Hermine: An Efficient Lattice-based FROST-like Threshold Signature

Giacomo Borin[1,2], Sofía Celi[3,4], Rafael del Pino[5], Thomas Espitau[5], Shuichi Katsumata[5,6], Guilhem Niot[5,7], Thomas Prest[5], Kaoru Takemure[5,6]

[1]IBM Research - Zurich
[2]University of Zurich
[3]Brave Research
[4]University of Bristol
[5]PQShield
[6]AIST
[7]Univ Rennes, CNRS, IRISA

Abstract

Threshold signatures have regained a strong interest recently, driven by applications in cryptocurrencies and NIST's ongoing call for threshold schemes. Among them, FROST— a *classical* threshold Schnorr signature scheme already in real-world deployment — stands out. Its appeal lies in three core features: *partially non-interactive signing*, *non-interactive identifiable abort (IA)*, and *proactive security*. In contrast, while *post-quantum* (PQ) threshold signatures have seen significant advances in recent years, no existing scheme simultaneously provides even two of these features. Considering the imminent need to migrate to PQ cryptography, this state-of-the-art remains unsatisfactory.

In this work, we propose Hermine, a lattice-based threshold signature that offers the full feature set of FROST under standard lattice assumptions. Hermine is designed to efficiently support the Medium scale of parties ($N \leq 64$) as defined in the NIST threshold call, producing a small Raccoon signature of size 11 KB. Our main technical contribution is an *everywhere-short* secret sharing scheme, which splits a *short* secret vector $\mathbf{s} \in \mathcal{R}_q^\ell$ into *short* shares and admits a *short* linear reconstruction algorithm. While the construction appears intuitive, its security proof requires a non-trivial, fine-grained analysis of the information about $\mathbf{s}$ inherently leaked by the short shares. Furthermore, we formalize game-based definitions of unforgeability and IA in the proactive setting, which may be of independent interest.

Note: This paper consolidates and significantly extends our unpublished results from preprints 2025/871 and 2025/1166. See Section 1.3 for details.

Contents

1 Introduction **3**
- 1.1 Our Contribution . . . 4
- 1.2 Related Work . . . 6
- 1.3 Relation to other prior and subsequent works . . . 6
- 1.4 Technical Overview . . . 7

2 Preliminaries **9**
- 2.1 General Notation . . . 9
- 2.2 Modulus Rounding . . . 10
- 2.3 Hardness Assumptions . . . 10
- 2.4 Forking Lemma with Oracle Access . . . 11

3 Proactively Secure Two-Round Threshold Signatures **12**
- 3.1 Syntax . . . 12
- 3.2 Security . . . 14

4 An Everywhere-Short Secret Sharing Scheme **15**
- 4.1 Construction of Vandermonde Secret Sharing . . . 15
- 4.2 Simulation Security . . . 16

5 Hermine: A FROST-like Two-round Threshold Signature **24**
- 5.1 Construction . . . 25

6 Security of Hermine **25**
- 6.1 Asymptotic Parameters . . . 25
- 6.2 Correctness and Identifiable Abort. . . . 26
- 6.3 Unforgeability . . . 30

7 Instantiation **44**

8 Implementation and Evaluation **45**

A Comparison with Other Secret Sharing **53**

B Optional algorithms of Vandermonde Short Secret Sharing **54**

C Detailed Performance Data **54**

1 Introduction

A T-out-of-N threshold signature scheme allows a signing key to be distributed among a set of N parties such that any subset of T parties can jointly generate a signature. Crucially, no adversary controlling up to $T-1$ parties can forge a signature, making them an essential primitive for distributing trust. Threshold signature have regained a strong interest recently, driven by the advent of blockchain technologies, cryptocurrencies, and NIST's call for threshold schemes [23, 22]. To date, numerous proposals for threshold signatures compatible with ECDSA, Schnorr, BLS, and including, post-quantum signatures have been developed.

Classical state-of-the-art. FROST [66] (and variants thereof [10, 81, 32]) stands as the state-of-the-art for threshold Schnorr signatures. It has been codified in an IRTF RFC [34], submitted to the NIST workshop on multi-party threshold cryptography [65], and has been already deployed in various real-world systems [87, 29, 84, 77]. What makes FROST standout in practice is its three core features:

1. *Partially non-interactive signing*: The signing protocol consists of two rounds with the first round being independent of both the signer set and the message. This allows the first round to be performed in an *offline* pre-processing phase, making the *online* phase non-interactive [22, Section 5.3.5].

2. *Non-interactive identifiable abort (NI IA)*: If the signing protocol fails to produce a valid signature, FROST enables a non-interactive identification of the malicious party from the transcript alone. Concretely, the validity of the partial signatures produced by individual parties can be locally verified.

3. *Proactive security*: FROST naturally supports proactive security against *mobile* adversaries [23, Section 9.2.2], who may progressively corrupt different parties across multiple executions of the protocol.[1] Even if the partial signing keys of $T-1$ parties are compromised over time, the parties can "refresh" their shares. This ensures security even if more than $T-1$ parties are corrupted during the entire lifespan of the signing key, provided fewer than T are corrupted within any single epoch.

Despite these advantages, one main limitation of FROST is that it only offers classical security and is vulnerable to quantum computers. The recent NIST PQC standardization efforts [3] and the Ethereum Foundation's commitment to post-quantum (PQ) cryptography [43] underscore the need to prepare a PQ secure threshold signature scheme capable of replacing FROST.

PQ threshold signatures. Lattice-based threshold signatures have evolved rapidly in recent years, starting from theoretical constructions utilizing heavy primitives like fully homomorphic encryption and trapdoor homomorphic commitments [17, 2, 59], to more practical designs [37, 49, 64, 20, 50, 78, 30, 88, 28, 15]. We can roughly group recent designs into three generations.

The first generation is the threshold signature scheme TRaccoon by del Pino et al. [37]. This was the first practical lattice-based threshold signature based on the signature scheme Raccoon [39, 38], submitted to the additional NIST call for signature proposals [74]. To get around lattice-specific issues (cf. [37, Sec. 2.2]), they introduce a technique called *masking*. Informally, the signers use a one-time additive mask in a way that allows individual signers to hide their responses while preserving correctness. This technique was later used by [49, 64, 20, 88], constructing variants of TRaccoon with different tradeoffs in round complexity, communication costs, and security. For instance, the signing protocol of [49] is partially non-interactive similar to FROST. However, due to masking, these schemes lack the mechanism to prevent misbehaving signers from causing the signing protocol to fail. Without it, a malicious party can covertly mount a DoS attack so that no signatures are ever generated.

The second generation addresses malicious behavior by achieving either robustness [50] or IA [78]. The Pelican scheme [50], based on the hash-and-sign signature Plover [47], provides robustness — guaranteeing a valid signature even if some parties misbehave — but requires a high overhead of at least $3T$ active signers. In a typical execution where all T parties behave honestly, this overhead incurs a high cost without any benefit. In contrast, Unmasked TRaccoon [78] adds IA to vanilla TRaccoon [37]. When TRaccoon fails to

[1] We note that there are various ways to formalize "proactive security", and we intentionally keep it abstract in the introduction.

output a signature, the parties run a lattice-based non-interactive zero-knowledge proof (NIZK) to prove that they generated their partial signatures correctly. However, as TRaccoon's (non-algebraic) masking prevents the use of an efficient lattice-based (algebraic) NIZK, [78] runs an *interactive* IA protocol to remove the mask, rendering it cumbersome for real-world deployment.

The third generation is threshold signatures focusing on compatibility with the NIST-standardized lattice-based signature ML-DSA [28, 15]. However, to maintain compatibility with the standard, these schemes are either restricted to a very small number of parties (e.g., $N \leq 6$) or suffer from extreme round complexity (ranging from tens to hundreds of rounds). Moreover, as the threshold implementation of ML-DSA requires to execute the notorious *rejection sampling* step, achieving an efficient IA is highly non-trivial. Roughly speaking, an honest having to reject and restart its signing execution is indistinguishable from a malicious party intentionally failing the protocol.

In summary, despite significant advancements in lattice-based threshold signatures, we have yet to see a scheme that satisfies both partially non-interactive signing and non-interactive IA, let alone proactive security. This leads to the main question of our work:

> *Can we construct a lattice-based threshold signature that supports all the core features offered by* FROST*?*

1.1 Our Contribution

In this work, we present Hermine, an efficient lattice-based threshold signature scheme that achieves the three core features of FROST: partially non-interactive signing, non-interactive identifiable abort (IA), and proactive security. Hermine is compatible with the Raccoon signature scheme [39, 38] with a signature of size 11 kB. In Table 1, we compare with other lattice-based threshold signature schemes.

Hermine is designed to support the Medium scale of parties (i.e., $N \leq 64$), as defined in the NIST call for threshold schemes [23, Table 17]. While certain applications may require Large ($N \leq 1024$) or Enormous ($N \geq 1025$) party sets, many real-world deployments of threshold signatures operate within the Medium range [87, 42, 24]. To show concrete efficiency, we implemented Hermine in Golang (see Section 8): all protocol phases admit concrete runtimes in the millisecond range, including per-party online signing, for all supported security levels and many thresholds. Notably, compared to the state-of-the-art two-round schemes of Ringtail [20] and EKT [49] in the Medium scale, our signature generation significantly outperforms theirs by eliminating the need to generate masks, which requires intensive hash computations.

At a high level, Hermine removes the masking from the two-round TRaccoon threshold signature by Espitau et al. [49, 88] to enable efficient non-interactive IA similar to FROST. Our technical contribution lies in the use of an *everywhere-short* secret sharing scheme (SSS) that splits a *short* secret vector $\mathbf{s} \in \mathcal{R}_q^\ell$ into *short* shares, which admits a *short* linear reconstruction algorithm (i.e., $\mathbf{s}$ can be reconstructed by simply summing the short shares).

While using everywhere-short SSS to remove the masking appears intuitive (or will so once reading the technical overview in Section 1.4), the security analysis of Hermine proves to be highly non-trivial, requiring a careful analysis of the specific SSS structure. The main challenge is that short secret shares leak information on the short signing key $\mathbf{s} \in \mathcal{R}_q^\ell$. For instance, a party might hold a short share $\mathbf{v} = \mathbf{s} + \mathbf{s}_i$ where $\mathbf{s}_i$ is a short vector. Because $\mathbf{s}_i$ is short, $\mathbf{v}$ leaks non-trivial information of $\mathbf{s}$; the difficulty lies in the fact that the reduction must now simulate $\mathbf{v}$ *without* knowing the secret $\mathbf{s}$. This type of reduction is unique to our setting, as most prior lattice-based cryptography employ a *not*-everywhere SSS [69, 17, 41, 2, 31, 30], and assume shares $\mathbf{s}_i$ are sampled uniformly at random over $\mathcal{R}_q^\ell$. Our proof consists of carefully tracking the shares of the everywhere-short SSS and arguing that a non-trivial portion of the signing key $\mathbf{s}$ remains secure. Concretely, we use the *algebraic one-more* MISIS (AOM-MISIS) problem [49, 88], known to be as hard as the standard MSIS and MLWE problems, to complete the proof.

As a concrete instantiation of our everywhere-short SSS, we introduce *Vandermonde* secret sharing, built on the scheme by Desmedt, Di Crescenzo, and Burmester [40]. While we could have built on other popular secret sharing schemes common in the lattice-based literature, e.g., replicated secret sharing and those by

Table 1: Lattice-based threshold signatures supporting FROST features.

Scheme	Rounds (Off + On)	Size(KB)		# Parties	Handling of	Proactive
		Sig.	Com.		malicious parties	Security
Pelican [50]	$(0+4)$	12.3	$26.8+56T$	M	Robust	✗
Pelican [50]	$(0+4)$	26.4	$53.8+116T$	L	Robust	✗
Unmasked TRaccoon[78]	$(0+3)$	12.7	28.2	L	NIZK-based IA	✗
EKT [49, 88]	$(1+1)$	10.8	276	L	✗	✗
CTZ [30]	$(1+1)$	440	$\geq$ MBs	M	✗	✗
Ringtail [20]	$(1+1)^{\dagger}$	13.7	613	L	✗	✗
Olingo [58]	$(0+3)^{\ddagger}$	≈ 10	$\geq$ MBs	L	NIZK-based IA	✗
Hermine	$(1+1)$	11.3	73.2	M	Non-Interactive IA	✓

The naming convention Medium (M) and Large (L) follow from [23, Table 17], indicating the scheme supports $N \leq 64$ and $N \leq 1024$ parties, respectively. "Non-interactive IA" implies the existence of a partial verification algorithm similarly to FROST. All schemes are based on MLWE and MSIS. †: Ringtail additionally requires the signer set as input in the offline phase. ‡: [58] achieves $(2+1)$ in the sequential setting. See Section 1.2 for the details.

Benaloh and Leichter [12], we observed that they mainly consider *asymptotic* efficiency and security. Notably, we demonstrate that Vandermonde sharing concretely outperforms all previous choices for the Medium scale of parties (i.e., $N \leq 64$), even if it performs asymptotically worse. We believe this analysis is of independent interest for general lattice-based cryptography relying on short secret sharing.

Next, relying on the linear homomorphism of Vandermonde secret sharing, we endow Hermine with proactive security via a key refresh mechanism[2]. This involves adding a fresh Vandermonde secret sharing of the zero vector $\mathbf{0} \in \mathcal{R}_q^{\ell}$ to existing shares. A subtle issue arises in the security proof when handling malicious key refresh. Namely, since the adversary can influence how shares are updated, the secret shares of honest parties may deviate from an "honest" distribution. This stands in contrast to threshold signatures *without* proactive security, where the state of honest parties is sampled by the reduction, and adds another layer of complexity to the above proof.

Lastly, we provide a formal game-based definition for unforgeability and non-interactive IA *with* proactive security. For unforgeability, we extend strong unforgeability from [73] to the proactive setting, with a focus on TS-UF-2 security from [10]. Roughly, we require unforgeability to hold provided that fewer than T parties are corrupted in any given epoch, where an epoch increments following the execution of an *honest* key refresh. Here, we consider a strong notion of unforgeability where even *malicious* key refreshes can be injected within epochs. In contrast, defining non-interactive IA with proactive security proves to be more subtle. Breaking IA becomes trivial if malicious key refreshes are permitted without constraint; for instance, an adversary could corrupt an honest party's share during a refresh, causing their subsequent partial signature to fail verification. While we could restrict the game to permit only honest key refreshes, such a definition is of limited value in practice. Instead, we explicitly incorporate a mechanism for honest parties to efficiently and non-interactively detect malicious refreshes that would trivially break IA, and our definition permits only those key refreshes that pass this detection. We believe our pragmatic definition of proactive IA to be of independent interest.

[2]We note that any linearly homomorphic everywhere-short SSS would suffice. We use Vandermonde for the explanation simply because Hermine uses it.

1.2 Related Work

Distributed key generation (DKG) for Hermine. In this work, we assume a trusted dealer for key generation. To relax this restriction, one could employ distributed key generation. An approach is to leverage verifiable short secret sharing used in [50]. Specifically, each party samples a random short vector and generates Vandermonde secret shares of it. After that, it sends shares to corresponding signers along with the proof ensuring the shortness of the sampled vector and the correct generation of the shares. We leave the concrete security analysis of this DKG for future work. There is a DKG protocol tailored to replicated secret sharing which was proposed by [15]. Whether this DKG can be extended to Vandermonde secret sharing is an open problem.

