mathbb{Z}_q$. We then compute the rank of each matrix using Gaussian elimination modulo q. Checking that all φ matrices are full-rank is equivalent to checking that $\bar{\mathbf{D}}$ is full-rank, but avoids heavy ring arithmetic, such as ring element division.

3. https://github.com/zeebo/blake3

4.2. Bandwidth

Table 3 presents the number of bytes sent across the network throughout in Ringtail, tRaccoon [PKM+24], and EKT [EKT24]. In particular, we report the size of the verification key and signature, as well as the the total number of bytes sent during key generation and signing.

At $t = 1024$, Ringtail's communication bandwidth in the online phase is 81% less than tRaccoon and 25% less than EKT, at the expense of a high-communication offline phase which can be preprocessed. As we will concretize in Section 4.4, minimizing the per-message bytes sent across the network is crucial since this time often dominates the end-to-end signing latency. The signature and verification key size is comparable across all three schemes.

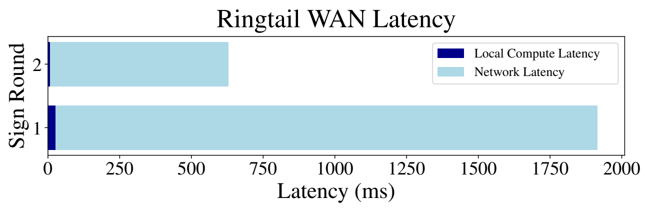

Figure 1: Global wide area network latency (ms) for Ringtail and for $t = \ell = 8$.

4.3. Local Per-Party Latency

Experiment Setup. We run our local per-party computation on one AWS `c5.4xlarge` instance with 16 vCPUs and 32 GB of memory. We use a single core without parallelism. To prevent multiple signers from competing for CPU resources, we perform each signer's computation serially.

Table 4 compares the per-party compute latency of Ringtail to tRaccoon. As the tRaccoon implementation is not yet open-source, we use their results as reported in [PKM+24].

We emphasize that since both schemes perform on the order of milliseconds, granular differences relating to implementation nuances, hardware, and randomness generation make a significant difference in the results, so these numbers are not a perfect comparison (e.g. verification exhibits a latency difference, though the schemes are identical). We were not able to identify these discrepancies since the tRaccoon code is not open source, but we still provide these numbers to show that, although the primary advantage of Ringtail is its one-shot online round, the local compute latency is also comparable. We average our results over 100 executions.

We expect the local latency to be higher than tRaccoon due to the additional computation of processing the $\mathbf{D}$ matrix in Algorithm 1, which is of higher dimension for the standard-assumption security proof. As we will show in Section 4.4, however, end-to-end latency is often dominated by network latency. Thus, we actually predict that Ringtail would be faster in a real-world deployment for two reasons:

1) In our online phase, fewer bytes must be transferred across the network than tRaccoon (Table 3).
2) Each additional roundtrip contributes significantly to the total signature generation time. tRaccoon has three online round, while we have one offline round and only one online round (Table 5).

Note that Sign_2 Preprocessing involves MAC verification, hashing $\mathbf{D}$, and the full-rankness check on $\bar{\mathbf{D}}$, as in Remark 3. These can be performed in parallel with the rest of the computation since they are message-independent.

4.4. Wide Area Network (WAN) Latency

In standard applications of threshold signatures (Section 1.2), a small number of signers are located in distinct trust domains, often in dispersed geographical regions. In these cases, the time to generate a signature is often dominated by network latency, making round-optimized protocols vital. Therefore, we evaluate Ringtail's performance in the WAN setting.

Experiment Setup. We execute the protocol using 8 AWS `c5.4xlarge` instances with 16 vCPUs and 32 GB of memory. We select machines in the following 8 regions across 5 continents, to achieve realistic network latency: `us-west-1` (Northern California), `us-east-1` (Northern Virginia), `eu-central-1` (Frankfurt), `eu-west-1` (Ireland), `ap-northeast-1` (Tokyo), `ap-southeast-1` (Singapore), `sa-east-1` (São Paulo), and `ap-southeast-2` (Sydney). We measure the round-trip-time and network throughput between each pair of servers in our setup; the average RTT is 170.11 ms and the average network throughput is 183.23 Mbps.

Table 5 and Fig. 1 show the compute-only, network, and end-to-end latency of Ringtail among our $t = \ell = 8$ servers. We average our results over 10 executions. As expected, the local per-party signing is significantly greater than Table 4 due to the network latency, amounting to 98.63% and 98.78% of the round 1 and 2 online end-to-end latencies, respectively. While tRaccoon does not provide WAN benchmarks, we predict that Ringtail's online phase would be substantially faster for two reasons: First, fewer bytes must be transferred across the network, and second, each additional roundtrip contributes significantly to the total signature generation time. Indeed, the fact that network latency dominates overall time highlights the importance of minimizing message-specific roundtrips.

5. Conclusion

In this work, we propose Ringtail, a practical two-round threshold signature scheme. Our scheme is proven secure against standard lattice assumptions (namely, LWE and SIS), and it is concretely efficient, even for a large number of parties. Our benchmarks show that Ringtail is competitive with other standard-assumption protocols that have more rounds of communication. Overall, our work demonstrates that lattice-based threshold signatures with a single round of online communication and solid security foundations can be efficient enough to be used in the real world.

Acknowledgments

We thank Elizabeth C. Crites for discussions on the security model of threshold signatures, and Jean-Philippe Bossuat for discussions about the Lattigo codebase. We thank anonymous reviewers of Symposium on Security and Privacy 2025 for helpful comments and constructive feedback.

C.B. is supported by the SNSF Ambizione grant No. PZ00P2_216140. R.L. is supported by the Research Council of Finland project No. 358951. G.M. is supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation). This paper was prepared in part for information purposes by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

[A+] M. R. Albrecht et al. Security estimates for lattice problems. GitHub repository. available at https://github.com/malb/lattice-estimator.

[AB21] H. K. Alper and J. Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In *CRYPTO 2021, Part I*, vol. 12825 of *LNCS*, pp. 157–188, Virtual Event, 2021. Springer, Heidelberg.

[AB23] S. Abramova and R. Böhme. Anatomy of a High-Profile data breach: Dissecting the aftermath of a Crypto-Wallet case. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 715–732, Anaheim, CA, 2023. USENIX Association.

[ADP24] N. A. Alkadri, N. Döttling, and S. Pu. Practical lattice-based distributed signatures for a small number of signers. Cryptology ePrint Archive, Paper 2024/449, 2024. https://eprint.iacr.org/2024/449.

[AL21] M. R. Albrecht and R. W. F. Lai. Subtractive sets over cyclotomic rings - limits of Schnorr-like arguments over lattices. In *CRYPTO 2021, Part II*, vol. 12826 of *LNCS*, pp. 519–548, Virtual Event, 2021. Springer, Heidelberg.

[APS15] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[ASY22] S. Agrawal, D. Stehlé, and A. Yadav. Round-optimal lattice-based threshold signatures, revisited. In *ICALP 2022*, vol. 229 of *LIPIcs*, pp. 8:1–8:20. Schloss Dagstuhl, 2022.

[BCJ08] A. Bagherzandi, J. H. Cheon, and S. Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In *ACM CCS 2008*, pp. 449–458. ACM Press, 2008.

[BCK+22] M. Bellare, E. C. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. Better than advertised security for non-interactive threshold signatures. In *CRYPTO 2022, Part IV*, vol. 13510 of *LNCS*, pp. 517–550. Springer, Heidelberg, 2022.

[BEP+21] P. Bert, G. Eberhart, L. Prabel, A. Roux-Langlois, and M. Sabt. Implementation of lattice trapdoors on modules and applications. In *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021*, pp. 195–214. Springer, Heidelberg, 2021.

[BGG+18] D. Boneh, R. Gennaro, S. Goldfeder, A. Jain, S. Kim, P. M. R. Rasmussen, and A. Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO 2018, Part I*, vol. 10991 of *LNCS*, pp. 565–596. Springer, Heidelberg, 2018.

[BKP13] R. Bendlin, S. Krehbiel, and C. Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In *ACNS 13*, vol. 7954 of *LNCS*, pp. 218–236. Springer, Heidelberg, 2013.

[BL90] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO'88*, vol. 403 of *LNCS*, pp. 27–35. Springer, Heidelberg, 1990.

[BLL+21] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. On the (in)security of ROS. In *EUROCRYPT 2021, Part I*, vol. 12696 of *LNCS*, pp. 33–53. Springer, Heidelberg, 2021.

[BN06] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *ACM CCS 2006*, pp. 390–399. ACM Press, 2006.

[BP23] L. Brandao and R. Peralta. Nist first call for multi-party threshold schemes, 2023.

[BTT22] C. Boschini, A. Takahashi, and M. Tibouchi. MuSig-L: Lattice-based multi-signature with single-round online phase. In *CRYPTO 2022, Part II*, vol. 13508 of *LNCS*, pp. 276–305. Springer, Heidelberg, 2022.

[CATZ24] R. Chairattana-Apirom, S. Tessaro, and C. Zhu. Partially non-interactive two-round lattice-based threshold signatures. Cryptology ePrint Archive, Paper 2024/467, 2024. https://eprint.iacr.org/2024/467.

[Che23] Y. Chen. DualMS: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In *CRYPTO 2023, Part V*, vol. 14085 of *LNCS*, pp. 716–747. Springer, Heidelberg, 2023.

[CKM23] E. C. Crites, C. Komlo, and M. Maller. Fully adaptive Schnorr threshold signatures. In *CRYPTO 2023, Part I*, vol. 14081 of *LNCS*, pp. 678–709. Springer, Heidelberg, 2023.

[CS19] D. Cozzo and N. P. Smart. Sharing the LUOV: threshold post-quantum signatures. In *Cryptography and Coding - 17th IMA International Conference, IMACC 2019, Oxford, UK, December 16-18, 2019, Proceedings*, vol. 11929 of *Lecture Notes in Computer Science*, pp. 128–153. Springer, 2019.

[DEF+19] M. Drijvers, K. Edalatnejad, B. Ford, E. Kiltz, J. Loss, G. Neven, and I. Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pp. 1084–1101. IEEE Computer Society Press, 2019.

[Des88] Y. Desmedt. Society and group oriented cryptography: A new concept. In *CRYPTO'87*, vol. 293 of *LNCS*, pp. 120–127. Springer, Heidelberg, 1988.

[DF90] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO'89*, vol. 435 of *LNCS*, pp. 307–315. Springer, Heidelberg, 1990.

[DKL+23] N. Döttling, D. Kolonelos, R. W. F. Lai, C. Lin, G. Malavolta, and A. Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT 2023, Part III*, vol. 14006 of *LNCS*, pp. 417–446. Springer, Heidelberg, 2023.

[DOTT22] I. Damgård, C. Orlandi, A. Takahashi, and M. Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. *Journal of Cryptology*, 35(2):14, 2022.

[EKT24] T. Espitau, S. Katsumata, and K. Takemure. Two-round threshold signature from algebraic one-more learning with errors. To appear in CRYPTO 2024, 2024.

[GKS24] K. D. Gur, J. Katz, and T. Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. To appear in PQCrypto 2024, 2024.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*, vol. 1. Cambridge University Press, Cambridge, UK, 2001.

[GPV07] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. Cryptology ePrint Archive, Report 2007/432, 2007. https://eprint.iacr.org/2007/432.

[GPV08] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *40th ACM STOC*, pp. 197–206. ACM Press, 2008.