Proactive security. There is an extensive literature on proactive security [75], in the context of, e.g., secret sharing (PSS) [61, 25, 83, 72, 46, 85, 86, 51], threshold RSA [53, 80, 54, 55, 63, 6, 62], threshold ECDSA [26, 27, 68], DL based schemes [60, 67], BLS based ones [16, 56]. Cimatti et al. [33] considered FROST in a dynamic setting where the number of users and the reconstruction threshold can be changed via refresh, though they do not provide any formal security definition. Boneh et al. [18] provides formal game-based definitions capturing proactive security for accountable threshold signatures. This definition is closest to ours but differs in how to model malicious key refreshes; while we provide explicit algorithms to check for an invalid refresh, they model it implicitly.

Concurrent and independent work. [58] constructs a threshold signature scheme Olingo using linearly homomorphic encryptions, and achieve IA by enforcing every party to prove correct computation using NIZK*s*. Olingo satisfies offline-online efficiency with a 2-round offline phase in the *sequential* setting (i.e., parties cannot participate in more than one session at a given time).

1.3 Relation to other prior and subsequent works

Prior works. This paper consolidates and significantly extends the unpublished results from preprints ePrint 2025/871 and ePrint 2025/1166 . Both were withdrawn from ePrint in May 2026 to avoid confusion with the current paper, which supersedes them. ePrint 2025/871 introduced the notion of everywhere short secret sharing (named Short DKG at the time) and applied it to achieve an efficient 3-round Threshold Raccoon with identifiable aborts. They proposed two instantiations based on replicated secret sharing and a coupon collector ramp secret sharing. ePrint 2025/1166 subsequently instantiated this everywhere short secret sharing using Vandermonde Secret Sharing to increase the scalability of the sharing to $N = 64$ parties, achieving a more efficient 3-round Threshold Raccoon with identifiable aborts.

Our current work unifies these results and introduces three major new contributions: (1) we formalize *proactive security* for everywhere short secret sharing; (2) we construct a *2-round offline-online efficient* Threshold Raccoon with identifiable aborts (improving upon the previous 3-round schemes); and (3) we achieve the stronger TS-UF-2 security notion, as opposed to the weaker TS-UF-0 considered in the preprints. Notably, reaching these stronger security guarantees (namely, contributions (1) and (3)) required a substantial strengthening of the underlying security definitions compared to the preprints.

For completeness, we note that ePrint 2025/871 also contains results on DKG for everywhere short secret sharing. We omitted those results as it is a relatively intuitive construction but requires a careful and heavy formalization that is orthogonal to the main contribution of this paper. Furthermore, ePrint 2025/1166 contained results on thresholdizing ML-DSA. As mentioned in our introduction, these results solve an independent question using different techniques; they have been accepted at USENIX Security 2026 and supersede that portion of ePrint 2025/1166 as ePrint 2026/013 .

Subsequent works. The Vandermonde Secret Sharing technique - and the corresponding security analysis - from our preliminary preprint (2025/1166) was subsequently utilized by ePrint 2026/021 [21] (c.f. Lemma 15) - published at PKC 2026 as "Amber" - to construct a threshold *KEM* with efficient identifiable aborts. By leveraging the additional security extensions of everywhere short secret sharing formalized in our current paper, Amber could potentially be extended to achieve proactive security. We believe this shows the versatility of our result and hope that our notion of everywhere short secret sharing will find further applications in other contexts.

1.4 Technical Overview

We provide a high-level overview of Hermine. Throughout this section, we omit subtle details (e.g., bit-dropping) to focus on the core intuition.

The underlying Raccoon signature. Raccoon [39, 38] is a variant of Lyubashevsky's lattice-based signature scheme [70, 71], which removes rejection sampling by exploiting the noise flooding technique [57, 2]. The verification key vk consists of a matrix $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$ and a vector $\mathbf{t} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}$ for short vectors $(\mathbf{s}, \mathbf{e}) \in \mathcal{R}_q^{\ell} \times \mathcal{R}_q^{k}$. To sign a message msg, the signing algorithm proceeds as follows:

1. Generate a commitment $\mathbf{w} = \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'$ for short vectors $\mathbf{r}$ and $\mathbf{e}'$,
2. Compute the challenge $c = H_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$ and output the signature $(c, \mathbf{z}, \mathbf{h})$, where $\mathbf{z} = c \cdot \mathbf{s} + \mathbf{r}$ and $\mathbf{h} = \mathbf{w} - \mathbf{A} \cdot \mathbf{z} + c \cdot \mathbf{t}$. Here, H_c is a random oracle outputting a small element $c \in \mathcal{R}_q$.

The verification algorithm consists of checking that $\mathbf{z}$ and $\mathbf{h}$ are short, and that $\mathbf{A} \cdot \mathbf{z} = c \cdot \mathbf{t} + \mathbf{w} - \mathbf{h}$ holds. The above structure is analogous to the classical Schnorr signature by noting $\mathbf{A} \cdot \mathbf{z} \approx c \cdot \mathbf{t} + \mathbf{w}$.

Insecure two-round threshold Raccoon. We now explain a naive (and insecure) way to adapt Raccoon to the threshold setting following the design of FROST [66], which yields a two-round threshold Schnorr signature.

The signing key $\mathbf{s}$ is distributed among N parties using Shamir's secret sharing such that $\mathbf{s} = \sum_{i \in \mathsf{act}}^{T} L_{\mathsf{act},i} \cdot [\![\mathbf{s}]\!]_i$ for any signer set $\mathsf{act} \in [N]$ of size T, where $L_{\mathsf{act},i}$ is the corresponding Lagrange coefficient. The *partial* verification key for party $i \in [N]$ is set to $\mathsf{vk}_i = \mathbf{A} \cdot [\![\mathbf{s}]\!]_i + \mathbf{e}_i$ for a short vector $\mathbf{e}_i$. The signing protocol proceeds as follows:

Offline phase: Party $i \in [N]$ outputs a *vector of* commitments $\overrightarrow{\mathbf{w}}_i = [\mathbf{w}_{i,1} | \cdots | \mathbf{w}_{i,\mathsf{rep}}]$ such that $\mathbf{w}_{i,b} = \mathbf{A} \cdot \mathbf{r}_{i,b} + \mathbf{e}'_{i,b}$ for short $\mathbf{r}_{i,b}$ and $\mathbf{e}'_{i,b}$. Importantly, this phase is independent of msg and the signer set act.

Online phase: Once given msg, act, and $\left(\overrightarrow{\mathbf{w}}_j\right)_{j \in \mathsf{act}}$, party i first computes $(\beta_b)_{b \in [\mathsf{rep}]} = H_\beta(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \left(\overrightarrow{\mathbf{w}}_j\right)_{j \in \mathsf{act}})$. Here, H_β is a random oracle outputting a list of small elements $\beta_b \in \mathcal{R}_q$. It then aggregates the vectors of commitments as $\mathbf{w} = \sum_{j \in \mathsf{act}} \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{w}_{j,b}$ and computes the challenge $c = H_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$. Lastly, it outputs a *partial* signature $\mathbf{z}_i = c \cdot L_{\mathsf{act},i} \cdot [\![\mathbf{s}]\!]_i + \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$.

Aggregation: The aggregator computes $\mathbf{z} = \sum_{i \in \mathsf{act}} \mathbf{z}_i$, $\mathbf{h} = \mathbf{w} - \mathbf{A} \cdot \mathbf{z} + c \cdot \mathbf{t}$, and outputs the signature $(c, \mathbf{z}, \mathbf{h})$.

It is easy to check that the scheme is functionally correct, satisfying Raccoon's verification algorithm. The vectorization of the commitments $\overrightarrow{\mathbf{w}}_i$ and their aggregation via random weights directly mirror FROST; the only minor difference lies in how rep and the weights β_b are chosen (see [49] for the details).[3]

However, on closer inspection, there are two key issues with the above scheme. First, in regards to security, as already observed in the original paper on TRaccoon [37, Sec. 2.2], the partial signature $\mathbf{z}_i$ leaks too much information about $[\![\mathbf{s}]\!]_i$, since the Lagrange coefficient $L_{\mathsf{act},i}$ is highly structured and can be arbitrarily large over mod q. Second, while a partial verification key $\mathsf{vk}_i = \mathbf{A} \cdot [\![\mathbf{s}]\!]_i + \mathbf{e}_i$ exists, it is unclear how it aids in achieving non-interactive IA as in FROST; the partial signature $\mathbf{z}_i$ is large and does not satisfy Raccoon's verification algorithm.

Partial solutions to the issue. Two approaches are known to overcome the first issue. One approach is the masking technique used by TRaccoon [37], later adopted by numerous other works [49, 64, 20, 88]. The key idea is to non-interactively generate a secret share of the zero vector and add it to the partial signature $\mathbf{z}_i$. Intuitively, the individual partial signatures no longer leak any information other than the fact that they aggregate to a valid signature $\mathbf{z}$. The second approach involves replacing the Shamir secret sharing scheme with a different secret sharing scheme which admits a *short* linear reconstruction algorithm. Concretely,

[3] More precisely, this aligns with FROST2 as all parties use the same weights to aggregate each signer's commitments. Since FROST2 is known to achieve TS-UF-2 [10], Hermine achieves the same security level.

[30] relies on the secret sharing of Benaloh and Leichter [12] so that a partial signature roughly looks like $\mathbf{z}_i = c \cdot M_{\mathsf{act},i} \cdot [\![\mathbf{s}]\!]_i + \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$ for a reconstruction coefficient $M_{\mathsf{act},i} \in \{0,1\}$.[4]

Unfortunately, neither approach solves the second issue, namely achieving non-interactive IA. While [30] came close, the shares $[\![\mathbf{s}]\!]_i$ are still arbitrarily large over $\mathcal{R}_q$, and therefore, the validity of the partial signature $\mathbf{z}_i$ cannot be checked non-interactively without resorting to heavy tools like NIZKs.

Using an everywhere-short secret sharing. In our work, we concretely solve the second issue with a natural idea: we rely on a secret sharing scheme that outputs a *short* share — combined with the original secret being *short* and the reconstruction coefficient being *small*, we call this an "everywhere-short" secret sharing. Note that since the partial signature $\mathbf{z}_i$ remains short, parties can locally run the Raccoon verification algorithm, checking $\mathbf{A} \cdot \mathbf{z}_i \approx c \cdot M_{\mathsf{act},i} \cdot \mathsf{vk}_i + \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{w}_{i,b}$; if verification fails, party i is identified as the one who caused the aggregated signature to be invalid.

This raises the question: what is the optimal way to construct an everywhere-short secret sharing? To answer it, we revisit the secret sharing scheme from Desmedt, Di Crescenzo, and Burmester [40], which has been recently used by Battagliola et al. [9] in the context of secret sharing in a (non-abelian) group. In essence, this is an algorithmic interpretation of Vandermonde's identity, based on the same combinatorial decomposition which states that for $0 \leqslant b \leqslant N$:

$$\binom{N}{T} = \sum_{k=0}^{T} \binom{b}{k} \cdot \binom{N-b}{T-k}. \tag{1}$$

Intuitively, Eq. (1) implies that for any set of T selected parties among N parties, when we split the set of parties into two halves (taking $b = \lfloor N/2 \rfloor$ to minimize complexity), one half will have k selected parties for some $0 \leqslant k \leqslant T$, while the other half will have exactly $T - k$ selected parties. Thus, in the *non*-everywhere short setting, the core idea of this sharing is, for each possible $0 \leqslant k \leqslant T$, to share the secret $\mathbf{s} \in \mathcal{R}_q^\ell$ into two uniformly random sub-secrets $\mathbf{s}_0$ and $\mathbf{s} - \mathbf{s}_0$, one for each half, with thresholds k and $T - k$ respectively, and recursively share these sub-secrets within each half. For example, in a 2-out-of-4 setting, parties 1, 2, 3, and 4 hold $(\mathbf{s}_0, \mathbf{s}_1)$, $(\mathbf{s}_0, \mathbf{s} - \mathbf{s}_1)$, $(\mathbf{s} - \mathbf{s}_0, \mathbf{s}_2)$, and $(\mathbf{s} - \mathbf{s}_0, \mathbf{s} - \mathbf{s}_2)$, respectively — it is easy to check that any two parties can reconstruct $\mathbf{s}$.

Our core modification for keeping the individual shares short is simple in hindsight and consists of sharing the secret $\mathbf{s}$ in two halves $\mathbf{s}_0$ and $\mathbf{s} - \mathbf{s}_0$ in the recursion step, with $\mathbf{s}_0$ being sampled from a *short* distribution instead of the classical uniform one. We can prove recursively that the shares will preserve a short norm and that the reconstruction coefficients are in $\{0,1\}$ as desired. We call this the *Vandermonde* secret sharing scheme.

We note that this modification of keeping the shares small can be applied to other popular secret sharing schemes common in the lattice-based literature, e.g., replicated secret sharing and those by Benaloh and Leichter [12]. However, as we see in Section A, Vandermonde sharing concretely outperforms all previous choices for the Medium scale of parties (i.e., $N \leq 64$).

Short shares are inherently leaky. While correctness is clear from construction, what makes Vandermonde secret sharing non-trivial is its *privacy* property. Since the secret $\mathbf{s}$ is shared in two halves $\mathbf{s}_0$ and $\mathbf{s}_1 = \mathbf{s} - \mathbf{s}_0$ with a *short* $\mathbf{s}_0$, it is no longer a proper secret sharing. Indeed, if $|\mathbf{s}_0| \ll |\mathbf{s}|$, the party holding $\mathbf{s}_1$ learns almost all information about $\mathbf{s}$! Conversely, we can take $|\mathbf{s}_0| \gg |\mathbf{s}|$ to statistically "flood" the information on $\mathbf{s}$ from $\mathbf{s}_1$ [57]. However, this leads to an impractical scheme since it necessitates using an exponentially large modulus q to argue that the shares $\mathbf{s}_0$ and $\mathbf{s}_1$ are (relatively) short.

Taming the leak. In this work, we carefully control the minimal noise $\mathbf{s}_0$ that needs to be added to the secret $\mathbf{s}$ so that the share $\mathbf{s}_1$ does not leak too much information about $\mathbf{s}$. To be more precise, our main technical contribution consists of answering the following questions:

(Q.1) How much information leaks from the secret shares held by a set of parties of size at most $T - 1$?

(Q.2) How much information on $\mathbf{s}$ can leak while maintaining the security of Hermine?

[4] To be precise, the secret sharing requires each party to contribute more than one share for reconstruction, so $\mathbf{z}_i$ will contain a sum of them as opposed to a single $[\![\mathbf{s}]\!]_i$.

Observe that the answer to Item (Q.1) is specific to the considered secret sharing scheme. We have to consider the worst-case party set $S^* \subset [N]$ with $|S^*| \leq T-1$ that minimizes the conditional min-entropy of $\mathbf{s}$, i.e., $H_\infty(\mathbf{s} \mid \text{all shares of party } i \in S^*)$. For our Vandermonde secret sharing, we tackle this by explicitly writing down the necessary leakage $L(\mathbf{s})$ on the secret $\mathbf{s}$ to construct an efficient algorithm $\mathsf{SimShares}(L(\mathbf{s}))$ that perfectly simulates the secret shares for all the parties in S^*. This indirectly proves that the conditional min-entropy of $\mathbf{s}$ is lower bounded by $H_\infty(\mathbf{s} \mid L(\mathbf{s}))$.

Regarding Item (Q.2), proving Hermine unforgeable requires the reduction to simulate the adversary's view without knowledge of the secret $\mathbf{s}$ – which includes the corruption queries, but also the signing queries. While the former is intrinsically tied to Item (Q.1), the latter introduces a new challenge: the reduction must simulate the partial signatures of honest parties, which are generated using their secret shares. However if SimShares can generate shares for *all* honest parties, it could combine these with the corrupted shares to reconstruct $\mathbf{s}$, contradicting the privacy guarantee of $\mathbf{s}$. To resolve this dilemma, we strengthen SimShares so that given any set of T parties, it can generate all their shares *except one*, designated as the "secure" share. The reduction can then simulate the partial signatures of all-but-one honest party with the shares generated on its own. The partial signature of the one honest party holding the secure share (which the reduction cannot know) will be simulated relying on the AOM-MISIS problem; this part of the simulation is similar to those used in prior works [49, 88].

Lastly, while this strategy suffices for TS-UF-0 security, additional subtleties arise when proving the stronger TS-UF-2 security, as the adversary can obtain a small number of partial signatures on the message to be forged. We cover this by adding a mechanism to require SimShares to prevent the honest share being used in those simulated partial signatures.

2 Preliminaries

2.1 General Notation

We assume familiarity with Landau's asymptotic notations: $O(n)$, $o(n)$, $\omega(n)$, $\Omega(n)$, $a(n) \sim b(n)$, and Knuth's Theta notation $\Theta(n)$. We denote the empty string by ε.

Sets and distributions.. For an integer $N > 0$, we note $[N] = \{1, \ldots, N\}$. For a lexicographically ordered set $P \subset [N]$, we denote the i-th element in P by $P[i]$ and the subsequence of P consisting of the elements from the i-th element to the j-th element by $P[i \ldots j]$. To denote the assign operation, we use $y \coloneqq f(x)$ when f is deterministic and $y \leftarrow f(x)$ when randomized. When S is a finite set, we note $\mathcal{U}(S)$ the uniform distribution over S, and shorthand $x \xleftarrow{\$} S$ for $x \leftarrow \mathcal{U}(S)$.

Algorithms. We use functions **require**(b) in individual algorithms and games for checking a boolean statement. If b is false, **require** makes the algorithm immediately return 0. Also, it aborts the entire game and returns 0 if b is false.

Linear algebra. For a fixed power-of-two n, we note $\mathcal{K} = \mathbb{Q}[x]/(x^n+1)$ and $\mathcal{R} = \mathbb{Z}[x]/(x^n+1)$, and $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. We assume q to be odd. Given $\mathbf{x} \in \mathcal{K}^\ell$, we denote $\|\mathbf{x}\|$ the Euclidean norm, and $\|\mathbf{x}\|_\infty$ the infinity norm of the $(n\,\ell)$-dimensional vector of the coefficients of $\mathbf{x}$. For elements in $\mathbb{Z}_q$ and $\mathcal{R}_q$, the absolute value and norms are defined using the representative with coefficients in $(-q/2, q/2]$. Unless specified otherwise, vectors are treated as *column* vectors. We note the set of all signed monomials $\mathbb{T} \coloneqq \{\pm 1, \pm x, \ldots, \pm x^{n-1}\} \subseteq \mathcal{R}_q$. We recall the following lemma, establishing the invertibility for $\mathbb{T}$ (see [13]).

Lemma 2.1. *For any distinct $\beta_1, \beta_2 \in \mathbb{T}$, $\beta_1 - \beta_2$ is invertible over $\mathcal{R}_q$ and $\left\|2 \cdot (\beta_1 - \beta_2)^{-1}\right\|_\infty \leqslant 1$.*

Definition 2.2 (Discrete Gaussians). *For $\sigma > 0, \mathbf{c} \in \mathbb{R}^N$, let $\rho_{\sigma,\mathbf{c}}(\mathbf{x}) = \exp(-\|\mathbf{x}-\mathbf{c}\|^2/(2\sigma^2))$. For any countable set $S \subset \mathbb{R}^N$, let $\rho_{\sigma,\mathbf{c}}(S) = \sum_{\mathbf{x}\in S} \rho_{\sigma,\mathbf{c}}(\mathbf{x})$. We define the discrete Gaussian distribution $D_{S,\sigma,\mathbf{c}}$ (also denoted $\mathcal{D}_{S,\sigma,\mathbf{c}}$) via $D_{S,\sigma,\mathbf{c}}(\mathbf{x}) = \rho_{\sigma,\mathbf{c}}(\mathbf{x})/\rho_{\sigma,\mathbf{c}}(S)$. We omit $\mathbf{c}$ when $\mathbf{c} = \mathbf{0}$. We may omit S when clear from context. We extend this definition to sets $S \subseteq \mathcal{K}$ (resp. $S \subseteq \mathcal{K}^k$) by identifying polynomials with their coefficient vectors in $\mathbb{R}^n$ (resp. $\mathbb{R}^{nk}$).*

We now recall useful convolution and tail-bound lemmata.

Lemma 2.3 (Simplified convolution lemma [76, Thm. 4.5]). *Let positive reals σ_1, σ_2 such that $\sigma_1^{-2} + \sigma_2^{-2} \leqslant \eta'_\varepsilon(\mathbb{Z}^n)^{-2}$ for some positive $\varepsilon \leqslant 1/4$. Then the distribution $D_{\mathbb{Z}^n,\sigma_1} + D_{\mathbb{Z}^n,\sigma_2}$ is within statistical distance 8ε of $D_{\mathbb{Z}^n,\sqrt{\sigma_1^2+\sigma_2^2}}$.*

Lemma 2.4 (Lemma 4.4 in [71], extended). *For any $\alpha > 1$, and any lattice $\Delta \subseteq \mathbb{R}^n$, $\Pr[\|\mathbf{z}\| > \alpha\sigma\sqrt{n} \mid \mathbf{z} \leftarrow \mathcal{D}_{\Delta,\sigma}] \leqslant C^n$, where $C = \alpha \cdot e^{\frac{1}{2}(1-\alpha^2)} < 1$. In particular, if $(\alpha \geqslant 1 + \frac{\kappa \log 2}{3n})$ or if $(\alpha \geqslant 1 + \sqrt{\frac{4\kappa \log 2}{n}})$, then $C^n \leqslant 2^{-\kappa}$.*

2.2 Modulus Rounding

Let $\nu \in \mathbb{N}^*$. Every $x \in \mathbb{Z}$ admits a unique decomposition $x = 2^\nu x_\top + x_\perp$, with $x_\perp \in [-2^{\nu-1}, 2^{\nu-1} - 1]$, and we define $\lfloor\cdot\rceil_\nu : \mathbb{Z} \to \mathbb{Z}$ by

$$\lfloor x \rceil_\nu = x_\top = \lfloor x/2^\nu \rceil\,, \qquad \lfloor t \rceil = \left\lfloor t + \tfrac{1}{2} \right\rfloor \text{ (half-up).}$$

If $q > 2^\nu$, we extend $\lfloor\cdot\rceil_\nu$ to $\mathbb{Z}_q$ by lifting $x \mapsto \bar{x} \in [0, q-1]$ and setting $\lfloor x \rceil_\nu = \lfloor \bar{x}/2^\nu \rceil = (\bar{x})_\top \in [0, \lfloor (q-1)/2^\nu \rceil]$, and apply it coefficient-wise to vectors. Here, $\overline{x}$ denotes the lift of x to $\mathbb{Z}^n$ with coefficients in $[0, q-1]$.

Remark 2.5. In [36, 37], $q_\nu = \lfloor q/2^\nu \rfloor$ is used. While this is necessary for the rounding of commitments $\mathbf{w}$ during signing (because of the hint mechanism), this was unnecessarily giving worse bounds for the public keys rounding. Here, we consider rounding outputs in $\mathbb{Z}$ and only apply modulo q_ν during signing before inputting the rounded commitments to the hash function.

Lemma 2.6. *Let $q, \nu \in \mathbb{N}^*$. For $x \in \mathbb{Z}_q$, $|x - 2^\nu \cdot \lfloor x \rceil_\nu| \leqslant 2^{\nu-1}$.*

Lemma 2.7 (extended from Lemma 3.2 [37]). *Let ν, q be positive integers such that $q > 2^\nu$ and $2^\nu \nmid q$ so that $\lfloor q/2^\nu \rfloor \neq \lceil q/2^\nu \rceil$. Let $q_\nu = \lfloor q/2^\nu \rceil$ and $q_\perp = q - 2^\nu \cdot q_\nu$. Then, for any $x \in \mathbb{Z}_q$, we have*

$$\left| x - 2^\nu \cdot \overline{\lfloor x \rceil_\nu \bmod q_\nu} \right| \leq 2^{\nu-1} + \max(0, q_\perp) \tag{2}$$

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

$$\left\| 2^\nu \cdot \overline{\lfloor \mathbf{x} + \boldsymbol{\delta} \rceil_\nu - \lfloor \mathbf{x} \rceil_\nu \bmod q_\nu} - \boldsymbol{\delta} \right\| \leqslant (3 \cdot 2^{\nu-1} + \max(0, q_\perp)) \cdot \sqrt{n}. \tag{3}$$

The proof of Theorem 2.7 is provided in **??**.

2.3 Hardness Assumptions

Our unforgeability proof relies on the Module Short Integer Solution problem (MSIS) and the Algebraic One-More MSIS problem (AOM-MISIS) introduced in [88]. The latter is a one-more variant of MSIS (in the spirit of the AOM-MLWE problem [49]), and admits hardness reductions from MSIS and MLWE in certain regimes. We recall the definitions below.

Definition 2.8 (MSIS). *Let k, ℓ, q be integers and $\beta > 0$. The advantage of $\mathcal{A}$ against $\mathsf{MSIS}_{q,k,\ell,\beta}$ is*

$$\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{A}}(\kappa) = \Pr\left[\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell},\ \mathbf{s} \xleftarrow{\$} \mathcal{A}(\mathbf{A}) : [\mathbf{A}\ \mathbf{I}]\mathbf{s} = \mathbf{0}\ \wedge\ 0 < \|\mathbf{s}\|_2 \leq \beta\right].$$

The $\mathsf{MSIS}_{q,k,\ell,\beta}$ assumption states that $\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{A}}(\kappa)$ is negligible for any efficient $\mathcal{A}$.

Definition 2.9 (MLWE). *Let k, ℓ, q be integers and $\beta > 0$. The advantage of $\mathcal{A}$ against $\mathsf{MLWE}_{q,k,\ell,\beta}$ is*

$$\mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{A}}(\kappa) = \Big|\Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, [\mathbf{A}\ \mathbf{I}]\mathbf{s})] - \Pr[1 \leftarrow \mathcal{A}(\mathbf{A}, \mathbf{b})]\Big|,$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{s}) \xleftarrow{\$} \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^k \times S$ and $S = \{\mathbf{x} \in \mathcal{R}^{\ell+k} : \|\mathbf{x}\|_\infty \leq \beta\}$. The $\mathsf{MLWE}_{q,k,\ell,\beta}$ assumption states that this advantage is negligible for any efficient $\mathcal{A}$.

$\mathsf{Game}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{A}}(1^\kappa)$

$B \leftarrow \emptyset, \mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}$
for $i \in [Q]$ **do** $\mathbf{s}_i \leftarrow D_{\mathcal{R}^{k+\ell},0,\sigma_i}; \mathbf{t}_i \leftarrow [\mathbf{A}\ \mathbf{I}]\cdot\mathbf{s}_i$
$(\mathbf{s}^*, \mathbf{b}^*, \mathbf{u}) \leftarrow \mathcal{A}^{\mathsf{O_{PI}}}(\mathbf{A}, \{\mathbf{t}_i\}_{i\in[Q]})$ // Obtain solution $(\mathbf{s}^*, \mathbf{b}^*, \mathbf{u}) \in \mathcal{R}_q^{k+\ell} \times \mathcal{R}_q^Q \times \mathcal{R}_q^Q$
require $\sum_{i\in[Q]} b_i^* \cdot \mathbf{t}_i = [\mathbf{A}\ \mathbf{I}]\cdot\mathbf{s}^* \wedge \mathbf{b}^{*\top}\cdot\mathbf{u} \neq 0$ // Need non-trivial solution
require $\forall \mathbf{b} \in B\ \ \mathbf{b}^\top\mathbf{u} = 0 \bmod q$ // Queried combinations must be orthogonal to $\mathbf{u}$
return $\{\|\mathbf{s}^*\|_2 \leqslant \beta_\mathbf{s} \wedge \left\|\left(\frac{u_1}{\sigma_1}, \ldots, \frac{u_Q}{\sigma_Q}\right)\right\|_2 \leqslant \beta_\mathbf{u} \wedge \left\|(b_1^*\sigma_1, ..., b_Q^*\sigma_Q)\right\|_1 \leqslant \beta_\mathbf{b}\}$

$\mathsf{O_{PI}}\left(\mathbf{b} \in \mathcal{R}_q^Q\right)$

$B \leftarrow B \cup \{\mathbf{b}\}$
return $\sum_{i\in[Q]} b_i \cdot \mathbf{s}_i \in \mathcal{R}_q^{k+\ell}$

Figure 1: The Algebraic One-More MSIS (AOM-MISIS) security game.

Definition 2.10 (Algebraic One-More MSIS (AOM-MISIS) [88]). *Let k, ℓ, q, Q be integers, $\beta_\mathbf{s}, \beta_\mathbf{b}, \beta_\mathbf{u} > 0$, and $\sigma_1, \ldots, \sigma_Q > 0$. The advantage of $\mathcal{A}$ against $\mathsf{AOM\text{-}MISIS}_{q,k,\ell,Q,\beta_\mathbf{s},\beta_\mathbf{b},\beta_\mathbf{u},(\sigma_i)_{i\in[Q]}}$ is*

$$\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{A}}(\kappa) = \Pr\left[\mathsf{Game}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{A}}(1^\kappa) = 1\right],$$

where $\mathsf{Game}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{A}}$ is given in Fig. 1. The corresponding assumption states that this advantage is negligible for any efficient $\mathcal{A}$.

Intuition. The challenger samples short $\mathbf{s}_i$ and publishes $\mathbf{t}_i = [\mathbf{A}\ \mathbf{I}]\mathbf{s}_i$. Oracle $\mathsf{O_{PI}}$ returns short preimages of queried combinations $\mathbf{b}$, namely $\sum_i b_i\mathbf{s}_i$. The adversary must output a *fresh* short preimage $\mathbf{s}^*$ for $\sum_i b_i^*\mathbf{t}_i$, in a way that cannot be "explained" by the oracle queries. The freshness is enforced by an auxiliary $\mathbf{u}$: all oracle queries satisfy $\mathbf{b}^\top\mathbf{u} = 0$ while the final vector satisfies $\mathbf{b}^{*\top}\mathbf{u} \neq 0$.