[JMW24] K. A. Jackson, C. A. Miller, and D. Wang. Evaluating the security of CRYSTALS-Dilithium in the quantum random oracle model. In *EUROCRYPT 2024, Part VI*, vol. 14656 of *LNCS*, pp. 418–446. Springer, 2024.

[KG20] C. Komlo and I. Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In *SAC 2020*, vol. 12804 of *LNCS*, pp. 34–65. Springer, Heidelberg, 2020.

[KLS18] E. Kiltz, V. Lyubashevsky, and C. Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In *EUROCRYPT 2018, Part III*, vol. 10822 of *LNCS*, pp. 552–586. Springer, Heidelberg, 2018.

[KLSS23] D. Kim, D. Lee, J. Seo, and Y. Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In *CRYPTO 2023, Part V*, vol. 14085 of *LNCS*, pp. 549–580. Springer, Heidelberg, 2023.

[KRT24] S. Katsumata, M. Reichle, and K. Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. To appear in CRYPTO 2024, 2024.

[lat23] Lattigo v5. Online: https://github.com/tuneinsight/lattigo, 2023. EPFL-LDS, Tune Insight SA.

[LDK+20] V. Lyubashevsky, L. Ducas, E. Kiltz, T. Lepoint, P. Schwabe, G. Seiler, D. Stehlé, and S. Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions.

[Lyu12] V. Lyubashevsky. Lattice signatures without trapdoors. In *EUROCRYPT 2012*, vol. 7237 of *LNCS*, pp. 738–755. Springer, Heidelberg, 2012.

[MBTPH20] C. V. Mouchet, J.-P. Bossuat, J. R. Troncoso-Pastoriza, and J.-P. Hubaux. Lattigo: A multiparty homomorphic encryption library in go. In *Proceedings of the 8th Workshop on Encrypted Computing and Applied Homomorphic Cryptography*, pp. 64–70, 2020.

[MP12] D. Micciancio and C. Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT 2012*, vol. 7237 of *LNCS*, pp. 700–718. Springer, Heidelberg, 2012.

[NRS21] J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In *CRYPTO 2021, Part I*, vol. 12825 of *LNCS*, pp. 189–221, Virtual Event, 2021. Springer, Heidelberg.

[PEK+23] R. D. Pino, T. Espitau, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M. O. Saarinen. Raccoon: A side-channel secure signature scheme. https://raccoonfamily.org/wp-content/uploads/2023/07/raccoon.pdf, 2023.

[PKM+24] R. D. Pino, S. Katsumata, M. Maller, F. Mouhartem, T. Prest, and M. O. Saarinen. Threshold Raccoon: Practical threshold signatures from standard lattice assumptions. In *EUROCRYPT 2024*, vol. 14652 of *LNCS*, pp. 219–248. Springer, 2024.

[Sha79] A. Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, 1979.

[Wol16] J. Wolff. How a 2011 hack you've never heard of changed the internet's infrastructure. https://slate.com/technology/2016/12/how-the-2011-hack-of-diginotar-changed-the-internets-infrastructure.html, 2016.

Appendix A. Proof of Theorem 3.3

In this section, we provide a security proof of Ringtail described in Algorithm 1. We first introduce some useful notations. Let $\mathcal{C} \subset [\ell]$ be a corruption set of size $t-1$ chosen by the adversary at the beginning of the unforgeabilty game. For each signing query, we denote by $\mathcal{T} \subseteq [\ell]$ the corresponding signing coalition[4]. For a fixed $\mathcal{T}$, we call $\mathcal{T} \cap \mathcal{C}$ a *corrupted coalition* and $\mathcal{T} \cap \mathcal{H}$ be an *honest coalition*, respectively. Typically, we denote an index of an honest party by i. For each honest coalition, we will denote its smallest index by $h := \min(\mathcal{T} \cap \mathcal{H})$. We also introduce a global counter $\mathsf{ctr}_{\mathsf{min}}$, denoting the number of queries made to $\mathcal{O}_{\mathsf{Sign}_1}$ so far with input $(h, \mathcal{T})$. For better readability, we slightly change the notations of sets kept tracked by oracles $\mathcal{O}_{\mathsf{Sign}_1}$ and $\mathcal{O}_{\mathsf{Sign}_2}$ in Game 1. We write "Store (resp. Load) $(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, \mathsf{st}_i)$" to denote the tuple $(\mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, \mathsf{st}_i)$ is stored in (resp. loaded from) the set S. Similarly, we write "Store (resp. Load) $(\text{FINISH}, \mathsf{eid}, i)$" to denote the tuple (eid, i) is stored in (resp. loaded from) the set S'. The oracles also keep track of potential online messages for *all* honest parties in the current signing coalition, deter mined by invocations of H_u with input $(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$. We denote each online message by a tuple $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$, where ctr_u is a counter that is initially set to 0 at the beginning of the game and keeps track of the number of fresh queries made to H_u. This counter will be used by oracles in intermediate hybrids we introduce later.

Proof. We prove this by a hybrid argument, and use the terms "hybrid" and "simulator" interchangeably. For completeness, in Algorithms 2 to 11 we present the pseudocode for intermediate hybrids to detail out every hop precisely. In the definitions of these hybrids, we omit subroutines which are unchanged compared to the previous hybrid. Below, we write Q for number of $\mathcal{O}_{\mathsf{Sign}_1}$ queries made by the adversary $\mathcal{A}$.

$\underline{\mathsf{Hyb}_0}$**:** This is equivalent to the unforgeability game as in Game 1 with the construction inlined, except that $\mathcal{O}_{\mathsf{Sign}_1}$ aborts if 1) it happens to generate $\mathbf{D}_i$ that was already generated by one of the previous calls to $\mathcal{O}_{\mathsf{Sign}_1}$, or 2) the adversary previously queried H_u with input including $\mathbf{D}_i$. Assuming that the event 1) does not occur, $\mathbf{D}_i$ can be used as a unique look-up key to load a recorded tuple $(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, \ldots)$. Assuming that the event 2) does not occur, we may apply syntactic changes to H_u such that it prepares a response $\mathbf{z}_i$ for every honest party i in $\mathcal{T} \cap \mathcal{H}$ as soon as it receives a query with input $(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D})_{j\in\mathcal{T}}, \mu)$. Concretely, the input to H_u can be

4. In our security model, an adversary may query $\mathcal{O}_{\mathsf{Sign}_1}$ for multiple honest parties with inconsistent $\mathcal{T}$, e.g., party 1 receives $\mathcal{T} = \{1,2,3\}$ and party 2 may receive $\mathcal{T}' = \{1,2,4\}$. However, such inconsistent coalitions will be always detected in the following call to $\mathcal{O}_{\mathsf{Sign}_2}$ since it validates that all honest parties agreed to participate in the *same* coalition. As remarked before, this is in practice guaranteed by having each party generate a signature on $(\mathcal{T}, i, \mathbf{D}_i)$ and attach the signature to $\mathbf{D}_i$.

used to derive $\mathbf{u}$, $\mathbf{h}$, and thus challenge c after making a query to H_c. As the input to H_u is consistent with that of PRF, one time row and column masks $\mathbf{m}_i, \mathbf{m}'_i$ can be derived once H_u is queried. Since the preimage $(\mathbf{r}^*_i, \mathbf{R}_i)$ can be uniquely obtained from $\mathbf{D}_i$, H_u can indeed compute $\mathbf{z}_i$ for every i in $\mathcal{T} \cap \mathcal{H}$. Clearly, this syntactic change does not alter the view of the adversary. We denote the routines for generating an offline message $\mathbf{D}_i$ by GenOffline, and for all honest online messages $(\mathbf{z}_i)_{i \in \mathcal{T} \cap \mathcal{H}}$ by GenOnline, respectively.

$\underline{\mathsf{Hyb}_s,\ s \in [Q]}$**:** In these hybrids, we change the way the oracles answer the first s signing queries with input $(i, \mathcal{T})$ such that i is the smallest index in the honest coalition, i.e., $i = h := \min(\mathcal{T} \cap \mathcal{H})$. In more detail, Hyb_s differs from Hyb_{s-1} in mainly two ways, which correspond to substituting GenOffline and GenOnline with two new subroutines, SimOffline and SimOnline. First, for queries to $\mathcal{O}_{\mathsf{Sign}}$ with $i = h$, SimOffline samples $\mathbf{D}_h$ by sampling its first column $\mathbf{d}_{h,0}$ uniformly at random, and the remaining columns (denoted by $\mathbf{D}_{h,1}$) with a trapdoor td_h. (To be more precise, it generates $\mathbf{D}'_{h,1}$ with a trapdoor and then sets $\mathbf{D}_{h,1}$ as its shift $\mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$ for the reason that becomes clear in later hybrids.) If the index $i \neq h$, it proceeds as in the real signing oracle. Second, for every query to H_u containing $(\mathbf{D}_j)_{j \in \mathcal{T} \cap \mathcal{H}}$ (from the queries to $\mathcal{O}_{\mathsf{Sign}_1}$), SimOnline does the following:

1) Sample c, $\mathbf{r}^*_h$, and $\mathbf{e}^*_h$ at random from their respective appropriate distributions.
2) Compute a session-wide response $\mathbf{z}^* = \mathbf{s}c + \mathbf{r}^*_h$ w.r.t. the original secret $\mathbf{s}$.
3) Use td_h to sample a short $\mathbf{u}$ such that

$$\mathbf{D}'_h \mathbf{u} = \mathbf{A}\mathbf{z}^* - \mathbf{b}c + \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}^*_h \qquad (2)$$

This step is carried out by preimage sampling algorithm from Theorem 2.8.
4) Sample a response $\mathbf{z}_i$ of parties $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ uniformly at random from $\mathcal{R}_q^n$.
5) Compute a response $\mathbf{z}_h$ of party h:

$$\begin{aligned}\mathbf{m} &= \sum_{i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j}) \\ \mathbf{z}_h &= \mathbf{z}^* + \mathbf{R}_h\mathbf{u} - (\sum_{j \in \mathcal{T}\cap\mathcal{C}} \mathbf{s}_j \cdot \lambda_{\mathcal{T},j}) \cdot c \\ &\quad - \sum_{i \in \mathcal{T}\cap\mathcal{H}\setminus\{h\}} (\mathbf{z}_i - \mathbf{r}^*_i - \mathbf{R}_i\mathbf{u}) + \mathbf{m}\end{aligned}$$

where $\mathbf{m}$ is an intermediate value consisting of masks and is never revealed.
6) Compute $\mathbf{h}$, and attempt to program $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$ with c. If H_c is already defined on this input, H_u aborts.

Note that the adversary may query H_u many times with different choices of $(\mathbf{D}_j)_{j \neq h}$ and μ before querying $\mathcal{O}_{\mathsf{Sign}_2}$ on eid associated with $\mathbf{D}_h$. At some point the adversary queries $\mathcal{O}_{\mathsf{Sign}_2}$ on $(\mathsf{eid}, h, (\mathbf{D}_j)_{j \neq h}, \mu)$ which fixes the first round messages of the adversarial parties. Note that the logic of $\mathcal{O}_{\mathsf{Sign}_2}$ ensures that H_u must be queried on $(\mathbf{D}_j)_{j \in \mathcal{T}}$ and μ, and hence $\mathbf{z}_h$ returned by $\mathcal{O}_{\mathsf{Sign}_2}$ is well-defined.

$\underline{\mathsf{Hyb}_{Q+1}}$**:** This hybrid is identical to Hyb_Q except that the simulator generates the secret keys of the corrupted users as $\mathbf{s}_j \xleftarrow{\$} \mathcal{R}_q^n$ for $j \in \mathcal{C}$ instead of executing Shamir sharing on $\mathbf{s}$. The hybrid remains well-defined since honest secret keys $(\mathbf{s}_i)_{i \in \mathcal{H}}$ are not used for answering oracle queries.