Theorem 2.11 ([88, Theorem 1]). *For any $\varepsilon \in (0,1)$, $\gamma > 1$, any parameters $(q, k, \ell, Q, \beta_\mathbf{s}, \beta_\mathbf{b}, \beta_\mathbf{u}, (\sigma_i)_{i\in[Q]})$ such that $(k+\ell)n \geqslant 2\kappa$ and $\sigma_i \geqslant \sqrt{\log(6(k+\ell)n)/\pi}$, and any AOM-MISIS adversary $\mathcal{A}$, there exist an adversary $\mathcal{B}_1$ against $\mathsf{MSIS}_{q,k,\ell,\beta_{\mathsf{sis}}}$ and two adversaries $\mathcal{B}_2$ and $\mathcal{B}_3$ against $\mathsf{MLWE}_{q,k,\ell-1,\beta_{\mathsf{lwe}}}$, such that*

$$\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{A}}(\kappa) \leqslant 2\delta_\gamma \left(\mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}_1}(\kappa) + \mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}_2}(\kappa) + Q\cdot 2^{-2\kappa+2}\right)^{\frac{\gamma}{\gamma-1}} + \mathsf{Adv}^{\mathsf{MLWE}}_{\mathcal{B}_3}(\kappa),$$

where $\delta_\gamma = \frac{1+\varepsilon}{1-\varepsilon}\cdot\exp((\gamma-1)\pi^2/\log(2(1+1/\varepsilon)))$, $\beta_{\mathsf{sis}} = \beta_\mathbf{s} + \beta_\mathbf{b}\sqrt{(k+\ell)n}$, $\beta_{\mathsf{lwe}} = 1/(\beta_\mathbf{u}\sqrt{(k+\ell)n\cdot\log(2(1+1/\varepsilon))/\pi})$, and $\mathsf{Time}(\mathcal{B}_1), \mathsf{Time}(\mathcal{B}_2), \mathsf{Time}(\mathcal{B}_3) \approx \mathsf{Time}(\mathcal{A})$.

2.4 Forking Lemma with Oracle Access

We recall a variant [45] of the forking lemma [79, 11] in which the forking algorithm is allowed accessing to a deterministic oracle. This type of formalization is useful when trying to reduce from an interactive assumption to the security of a scheme.

Lemma 2.12 (Forking Lemma with Oracle Access). *Fix an integer $q_{\mathsf{Fork}} \geq 1$ and a set $\mathcal{H}$ of size $h \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that has oracle access to a deterministic algorithm O, where on input par, $\overrightarrow{h} := (h_1, \cdots, h_{q_{\mathsf{Fork}}})$, algorithm $\mathcal{A}$ returns $J \in [0, \cdots, q_{\mathsf{Fork}}]$ and an arbitrary string σ. Let IG be*

a randomized algorithm called the input generator. *The accepting probability of* $\mathcal{A}$*, denoted* acc*, is defined below:*

$$\mathsf{acc} = \Pr\left[(\mathsf{par}, \overline{\mathsf{par}}) \xleftarrow{\$} \mathsf{IG},\ \overrightarrow{h} \xleftarrow{\$} \mathcal{H}^{q_{\mathsf{Fork}}},\ (J, \sigma) \xleftarrow{\$} \mathcal{A}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par}, \overrightarrow{h}) : J \geq 1\right].$$

The forking algorithm $\mathsf{Fork}_{\mathcal{A}}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}$ *associated to* $\mathcal{A}$ *is a randomized oracle-calling algorithm that takes input* par *and proceeds as in Fig. 2. Let*

$$\mathsf{frk} = \Pr\left[(\mathsf{par}, \overline{\mathsf{par}}) \xleftarrow{\$} \mathsf{IG};\ (b, (\sigma_1, \sigma_2)) \xleftarrow{\$} \mathsf{Fork}_{\mathcal{A}}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par}) \ :\ b = 1\right].$$

Then, $\mathsf{frk} \geq \mathsf{acc} \cdot \left(\frac{\mathsf{acc}}{q} - \frac{1}{h}\right)$.

Algorithm $\mathsf{Fork}_{\mathcal{A}}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par})$

1 : $\mathsf{coin} \xleftarrow{\$} \{0,1\}^{\ell_{\mathcal{A}}}$ // $\ell_{\mathcal{A}}$-bit randomness used by $\mathcal{A}$
2 : $\overrightarrow{h} := (h_1, \cdots, h_q) \xleftarrow{\$} \mathcal{H}^{q_{\mathsf{Fork}}}$
3 : $(I, \sigma) := \mathcal{A}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par}, \overrightarrow{h}; \rho)$
4 : **if** $I = 0$ **then**
5 : **return** $(0, (\bot, \bot))$
6 : $(h'_I, \cdots, h'_q) \xleftarrow{\$} \mathcal{H}^{q_{\mathsf{Fork}} - I + 1}$
7 : $\overrightarrow{h}' := (h_1, \cdots, h_{I-1}, h'_I, \cdots, h'_q)$
8 : $(I', \sigma') := \mathcal{A}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par}, \overrightarrow{h}'; \rho)$
9 : **if** $I = I' \ \wedge\ h_I \neq h'_I$ **then**
10 : **return** $(1, (\sigma, \sigma'))$
11 : **else**
12 : **return** $(0, (\bot, \bot))$

Figure 2: Description of the oracle-calling forking algorithm $\mathsf{Fork}_{\mathcal{A}}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}$.

3 Proactively Secure Two-Round Threshold Signatures

We define a *two-round* threshold signature scheme, modeling unforgeability and *non-interactive* identifiable aborts (NI-IA) with *proactive* security, also known as security against a *mobile* adversary [23, Section 9.2.2].

3.1 Syntax

We first provide the core syntax of threshold signature following [49], and provide additional algorithms for capturing NI-IA and proactive security.

$\mathsf{TS.Setup}(1^\kappa, N, T) \to \mathsf{tspar}$: takes as input a security parameter κ, the number of parties N, and the threshold T, and outputs public parameters tspar.

$\mathsf{TS.Keygen}(\mathsf{tspar}) \to \left(\mathsf{vk}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i \in [N]}\right)$: takes as input public parameters tspar, and outputs the verification key vk, and a partial verification key vk_i (used for abort identification) and secret key share sk_i for each signer $i \in [N]$. It implicitly sets up an empty state $\mathsf{st}_i := \emptyset$ for all N signers. We assume vk include tspar.

$\mathsf{TS.Preprocess}(\mathsf{vk}, i, \mathsf{sk}_i, \mathsf{st}_i) \to (\mathsf{st}_i, \mathsf{pp}_i)$: signing algorithm for a pre-processing round that takes as input a verification key vk, an index i of a signer, a secret key share sk_i and a state st_i of signer i, and outputs a *pre-processing token* pp_i and an updated state st_i.

$\mathsf{TS}.\mathsf{Sign}(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, i, (\mathsf{pp}_j)_{j\in\mathsf{act}}, \mathsf{sk}_i, \mathsf{st}_i) \to (\mathsf{st}_i, \mathsf{sig}_i)$: signing algorithm that takes as input a verification key vk, a signer set act, a message msg, an index $i \in \mathsf{act}$ of a signer, a tuple of pre-processing tokens $(\mathsf{pp}_j)_{j\in\mathsf{act}}$, a secret key share sk_i, and a state st_i of signer i and outputs a partial signature sig_i and an updated state st_i. We often denote $(\mathsf{pp}_j)_{j\in\mathsf{act}}$ by $\overrightarrow{\mathsf{pp}}$ for simplicity.

$\mathsf{TS}.\mathsf{Agg}(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, (\mathsf{pp}_i, \mathsf{sig}_i)_{i\in\mathsf{act}}) \to \mathsf{sig} \mid \perp$: aggregation algorithm that takes as input a verification key vk, a signer set act, a message msg, and a tuple of pre-processing tokens and partial signatures $(\mathsf{pp}_i, \mathsf{sig}_i)_{i\in\mathsf{act}}$, and outputs a combined signature sig (or $\perp$ in case of failure).

$\mathsf{TS}.\mathsf{Verify}(\mathsf{vk}, \mathsf{msg}, \mathsf{sig}) \to \mathsf{true} \mid \mathsf{false}$: verification algorithm that takes as input a verification key vk, a message msg, and a signature sig, and outputs true if the signature is valid and false otherwise.

Non-interactive IA. To support NI-IA, we adapt the notion from [81], introducing an additional algorithm to verify the validity of the partial signature output by party i.

$\mathsf{TS}.\mathsf{PartialVerify}(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, (\mathsf{pp}_j)_{j\in\mathsf{act}}, i, \mathsf{vk}_i, \mathsf{sig}_i) \to \mathsf{true} \mid \mathsf{false}$: takes as input a verification key vk, a signer set act, a message msg, a tuple of pre-processing tokens $(\mathsf{pp}_j)_{j\in\mathsf{act}}$, a signer index i, a partial verification key vk_i, and the partial signature sig_i, and outputs false if the signer i misbehaved in the session and true otherwise.

Proactive security. To support proactive security, we add a *key refresh* algorithm. This updates the secret key share sk_i, ensuring that even if a share from a previous epoch was compromised, the refreshed share for the subsequent epoch remains secure. For simplicity, we consider a setting where a single (potentially malicious) dealer is responsible for distributing refresh tokens to each signer.

$\mathsf{TS}.\mathsf{RefreshDealer}(T, N, \mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]}) \to ((\mathsf{vk}'_i)_{i\in[N]}, (\mathsf{rf}_i)_{i\in[N]})$: takes as input the threshold T, the number of parties N, a verification key vk, and a partial verification key vk_i for each signer, and outputs, for each signer $i \in [N]$, an updated partial verification key vk'_i and a refresh token rf_i.

$\mathsf{TS}.\mathsf{RefreshUpdate}(T, N, \mathsf{sk}_i, \mathsf{rf}_i) \to \mathsf{sk}'_i$: takes as input the threshold T, the number of parties N, a secret key share sk_i of signer i, and a refresh token rf_i, and outputs an updated secret key share sk_i.

Analogous to $\mathsf{PartialVerify}$, we introduce an algorithm to verify the validity of the updated partial verification keys and refresh tokens. This mechanism is essential in practice, as it guarantees that an honest party updates its secret key share only upon receiving a valid refresh token. Without this mechanism, the secret share sk_i could become malformed, resulting in an honest party being falsely accused of misbehavior in a signing protocol (i.e., trivially break IA security).

$\mathsf{TS}.\mathsf{RefreshVerify}((\mathsf{vk}_i)_{i\in[N]}, (\mathsf{vk}'_i)_{i\in[N]}) \to \mathsf{true} \mid \mathsf{false}$: checks that the updated partial keys $(\mathsf{vk}'_i)_i$ are consistent with $(\mathsf{vk}_i)_i$ (detects a dealer who hands out invalid updates).

$\mathsf{TS}.\mathsf{RefreshVerifyInd}(\mathsf{vk}_i, \mathsf{vk}'_i, \mathsf{rf}_i) \to \mathsf{true} \mid \mathsf{false}$: checks that rf_i correctly links vk_i to vk'_i (detects invalid per-signer refresh tokens).

Lastly, we define *correctness* for the refreshing mechanism, ensuring that a honest refresh will always pass the verification checks.

Definition 3.1. *The refreshing mechanism of* TS *is said correct if* $\mathsf{TS}.\mathsf{RefreshDealer}(T, N, \mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]})$ *outputs* $((\mathsf{vk}'_i)_{i\in[N]}, (\mathsf{rf}_i)_{i\in[N]})$ *such that:*

- $\mathsf{TS}.\mathsf{RefreshVerify}((\mathsf{vk}_i)_{i\in[N]}, (\mathsf{vk}'_i)_{i\in[N]}) = \mathsf{true}$ *with overwhelming probability,*
- $\mathsf{TS}.\mathsf{RefreshVerifyInd}(\mathsf{vk}_i, \mathsf{vk}'_i, \mathsf{rf}_i) = \mathsf{true}$ *with overwhelming probability.*

3.2 Security

We are now ready to define unforgeability and NI-IA with proactive securtiy.

Proactive unforgeability. We extend strong unforgeability from [73] to the proactive setting, with a focus on TS-UF-2 security from [10]. We consider an adversary that selectively corrupts parties on each honest refresh. Malicious refreshes are supported via a refresh update oracle.[5]

Definition 3.2. *Let* TS *be a two-round threshold signature scheme. We define the advantage* $\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$ *of an adversary* $\mathcal{A}$ *against the strong unforgeability with proactive security of* TS *as the probability that* $\mathcal{A}$ *wins in the game* $\mathsf{Game}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$ *depicted in Figure 3. The scheme* TS *is said to be* ts-suf-2 *if for any efficient adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$ *is negligible in* κ.

$\mathsf{Game}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$

$Q \coloneqq \emptyset$; $\mathsf{id}_{\mathsf{rf}} \coloneqq 0$; $\mathsf{Signed}[\cdot] \coloneqq \{\}$
$\mathsf{tspar} \leftarrow \mathsf{TS.Setup}(1^\kappa, N, T)$
$(\mathsf{CS}^{(0)}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}^H(\mathsf{tspar})$
$\mathsf{HS}^{(0)} \coloneqq [N] \backslash \mathsf{CS}^{(0)}$
for $i \in \mathsf{HS}^{(0)}$ **do** $\mathsf{st}_i \coloneqq \emptyset$
$(\mathsf{vk}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[N]}) \leftarrow \mathsf{TS.Keygen}(\mathsf{tspar})$
$O \coloneqq H, \mathsf{O}_{\mathsf{vk}_i}, \mathsf{O}_{\mathsf{TS.Preprocess}}, \mathsf{O}_{\mathsf{TS.Sign}}, \mathsf{O}_{\mathsf{TS.RefreshUpdate}}, \mathsf{O}_{\mathsf{TS.RefreshDealer}}$
$\mathsf{msg}^*, (\mathsf{sig}^*_j)_{j\in[\ell]} \leftarrow \mathcal{A}^O(\mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(0)}}, \mathsf{st}_{\mathcal{A}})$
require $(\mathsf{sig}^*_j)_{j\in[\ell]}$ pairwise distinct
$s \coloneqq 0$ // Count signatures on msg^*
for $\mathsf{ctnt} \in Q$ s.t. $\mathsf{ctnt} = (\cdot, \cdot, \mathsf{msg}^*, \cdot)$ **do**
  Parse $\mathsf{ctnt} = (\mathsf{id}^*, \mathsf{act}^*, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}^*)$
  if $|\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$ **then** $s \leftarrow s + 1$
return $\forall j \in [\ell], \mathsf{TS.Verify}(\mathsf{vk}, \mathsf{msg}^*, \mathsf{sig}^*_j) \wedge \ell > s$

$\mathsf{O}_{\mathsf{TS.Preprocess}}(i)$

require $i \in \mathsf{HS}^{(\mathsf{id}_{\mathsf{rf}})}$
$(\mathsf{st}_i, \mathsf{pp}_i) \leftarrow \mathsf{TS.Preprocess}(\mathsf{vk}, i, \mathsf{sk}_i, \mathsf{st}_i)$
return pp_i

$\mathsf{O}_{\mathsf{TS.Sign}}(\mathsf{act}, \mathsf{msg}, i, \overrightarrow{\mathsf{pp}})$

require $\mathsf{act} \subset [N] \wedge |\mathsf{act}| = T \wedge i \in \mathsf{HS}^{(\mathsf{id}_{\mathsf{rf}})} \cap \mathsf{act}$
$(\mathsf{st}_i, \mathsf{sig}_i) \leftarrow \mathsf{TS.Sign}(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, i, \overrightarrow{\mathsf{pp}}, \mathsf{sk}_i, \mathsf{st}_i)$
if $\mathsf{sig}_i \neq \perp$ **then**
  $\mathsf{ctnt} \leftarrow (\mathsf{id}_{\mathsf{rf}}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$
  $Q \leftarrow Q \cup \{\mathsf{ctnt}\}$; $\mathsf{Signed}[\mathsf{ctnt}] \leftarrow \mathsf{Signed}[\mathsf{ctnt}] \cup \{i\}$
return sig_i

$\mathsf{O}_{\mathsf{TS.RefreshUpdate}}(i, \mathsf{rf}_i)$

require $i \in \mathsf{HS}^{(\mathsf{id}_{\mathsf{rf}})}$
$\mathsf{sk}_i \leftarrow \mathsf{TS.RefreshUpdate}(T, N, \mathsf{sk}_i, \mathsf{rf}_i)$

$\mathsf{O}_{\mathsf{vk}_i}(i, \mathsf{vk}'_i)$

$\mathsf{vk}_i \leftarrow \mathsf{vk}'_i$

$\mathsf{O}_{\mathsf{TS.RefreshDealer}}((\mathsf{vk}_i)_{i\in[N]}, \mathsf{CS}', (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}})}})$

require $\mathsf{CS}' \subset [N] \wedge |\mathsf{CS}'| < T$
$((\mathsf{vk}'_i)_{i\in[N]}, (\mathsf{rf}_i)_{i\in[N]}) \leftarrow \mathsf{TS.RefreshDealer}(T, N, \mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]})$
for $i \in [N]$ **do** // Refresh keys and reset the state of participants
  $\mathsf{sk}_i \leftarrow \mathsf{TS.RefreshUpdate}(T, N, \mathsf{sk}_i, \mathsf{rf}_i)$; $\mathsf{st}_i \leftarrow \{\}$
$\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}}+1)} \coloneqq \mathsf{CS}'$; $\mathsf{HS}^{(\mathsf{id}_{\mathsf{rf}}+1)} \coloneqq [N] \backslash \mathsf{CS}'$; $\mathsf{id}_{\mathsf{rf}} \leftarrow \mathsf{id}_{\mathsf{rf}} + 1$
return $((\mathsf{vk}'_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}})}})$

Figure 3: Strong unforgeability game with proactive security. The adversary $\mathcal{A}$ wins if the game returns 1, i.e. if it forged a new signature.

Proactive NI-IA. We extend the NI-IA definition from [81] to the proactive setting. The adversary aims to frame a target honest signer i^* by causing PartialVerify to fail on their signature, potentially after executing key refreshes. As mentioned in Section 1.1, to prevent trivial framing via malformed updates, the game updates i^*'s key only if the new verification keys and refresh tokens satisfy RefreshVerify and RefreshVerifyInd, respectively.

Definition 3.3. *Let* TS *be a two-round threshold signature scheme. We define the advantage* $\mathsf{Adv}^{\mathsf{ts\text{-}ia}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$ *of an adversary* $\mathcal{A}$ *against the abort identification security of* TS *as the probability that* $\mathcal{A}$ *wins in the game* $\mathsf{Game}^{\mathsf{ts\text{-}ia}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T, i^*)$ *depicted in Figure 4, where* $i^* \in [N]$ *is the index of an honest signer to frame. The scheme* TS *is said to be* ts-ia *if for any efficient adversary* $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ts\text{-}ia}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$ *is negligible in* κ.

[5] While not strictly required, we assume a strong notion where unforgeability holds even if the parties accidentally updates their secret shares with an invalid refresh token (i.e., the game does not run RefreshVerify nor RefreshVerifyInd).

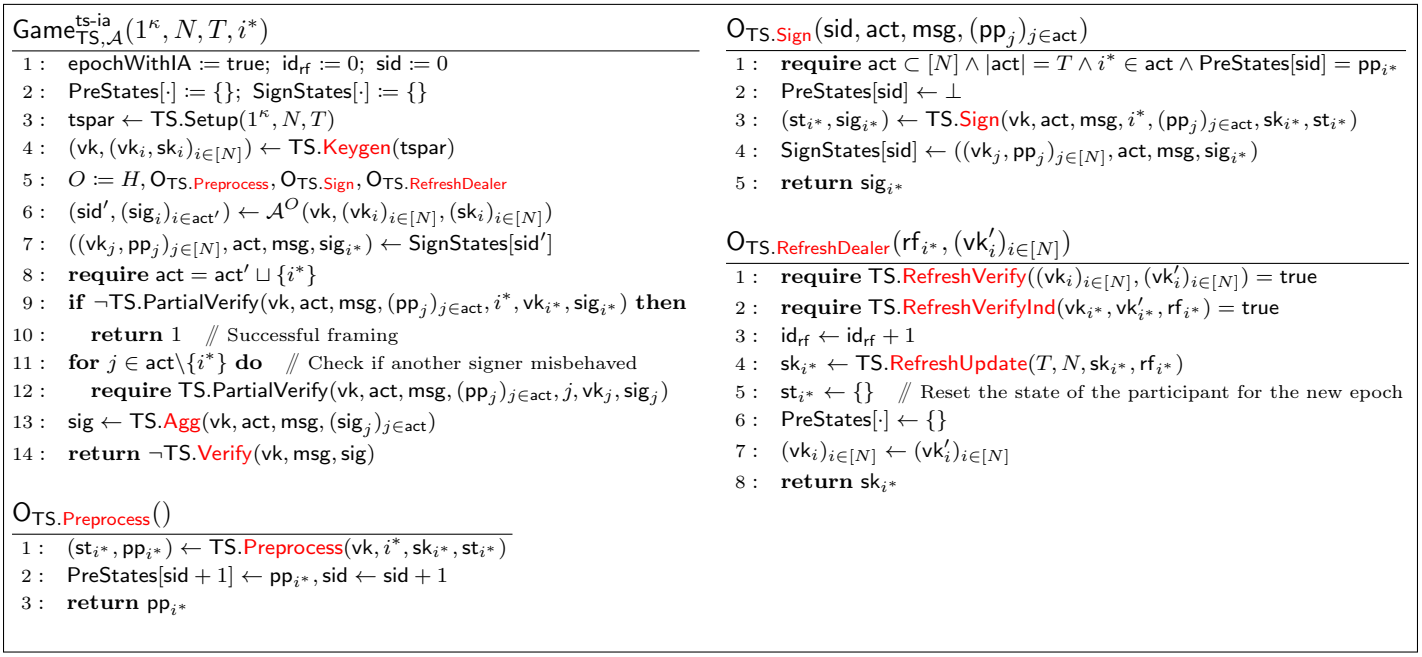


Figure 4: Non-interactive IA game with proactive security. The adversary $\mathcal{A}$ wins if the game returns 1, i. e. if it successfully frames the honest signer i^* or produces an invalid signature without any misbehaving signer detected in the signing set.

Remark 3.4 (Correctness of threshold signature). Note that security of NI-IA ensures correctness of a threshold signature scheme as it guarantees that honest parties are never marked as misbehaving, and subsequently that the final signature is always valid when all parties behave honestly.

4 An Everywhere-Short Secret Sharing Scheme

In this section, we propose an *everywhere-short* secret sharing scheme (SSS), called *Vandermonde* secret sharing, that splits a *short* secret $\mathbf{s} \in \mathcal{R}_q^\ell$ into *short* shares, which admits a *short* linear reconstruction algorithm (i.e., $\mathbf{s}$ can be reconstructed by simply summing the short shares).

4.1 Construction of Vandermonde Secret Sharing

Our Vandermonde secret sharing is an adaptation of the secret sharing scheme by Desmedt, Di Crescenzo, and Burmester [40], allowing to output *short* shares. A formal description of the Vandermonde secret sharing is given in Fig. 5, where the sharing is recursively run using VandShare and VandRecover.

We introduce the distribution χ from which we sample the partial secrets x_0 in the recursion (line 14 of VandShare), and derive x_1 as $x - x_0$. When $\chi = \mathcal{U}(\mathbb{Z}_q)$ with $x \in \mathbb{Z}_q$, we recover the original scheme from [40], with perfect privacy. In our case, we instantiate χ with a discrete Gaussian (see Table 2), outputting elements of small norm. We also rename the secret and partials secrets as x, x_L, x_R in our formal descriptions to more easily distinguish them from Hermine's secret when reasoning recursively about our secret sharing.

Correctness. We can see that our secret sharing is correct. Namely, given a list of shares produced by $\mathsf{Shr} \leftarrow \mathsf{VandShare}(x, [N], T, \varepsilon, \{\})$ for any secret x, any set $\mathsf{act} \subseteq [N]$ of size T can reconstruct x by running $\mathsf{Idx} \leftarrow \mathsf{VandRecover}([N], \mathsf{act}, \varepsilon, \{\})$ and computing $x = \sum_{i\in\mathsf{act}} \mathsf{Shr}[i][\mathsf{Idx}[i]]$. Notably, x is reconstructed by simply adding T shares.

VandShare$(x, \mathcal{P}, T, \mathsf{idx}, \mathsf{Shr})$

1 : **if** $T = 1$ **then**
2 : **for** $i \in \mathcal{P}$ **do** $\mathsf{Shr}[i][\mathsf{idx}] \leftarrow x$
3 : **else**
4 : $b \leftarrow \lfloor |\mathcal{P}|/2 \rfloor, N \leftarrow |\mathcal{P}|$
5 : $\mathcal{P}_L, \mathcal{P}_R \leftarrow \mathcal{P}[1 \dots b], \mathcal{P}[b+1 \dots N]$
6 : $\underline{T}_L, \overline{T}_L \leftarrow \max(0, T-(N-b)), \min(T, b)$
7 : **for** $k \in [\underline{T}_L, \overline{T}_L]$ **do**
8 : $\mathsf{idx}_L \leftarrow (\mathsf{idx} \| \texttt{":L:"} \| k)$, $\mathsf{idx}_R \leftarrow (\mathsf{idx} \| \texttt{":R:"} \| (T-k))$
9 : **if** $k = 0$ **then**
10 : $\mathsf{Shr} \leftarrow$ VandShare$(x, \mathcal{P}_R, T, \mathsf{idx}_R, \mathsf{Shr})$
11 : **elseif** $k = T$ **then**
12 : $\mathsf{Shr} \leftarrow$ VandShare$(x, \mathcal{P}_L, T, \mathsf{idx}_L, \mathsf{Shr})$
13 : **else**
14 : $x_L \leftarrow \chi, x_R \leftarrow (x - x_L) \bmod q$
15 : $\mathsf{Shr} \leftarrow$ VandShare$(x_L, \mathcal{P}_L, k, \mathsf{idx}_L, \mathsf{Shr})$
16 : $\mathsf{Shr} \leftarrow$ VandShare$(x_R, \mathcal{P}_R, T-k, \mathsf{idx}_R, \mathsf{Shr})$
17 : **return** Shr

VandRecover$(\mathcal{P}, \mathsf{act}, \mathsf{idx}, \mathsf{Idx})$

1 : $N \leftarrow |\mathcal{P}|$; $T \leftarrow |\mathsf{act}|$
2 : **if** $T = 1$ **then**
3 : **for** $i \in \mathsf{act}$ **do** $\mathsf{Idx}[i] \leftarrow \mathsf{idx}$
4 : **else**
5 : $b \leftarrow \lfloor N/2 \rfloor$
6 : $\mathcal{P}_L, \mathcal{P}_R \leftarrow \mathcal{P}[1 \dots b], \mathcal{P}[b+1 \dots N]$
7 : $\mathsf{act}_L, \mathsf{act}_R \leftarrow \mathsf{act} \cap \mathcal{P}_L, \mathsf{act} \cap \mathcal{P}_R$
8 : $k \leftarrow |\mathsf{act}_L|$
9 : $\mathsf{idx}_L \leftarrow (\mathsf{idx} \| \texttt{":L:"} \| k)$, $\mathsf{idx}_R \leftarrow \mathsf{idx} \| \texttt{":R:"} \| (T-k)$
10 : **if** $k = 0$ **then**
11 : $\mathsf{Idx} \leftarrow$ VandRecover$(\mathcal{P}_R, \mathsf{act}_R, \mathsf{idx}_R, \mathsf{Idx})$
12 : **elseif** $k = T$ **then**
13 : $\mathsf{Idx} \leftarrow$ VandRecover$(\mathcal{P}_L, \mathsf{act}_L, \mathsf{idx}_L, \mathsf{Idx})$
14 : **else**
15 : $\mathsf{Idx} \leftarrow$ VandRecover$(\mathcal{P}_L, \mathsf{act}_L, \mathsf{idx}_L, \mathsf{Idx})$
16 : $\mathsf{Idx} \leftarrow$ VandRecover$(\mathcal{P}_R, \mathsf{act}_R, \mathsf{idx}_R, \mathsf{Idx})$
17 : **return** Idx

Figure 5: Our *short* Vandermonde secret sharing. To share the original secret, we initialize the set of parties $\mathcal{P}$ to $[N]$, the label idx to the empty string ε, and the lists $\mathsf{Shr}[\cdot][\cdot]$ and $\mathsf{Idx}[\cdot]$ to $\{\}$. $\mathcal{P}$ and act are always lexicographically ordered.

In more detail, correctness follows from the combinatorial decomposition of the Vandermonde's identity (cf. Eq. (1)). Consider a set of T parties $\mathsf{act} \subseteq [N]$: the base case $T = 1$ is trivial since all parties receive the secret x as shares; for $T > 1$, at each step of the recursion, there will be a k such that k of the parties are in $\mathcal{P}_L$ and $T-k$ in $\mathcal{P}_R$ (as computed in VandRecover), thus allowing to reconstruct both x_L and x_R and consequently the secret $x = x_L + x_R$. Note that we make sure that the indices used to store and retrieve the shares are never ambiguous by appending the path taken in the recursion to the label idx.

Comparison with other sharing schemes. Our modification for keeping the shares small could be applied to other secret sharing schemes such as replicated secret sharing and those by Benaloh and Leichter [12], often used in the lattice-based literature. However, for the range of parameters we consider in this work ($N \leqslant 64$), the Vandermonde secret sharing appears as the *only practical solution*, with a reasonable number of shares per party. We provide the detailed comparison in Section A. Lastly, for the sake of reference, we include a minor optimization of the Vandermonde secret sharing in Section B, where we propose a more efficient sharing strategy for specific pairs of T and N.

4.2 Simulation Security

Overview. As discussed in Section 1.4, the privacy guarantee of the secret $x \in \mathbb{Z}_q$ depends on the specific distribution χ. For instance, if $\chi = \mathcal{U}(\mathbb{Z}_q)$, we can prove that x is perfectly hidden even given the set of shares $(\mathsf{Shr}[i])_{i \in \mathsf{CS}}$ for any set of "corrupt" parties $\mathsf{CS} \subset [N]$ with $|\mathsf{CS}| \leq T-1$; this is the standard privacy notion for secret sharing. However, when χ is a distribution of small elements, which is the case of interest to us, the shares leak non-trivial information on x.

We therefore diverge from the standard notion of privacy and introduce a *simulation-based* security notion (in Theorem 4.1), taking into account the amount of leakage of the secret x. Specifically, we require that there exists a simulator SimShares that takes as input *hints* on the secret x of the form $x + x_0$ for several x_0 sampled from χ such that

(P.1) For any set $\mathsf{CS} \subset [N]$ of size at most $T-1$, SimShares can perfectly simulate shares $(\mathsf{Shr}[i])_{i \in \mathsf{CS}}$ for all parties in CS.

We show this property by observing that in Vandermonde secret sharing, every partial secrets in the recursion can be interpreted as hints on x, and that the total leakage x can be captured by the leakage of several hints $x + x_0$ on the original secret x. We highlight that if the simulator did not take the hints as input, it would match the standard definition of perfect privacy.