$\underline{\mathsf{Hyb}_{Q+2}}$**:** This hybrid is identical to Hyb_{Q+1} except that the public key (before rounding) $\mathbf{b}$ is sampled uniformly at random, while $\mathbf{s}$ and $\mathbf{e}$ sampled independently of $\mathbf{b}$ are used for generating $\mathbf{z}^*$ and $\mathbf{u}$.

$\underline{\mathsf{Hyb}_{Q+3}}$**:** This hybrid is identical to Hyb_{Q+2} except that the simulator guesses a query to H_c that determines $\tilde{c}$ as part of the forgery tuple $(\tilde{c}, \tilde{\mathbf{z}}, \tilde{\mathbf{\Delta}})$ and aborts if $\tilde{c}$ was previously programmed by SimOnline. That is, the simulator picks a uniformly random $\tilde{q} \in [Q_c + Q_u]$ at the beginning of the game, and skips programming H_c inside SimOnline (Line 14) if its caller H_u was invoked as the $\tilde{q}$-the query to random oracles (assuming that there is a common counter for queries to both H_c and H_u); else, it invokes SimOnline as in the previous hybrid. Similar to MuSig-L [BTT22], we need to add this step to make sure that H_u does not program H_c for all queries, since one of them might contain a message $\tilde{\mu}$ used for a forgery.

Lemma A.1 shows that the unforgeability experiment $\mathsf{Exp}^{\mathsf{TUF}}_{\mathsf{TS},\mathcal{A}}$ and Hyb_0 are statistically indistinguishable. Lemma A.2 shows that Hyb_Q and Hyb_{Q+1} are identical. Lemma A.3 shows that Hyb_{Q+1} and Hyb_{Q+2} are computationally indistinguishable under the Hint-LWE assumption. Lemma A.4 shows that the probability of the adversary winning in Hyb_{Q+3} decreases by a multiplicative factor $1/(Q_u + Q_c)$ compared to Hyb_{Q+2}. Lemma A.5 shows that the probability of the adversary winning in Hyb_{Q+3} is negligible under the SelfTargetSIS assumption. Finally, Lemma A.6 shows that Hyb_{s-1} and Hyb_s are indistinguishable for $s \in [Q]$. Putting together, we obtain the concrete advantage bound stated in Theorem 3.3. □

Lemma A.1. *The unforgeability experiment* $\mathsf{Exp}^{\mathsf{TUF}}_{\mathsf{TS},\mathcal{A}}$ *and* Hyb_0 *are statistically indistinguishable, i.e.*

$$|\mathbf{Adv}^{\mathsf{TUF}}_{\mathcal{A}}(\lambda) - \Pr[\mathsf{Hyb}_0(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda).$$

Proof. Unless $\mathsf{Hyb}_0(1^\lambda)$ aborts, the view of the adversary is identical in both experiments. To bound the probability that it aborts, we evaluate the min-entropy of the first column $\mathbf{d}_{i,0} = \mathbf{A}\mathbf{r}^*_i + \mathbf{e}^*_i$ which has at least the min-entropy of $\mathbf{e}^*_i$. The min-entropy of a one-dimensional Gaussian of parameter σ^* over $\mathbb{Z}$ is given by $\log_2 \rho_{\sigma^*}(\mathbb{Z})$, which is ε-close to $\log_2(\sigma^*\sqrt{2\pi e})$ since $\sigma^* > \eta_\varepsilon(\mathbb{Z})$. It follows that $\mathbf{D}_i$ has min-entropy at least $m\varphi \log_2(\sigma^*\sqrt{2\pi e})$. Therefore, the probability of any possible value of $\mathbf{D}_i$ is bounded by $(1/(\sigma^*\sqrt{2\pi e}))^{m\varphi}$, which is negligible in λ. Since there are at most $Q + Q_u$ existing values of $\mathbf{D}_i$ and the adversary queries $\mathcal{O}_{\mathsf{Sign}_1}$ at most Q times, by the union bound, the probability that Hyb_0 aborts is negligible. □

Lemma A.2. Hyb_Q *and* Hyb_{Q+1} *are perfectly indistinguishable, i.e.*

$$\Pr[\mathsf{Hyb}_Q(1^\lambda) = 1] = \Pr[\mathsf{Hyb}_{Q+1}(1^\lambda) = 1].$$

Proof. This trivially follows from the security of Sharmir secret sharing. That is, the distribution of at most $t-1$ shares is uniformly random and independent of the shared secret $\mathbf{s}$. $\square$

Lemma A.3. Hyb_{Q+1} *and* Hyb_{Q+2} *are indistinguishable under the* $\mathsf{hLWE}_{\mathcal{R},q,m,n,Q_u,\sigma_e,\sigma^*,C}$ *assumption (see Definition 2.5). Concretely, there exists a PPT adversary* $\mathcal{B}$ *such that*

$$|\Pr[\mathsf{Hyb}_{Q+1}(1^\lambda) = 1] - \Pr[\mathsf{Hyb}_{Q+2}(1^\lambda) = 1]| \\ \leq \mathbf{Adv}^{\mathsf{hLWE}}_{\mathcal{R},q,m,n,Q_u,\sigma_e,\sigma^*,C}(\lambda)$$

Proof. Upon receiving an LWE instance $(\mathbf{A}^*, \mathbf{y}^*, \mathbf{l}^*, \mathbf{c}^*)$, the reduction $\mathcal{B}$ sets $\mathbf{A} = \mathbf{A}^*$ and $\mathbf{b} = \mathbf{y}^*$. It then parses $\mathbf{l}^*, \mathbf{c}^*$ as

$$\mathbf{l}^* = \left\{ \begin{pmatrix} \mathbf{s}c_i + \mathbf{r}_i^* \\ \mathbf{e}c_i + \mathbf{e}_i^* \end{pmatrix} \right\}_{i=1}^{Q_u} \quad \mathbf{c}^* = \{c_i\}_{i=1}^{Q_u}$$

and simulates the rest of Hyb_Q or Hyb_{Q+1} for $\mathcal{A}$ using the first component of each element of $\mathbf{l}^*$ as $\mathbf{z}^*$, and the second component as the last two terms of $\mathbf{w}'_h$ (i.e., the RHS of (2)), and $\mathbf{c}$ as challenge that gets programmed by $\mathsf{SimOnline}$. If the instance is real (resp. uniform), it is clear that the view of the adversary $\mathcal{A}$ is identical to that of Hyb_{Q+1} (resp. Hyb_{Q+2}). The advantage of $\mathcal{B}$ is thus exactly the probability of $\mathcal{A}$ distinguishing Hyb_{Q+1} from Hyb_{Q+2}. $\square$

Lemma A.4.
$\Pr[\mathsf{Hyb}_{Q+2}(1^\lambda) = 1] \leq (Q_c + Q_u) \cdot \Pr[\mathsf{Hyb}_{Q+3}(1^\lambda) = 1].$

Proof. Since the adversary sets the output of H_c at most $Q_c + Q_u$ times (both directly to H_c and implicitly through queries to H_u), the probability that the simulator correctly guesses a critical query to H_c used for a forgery is at least $1/(Q_c + Q_u)$. $\square$

Lemma A.5. *The probability that* $\mathcal{A}$ *wins in* Hyb_{Q+3} *is negligible under the* $\mathsf{stSIS}_{\mathcal{R},q,m,n+1,C,\beta}$ *assumption (see Definition 2.7). Concretely, if* $\mathcal{A}$ *makes at most* Q_c *queries to the random oracle* H_c *and* Q_u *queries to* H_u*, there exists a PPT adversary* $\mathcal{B}$ *such that*

$$\Pr[\mathsf{Hyb}_{Q+3}(1^\lambda) = 1] \leq \mathbf{Adv}^{\mathsf{stSIS}}_{\mathcal{R},q,m,n+1,C,\beta}(\lambda)$$

where $\beta = B_2 + \sqrt{\kappa} + (\kappa \cdot 2^\xi + 2^{\nu+1}) \cdot \sqrt{\varphi m}$.

Proof. At this stage, note that $(\mathbf{s}, \mathbf{e})$ used by the simulator are sampled independently of uniform $\mathbf{b}$, and that $\tilde{c}$ as part of the forgery tuple is not derived by programming H_c. Thus, a reduction against stSIS can relay all queries to H_c made by the adversary to G in the stSIS game. The rest follows from unforgeability of tRaccoon (see [PKM+24, Lemma 7.4]), so we omit the proof. $\square$

Lemma A.6. *For* $s \in [Q]$, $\mathsf{Hyb}_{s-1} \approx_c \mathsf{Hyb}_s$*. Concretely,*

$$\begin{aligned} &|\Pr[\mathsf{Hyb}_{s-1}(1^\lambda) = 1] - \Pr[\mathsf{Hyb}_s(1^\lambda)]| \\ &\leq Q_u\Big(\bar{d} \cdot \mathbf{Adv}^{\mathsf{hLWE}}_{\mathcal{R},q,m,n,1,\sigma_E,\sigma^*,\mathcal{D}_{\sigma_u}}(\lambda) \\ &\quad + (\bar{d} - 2m) \cdot \mathbf{Adv}^{\mathsf{dLWE}'}_{\mathcal{R},q,m,m,\sigma_{\mathsf{td}}}(\lambda) \\ &\quad + Q_u \cdot \mathbf{Adv}^{\mathsf{dLWE}'}_{\mathcal{R},q,m,n,\sigma^*}(\lambda)\Big) + \mathbf{Adv}^{\mathsf{PRF}}(\lambda) + \mathsf{negl}(\lambda). \end{aligned}$$

Proof. We prove the claim via a sequence of sub-hybrids

$$\mathsf{Hyb}_{s-1} \approx_c \mathsf{Hyb}_{s,0}, \ldots, \mathsf{Hyb}_{s,16} = \mathsf{Hyb}_s,$$

each of which only changes how the game behaves at signing query s. Changes are grouped in two new subroutines, $\mathsf{PartSimOffline}$ and $\mathsf{PartSimOnline}$, that are substituted to $\mathsf{SimOffline}$ in $\mathsf{Hyb}_{s,5}$ and to $\mathsf{SimOnline}$ in $\mathsf{Hyb}_{s,0}$ respectively to simulate the s-th crucial query with $i = h$. To shorten probability expressions, we introduce the shorthand

$$\rho_x := \Pr[\mathsf{Hyb}_x(1^\lambda) = 1].$$

$\underline{\mathsf{Hyb}_{s,0}}$ We change how the challenge c is generated: H_u samples c uniformly at random, computes $\mathbf{h}$, and then programs $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$ with c. If H_c was already defined on this input, it aborts.