Unfortunately, in the context of our threshold signature, even this generalized simulation-based definition is insufficient. This is because an adversary can ask any set $\mathsf{act} \subset [N]$ of size T to engage in a signing protocol. From an unforgeability proof perspective, this means the reduction must simulate the partial signatures of the "honest" parties $\mathsf{act} \cap \mathsf{HS}$ where $\mathsf{HS} = [N] \backslash \mathsf{CS}$. However, this is a conflicting requirement: if $\mathsf{SimShares}$ could generate *all* honest shares explicitly, then it could also reconstruct the secret x fully by combining with the shares of $T - 1$ corrupted parties, effectively invalidating any privacy guarantee.

To circumvent this issue, we require the simulator $\mathsf{SimShares}$ to satisfy an additional property on the honest parties shares:

(P.2) For any set $\mathsf{act} \subset [N]$ of size T, and $\mathsf{act}' \subset \mathsf{act}$ of size less than $T - |\mathsf{CS}|$, $\mathsf{SimShares}$ can perfectly simulate $(\mathsf{Shr}[i][\mathsf{Idx}[i]])_{\mathsf{act}''}$ where act'' is some set of size $T - 1$ with $\mathsf{act}' \subseteq \mathsf{act}'' \subset \mathsf{act}$, where $\mathsf{Idx} \leftarrow \mathsf{VandRecover}(\mathcal{P}, \mathsf{act}, \varepsilon, \{\})$.

That is, while the simulator cannot simulate the entire shares of the honest parties, it can simulate *all but one* of the *individual* shares to reconstruct the secret x; namely, $\mathsf{Shr}[j^*][\mathsf{Idx}[j^*]] = x - \sum_{i \in \mathsf{act}'} \mathsf{Shr}[i][\mathsf{Idx}[i]]$ where $j^* = \mathsf{act} \backslash \mathsf{act}''$. Intuitively, this captures that even given $T - |\mathsf{CS}| - 1$ shares of (potentially all honest) parties is not enough to reconstruct the secret x, even when combined with the $|\mathsf{CS}|$ shares of corrupted parties. Looking ahead, the partial signature of this special honest party j^*, which the simulator cannot simulate, will be handled in our unforgeability proof of $\mathsf{Hermine}$ by invoking the $\mathsf{AOM\text{-}MISIS}$ problem.[6]

Note that (P.2) implies that the simulator is provided with the set act and act'.[7] However, these sets may not be known in advance by a reduction leveraging our simulation property, and guessing them would lead to an exponential blowup. One way to circumvent this is to rather provide the simulator with the index j^* of the special honest party, and require (P.2) to hold for any choice of act and act' such that j^* would have been the special party (captured by the introduction of a function $\mathsf{SecureUser}$ in our simulation property). This user index can be easily guessed by the reduction with only a linear loss in N.

Additional considerations. To support proactive security, we add a Vandermonde secret sharing of 0 to the existing shares. For a standard perfectly private sharing, any subset of size at most $T - 1$ of the new shares (i.e., set of corrupted parties in the next epoch) will be independent of the old shares. However, in our setting, we must take into account of the non-trivial leakage, and model this using a dedicated simulator $\mathsf{SimRefresh}$.

Simulation security. Before stating the theorem, we introduce two bookkeeping quantities $B_{\mathsf{shr}}(N, T)$, and $B_{\mathsf{rf}}(N, T)$. They upper bound the number of *hints* that needs to be consumed by the simulator. Concretely, a hint is a value of the form $h = x + x_i$ where $x_i \leftarrow \chi$, and each time the simulator needs to embed the unknown secret into a "secure" share (rather than sampling it directly), it consumes one such hint; these quantities are defined by counting this usage in the recursive constructions (see the proof of Theorem 4.1). The base case is $B_{\mathsf{shr}}(N, T) = B_{\mathsf{rf}}(N, T) \coloneqq 0$ if $T \in \{0, 1\}$. Otherwise, they are defined recursively. Let $N_L \coloneqq \lfloor N/2 \rfloor$, $N_R \coloneqq \lceil N/2 \rceil$, $T_R \coloneqq T - T_L$, with $T_L \in \big[\max(0, T - N_R), \min(N_L, T)\big]$:

$$B_{\mathsf{shr}}(N, T) \coloneqq \sum_{T_L} \max\Big\{1 + B_{\mathsf{shr}}(N_L, T_L),\ B_{\mathsf{shr}}(N_R, T_R)\Big\},$$

$$B_{\mathsf{rf}}(N, T) \coloneqq \sum_{T_L} \max\big\{B_{\mathsf{rf}}(N_L, T_L),\ B_{\mathsf{rf}}(N_R, T_R), 1 + B_{\mathsf{shr}}(N_L, T_L) + B_{\mathsf{shr}}(N_R, T_R)\big\}.$$

[6] To be specific, thanks to Item (P.2), the only non-trivial information of x (other than the hints $\mathsf{SimShares}$ takes as input) that leaks is through party j^*'s partial signature. The reduction uses $\mathsf{AOM\text{-}MISIS}$ to simulate this and our proof relies on the fact that each signing protocol query exhausts only one query to $\mathsf{AOM\text{-}MISIS}$.

[7] In the context of threshold signatures, act is the signer set associated to the forgery and act' is the parties that participated in that signing session (cf., $\mathsf{TS\text{-}UF\text{-}2}$ security).

Theorem 4.1. *Let N be the number of parties, T be the threshold, and $\mathbf{s}$ be a secret vector.*[8] *There exist two efficient algorithms,* $\mathsf{SecureUser}$ *and* $\mathsf{SimShares}$*, that together ensure the following properties:*

1. ***Identifying a Secure Participant:*** $\mathsf{SecureUser}(\mathsf{CS}, \mathsf{act}, \mathsf{act}')$ *takes a set of corrupted parties* $\mathsf{CS} \subset [N]$*, a set* $\mathsf{act} \subset [N]$ *of size* T*, and a subset* $\mathsf{act}' \subset \mathsf{act} \cap \mathsf{HS}$ *of parties with* $\mathsf{HS} = [N] \backslash \mathsf{CS}$*. If* $|\mathsf{act}'| < T - |\mathsf{CS}|$*, it outputs a designated "secure" participant* $j^* \in (\mathsf{act} \cap \mathsf{HS}) \setminus \mathsf{act}'$*.*

2. ***Simulating Shares:*** $\mathsf{SimShares}(T, \mathsf{CS}, j^*, (h_i)_{i \in [B_{\mathsf{shr}}(N,T)]})$ *takes as input the set of parties* $\mathcal{P}$*, the threshold* T*, the set of corrupted parties* CS *such that* $|\mathsf{CS}| < T$*, a party index* j^**, and* $B_{\mathsf{shr}}(N, T)$ *hints on* $\mathbf{s}$ *of the form* $\mathbf{s} + \mathbf{s}_i$ *where* $N = |\mathcal{P}|$ *and* $\mathbf{s}_i \leftarrow \chi$*. It then produces a set of simulated shares and corresponding simulation bits,* $\{\mathsf{Shr}'[i][\mathsf{idx}], b[i][\mathsf{idx}]\}_{i,\mathsf{idx}}$*, for all parties and share indices, satisfying:*

   (a) *For any corrupted party* $i \in \mathsf{CS}$*, the simulator outputs their true share, indicated by* $b[i][\mathsf{idx}] = 0$ *for all* idx*.*

   (b) *For any set* act*, let* $\mathsf{Idx} = \mathsf{VandRecover}([N], \mathsf{act}, , \varepsilon, \{\})$*. Then, there is only one participant* $i \in \mathsf{act} \cap \mathsf{HS}^{(0)}$ *such that* $b[i][\mathsf{Idx}[i]] = 1$ *where* $\mathsf{HS}^{(0)} = \mathsf{act} \setminus \mathsf{CS}^{(0)}$*.*

   (c) *For any set* act*, and set* act'*, let* $\mathsf{Idx} = \mathsf{VandRecover}([N], \mathsf{act}, \varepsilon, \{\})$ *and* $j^* = \mathsf{SecureUser}(\mathsf{CS}^{(0)}, \mathsf{act}, \mathsf{act}')$*. Then:*

      i. $b[j^*][\mathsf{Idx}[j^*]] = 1$ *for the designated secure party* j^**. This signifies that the simulator does not know this share directly.*

      ii. $b[i][\mathsf{Idx}[i]] = 0$ *for any other party* $i \in \mathsf{act} \setminus \{j^*\}$*. This means the simulator can compute their shares.*

   (d) ***Correct Distribution:*** *The collection of shares reconstructed from the simulator's output, defined by the rule, for each* i *and* idx*,*

$$\mathsf{Shr}[i][\mathsf{idx}] = \begin{cases} \mathbf{s} - \mathsf{Shr}'[i][\mathsf{idx}] & \text{if} \quad b[i][\mathsf{idx}] = 1 \\ \mathsf{Shr}'[i][\mathsf{idx}] & \text{if} \quad b[i][\mathsf{idx}] = 0 \end{cases}, \tag{4}$$

   is identically distributed as a true Vandermonde sharing of the secret $\mathbf{s}$ *produced by* $\mathsf{VandShare}(\mathbf{s}, [N], T, \varepsilon, \{\})$*.*

To support proactive refreshing, we define an additional simulator $\mathsf{SimRefresh}(T, \mathsf{CS}, j^*, (h_i)_{i \in [B_{\mathsf{rf}}(N,T)]}, \mathsf{Shr}', b) \to (\mathsf{Shr}', b)$ *that, given a new set of corrupted parties, the threshold* T*, the party index* j^**,* $B_{\mathsf{rf}}(N, T)$ *fresh hints on* $\mathbf{s}$*, and the previously simulated shares and simulation bits, can produce simulated refreshed shares and simulation bits. We require that it maintains the same constraints on* b *as* $\mathsf{SimShares}$*, and correctness property for the refreshed shares, defined by the sum of the shares before refresh and the shares of a Vandermonde sharing of* $\mathbf{0}$*.*

Proof of Lemma 4.1. Our simulation algorithms are recursive. In addition to the inputs stated in the theorem, we assume that they accept a set of users $\mathcal{P}$ as input, which is the set of users involved in the current step of recursion. The initial call to these algorithms will be with $\mathcal{P} = [N]$. For simplicity, we omit recursion-depth subscripts; variables represent the values associated with the current step. Additionally, we treat idx as the global path from the root, implicitly assuming the path prefix is inherited.

Defining $\mathsf{SecureUser}$**.** Let us start by defining $\mathsf{SecureUser}$. We will proceed recursively, reducing the number of users N at each step to ensure proper definition. At a high level, we choose a path in the Vandermonde tree that leads to a user j^* whose share is not corrupted and has not participated in the current protocol instance. Looking forward, given the index j^*, we will later be able to generate simulated shares so that for any instance where j^* is the designated secure user, the shares of other users are directly output by the simulator, while the share of user j^* is not, although the difference between the secret $\mathbf{s}$ and their

[8] To be consistent with how Vandermonde secret sharing is used in our scheme, we consider a vector $\mathbf{s}$ for the secret without loss of generality.

share is, capturing the embedding of the secret in j^*'s share. We will proceed by "virtually" corrupting the shares on all the co-paths of j^* in the tree. We will maintain the invariant that at each level of recursion, $|\mathsf{act}'| < T - |\mathsf{CS}|$ and $|\mathsf{CS}| < T$. For clarity, we denote the total number of users in $\mathcal{P}$ as N. We define our recursive algorithm $\mathsf{SecureUser}(\mathcal{P}, \mathsf{CS}, \mathsf{act}, \mathsf{act}')$ as follows:

- **Base Case ($T = 1$):** If $T = 1$, the condition $|\mathsf{act}'| < T - |\mathsf{CS}|$ implies $|\mathsf{act}'| < 1 - |\mathsf{CS}|$, so act' must be empty and $|\mathsf{CS}|$ must be 0. We return the honest user in act as j^*. Notice that act contains exactly one user in this case as $|\mathsf{act}| = T = 1$.
- **Recursive Case ($N \geqslant T > 1$):** We split the set of users $\mathcal{P}$ into two halves: $\mathcal{P}_L$ with b users, and $\mathcal{P}_R$ with $N - b$ users. Also, as in the reconstruction algorithm VandRecover, we determine a sub-threshold k for $\mathcal{P}_L$ and $T - k$ for $\mathcal{P}_R$, such that $|\mathsf{act} \cap \mathcal{P}_L| = k$ and $|\mathsf{act} \cap \mathcal{P}_R| = T - k$. We denote the set of corrupted users (resp. the users who already participated) in each half by CS_L and CS_R (resp. act'_L and act'_R).

  The core idea is to find a sub-sharing (either in $\mathcal{P}_L$ or $\mathcal{P}_R$) where the recursion invariant holds, allowing us to continue.

  - **Case 1: One sub-sharing is fully compromised.** If $|\mathsf{CS}_L| \geqslant k$ or $|\mathsf{CS}_R| \geqslant T - k$, the adversary can reconstruct the corresponding sub-secret (x_L or x_R) on their own. The security of the secret $\mathbf{s}$ now rests entirely on the other half. We can safely recurse into that other half. For instance, if $|\mathsf{CS}_L| \geqslant k$, we recurse on $\mathcal{P}_{\mathsf{next}} = \mathcal{P}_R$ with threshold $T_{\mathsf{next}} = T - k$. Then, we set $\mathsf{act}'_{\mathsf{next}} = \mathsf{act}'_R$ and $\mathsf{CS}_{\mathsf{next}} = \mathsf{CS}_R$. The recursion condition holds: $|\mathsf{act}'_{\mathsf{next}}| = |\mathsf{act}'_R| \leqslant |\mathsf{act}'| < T - |\mathsf{CS}| = T - (|\mathsf{CS}_L| + |\mathsf{CS}_R|) \leqslant T - k - |\mathsf{CS}_R| = T_{\mathsf{next}} - |\mathsf{CS}_{\mathsf{next}}|$. The argument is symmetric if $|\mathsf{CS}_R| \geqslant T - k$.
  - **Case 2: Neither sub-sharing is fully compromised.** If $|\mathsf{CS}_L| < k$ and $|\mathsf{CS}_R| < T - k$, we prove by contradiction that at least one half must satisfy the recursion invariant. Assume by contradiction that both halves violate the invariant:

    $$|\mathsf{act}'_L| \geqslant k - |\mathsf{CS}_L| \quad \text{and} \quad |\mathsf{act}'_R| \geqslant (T - k) - |\mathsf{CS}_R|$$

    Summing these inequalities gives:

    $$|\mathsf{act}'| = |\mathsf{act}'_L| + |\mathsf{act}'_R| \geqslant (k - |\mathsf{CS}_L|) + ((T - k) - |\mathsf{CS}_R|) = T - |\mathsf{CS}|$$

    This contradicts our invariant that $|\mathsf{act}'| < T - |\mathsf{CS}|$. Therefore, at least one of the conditions $|\mathsf{act}'_L| < k - |\mathsf{CS}_L|$ or $|\mathsf{act}'_R| < (T-k) - |\mathsf{CS}_R|$ must hold, and we can recurse into the corresponding half to define j^* as in Case 1.