To evaluate the min-entropy of $\lfloor \mathbf{h} \rceil_\nu$, we require honest signers to check that $\bar{\mathbf{D}}$ is full-rank over $\mathcal{R}_q$, where $\bar{\mathbf{D}}$ is the second component of the decomposed $\mathbf{D} = [\mathbf{d} \,|\, \bar{\mathbf{D}}]$. Up to a possible permutation of the columns, we may therefore assume that $\bar{\mathbf{D}}$ is of the form $[\bar{\mathbf{D}}' \,|\, \bar{\mathbf{D}}'']$ with $\bar{\mathbf{D}}' \in \mathcal{R}_q^{m \times m}$ invertible. It follows that $\mathbf{h} = \mathbf{d} + \bar{\mathbf{D}}'\mathbf{u}' + \bar{\mathbf{D}}''\mathbf{u}''$ with $\mathbf{u} = [\mathbf{u}' \,|\, \mathbf{u}''] \sim \mathcal{D}_{\sigma_u}^{m+(\bar{d}-m)}$. Since $\mathbf{u}'$ and $\mathbf{u}''$ are independent, $\mathbf{h}$ has min-entropy greater or equal to the min-entropy of $\bar{\mathbf{D}}'\mathbf{u}'$, which, since $\bar{\mathbf{D}}'$ is invertible, is equal to the min-entropy of $\mathbf{u}'$. Now, the min-entropy of a one-dimensional Gaussian of parameter σ_u over $\mathbb{Z}$ is given by $\log_2 \rho_{\sigma_u}(\mathbb{Z})$, which is ε-close to $\log_2(\sigma_u\sqrt{2\pi e})$ since $\sigma_u > \eta_\varepsilon(\mathbb{Z})$. It follows that $\mathbf{h}$ has min-entropy at least $m\varphi \log_2(\sigma_u\sqrt{2\pi e})$.

Finally, since preimages under the map $x \mapsto \lfloor x \rceil_\nu$ over $\mathbb{Z}_q$ have at most $2^\nu(1 + q/2^\nu - q_\nu)$ elements, the min-entropy of $\lfloor \mathbf{h} \rceil_\nu$ is reduced by at most $\nu + \log_2(1+\delta)$ per coefficient over $\mathbb{Z}$, where $\delta = q/2^\nu - q_\nu$.[5] In other words, the min-entropy of $\lfloor \mathbf{h} \rceil_\nu$ is at least $m\varphi\big(\log_2(\sigma_u\sqrt{2\pi e}) - \nu - \log_2(1+\delta)\big)$.

Therefore, the probability of any possible value of $\lfloor \mathbf{h} \rceil_\nu$ is bounded by $\big(2^\nu \cdot (1+\delta)/(\sigma_u\sqrt{2\pi e})\big)^{m\varphi}$, which is chosen as negligible in λ. Since the adversary makes at most $Q_c + Q_u$ queries to H_c (both directly and indirectly through H_u) and the number of attempts to program H_c by $\mathsf{PartSimOnline}$ is at most Q_u, by the union bound,

5. Recall $\lfloor x \rceil_\nu = x_{\mathsf{hi}} \bmod q_\nu$, which is always just x_{hi}, except for one value of x_{hi}, namely $x_{\mathsf{hi}} = q_\nu$. For all the other values we have at most 2^ν preimages. The number of preimages of 0 can be obtained by counting the the integers in $[0, 2^{\nu-1} - 1] \cup [2^\nu q_\nu - 2^{\nu-1}, q-1]$, which is $2^\nu + (q - 2^\nu q_\nu) = 2^\nu \cdot (1+\delta)$ in total. As we choose q very close to a power of two in our concrete parameter sets, δ is actually very small, and we can essentially ignore the $(1+\delta)$ factor when setting parameters.

the probability that $\mathsf{Hyb}_{s,0}$ aborts is negligible (assuming Q_u, Q_c are polynomial in λ). That is,

$$|\rho_{s,0} - \rho_{s-1}| \le \mathsf{negl}(\lambda)$$

$\underline{\mathsf{Hyb}_{s,1}\text{-}\mathsf{Hyb}_{s,4}}$ These steps are analogous to the proof for tRaccoon. In $\mathsf{Hyb}_{s,1}$, PartSimOnline uniformly samples pair-wise masks for all honest parties, i.e., $\mathbf{m}_{i,j} \xleftarrow{\$} \mathcal{R}_q^n$ for $i, j \in \mathcal{T} \cap \mathcal{H}$. On the other hand, pair-wise masks for corrupt parties are generated using PRF as before. Since honest parties never reuse $\mathbf{D}_i$, the security of PRF guarantees that its outputs are computationally indistinguishable from the uniform distribution over $\mathcal{R}_q^n$, that is,

$$|\rho_{s,1} - \rho_{s,0}| \le \mathbf{Adv}^{\mathsf{PRF}}(\lambda).$$

In $\mathsf{Hyb}_{s,2}$, PartSimOnline uniformly samples row mask $\mathbf{m}_i$ for every honest party, column mask $\mathbf{m}'_i$ for every honest party except h, and uniquely determines $\mathbf{m}'_h$ as follows:

$$\mathbf{m}'_h := \sum_{i \in \mathcal{T} \cap \mathcal{H}} \mathbf{m}_i - \sum_{i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}} \mathbf{m}'_i + \sum_{i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$$

Since pair-wise masks are generated uniformly in the previous hybrid and we set $\mathbf{m}'_h$ such that the sum of column masks and row masks cancel out, this hybrid remains perfectly indistinguishable from the previous one, that is,

$$\rho_{s,2} = \rho_{s,1}.$$

In $\mathsf{Hyb}_{s,3}$, PartSimOnline uniformly samples $\mathbf{z}_i$ for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ and determines their row masks as

$$\mathbf{m}'_i := \mathbf{z}_i - \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c - \mathbf{r}_i^* - \mathbf{R}_i \mathbf{u} + \mathbf{m}_i \bmod q.$$

Since $\mathbf{m}'_i$ was uniformly sampled in the previous hybrid, this hybrid remains perfectly indistinguishable from the previous one, that is,

$$\rho_{s,3} = \rho_{s,2}.$$

In $\mathsf{Hyb}_{s,4}$, we rewrite $\mathbf{z}_h$ by plugging in the current constraints on $\mathbf{m}'_h$ into it:

$$\begin{aligned} \mathbf{z}_h &= \mathbf{s}_h \cdot \lambda_{\mathcal{T},h} \cdot c + \mathbf{r}_h^* + \mathbf{R}_h \mathbf{u} + \mathbf{m}'_h - \mathbf{m}_h \\ &= \mathbf{s}c + \mathbf{r}_h^* + \mathbf{R}_h \mathbf{u} - \Big(\sum_{j \in \mathcal{T} \cap \mathcal{C}} \mathbf{s}_j \lambda_{\mathcal{T},j}\Big) \cdot c \\ &\quad - \sum_{i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}} (\mathbf{z}_i - \mathbf{r}_i^* - \mathbf{R}_i \mathbf{u}) + \mathbf{m} \bmod q \end{aligned}$$

where $\mathbf{m}$ is as defined in Step 5 of Hyb_s. That is, we expressed $\mathbf{z}_h$ in terms of the shared secret $\mathbf{s}$ and corrupt key shares. Looking ahead, we regard the first two (resp. first three) terms of the last expression as a session-wide response $\mathbf{z}^*$ (resp. shifted session-wide response $\mathbf{z}'$) and uses them for simulation throughout. Since this is a syntactic change, we have

$$\rho_{s,4} = \rho_{s,3}.$$

$\underline{\mathsf{Hyb}_{s,5}}$ The game guesses a query index $q^* \in [Q_u]$ just for the s-th crucial query in a new subroutine PartSimOffline (see Algorithm 6). The subroutine PartSimOffline internally samples $c, \mathbf{u}$ assuming that the q^*-th query to H_u is used by $\mathcal{O}_{\mathsf{Sign}_2}$ upon receiving $i = h = \min(\mathcal{T} \cap \mathcal{C})$. Assuming that the guess is correct, since $\mathbf{d}_{h,0}$ can be determined by the remaining elements in the transcript, we can set

$$\begin{aligned} \mathbf{z}' &:= \mathbf{s}c + \mathbf{r}_h^* + \mathbf{R}_h \mathbf{u} \\ \mathbf{d}_{h,0} &:= \mathbf{A}\mathbf{z}' - \mathbf{b}c - \mathbf{D}_{h,1}\mathbf{u} + \mathbf{e}c + \mathbf{e}_h^* + \mathbf{E}_h \mathbf{u} \end{aligned}$$

One can check the RHS of the above indeed coincides with $\mathbf{A}\mathbf{r}_h^* + \mathbf{e}_h^*$. Then we modify PartSimOnline such that it simulates $\mathbf{z}_h$ using $\mathbf{z}'$ preprocessed by PartSimOffline only if $\mathsf{ctr}_u = q^*$, i.e., if PartSimOnline gets triggered by the q^*-th query to H_u; else, it sets $\mathbf{z}_h = \perp$. If the input $(\mathsf{eid}, h, (\mathbf{D}_j)_{j \in \mathcal{T} \setminus \{h\}}, \mu)$ to the online signing oracle $\mathcal{O}_{\mathsf{Sign}_2}$ was not contained in the q^*-th query to H_u, then the game aborts. Conditioned on the correct guess, the view of the adversary is identical to the previous one, thus

$$\rho_{s,4} \le Q_u \cdot \rho_{s,5}.$$

$\underline{\mathsf{Hyb}_{s,6}}$ We change the way $\mathbf{D}_{h,1}$ is generated such that it is sampled uniformly from $\mathcal{R}_q^{m \times \bar{d}}$. By embedding a given instance in $\mathbf{D}_{h,1}$, we can construct a reduction solving the $\mathsf{hLWE}_{\mathcal{R},q,m,n,1,\sigma_E,\sigma^*,\mathcal{D}_{\sigma_u}}$ problem given an adversary distinguishing $\mathsf{Hyb}_{s,6}$ from $\mathsf{Hyb}_{s,5}$, that is,

$$|\rho_{s,6} - \rho_{s,5}| \le \bar{d} \cdot \mathbf{Adv}^{\mathsf{hLWE}}_{\mathcal{R},q,m,n,1,\sigma_E,\sigma^*,\mathcal{D}_{\sigma_u}}(\lambda).$$

Concretely, we replace k-th column $\mathbf{d}_{h,1}^{(k)}$ of $\mathbf{D}_{h,1}$ by a uniformly random vector for $k = 1, \ldots \bar{d}$. Upon receiving an Hint-LWE instance $(\mathbf{A}^*, \mathbf{y}^*, \mathbf{l}^*, \mathbf{c}^*)$, the reduction $\mathcal{B}$ sets $\mathbf{A} = \mathbf{A}^*$ and $\mathbf{d}_{h,1}^{(k)} = \mathbf{y}^*$. It then parses $\mathbf{l}^*, \mathbf{c}^*$ as

$$\mathbf{l}^* = \begin{pmatrix} \mathbf{r}_h^* + \mathbf{r}_{h,1}^{(k)} u^k \\ \mathbf{e}_h^* + \mathbf{e}_{h,1}^{(k)} u^k \end{pmatrix} \quad \mathbf{c}^* = u^{(k)}$$

and simulates the rest of $\mathsf{Hyb}_{s,5}$ or $\mathsf{Hyb}_{s,6}$ for $\mathcal{A}$ using the first component of $\mathbf{l}^*$ to compute $\mathbf{z}'$, the second component as the last two terms of $\mathbf{d}_{h,0}$, and $u^{(k)}$ as the k-th element of $\mathbf{u}$ that gets programmed by H_u. If the instance is real (resp. uniform), it is clear that the view of the adversary $\mathcal{A}$ is identical to that of $\mathsf{Hyb}_{s,5}$ (resp. $\mathsf{Hyb}_{s,6}$). The advantage of $\mathcal{B}$ is thus exactly the probability of $\mathcal{A}$ distinguishing $\mathsf{Hyb}_{s,5}$ from $\mathsf{Hyb}_{s,6}$.