Defining SimShares. We now turn to defining SimShares. We will again proceed by recursion on the number of users N and show that we can simulate shares with the required properties using $B_{\mathsf{shr}}(N, T)$ hints on the secret $\mathbf{s}$. Here, $B_{\mathsf{shr}}(N, T)$ is defined as the maximum number of hints needed to simulate shares for N users with threshold T. We define our recursive algorithm $\mathsf{SimShares}(\mathcal{P}, T, \mathsf{CS}, j^*, (h_i)_{i \in [B_{\mathsf{shr}}(N,T)]})$ for the secret $\mathbf{s}$ as follows:

- **Base Case ($T = 1$):** All users receive $\mathbf{s}$ as their share. We set $\mathsf{Shr}'[u][\mathsf{idx}] = 0$ and $b[u][\mathsf{idx}] = 1$ for all $u \in \mathcal{P}$ and return them as the output.
- **Recursive Case ($N \geqslant T > 1$):** We split users in $\mathcal{P}$ into $\mathcal{P}_L$ and $\mathcal{P}_R$. For a given recursive step in the sharing, the secret $\mathbf{s}$ is split into $x_L \leftarrow \chi$ and $x_R = \mathbf{s} - x_L$. The core of the simulation is to decide to reveal one of x_L or x_R to the simulator, while embedding the secret in the other half and use the induction hypothesis to simulate its shares. We refer to the half in which the corresponding share is revealed as *the revealed half*, and the other half as *the secure half*. In short, the simulator honestly generates shares of the secret for the revealed half and simulates shares of the secret for the secure half in the recursive manner. Below, we provide the concrete procedure.

1. **Split $\mathcal{P}$ into the "revealed" half and the "secure" half:** The choice of which half to be the secure half depends on the location of the user j^* and the distribution of corrupted parties. If the half containing j^* is not fully compromised by existing corruptions (i.e., its sub-threshold is greater than the number of corrupted parties within it), we regard this set as the secure half. Otherwise, the other half is the secure half.[9] It is easy to check that at any level of the recursion, there is at least one secure half (by the same contradiction argument as in the definition of SecureUser). Note that we do not need to satisfy the properties for the user j^* when it falls in a revealed half, since j^* will never be selected as secure in that case by construction of SecureUser.

   Let $(\mathsf{rvl}, \mathsf{scr}) \in \{(L,R),(R,L)\}$ be subscripts such that $\mathcal{P}_{\mathsf{rvl}}$ is the revealed half and $\mathcal{P}_{\mathsf{scr}}$ is the secure half, respectively. This means the simulator will learn x_{rvl} but not x_{scr}. Also, we denote corresponding sub-threshold by T_{rvl} and T_{scr}. Notice that $T_{\mathsf{rvl}} + T_{\mathsf{scr}} = T$ holds.
2. **Generate the "revealed" share:** In the simulation, if $\mathsf{rvl} = L$, we sample $x_L \leftarrow \chi$. If $\mathsf{rvl} = R$, we choose $x_R = h_i$ for some hint $h_i = \mathbf{s} + \mathbf{s}'_i$ on $\mathbf{s}$.
3. **Generate shares for the revealed half $\mathcal{P}_{\mathsf{rvl}}$:** The simulator runs the real sharing algorithm VandShare under the secret x_{rvl}, the user set $\mathcal{P}_{\mathsf{rvl}}$, and the threshold T_{rvl}. This algorithm finally outputs shares of x_{rvl} for $\mathcal{P}_{\mathsf{rvl}}$. Then, it sets the simulation bit $b[u][\mathsf{idx}] = 0$ and $\mathsf{Shr}'[u][\mathsf{idx}]$ to the share corresponding to the path idx for all $u \in \mathcal{P}_{\mathsf{rvl}}$ and idx in this sub-tree.
4. **Recurse on the "secure" half ($\mathcal{P}_{\mathsf{scr}}$):** The simulator needs to simulate shares for $\mathcal{P}_{\mathsf{scr}}$ for the secret $x_{\mathsf{scr}} = \mathbf{s} - x_{\mathsf{rvl}}$. It cannot compute x_{scr} directly since it does not have access to $\mathbf{s}$. However, it has hints of the form $h_i = \mathbf{s} + \mathbf{s}_i$. To run SimShares under $\mathcal{P}_{\mathsf{scr}}$ and this sub-secret x_{scr}, it transforms these hints into hints h'_i on x_{scr} by computing $h'_i = h_i - x_{\mathsf{rvl}} = (\mathbf{s} - x_{\mathsf{rvl}}) + \mathbf{s}_i = x_{\mathsf{scr}} + \mathbf{s}_i$. With these new hints, it makes a recursive call: $\mathsf{SimShares}(\mathcal{P}_{\mathsf{scr}}, T_{\mathsf{scr}}, \mathsf{CS}_{\mathsf{scr}}, j^*, (h'_i)_i)$ where $\mathsf{CS}_{\mathsf{scr}} = \mathsf{CS} \cap \mathcal{P}_{\mathsf{scr}}$. As a result of recursion, it outputs the simulated shares Shr' and bits b. The simulator updates Shr' by $\mathsf{Shr}'[u][\mathsf{idx}] \leftarrow \mathsf{Shr}'[u][\mathsf{idx}] + x_{\mathsf{rvl}}$ for $u \in \mathcal{P}_{\mathsf{scr}}$ and idx such that $b[u][\mathsf{idx}] = 1$.

(*Checking properties.*) We now show that Shr' and bits b generated by SimShares at any level of recursion satisfy the requirements in this lemma. By construction, the number of hints suffices to run the algorithm. We first check property (a). We set $b[i][\mathsf{idx}]$ to 1 only when its path falls into the base case. Recall that CS at the root of recursion satisfy $|\mathsf{CS}| < T$. By construction of SimShares, a corrupt set $\mathsf{CS}_{\mathsf{scr}}$ for a secure half $\mathcal{P}_{\mathsf{scr}}$ also satisfies $|\mathsf{CS}_{\mathsf{scr}}| < T_{\mathsf{scr}}$. Since we recursively run SimShares only under a secure half, $\mathsf{CS}_{\mathsf{scr}}$ is empty when this step falls into the base case. Thus, no corrupted user is assigned 1, which ensures (a).

We move to (b) and (c). When we reach the base case $T = 1$ for the shares used to reconstruct for the set act, there is only one user i in $\mathsf{act} \cap \mathcal{P}_{\mathsf{scr}}$. Indeed, if there were more than one such user, then those users would be assigned the same secret under act, which would contradict the threshold requirement. Due to property (a), $\mathcal{P}_{\mathsf{scr}}$ contains only honest users. Thus (b) holds. Furthermore, if j^* is the user returned by SecureUser for the set act, then j^* is necessarily the user that is reached in the base case for the set act. Indeed, by construction of SimShares, we also choose the path leading to j^* as secure (and the co-path as revealed) whenever the half containing j^* is not fully corrupted. Therefore (c) also holds.

Finally, we check (d). In the base case, it is easy to check this as $\mathbf{s} - \mathsf{Shr}'[u][\mathsf{idx}] = \mathbf{s} - 0 = \mathbf{s}$. For a revealed half in the recursive case, Shr' is set to the corresponding share of x_{rvl} generated by VandShare. Recall that shares of a parent secret $\mathbf{s}$ consist of shares of x_L and shares of x_R. Thus Shr' in this case are distributed correctly. For a secure half, we use induction to show it. The simulated shares Shr' and bits b produced by SimShares on $\mathcal{P}_{\mathsf{scr}}$ for a sub-secret x_{scr} satisfy required properties. For (u, idx) such that $b[u][\mathsf{idx}]$ is 0, $\mathsf{Shr}'[u][\mathsf{idx}]$ is identically distributed to a real share of x_{scr}, namely, a share of $\mathbf{s}$, which ensures (d) under $\mathbf{s}$. For (u, idx) such that $b[u][\mathsf{idx}] = 1$, let $\mathsf{Shr}'_{\mathsf{old}}$ and $\mathsf{Shr}'_{\mathsf{new}}$ be Shr' before and after updating, respectively, such that $\mathsf{Shr}'_{\mathsf{new}}[u][\mathsf{idx}] = \mathsf{Shr}'_{\mathsf{old}}[u][\mathsf{idx}] + x_{\mathsf{rvl}}$. Now we have

$$\mathbf{s} - \mathsf{Shr}'_{\mathsf{new}}[u][\mathsf{idx}] = (x_{\mathsf{rvl}} + x_{\mathsf{scr}}) - (\mathsf{Shr}'_{\mathsf{old}}[u][\mathsf{idx}] + x_{\mathsf{rvl}}) = x_{\mathsf{scr}} - \mathsf{Shr}'_{\mathsf{old}}[u][\mathsf{idx}].$$

By the induction hypothesis (i.e., property (d) for the recursive call on secret x_{scr}), the value $x_{\mathsf{scr}} - \mathsf{Shr}'_{\mathsf{old}}[u][\mathsf{idx}]$ is distributed identically to the corresponding honest share of x_{scr} on that path. Since we have just shown

[9] It is fine that j^* is involved in a fully corrupted sub-sharing, as whenever it will use the shares from this sharing, it will not be selected by SecureUser and thus will not need to satisfy property (b).

that $\mathbf{s} - \mathsf{Shr}'_{\mathsf{new}}[u][\mathsf{idx}] = x_{\mathsf{scr}} - \mathsf{Shr}'_{\mathsf{old}}[u][\mathsf{idx}]$, it follows that $\mathbf{s} - \mathsf{Shr}'_{\mathsf{new}}[u][\mathsf{idx}]$ is distributed identically to the corresponding honest share of $\mathbf{s}$. Therefore, property (d) holds.

Extending Simulation for Proactive Refreshing. The simulator $\mathsf{SimRefresh}$ updates the simulated shares Shr' and bits b to account for a public refresh vector (in the form of a sharing of $\mathbf{0}$). To construct $\mathsf{SimRefresh}$ recursively, we prove a slightly stronger simulation result where the refresh vector is in the form of a sharing of a public vector $\mathbf{s}_r$ (not necessarily $\mathbf{0}$). This handles the recursive step where we sample sub-secrets for the refresh sharing. We add superscripts $\cdot^{(e)}$ and $\cdot^{(e+1)}$ to variables to denote whether they are associated with the current epoch e or the next epoch $e+1$.

We wish to maintain the same properties as $\mathsf{SimShares}$ after refreshing, but with the new set of corrupted parties. The main challenge arises *when the secure half flips due to new corruptions* (e.g., from $\mathcal{P}_L$ to $\mathcal{P}_R$). The "flip" scenario requires revealing refreshed shares without compromising secure shares in the previous epoch. To deal with this, we need a way to reveal all refreshed shares for new revealed halves in the flipped case.

The following lemma solves this by providing a mechanism to reveal all shares of a refreshed sub-secret $x_{\mathsf{rvl}}^{(e+1)}$ using the knowledge of it and hints on the previous secure sub-secret $x_{\mathsf{scr}}^{(e)}$. To show the following lemma, we use the fact that we can reconstruct any sub-secret for revealed halves by using $\mathsf{Shr}'^{(e)}$, which we will show later.

Lemma 4.2. *Let $x_{\mathsf{rvl}}^{(e+1)}$ be a next revealed sub-secret, $x_{\mathsf{scr}}^{(e)}$ be a secure sub-secret vector in epoch e, and $\hat{x}_{\mathsf{rvl}}$ be a share of a refresh vector $\mathbf{s}_r$ corresponding to the next revealed half such that $x_{\mathsf{rvl}}^{(e+1)} = x_{\mathsf{scr}}^{(e)} + \hat{x}_{\mathsf{rvl}}$ and the distribution of it is identical to a real share. Let $\mathsf{Shr}'^{(e)}$ and bits $b^{(e)}$ be previously simulated shares and bits. There exists an efficient algorithm $\mathsf{ShareReveal}(\mathcal{P}, T, x_{\mathsf{rvl}}^{(e+1)}, (h_i)_{i\in[B_{\mathsf{shr}}(N,T)]}, \mathsf{Shr}'^{(e)}, b^{(e)})$ that takes as input the set of parties $\mathcal{P}$ corresponding to this path, $x_{\mathsf{rvl}}^{(e+1)}$, $B_{\mathsf{shr}}(N,T)$ hints on $x_{\mathsf{scr}}^{(e)}$, $\mathsf{Shr}'^{(e)}$, and $b^{(e)}$ and outputs all shares of $x_{\mathsf{rvl}}^{(e+1)}$ such that they are distributed correctly.*

Proof of Lemma 4.2. The goal is to produce all shares of a next revealed sub-secret $x_{\mathsf{rvl}}^{(e+1)} = x_{\mathsf{scr}}^{(e)} + \hat{x}_{\mathsf{rvl}}$ leveraging a simulated sharing $\mathsf{Shr}'^{(e)}$ and hints. We note that one half in the sub-sharing of $x_{\mathsf{scr}}^{(e)}$ is a secure half since $x_{\mathsf{scr}}^{(e)}$ is a secure share. To show this lemma, we proceed recursively. We define our recursive algorithm $\mathsf{ShareReveal}(\mathcal{P}, T, x_{\mathsf{rvl}}^{(e+1)}, (h_i)_{i\in[B_{\mathsf{shr}}(N,T)]}, \mathsf{Shr}'^{(e)}, b^{(e)})$ and show that the shares output follow the correct distribution. We prove it by induction. Let us assume that $x_{\mathsf{rvl}}^{(e+1)}$ at a given level of the sharing's recursion follows the correct distribution.

- **Base Case ($T = 1$):** The refreshed share for all users is simply $x^{(e+1)}$. The simulator is given this value as input, so it can return it as the share. The distribution of shares trivially remains the same.
- **Recursive Case ($N \geqslant T > 1$):** In this case, we will consider a sub-sharing of $x_{\mathsf{rvl}}^{(e+1)}$ into $(y_L^{(e+1)}, y_R^{(e+1)})$ and wish to run $\mathsf{ShareReveal}$ recursively. Let the sharing of $x_{\mathsf{scr}}^{(e)}$ be split into $y_{\mathsf{rvl}}^{(e)}$ and $y_{\mathsf{scr}}^{(e)}$. The revealed share $y_{\mathsf{rvl}}^{(e)}$ can be reconstructed from $\mathsf{Shr}'^{(e)}$, while $y_{\mathsf{scr}}^{(e)}$ can't be. Also, let $\hat{x}_{\mathsf{rvl}}$ in this refreshing be split into $\hat{y}_L$ and $\hat{y}_R$. We note that the simulator cannot access $\hat{x}_{\mathsf{rvl}}$. For convenience, we denote shares of $x_{\mathsf{rvl}}^{(e+1)}$ and $\hat{x}_{\mathsf{rvl}}$ corresponding to the path of the previous revealed sub-secret $y_{\mathsf{rvl}}^{(e)}$ (resp. secure sub-secret $y_{\mathsf{scr}}^{(e)}$) by $y_{\mathsf{rvl}}^{(e+1)}$ and $\hat{y}_{\mathsf{rvl}}$ (resp. $y_{\mathsf{scr}}^{(e+1)}$ and $\hat{y}_{\mathsf{scr}}$), respectively. Note that despite the subscript scr, $y_{\mathsf{scr}}^{(e+1)}$ is revealed in the next epoch. We distinguish it only to indicate that it corresponds to the path of the previous secure sub-secret $y_{\mathsf{scr}}^{(e)}$.