$\underline{\mathsf{Hyb}_{s,7}}$, we first sample uniformly random $\mathbf{D}'_{h,1}$ and shift it by $\mathbf{A}\mathbf{R}_h + \mathbf{E}_h$ to obtain $\mathbf{D}_{h,1}$. We do so in order to drop $\mathbf{d}_{h,0}$'s dependency on $\mathbf{u}$. Moreover, the hybrid aborts if $\mathbf{D}'_{h,1}$ is not full-rank over $\mathcal{R}_q^{m \times \bar{d}}$. By Corollary 3.2, $\mathbf{D}'_{h,1}$ is *not* full-rank with probability at most $\varphi / q^{\bar{d}-m}$, which is negligible. Unless the PartSimOffline aborts, the view of the adversary is unchanged, thus $|\rho_{s,7} - \rho_{s,6}| \le \mathsf{negl}(\lambda)$.

$\underline{\mathsf{Hyb}_{s,8}}$ PartSimOffline generates $\mathbf{D}'_{h,1}$ by invoking a trapdoor generation algorithm GenTrap. The distribution of $\mathbf{D}'_{h,1}$ is indistinguishable with uniform distribution over full-rank matrices in $\mathcal{R}_q^{m \times \bar{d}}$ due to the computational indistinguishability of GenTrap (Theorem 2.8) under the

$\mathsf{dLWE}'_{\mathcal{R},m,m,q,\sigma_{\mathsf{td}}}$ assumption (since the LWE secret is also sampled from Gaussian with parameter σ_{td}), that is,

$$|\rho_{s,8} - \rho_{s,7}| \leq (\bar{d} - 2m) \cdot \mathbf{Adv}^{\mathsf{dLWE}'}_{\mathcal{R},m,m,q,\sigma_{\mathsf{td}}}(\lambda)$$

where $\bar{d} - 2m$ comes from the width of trapdoor $\mathbf{T}$ in Theorem 2.8.

$\underline{\mathsf{Hyb}_{s,9}}$ we change the way $\mathbf{u}$ is generated such that the game first samples uniform $\mathbf{w}'_h$ from $\mathcal{R}_q^m$ and then samples Gaussian $\mathbf{u}$ from the lattice coset $\Lambda_q^{\mathbf{w}'_h}(\mathbf{D}'_{h,1}) = \{\mathbf{x} \in \mathcal{R}^{\bar{d}} : \mathbf{D}'_{h,1}\mathbf{x} = \mathbf{w}'_h\}$ using the preimage sampling algorithm $\mathsf{SamplePre}$. The joint distribution of $(\mathbf{D}'_{h,1}, \mathbf{u}, \mathbf{w}'_h)$ in $\mathsf{Hyb}_{s,9}$ is statistically indistinguishable from the one in $\mathsf{Hyb}_{s,8}$ due to the property of $\mathsf{SamplePre}$ (Theorem 2.8), that is,

$$|\rho_{s,9} - \rho_{s,8}| \leq \mathsf{negl}(\lambda).$$

$\underline{\mathsf{Hyb}_{s,10}}$ We apply minor syntactic changes to $\mathsf{PartSimOffline}$ such that it first sets a session-wide response $\mathbf{z}^*$ determined by $\mathbf{s}$ and compute $\mathbf{z}_h$ and $\mathbf{d}_{h,0}$ from $\mathbf{z}^*$:

$$\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^* \qquad \mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$$
$$\mathbf{d}_{h,0} := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{w}'_h + \mathbf{e}c + \mathbf{e}_h^*$$

This is equivalent to the previous hybrid, thus $\rho_{s,10} = \rho_{s,9}$.

$\underline{\mathsf{Hyb}_{s,11}}$ We apply minor syntactic changes to $\mathsf{PartSimOffline}$ such that it first generates $\mathbf{d}_{h,0}$ uniformly and sets $\mathbf{w}'_h$. This is equivalent to the previous hybrid, thus

$$\rho_{s,11} = \rho_{s,10}.$$

$\underline{\mathsf{Hyb}_{s,12}}$ We apply minor syntactic changes to $\mathsf{PartSimOffline}$ and $\mathsf{PartSimOnline}$ so that most work of transcript simulation is deferred to $\mathsf{PartSimOnline}$. In this subhybrid, $\mathsf{PartSimOffline}$ generates a uniformly random $\mathbf{d}_{h,0}$ from $\mathcal{R}_q^m$, and then later $\mathsf{PartSimOnline}$ sets

$$\mathbf{w}'_h = \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$$

after generating $c, \mathbf{r}_h^*, \mathbf{e}_h^*$ from the corresponding distributions and setting $\mathbf{z}^*$ as before. Note that in the previous subhybrid $c, \mathbf{z}^*, \mathbf{e}_h^*, \mathbf{w}'_h$ are not revealed when $\mathbf{d}_{h,0}$ is output by $\mathcal{O}_{\mathsf{Sign}_1}$. Thus, this hybrid is equivalent to the previous one, that is, $\rho_{s,12} = \rho_{s,11}$.

$\underline{\mathsf{Hyb}_{s,13}}$ We apply minor syntactic changes to $\mathsf{PartSimOnline}$. For all $\bar{q}$-th queries to H_u with $\bar{q} \neq q^*$, we have $\mathsf{PartSimOnline}$ (triggered by queries to H_u) define the target vector $\mathbf{w}'_h$ as follows:

$$\mathbf{w}'_h := \mathbf{b}' + \mathbf{A}\mathbf{s}c - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c$$

where $\mathbf{b}' \xleftarrow{\$} \mathcal{R}_q^m$ and $\mathbf{d}_{h,0}$ is the first column of $\mathbf{D}_h$. This hybrid is equivalent to the previous one, that is, $\rho_{s,13} = \rho_{s,12}$

$\underline{\mathsf{Hyb}_{s,14}}$ We change the behavior of $\mathsf{PartSimOnline}$ such that it generates $\mathbf{b}'$ as an LWE instance. Concretely, instead of sampling uniformly, we let $\mathbf{b}' = \mathbf{A}\mathbf{r}_h^* + \mathbf{e}_h^*$ and define the target vector $\mathbf{w}'_h$ using $\mathbf{b}'$ as before. By embedding a given instance in $\mathbf{b}'$, we can construct a reduction solving the $\mathsf{dLWE}'_{\mathcal{R},m,n,q,\sigma^*}$ problem given an adversary distinguishing $\mathsf{Hyb}_{s,14}$ from $\mathsf{Hyb}_{s,13}$, that is,

$$|\rho_{s,14} - \rho_{s,13}| \leq Q_u \cdot \mathbf{Adv}^{\mathsf{dLWE}'}_{\mathcal{R},m,n,q,\sigma^*}(\lambda)$$

$\underline{\mathsf{Hyb}_{s,15}}$ We apply minor syntactic changes to $\mathsf{PartSimOnline}$ such that it prepares simulated transcripts for all queries to H_u. $\rho_{s,15} = \rho_{s,14}$

$\underline{\mathsf{Hyb}_{s,16}}$ Note that at this stage, no guessing argument is required since $\mathbf{d}_{h,0}$ is uniform sampled independently of the rest of the transcript and H_u can prepare transcripts for arbitrary queries using the trapdoor for $\mathbf{D}_{h,1}$. Thus, in $\mathsf{Hyb}_{s,16}$ the oracle $\mathcal{O}_{\mathsf{Sign}_2}$ does not abort even when the guess is incorrect.

$$\rho_{s,15} = \rho_{s,16}/Q_u.$$

$\mathsf{Hyb}_{s,16}$ is in fact equivalent to Hyb_s, where the s-th signing query is answered using a simulated transcript, i.e. $\rho_{s,16} = \rho_s$. This concludes the proof. □

Algorithm 2: Hyb_s for $s = 0, \dots, Q$: Simulating the first s signing queries with input $(h, \mathcal{T})$

```
Gen(1^ℓ, 1^t)
1:  $\mathbf{s} \xleftarrow{\$} \mathcal{D}^n_{\sigma_e}$
2:  $\mathbf{e} \leftarrow \mathcal{D}^m_{\sigma_e}$
3:  $\mathbf{b} := \mathbf{A}\mathbf{s} + \mathbf{e} \bmod q$
4:  $\mathsf{pk} := \tilde{\mathbf{b}} = \lfloor \mathbf{b} \rceil_\xi$
5:  $(\mathbf{s}_1, \dots, \mathbf{s}_\ell) \leftarrow \mathsf{Share}(\mathbf{s}, q, t, \ell)$
6:  for $i \in [\ell]$ do
7:      for $j \in [\ell]$ do
8:          $\mathsf{sd}_{i,j} \xleftarrow{\$} \{0,1\}^l$
9:  for $i \in [\ell]$ do
10:     $\mathsf{sk}_i := (\mathbf{s}_i, (\mathsf{sd}_{i,j}, \mathsf{sd}_{j,i})_{j \in [\ell]})$
11: return $(\mathsf{pk}, (\mathsf{sk}_i)_{i \in [\ell]})$
```

```
BaD_δ((D_j)_{j∈T})
1:  $\mathbf{D} := \sum_{j \in \mathcal{T}} \mathbf{D}_j$
2:  $[\mathbf{d} | \bar{\mathbf{D}}] := \mathbf{D}$
3:  if $\bar{\mathbf{D}} \in \mathcal{R}_q^{m \times \bar{d}}$ is not full-rank then
4:      return 1
5:  else
6:      return 0
```

```
O_Sign1(i, T)
1:  if $(i \notin \mathcal{H}) \lor (i \notin \mathcal{T}) \lor (|\mathcal{T}| < t) \lor (\mathcal{T} \not\subseteq [\ell])$ then
2:      return $\perp$
3:  $\mathsf{eid} \xleftarrow{\$} \{0,1\}^*$
4:  $h := \min(\mathcal{T} \cap \mathcal{H})$
5:  if $i = h$ then
6:      Increment $\mathsf{ctr}_{\mathsf{min}}$
7:      if $\mathsf{ctr}_{\mathsf{min}} > s$ then
8:          $\mathbf{D}_i \leftarrow \mathsf{GenOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
9:      else
10:         $\mathbf{D}_i \leftarrow \mathsf{SimOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
11: else
12:     $\mathbf{D}_i \leftarrow \mathsf{GenOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
13: if $(\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$ is defined for some $((\mathbf{D}_j)_{j \in \mathcal{T} \setminus \{i\}}, \mu)) \lor$
    $(\exists(\text{OFFLINE}, \mathsf{eid}', i, \mathcal{T}, \mathbf{D}_i, \dots)$ for some $\mathsf{eid}' \neq \mathsf{eid})$ then
14:     Abort
15: return $(\mathsf{eid}, \mathbf{D}_i)$
```

```
O_Sign2(eid, i, (D_j)_{j∈T\{i}}, μ)
1:  if $(\nexists(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \dots)) \lor \exists(\text{FINISH}, \mathsf{eid}, i) \lor (\exists\, j \in (\mathcal{T} \cap \mathcal{H}) \setminus$
    $\{i\} \mid \nexists\, (\text{OFFLINE}, \cdot, j, \mathcal{T}, \mathbf{D}_j, \dots))$ then
2:      return $\perp$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, \dots, \mathsf{ctr}_{\mathsf{min}})$
4:  if $\mathsf{BaD}_\delta((\mathbf{D}_j)_{j \in \mathcal{T}})$ then
5:      return $\perp$
6:  Run $\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$
7:  Load $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
8:  $\mathcal{M} := \mathcal{M} \cup \{\mu\}$
9:  Store $(\text{FINISH}, \mathsf{eid}, i)$
10: return $\mathbf{z}_i$
```

```
H_u(pp, pk, T, (D_j)_{j∈T}, μ)
1:  if the response is defined as $\mathbf{u}$ then
2:      return $\mathbf{u}$
3:  Increment $\mathsf{ctr}_u$
4:  if $(\exists j \in \mathcal{T} \cap \mathcal{H} \mid \nexists(\text{OFFLINE}, \cdot, j, \mathcal{T}, \mathbf{D}_j, \dots)) \lor \mathsf{BaD}_\delta((\mathbf{D}_j)_{j \in \mathcal{T}})$ then
5:      $\mathbf{u} \leftarrow \mathcal{D}^{\bar{d}}_{\sigma_u}$
6:      $\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T} \cap \mathcal{H}}, \mu) := \mathbf{u}$
7:      return $\mathbf{u}$
8:  $h := \min(\mathcal{T} \cap \mathcal{H})$
9:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, (\dots), \mathsf{ctr}_{\mathsf{min}})$
10: if $\mathsf{ctr}_{\mathsf{min}} > s$ then
11:     $(\mathbf{u}, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}}) \leftarrow \mathsf{GenOnline}(\mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$
12: else
13:     $(\mathbf{u}, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}}) \leftarrow \mathsf{SimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$
14: $\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu) := \mathbf{u}$
15: Store $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
16: return $\mathbf{u}$
```

```
H_c(pp, pk, ⌊h⌉_ν, μ)
1:  if the response is defined as $c$ then
2:      return $c$
3:  else
4:      $c \xleftarrow{\$} C$
5:      $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu) := c$
6:      return $c$
```

```
GenOffline(eid, i, T, ctr_min)
1:  $\mathbf{r}_i^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_i^* \leftarrow \mathcal{D}^m_{\sigma^*}$
2:  $\mathbf{R}_i \leftarrow \mathcal{D}^{n \times \bar{d}}_{\sigma_E}$; $\mathbf{E}_i \leftarrow \mathcal{D}^{m \times \bar{d}}_{\sigma_E}$
3:  $\mathbf{D}_i := [\mathbf{d}_{i,0} | \mathbf{D}_{i,1}] = \mathbf{A}[\mathbf{r}_i^* \,|\, \mathbf{R}_i] + [\mathbf{e}_i^* \,|\, \mathbf{E}_i] \bmod q \in \mathcal{R}_q^{m \times d}$
4:  Store $(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \mathsf{ctr}_{\mathsf{min}})$
5:  return $(\mathbf{D}_i)$
```

```
SimOffline(eid, h, T, ctr_min)
1:  $\mathbf{R}_h \leftarrow \mathcal{D}^{n \times \bar{d}}_{\sigma_E}$; $\mathbf{E}_h \leftarrow \mathcal{D}^{m \times \bar{d}}_{\sigma_E}$
2:  $(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m, \bar{d})$
3:  if $\mathbf{D}'_{h,1}$ is not full rank then
4:      Ask caller to abort
5:  $\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
6:  $\mathbf{d}_{h,0} \xleftarrow{\$} \mathcal{R}_q^m$
7:  $\mathbf{D}_h := [\mathbf{d}_{h,0} \,|\, \mathbf{D}_{h,1}]$
8:  Store $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, (\mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h), \mathsf{ctr}_{\mathsf{min}})$
9:  return $(\mathbf{D}_h)$
```

```
GenOnline(T, (D_j)_{j∈T}, μ)
1:  for $i \in \mathcal{T} \cap \mathcal{H}$ do
2:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3:  $\mathbf{u} \leftarrow \mathcal{D}^{\bar{d}}_{\sigma_u}$
4:  $\mathbf{D} := \sum_{j \in \mathcal{T}} \mathbf{D}_j$
5:  $\mathbf{h} := \mathbf{D} \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix} \bmod q$
6:  $c \leftarrow \mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$
7:  for $i \in \mathcal{T} \cap \mathcal{H}$ do
8:      $\mathbf{m}_i := \sum_{j \in \mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu))$
9:      $\mathbf{m}'_i := \sum_{j \in \mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu))$
10:     $\mathbf{z}_i := \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \,|\, \mathbf{R}_i] \cdot \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix} + \mathbf{m}'_i - \mathbf{m}_i \bmod q$
11: return $(\mathbf{u}, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
```

```
SimOnline(T, (D_j)_{j∈T}, μ)
1:  $h := \min(\mathcal{T} \cap \mathcal{H})$
2:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, (\mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h), \cdot)$
3:  for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
4:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
5:  $[\mathbf{d}_{h,0} \,|\, \mathbf{D}_{h,1}] := \mathbf{D}_h$
6:  $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}^m_{\sigma^*}$
7:  $\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^*$
8:  $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
9:  $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathbf{D}'_{h,1}, \mathsf{td}_h, \mathbf{w}'_h)$
10: $\mathbf{D} := \sum_{j \in \mathcal{T}} \mathbf{D}_j$
11: $\mathbf{h} := \mathbf{D} \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix} \bmod q$
12: if $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$ is defined then
13:     Ask caller to abort
14: $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu) := c$
15: for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
16:     $\mathbf{z}_i \xleftarrow{\$} \mathcal{R}_q^n$
17: for $i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}$ do
18:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k \in \mathcal{T}}, \mu))$
19:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k \in \mathcal{T}}, \mu))$
20: $\mathbf{m} := \sum_{i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
21: $\mathbf{z}_h := \mathbf{z}^* + \mathbf{R}_h\mathbf{u} - (\sum_{j \in \mathcal{T} \cap \mathcal{C}} \mathbf{s}_j \cdot \lambda_{\mathcal{T},j})c$
        $- \sum_{i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}} (\mathbf{z}_i - \mathbf{r}_i^* - \mathbf{R}_i\mathbf{u}) + \mathbf{m}$
22: return $(\mathbf{u}, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
```

Algorithm 3: $\mathsf{Hyb}_{s,0}$: Abort if H_c is defined

```
$\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$

    ⋮ // Identical to the previous hybrid
10: if $\mathsf{ctr}_{\mathsf{min}} > s$ then
11:     $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}}) \leftarrow \mathsf{GenOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
12: else if $\mathsf{ctr}_{\mathsf{min}} = s$ then
13:     $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}}) \leftarrow \mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
14: else
15:     $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}}) \leftarrow \mathsf{SimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
16: $\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu) := \mathbf{u}$
17: Store $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
18: return $\mathbf{u}$
```

```
$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
1: for $i \in \mathcal{T}\cap\mathcal{H}$ do
2:     Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3: $c \xleftarrow{\$} C$; $\mathbf{u} \leftarrow \mathcal{D}_{\sigma_u}^{\bar{d}}$
4: $\mathbf{D} := \sum_{j\in\mathcal{T}} \mathbf{D}_j$
5: $\mathbf{h} := \mathbf{D}\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$
6: if $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor\mathbf{h}\rceil_\nu, \mu)$ is defined then
7:     Ask caller to abort
8: $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor\mathbf{h}\rceil_\nu, \mu) := c$
9: for $i \in \mathcal{T}\cap\mathcal{H}$ do
10:     $\mathbf{m}_i := \sum_{j\in\mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu))$
11:     $\mathbf{m}_i' := \sum_{j\in\mathcal{T}} \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu))$
12:     $\mathbf{z}_i := \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \mid \mathbf{R}_i] \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} + \mathbf{m}_i' - \mathbf{m}_i \bmod q$
13: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```

Algorithm 4: $\mathsf{Hyb}_{s,x}$ for $x = 1, 2$: Truly random pair-wise masks for honest parties, determine a column mask for party h

```
$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 1$

    ⋮ // Identical to the previous hybrid
9: for $i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{H}$ do
10:     $\mathbf{m}_{i,j} \xleftarrow{\$} \mathcal{R}_q^n$
11: for $i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{C}$ do
12:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
13:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
14: for $i \in \mathcal{T}\cap\mathcal{H}$ do
15:     $\mathbf{m}_i := \sum_{j\in\mathcal{T}} \mathbf{m}_{i,j}$
16:     $\mathbf{m}_i' := \sum_{j\in\mathcal{T}} \mathbf{m}_{j,i}$
17:     $\mathbf{z}_i := \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \mid \mathbf{R}_i] \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} + \mathbf{m}_i' - \mathbf{m}_i \bmod q$
18: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```

```
$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 2$

    ⋮ // Identical to the previous hybrid
9: for $i \in \mathcal{T}\cap\mathcal{H}\setminus\{h\}$ do
10:     $\mathbf{m}_i \xleftarrow{\$} \mathcal{R}_q^n$
11:     $\mathbf{m}_i' \xleftarrow{\$} \mathcal{R}_q^n$
12: $\mathbf{m}_h \xleftarrow{\$} \mathcal{R}_q^n$
13: for $i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{C}$ do
14:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
15:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
16: $\mathbf{m}_h' := \sum_{i\in\mathcal{T}\cap\mathcal{H}} \mathbf{m}_i - \sum_{i\in\mathcal{T}\cap\mathcal{H}\setminus\{h\}} \mathbf{m}_i' + \sum_{i\in\mathcal{T}\cap\mathcal{H}, j\in\mathcal{T}\cap\mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
17: for $i \in \mathcal{T}\cap\mathcal{H}$ do
18:     $\mathbf{z}_i := \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c + [\mathbf{r}_i^* \mid \mathbf{R}_i] \cdot \begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} + \mathbf{m}_i' - \mathbf{m}_i \bmod q$
19: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```

Algorithm 5: $\mathsf{Hyb}_{s,x}$ for $x = 3, 4$: Sample all-but-one $\mathbf{z}_i$ uniformly, determine $\mathbf{z}_h$ from $\mathbf{s}$ and corrupt shares

```
$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 3$

    ⋮ // Identical to the previous hybrid
9: for $i \in \mathcal{T}\cap\mathcal{H}\setminus\{h\}$ do
10:     $\mathbf{m}_i \xleftarrow{\$} \mathcal{R}_q^n$
11:     $\mathbf{z}_i \xleftarrow{\$} \mathcal{R}_q^n$
12:     $\mathbf{m}_i' := \mathbf{z}_i - \mathbf{s}_i \cdot \lambda_{\mathcal{T},i} \cdot c - \mathbf{r}_i^* - \mathbf{R}_i\mathbf{u} + \mathbf{m}_i \bmod q$
13: $\mathbf{m}_h \xleftarrow{\$} \mathcal{R}_q^n$
14: for $i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{C}$ do
15:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
16:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
17: $\mathbf{m} := \sum_{i\in\mathcal{T}\cap\mathcal{H}, j\in\mathcal{T}\cap\mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
18: $\mathbf{m}_h' := \mathbf{m} + \sum_{i\in\mathcal{T}\cap\mathcal{H}} \mathbf{m}_i - \sum_{i\in\mathcal{T}\cap\mathcal{H}\setminus\{h\}} \mathbf{m}_i'$
19: $\mathbf{z}_h := \mathbf{s}_h \cdot \lambda_{\mathcal{T},h} \cdot c + \mathbf{r}_h^* + \mathbf{R}_h\mathbf{u} + \mathbf{m}_h' - \mathbf{m}_h \bmod q$
20: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```

```
$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 4$

    ⋮ // Identical to the previous hybrid
9: for $i \in \mathcal{T}\cap\mathcal{H}\setminus\{h\}$ do
10:     $\mathbf{z}_i \xleftarrow{\$} \mathcal{R}_q^n$
11: for $i \in \mathcal{T}\cap\mathcal{H}, j \in \mathcal{T}\cap\mathcal{C}$ do
12:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
13:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
14: $\mathbf{m} := \sum_{i\in\mathcal{T}\cap\mathcal{H}, j\in\mathcal{T}\cap\mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
15: $\mathbf{z}_h := \mathbf{s}c + \mathbf{r}_h^* + \mathbf{R}_h\mathbf{u} - (\sum_{j\in\mathcal{T}\cap\mathcal{C}} \mathbf{s}_j\lambda_{\mathcal{T},j}) \cdot c$
        $- \sum_{i\in\mathcal{T}\cap\mathcal{H}\setminus\{h\}} (\mathbf{z}_i - \mathbf{r}_i^* - \mathbf{R}_i\mathbf{u}) + \mathbf{m} \bmod q$
16: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```

Algorithm 6: $\mathsf{Hyb}_{s,x}$ for $x = 5, 6, 7, 8$: Guess a critical query to H_u, replace $\mathbf{D}_{h,1}$ with pseudorandom matrix with a trapdoor

```
$\mathcal{O}_{\mathsf{Sign}_1}(i, \mathcal{T})$
 1: if $(i \notin \mathcal{H}) \lor (i \notin \mathcal{T}) \lor (|\mathcal{T}| < t) \lor (\mathcal{T} \nsubseteq [\ell])$ then
 2:     return $\perp$
 3: $\mathsf{eid} \xleftarrow{\$} \{0,1\}^*$
 4: $h := \min(\mathcal{T} \cap \mathcal{H})$
 5: if $i = h$ then
 6:     Increment $\mathsf{ctr}_{\mathsf{min}}$
 7:     if $\mathsf{ctr}_{\mathsf{min}} > s$ then
 8:         $\mathbf{D}_i \leftarrow \mathsf{GenOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
 9:     else if $\mathsf{ctr}_{\mathsf{min}} = s$ then
10:         $\mathbf{D}_i \leftarrow \mathsf{PartSimOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
11:     else
12:         $\mathbf{D}_i \leftarrow \mathsf{SimOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
13: else
14:     $\mathbf{D}_i \leftarrow \mathsf{GenOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
15: if $(\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$ is defined for some $((\mathbf{D}_j)_{j \in \mathcal{T} \setminus \{i\}}, \mu)) \lor$
    $(\exists(\text{OFFLINE}, \mathsf{eid}', i, \mathcal{T}, \mathbf{D}_i, \ldots)$ for some $\mathsf{eid}' \neq \mathsf{eid})$ then
16:     Abort
17: return $(\mathsf{eid}, \mathbf{D}_i)$

$\mathsf{PartSimOffline}(\mathsf{eid}, h, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$
 1: $q^* \xleftarrow{\$} [Q_u]$
 2: $\mathbf{R}_h \leftarrow \mathcal{D}_{\sigma_E}^{n \times \bar{d}}$; $\mathbf{E}_h \leftarrow \mathcal{D}_{\sigma_E}^{m \times \bar{d}}$
 3: if $x = 5$ then
 4:     $\mathbf{D}_{h,1} := \mathbf{A}\mathbf{R}_h + \mathbf{E}_h \bmod q \in \mathcal{R}_q^{m \times \bar{d}}$
 5: else if $x = 6$ then
 6:     $\mathbf{D}_{h,1} \xleftarrow{\$} \mathcal{R}_q^{m \times \bar{d}}$
 7: else if $x = 7$ then
 8:     $\mathbf{D}'_{h,1} \xleftarrow{\$} \mathcal{R}_q^{m \times \bar{d}}$
 9:     if $\mathbf{D}'_{h,1}$ is not full rank then
10:         Ask caller to abort
11:     $\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
12: else
13:     $(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
14:     if $\mathbf{D}'_{h,1}$ is not full rank then
15:         Ask caller to abort
16:     $\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
17: $\mathbf{u} \leftarrow \mathcal{D}_{\sigma_u}^{\bar{d}}$
18: $\mathbf{w}_h := \mathbf{D}_{h,1}\mathbf{u}$
19: $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}_{\sigma^*}^{n}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}_{\sigma^*}^{m}$
20: $\mathbf{z}' := \mathbf{s}c + \mathbf{r}_h^* + \mathbf{R}_h\mathbf{u}$
21: $\mathbf{d}_{h,0} := \mathbf{A}\mathbf{z}' - \mathbf{b}c - \mathbf{w}_h + \mathbf{e}c + \mathbf{e}_h^* + \mathbf{E}_h\mathbf{u}$
22: $\mathbf{D}_h := [\mathbf{d}_{h,0} | \mathbf{D}_{h,1}]$
23: Store $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, c, \mathbf{u}, \mathbf{z}', \mathsf{ctr}_{\mathsf{min}})$
24: return $\mathbf{D}_h$
```

```
$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{eid}, i, (\mathbf{D}_j)_{j \in \mathcal{T} \setminus \{i\}}, \mu)$
    $\vdots$ // Identical to the previous hybrid
 7: Load $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
 8: if $i = h \land \mathsf{ctr}_{\mathsf{min}} = s \land \mathsf{ctr}_u \neq q^*$ then
 9:     Abort
10: $\mathcal{M} := \mathcal{M} \cup \{\mu\}$
11: Store $(\text{FINISH}, \mathsf{eid}, i)$
12: return $(\mathbf{z}_i)$

$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j \in \mathcal{T}}, \mu)$
 1: for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
 2:     Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
 3: if $\mathsf{ctr}_u = q^*$ then
 4:     Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, c, \mathbf{u}, \mathbf{z}', \mathsf{ctr}_{\mathsf{min}})$
 5: else
 6:     $c \xleftarrow{\$} C$; $\mathbf{u} \leftarrow \mathcal{D}_{\sigma_u}^{\bar{d}}$
 7:     $\mathbf{z}' := \perp$
 8: $\mathbf{D} := \sum_{j \in \mathcal{T}} \mathbf{D}_j$
 9: $\mathbf{h} := \mathbf{D} \begin{pmatrix} 1 \\ \mathbf{u} \end{pmatrix} \bmod q$
10: if $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu)$ is defined then
11:     Ask caller to abort
12: $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor \mathbf{h} \rceil_\nu, \mu) := c$
13: for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
14:     $\mathbf{z}_i \xleftarrow{\$} \mathcal{R}_q^n$
15: for $i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}$ do
16:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k \in \mathcal{T}}, \mu))$
17:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k \in \mathcal{T}}, \mu))$
18: $\mathbf{m} := \sum_{i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
19: if $\mathbf{z}' = \perp$ then
20:     $\mathbf{z}_h := \perp$
21: else
22:     $\mathbf{z}_h := \mathbf{z}' - (\sum_{j \in \mathcal{T} \cap \mathcal{C}} \mathbf{s}_j \lambda_{\mathcal{T},j}) \cdot c$
            $- \sum_{i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}} (\mathbf{z}_i - \mathbf{r}_i^* - \mathbf{R}_i\mathbf{u}) + \mathbf{m} \bmod q$
23: return $(\mathbf{u}, (\mathbf{z}_j)_{j \in \mathcal{T} \cap \mathcal{H}})$
```

Algorithm 7: $\mathsf{Hyb}_{s,9}$: Sample $\mathbf{u}$ using a trapdoor for (shifted) $\mathbf{D}_{h,1}$

```
PartSimOffline(eid, $i$, $\mathcal{T}$, $\mathsf{ctr}_{\mathsf{min}}$)
$q^* \xleftarrow{\$} [Q_u]$
$\mathbf{R}_h \leftarrow \mathcal{D}_{\sigma_E}^{n\times\bar{d}}$; $\mathbf{E}_h \leftarrow \mathcal{D}_{\sigma_E}^{m\times\bar{d}}$
$(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
if $\mathbf{D}'_{h,1}$ is not full rank then
    Ask caller to abort
$\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
$\mathbf{w}'_h \xleftarrow{\$} \mathcal{R}_q^m$
$\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
$\mathbf{w}_h := \mathbf{w}'_h + (\mathbf{A}\mathbf{R}_h + \mathbf{E}_h)\mathbf{u}$
$c \xleftarrow{\$} C$; $\mathbf{r}^*_h \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}^*_h \leftarrow \mathcal{D}^m_{\sigma^*}$
$\mathbf{z}' := \mathbf{s}c + \mathbf{r}^*_h + \mathbf{R}_h\mathbf{u}$
$\mathbf{d}_{h,0} := \mathbf{A}\mathbf{z}' - \mathbf{b}c - \mathbf{w}_h + \mathbf{e}c + \mathbf{e}^*_h + \mathbf{E}_h\mathbf{u}$
$\mathbf{D}_h := [\mathbf{d}_{h,0}|\mathbf{D}_{h,1}]$
Store ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $c$, $\mathbf{u}$, $\mathbf{z}'$, $\mathbf{D}'_h$, $\mathsf{td}_h$, $\mathsf{ctr}_{\mathsf{min}}$)
return $\mathbf{D}_h$
```

```
PartSimOnline($\mathcal{T}$, $(\mathbf{D}_j)_{j\in\mathcal{T}}$, $\mu$)
for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
    Load ($\textsc{offline}$, $\cdot$, $i$, $\mathcal{T}$, $\mathbf{D}_i$, $(\mathbf{r}^*_i, \mathbf{R}_i)$, $\cdot$)
if $\mathsf{ctr}_u = q^*$ then
    Load ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $c$, $\mathbf{u}$, $\mathbf{z}'$, ...)
else
    Load ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, ..., $\mathbf{D}'_h$, $\mathsf{td}_h$, $\cdot$)
    $c \xleftarrow{\$} C$
    $\mathbf{w}'_h \xleftarrow{\$} \mathcal{R}_q^m$
    $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
    $\mathbf{z}' := \perp$
⋮ // Identical to the previous
```

Algorithm 8: $\mathsf{Hyb}_{s,x}$ for $x = 10, 11$: Syntactic changes

```
PartSimOffline(eid, $i$, $\mathcal{T}$, $\mathsf{ctr}_{\mathsf{min}}$) for $x = 10$
$q^* \xleftarrow{\$} [Q_u]$
$\mathbf{R}_h \leftarrow \mathcal{D}_{\sigma_E}^{n\times\bar{d}}$; $\mathbf{E}_h \leftarrow \mathcal{D}_{\sigma_E}^{m\times\bar{d}}$
$(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
if $\mathbf{D}'_{h,1}$ is not full rank then
    Ask caller to abort
$\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
$\mathbf{w}'_h \xleftarrow{\$} \mathcal{R}_q^m$
$\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
$\mathbf{w}_h := \mathbf{w}'_h + (\mathbf{A}\mathbf{R}_h + \mathbf{E}_h)\mathbf{u}$
$c \xleftarrow{\$} C$; $\mathbf{r}^*_h \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}^*_h \leftarrow \mathcal{D}^m_{\sigma^*}$
$\mathbf{z}^* := \mathbf{s}c + \mathbf{r}^*_h$
$\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
$\mathbf{d}_{h,0} := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{w}'_h + \mathbf{e}c + \mathbf{e}^*_h$
$\mathbf{D}_h := [\mathbf{d}_{h,0}|\mathbf{D}_{h,1}]$
Store ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $c$, $\mathbf{u}$, $\mathbf{z}'$, $\mathbf{D}'_h$, $\mathsf{td}_h$, $\mathsf{ctr}_{\mathsf{min}}$)
return $\mathbf{D}_h$
```

```
PartSimOffline(eid, $i$, $\mathcal{T}$, $\mathsf{ctr}_{\mathsf{min}}$) for $x = 11$
$q^* \xleftarrow{\$} [Q_u]$
$\mathbf{R}_h \leftarrow \mathcal{D}_{\sigma_E}^{n\times\bar{d}}$; $\mathbf{E}_h \leftarrow \mathcal{D}_{\sigma_E}^{m\times\bar{d}}$
$(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
if $\mathbf{D}'_{h,1}$ is not full rank then
    Ask caller to abort
$\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
$\mathbf{d}_{h,0} \xleftarrow{\$} \mathcal{R}_q^m$
$c \xleftarrow{\$} C$; $\mathbf{r}^*_h \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}^*_h \leftarrow \mathcal{D}^m_{\sigma^*}$
$\mathbf{z}^* := \mathbf{s}c + \mathbf{r}^*_h$
$\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}^*_h$
$\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
$\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
$\mathbf{D}_h := [\mathbf{d}_{h,0}|\mathbf{D}_{h,1}]$
Store ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $c$, $\mathbf{u}$, $\mathbf{z}'$, $\mathbf{D}'_h$, $\mathsf{td}_h$, $\mathsf{ctr}_{\mathsf{min}}$)
return $\mathbf{D}_h$
```

Algorithm 9: $\mathsf{Hyb}_{s,12}$: Defer generation of $c, \mathbf{u}, \mathbf{z}'$ to PartSimOnline

```
PartSimOffline(eid, $h$, $\mathcal{T}$, $\mathsf{ctr}_{\mathsf{min}}$)
$q^* \xleftarrow{\$} [Q_u]$
$\mathbf{R}_h \leftarrow \mathcal{D}_{\sigma_E}^{n\times\bar{d}}$; $\mathbf{E}_h \leftarrow \mathcal{D}_{\sigma_E}^{m\times\bar{d}}$
$(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
if $\mathbf{D}'_{h,1}$ is not full rank then
    Ask caller to abort
$\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
$\mathbf{d}_{h,0} \xleftarrow{\$} \mathcal{R}_q^m$
$\mathbf{D}_h := [\mathbf{d}_{h,0}|\mathbf{D}_{h,1}]$
Store ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $\mathbf{D}'_h$, $\mathbf{R}_h$, $\mathsf{td}_h$, $\mathsf{ctr}_{\mathsf{min}}$)
return $\mathbf{D}_h$
```

```
PartSimOnline($\mathcal{T}$, $(\mathbf{D}_j)_{j\in\mathcal{T}}$, $\mu$)
for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
    Load ($\textsc{offline}$, $\cdot$, $i$, $\mathcal{T}$, $\mathbf{D}_i$, $(\mathbf{r}^*_i, \mathbf{R}_i)$, $\cdot$)
Load ($\textsc{offline}$, eid, $h$, $\mathcal{T}$, $\mathbf{D}_h$, $\mathbf{D}'_h$, $\mathbf{R}_h$, $\mathsf{td}_h$, $\cdot$)
$[\mathbf{d}_{h,0}|\mathbf{D}_{h,1}] := \mathbf{D}_h$
$c \xleftarrow{\$} C$; $\mathbf{r}^*_h \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}^*_h \leftarrow \mathcal{D}^m_{\sigma^*}$
$\mathbf{z}^* := \mathbf{s}c + \mathbf{r}^*_h$
if $\mathsf{ctr}_u = q^*$ then
    $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}^*_h$
    $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
    $\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
else
    $\mathbf{w}'_h \xleftarrow{\$} \mathcal{R}_q^m$
    $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
    $\mathbf{z}' := \perp$
⋮ // Identical to the previous
```

Algorithm 10: $\mathsf{Hyb}_{s,x}$ for $x = 13, 14, 15$: Make $\mathsf{PartSimOnline}$ behave consistently for $\mathsf{ctr}_u = q^*$ and $\mathsf{ctr}_u \neq q^*$

$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 13$

```
1:  for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
2:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, \mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h, \cdot)$
4:  $[\mathbf{d}_{h,0}|\mathbf{D}_{h,1}] := \mathbf{D}_h$
5:  $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}^m_{\sigma^*}$
6:  $\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^*$
7:  if $\mathsf{ctr}_u = q^*$ then
8:      $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
9:      $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
10:     $\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
11: else
12:     $\mathbf{b}' \xleftarrow{\$} \mathcal{R}_q^m$
13:     $\mathbf{w}'_h := \mathbf{b}' + \mathbf{A}\mathbf{s}c - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c$
14:     $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
15:     $\mathbf{z}' := \bot$
    ⋮ // Identical to the previous
```

$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 14$

```
1:  for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
2:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, \mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h, \cdot)$
4:  $[\mathbf{d}_{h,0}|\mathbf{D}_{h,1}] := \mathbf{D}_h$
5:  $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}^m_{\sigma^*}$
6:  $\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^*$
7:  if $\mathsf{ctr}_u = q^*$ then
8:      $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
9:      $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
10:     $\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
11: else
12:     $\mathbf{b}' := \mathbf{A}\mathbf{r}_h^* + \mathbf{e}_h^*$
13:     $\mathbf{w}'_h := \mathbf{b}' + \mathbf{A}\mathbf{s}c - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c$
14:     $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
15:     $\mathbf{z}' := \bot$
    ⋮ // Identical to the previous
```

$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$ for $x = 15$

```
1:  for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
2:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, \mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h, \cdot)$
4:  $[\mathbf{d}_{h,0}|\mathbf{D}_{h,1}] := \mathbf{D}_h$
5:  $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}^m_{\sigma^*}$
6:  $\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^*$
7:  if $\mathsf{ctr}_u = q^*$ then
8:      $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
9:      $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
10:     $\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
11: else
12:     $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
13:     $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
14:     $\mathbf{z}' := \mathbf{z}^* + \mathbf{R}_h\mathbf{u}$
    ⋮ // Identical to the previous
```

Algorithm 11: $\mathsf{Hyb}_{s,16}$: Stop guessing a critical query to H_u

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{eid}, h, (\mathbf{D}_j)_{j\in\mathcal{T}\setminus\{h\}}, \mu)$

```
1:  if $(\nexists(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \ldots)) \lor \exists(\text{FINISH}, \mathsf{eid}, i) \lor (\exists\ j \in (\mathcal{T} \cap \mathcal{H}) \setminus$
    $\{i\} \mid \nexists\ (\text{OFFLINE}, \cdot, j, \mathcal{T}, \mathbf{D}_j, \ldots))$ then
2:      return $\bot$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, i, \mathcal{T}, \mathbf{D}_i, \ldots, \mathsf{ctr}_{\mathsf{min}})$
4:  if $\mathsf{BaD}_\delta((\mathbf{D}_j)_{j\in\mathcal{T}})$ then
5:      return $\bot$
6:  Run $\mathsf{H}_u(\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$
7:  Load $(\text{ONLINE}, \mathsf{ctr}_u, \mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
8:  $\mathcal{M} := \mathcal{M} \cup \{\mu\}$
9:  Store $(\text{FINISH}, \mathsf{eid}, i)$
10: return $\mathbf{z}_i$
```

$\mathsf{PartSimOffline}(\mathsf{eid}, i, \mathcal{T}, \mathsf{ctr}_{\mathsf{min}})$

```
1:  $\mathbf{R}_h \leftarrow \mathcal{D}^{n\times\bar{d}}_{\sigma_E}$; $\mathbf{E}_h \leftarrow \mathcal{D}^{m\times\bar{d}}_{\sigma_E}$
2:  $(\mathbf{D}'_{h,1}, \mathsf{td}_h) \leftarrow \mathsf{GenTrap}(\mathcal{R}_q, m)$
3:  if $\mathbf{D}'_{h,1}$ is not full rank then
4:      Ask caller to abort
5:  $\mathbf{D}_{h,1} := \mathbf{D}'_{h,1} + \mathbf{A}\mathbf{R}_h + \mathbf{E}_h$
6:  $\mathbf{d}_{h,0} \xleftarrow{\$} \mathcal{R}_q^m$
7:  $\mathbf{D}_h := [\mathbf{d}_{h,0}|\mathbf{D}_{h,1}]$
8:  Store $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, (\mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h), \mathsf{ctr}_{\mathsf{min}})$
9:  return $\mathbf{D}_h$
```

$\mathsf{PartSimOnline}(\mathcal{T}, (\mathbf{D}_j)_{j\in\mathcal{T}}, \mu)$

```
1:  for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
2:      Load $(\text{OFFLINE}, \cdot, i, \mathcal{T}, \mathbf{D}_i, (\mathbf{r}_i^*, \mathbf{R}_i), \cdot)$
3:  Load $(\text{OFFLINE}, \mathsf{eid}, h, \mathcal{T}, \mathbf{D}_h, \mathbf{D}'_h, \mathbf{R}_h, \mathsf{td}_h, \cdot)$
4:  $[\mathbf{d}_{h,0}|\mathbf{D}_{h,1}] := \mathbf{D}_h$
5:  $c \xleftarrow{\$} C$; $\mathbf{r}_h^* \leftarrow \mathcal{D}^n_{\sigma^*}$; $\mathbf{e}_h^* \leftarrow \mathcal{D}^m_{\sigma^*}$
6:  $\mathbf{z}^* := \mathbf{s}c + \mathbf{r}_h^*$
7:  $\mathbf{w}'_h := \mathbf{A}\mathbf{z}^* - \mathbf{b}c - \mathbf{d}_{h,0} + \mathbf{e}c + \mathbf{e}_h^*$
8:  $\mathbf{u} \leftarrow \mathsf{SamplePre}(\mathsf{td}_h, \mathbf{w}'_h, \mathbf{D}'_{h,1})$
9:  $\mathbf{D} := \sum_{j\in\mathcal{T}} \mathbf{D}_j$
10: $\mathbf{h} := \mathbf{D}\begin{pmatrix}1\\ \mathbf{u}\end{pmatrix} \bmod q$
11: if $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor\mathbf{h}\rceil_\nu, \mu)$ is defined then
12:     Ask caller to abort
13: $\mathsf{H}_c(\mathsf{pp}, \mathsf{pk}, \lfloor\mathbf{h}\rceil_\nu, \mu) := c$
14: for $i \in \mathcal{T} \cap \mathcal{H} \setminus \{h\}$ do
15:     $\mathbf{z}_i \xleftarrow{\$} \mathcal{R}_q^n$
16: for $i \in \mathcal{T} \cap \mathcal{H}, j \in \mathcal{T} \cap \mathcal{C}$ do
17:     $\mathbf{m}_{i,j} := \mathsf{PRF}(\mathsf{sd}_{i,j}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
18:     $\mathbf{m}_{j,i} := \mathsf{PRF}(\mathsf{sd}_{j,i}, (\mathsf{pp}, \mathsf{pk}, \mathcal{T}, (\mathbf{D}_k)_{k\in\mathcal{T}}, \mu))$
19: $\mathbf{m} := \sum_{i\in\mathcal{T}\cap\mathcal{H}, j\in\mathcal{T}\cap\mathcal{C}}(\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$
20: $\mathbf{z}_h := \mathbf{z}^* + \mathbf{R}_h\mathbf{u} - (\sum_{j\in\mathcal{T}\cap\mathcal{C}} \mathbf{s}_j \cdot \lambda_{\mathcal{T},j}) \cdot c$
        $- \sum_{i\in\mathcal{T}\cap\mathcal{H}\setminus\{h\}}(\mathbf{z}_i - \mathbf{r}_i^* - \mathbf{R}_i\mathbf{u}) + \mathbf{m} \bmod q$
21: return $(\mathbf{u}, (\mathbf{z}_j)_{j\in\mathcal{T}\cap\mathcal{H}})$
```