  Below, we show how to simulate refreshed shares in recursive case:

  1. **Generate a share $\hat{y}_{\mathsf{rvl}}$:** This is a crucial step of this algorithm. We can generate a share $\hat{y}_{\mathsf{rvl}}$ of $\hat{x}_{\mathsf{rvl}}$ such that $(\hat{y}_{\mathsf{rvl}}, \hat{x}_{\mathsf{rvl}} - \hat{y}_{\mathsf{rvl}})$ follows the correct distribution of a sharing of $\hat{x}_{\mathsf{rvl}}$. Specifically, if $\hat{y}_{\mathsf{rvl}}$ is $\hat{y}_L$, it honestly generates a share $\hat{y}_{\mathsf{rvl}}$ from χ. If $\hat{y}_{\mathsf{rvl}} = \hat{y}_R$, we compute a share of the form

$\hat{x}_{\mathsf{rvl}} - \hat{y}_L$ leveraging a hint $h_i = x_{\mathsf{scr}}^{(e)} + \mathbf{s}_i$ by

$$x_{\mathsf{rvl}}^{(e+1)} - h_i = (x_{\mathsf{scr}}^{(e)} + \hat{x}_{\mathsf{rvl}}) - (x_{\mathsf{scr}}^{(e)} + \mathbf{s}_i) = \hat{x}_{\mathsf{rvl}} - \mathbf{s}_i$$

and regarding $\mathbf{s}_i$ as $\hat{y}_L$.

2. **Compute shares of $y_{\mathsf{rvl}}^{(e+1)}$:** The simulator knows both previous shares $\mathsf{Shr}'^{(e)}[u][\mathsf{idx}]$ with $b^{(e)}[u][\mathsf{idx}] = 0$ for paths under this half and $\hat{y}_{\mathsf{rvl}}$. Thus, it honestly generates shares $(\hat{y}_{\mathsf{rvl}}[u][\mathsf{idx}])_{u,\mathsf{idx}}$ of $\hat{y}_{\mathsf{rvl}}$ by using VandShare and computes shares of $y_{\mathsf{rvl}}^{(e+1)}$ by $\mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + \hat{y}_{\mathsf{rvl}}[u][\mathsf{idx}]$ for such paths. It is easy to check that their distribution is correct.
3. **Compute shares of $y_{\mathsf{scr}}^{(e+1)}$:** In this case, we cannot access $\hat{y}_{\mathsf{scr}}$ and $y_{\mathsf{scr}}^{(e)}$. We observe that we can compute the refreshed sub-secret $y_{\mathsf{scr}}^{(e+1)}$ for this half as

$$y_{\mathsf{scr}}^{(e+1)} = x_{\mathsf{rvl}}^{(e+1)} - y_{\mathsf{rvl}}^{(e+1)} = x_{\mathsf{rvl}}^{(e+1)} - (y_{\mathsf{rvl}}^{(e)} + \hat{y}_{\mathsf{rvl}})$$

where $y_{\mathsf{rvl}}^{(e)}$ can be reconstructed from $\mathsf{Shr}'^{(e)}$. Additionally, we can derive hints h'_i on $y_{\mathsf{scr}}^{(e)}$ from a hint h_i on $x_{\mathsf{scr}}^{(e)}$, by computing

$$h_i - y_{\mathsf{rvl}}^{(e)} = (x_{\mathsf{scr}}^{(e)} + \mathbf{s}_i) - y_{\mathsf{rvl}}^{(e)} = y_{\mathsf{scr}}^{(e)} + \mathbf{s}_i.$$

Then we call $\mathsf{ShareReveal}(\mathcal{P}_{\mathsf{scr}}, T_{\mathsf{scr}}, y_{\mathsf{scr}}^{(e+1)}, (h'_i)_i, \mathsf{Shr}'^{(e)}, b^{(e)})$, where $\mathcal{P}_{\mathsf{scr}}$ and T_{scr} are a set of users and sub-threshold for sub-sharing under this secure half, and obtain shares of $y_{\mathsf{scr}}^{(e+1)}$. These shares are generated when the recursion steps reach the base case or the above case. Also, since $\hat{y}_{\mathsf{rvl}}$ is identically distributed to that in the real refreshing algorithm, the distribution of $y_{\mathsf{scr}}^{(e+1)}$ is also identical to that of the real refreshing. Thus the distribution of shares of $y_{\mathsf{scr}}^{(e+1)}$ produced by this recursion remains identical.

Recall that shares of a parent secret x are identical to shares of sub-sharing of x_L and x_R. Therefore, the simulator returns shares of $y_{\mathsf{rvl}}^{(e+1)}$ and $y_{\mathsf{scr}}^{(e+1)}$ as shares of $x_{\mathsf{rvl}}^{(e+1)}$.

By construction, the number of hints suffices to run the algorithm. We first run our recursive algorithm under $x_{\mathsf{rvl}}^{(e+1)}$ given as input, which follows the correct distribution. Therefore, as a result of recursion, we obtain all shares of $x_{\mathsf{rvl}}^{(e+1)}$ of which distribution is identical to the real refreshing. This completes the proof. □

We now turn to defining $\mathsf{SimRefresh}$. We wish to simulate the refreshed shares $\mathsf{Shr}'^{(e+1)}$ and simulation bits $b^{(e+1)}$ given the previous shares $\mathsf{Shr}'^{(e)}$ and bits $b^{(e)}$. We proceed recursively, proving a slightly stronger result for a *public* refresh vector $\mathbf{s}_r$, which is initially $\mathbf{0}$ and is a vector during recursion. To handle this public refresh vector, our recursive algorithm will take it as input explicitly. We will check that the simulator always knows $\mathbf{s}_r$ when it runs $\mathsf{SimRefresh}$ during the recursion. Below we define our recursive algorithm $\mathsf{SimRefresh}(\mathcal{P}, T, \mathsf{CS}^{(e+1)}, j^*, (h_i)_{i\in[B_{\mathsf{rf}}(N,T)]}, \mathsf{Shr}'^{(e)}, b^{(e)}, \mathbf{s}_r)$.

- **Base Case ($T = 1$):** All users obtain $\mathbf{s} + \mathbf{s}_r$ as their share. The simulator sets $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = 0$ and $b^{(e+1)}[u][\mathsf{idx}] = 1$ for all $u \in \mathcal{P}$.
- **Recursive Case ($N \geqslant T > 1$):** The first step in this case is to split $\mathcal{P}$ into the revealed and secure halves in the same manner as $\mathsf{SimShares}$ using new corrupt set $\mathsf{CS}^{(e+1)}$. Let $\mathcal{P}_{\mathsf{rvl}}^{(e)}$ and $\mathcal{P}_{\mathsf{rvl}}^{(e+1)}$ be the revealed half in epoch e and $e+1$, respectively. Also, let $x_{\mathsf{rvl}}^{(e)}$ and $x_{\mathsf{rvl}}^{(e+1)}$ be the share for the revealed half in epoch e and $e+1$, respectively. For the secure half, we denote by $\mathcal{P}_{\mathsf{scr}}^{(e)}$, $\mathcal{P}_{\mathsf{scr}}^{(e+1)}$, $x_{\mathsf{scr}}^{(e)}$ and $x_{\mathsf{scr}}^{(e+1)}$ similarly. Notice that $\mathbf{s} = x_{\mathsf{rvl}}^{(e)} + x_{\mathsf{scr}}^{(e)}$ and $\mathbf{s} + \mathbf{s}_r = x_{\mathsf{rvl}}^{(e+1)} + x_{\mathsf{scr}}^{(e+1)}$. Moreover, let the sharing of $\mathbf{s}_r$ in refreshing be split into $\hat{x}_L$ and $\hat{x}_R$. For convenience, we denote by $\hat{x}_{\mathsf{rvl}}$ and $\hat{x}_{\mathsf{scr}}$ shares corresponding to the revealed half and secure half *in epoch* $e+1$, respectively.

  Next, we generate $\mathsf{Shr}'^{(e+1)}$ and $b'^{(e+1)}$ recursively. Now we distinguish two cases: "non-flipped" case and "flipped" case.

– **Non-flipped Case** $(\mathcal{P}^{(e)}_{\mathsf{rvl}}, \mathcal{P}^{(e)}_{\mathsf{scr}}) = (\mathcal{P}^{(e+1)}_{\mathsf{rvl}}, \mathcal{P}^{(e+1)}_{\mathsf{scr}})$**:** This is the simpler case. In this case, the simulator can honestly generate shares of $\mathbf{s}_r$, $\hat{x}_{\mathsf{rvl}}$ and $\hat{x}_{\mathsf{scr}} = -\hat{x}_{\mathsf{rvl}}$ since it knows $\mathbf{s}_r$.

  * For the revealed half $\mathcal{P}^{(e+1)}_{\mathsf{rvl}}$, it honestly generates shares $z[u][\mathsf{idx}]$ of $\hat{x}_{\mathsf{rvl}}$ by VandShare and computes $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + z[u][\mathsf{idx}]$ for all u and idx under this path. All bits are set to 0.
  * For the secure half $\mathcal{P}^{(e+1)}_{\mathsf{scr}}$, it recursively runs SimRefresh under the set $\mathcal{P}_{\mathsf{scr}}$, the sub-secret $x^{(e+1)}_{\mathsf{scr}} = x^{(e)}_{\mathsf{scr}} + \hat{x}_{\mathsf{scr}}$, and the refresh vector $\hat{x}_{\mathsf{scr}}$, which is not known to the simulator, using hints h'_i on $x^{(e)}_{\mathsf{scr}}$ derived from hints h_i on $\mathbf{s}$ by $h'_i = h_i - x^{(e)}_{\mathsf{rvl}} = (\mathbf{s} + \mathbf{s}_i) - x^{(e)}_{\mathsf{scr}} = x^{(e)}_{\mathsf{scr}} + \mathbf{s}_i$. As a result of recursion, it obtains $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ and $b^{(e+1)}[u][\mathsf{idx}]$ for all u and idx of this sub-sharing. For all u and idx such that $b^{(e+1)}[u][\mathsf{idx}] = 1$, we update $\mathsf{Shr}'^{(e+1)}$ by $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] + x^{(e+1)}_{\mathsf{scr}}$, where $x^{(e+1)}_{\mathsf{scr}}$ can be computed by $x^{(e)}_{\mathsf{rvl}} + \hat{x}_{\mathsf{rvl}}$ and $x^{(e)}_{\mathsf{rvl}}$ can be reconstructed from $\mathsf{Shr}'^{(e)}$.

– **Flipped Case** $(\mathcal{P}^{(e)}_{\mathsf{rvl}}, \mathcal{P}^{(e)}_{\mathsf{scr}}) = (\mathcal{P}^{(e+1)}_{\mathsf{scr}}, \mathcal{P}^{(e+1)}_{\mathsf{rvl}})$**:** First, the simulator samples a share for the sub-secret $\mathbf{s} + \mathbf{s}_r$. Concretely, we sample y such that $(y, \mathbf{s} + \mathbf{s}_r - y)$ follows the correct distribution of the refreshed shares $(x^{(e+1)}_{\mathsf{rvl}}, x^{(e+1)}_{\mathsf{scr}}) = (x^{(e)}_{\mathsf{scr}} + \hat{x}_{\mathsf{rvl}}, x^{(e)}_{\mathsf{rvl}} + \hat{x}_{\mathsf{scr}})$, by using one hint h_i on $\mathbf{s}$ such that $h_i = \mathbf{s} + \mathbf{s}_i$ and relying on $\mathbf{s}_r$ being public. This captures the fact that the simulator will know the refreshed share in $\mathcal{P}^{(e+1)}_{\mathsf{rvl}}$, but not in $\mathcal{P}^{(e+1)}_{\mathsf{scr}}$. Specifically, it computes y by

$$y = h_i - x^{(e)}_{\mathsf{rvl}} = \mathbf{s} + \mathbf{s}_i - x^{(e)}_{\mathsf{rvl}} = x^{(e)}_{\mathsf{scr}} + \mathbf{s}_i.$$

Next, the simulator generates $\mathsf{Shr}'^{(e+1)}$ and $b^{(e+1)}$ for the revealed half $\mathcal{P}^{(e+1)}_{\mathsf{rvl}}$ relying on ShareReveal in Theorem 4.2. It first sets $x^{(e+1)}_{\mathsf{rvl}} = y$ by regarding $\mathbf{s}_i$ as $\hat{x}_{\mathsf{rvl}}$. Then it runs ShareReveal on $\mathcal{P}^{(e+1)}_{\mathsf{scr}}$ and this sub-secret $x^{(e+1)}_{\mathsf{rvl}}$ leveraging hints h'_i on $x^{(e)}_{\mathsf{scr}}$ derived from hints h_i and $x^{(e)}_{\mathsf{rvl}}$. It sets $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ to outputs by ShareReveal and $b^{(e+1)}[u][\mathsf{idx}] = 0$ for all u and idx of this sub-sharing.

Finally, it defines $\mathsf{Shr}'^{(e+1)}$ and $b^{(e+1)}$ for the secure half $\mathcal{P}^{(e+1)}_{\mathsf{scr}}$. This is achieved by calling SimShares on a secret $\hat{x}_{\mathsf{scr}}$. Specifically, we run $\mathsf{SimShares}(\mathcal{P}^{(e+1)}_{\mathsf{scr}}, T_{\mathsf{scr}}, \mathsf{CS}_{\mathsf{scr}}, j^*, (h''_i)_{i \in [B_{\mathsf{shr}}(N_{\mathsf{scr}}, T_{\mathsf{scr}})]})$ where $N_{\mathsf{scr}} = |\mathcal{P}^{(e+1)}_{\mathsf{scr}}|$ and T_{scr} is the sub-threshold for $\mathcal{P}^{(e+1)}_{\mathsf{scr}}$, $\mathsf{CS}_{\mathsf{scr}} = \mathcal{P}^{(e+1)}_{\mathsf{scr}} \cap \mathsf{CS}^{(e+1)}$, and h''_i is a hint on $\hat{x}_{\mathsf{scr}}$. We can derive these hints $(h''_i)_i$ using hints h_i and known values as follows:

$$\begin{aligned} h''_i &= h_i + \mathbf{s}_r - y - x^{(e)}_{\mathsf{rvl}} \\ &= (\mathbf{s} + \mathbf{s}_i) + \mathbf{s}_r - (x^{(e)}_{\mathsf{scr}} + \hat{x}_{\mathsf{rvl}}) - x^{(e)}_{\mathsf{rvl}} \\ &= \mathbf{s}_r - \hat{x}_{\mathsf{rvl}} + \mathbf{s}_i = \hat{x}_{\mathsf{scr}} + \mathbf{s}_i. \end{aligned}$$

This algorithm returns simulated shares w of $\hat{x}_{\mathsf{scr}}$ and bits b_w satisfying the requirements of this lemma. We denote a simulated share and a bit corresponding to the path of $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ by $w[u][\mathsf{idx}]$ and $b_w[u][\mathsf{idx}]$.

We set $b^{(e+1)}[u][\mathsf{idx}] = b_w[u][\mathsf{idx}]$ and compute $\mathsf{Shr}'^{(e+1)}$ depending on the bit of $b^{(e+1)}[u][\mathsf{idx}]$. For u and idx such that $b^{(e+1)}[u][\mathsf{idx}] = 0$, we compute it by $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + w[u][\mathsf{idx}]$. Otherwise, we compute $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = w[u][\mathsf{idx}] - \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + x^{(e)}_{\mathsf{rvl}} + y$.

Before showing that $\mathsf{Shr}'^{(e+1)}$ and $b'^{(e+1)}$ satisfies the required properties, we check two facts: (1) the simulator always knows $\mathbf{s}_r$ when running SimRefresh recursively, and (2) we can reconstruct any sub-secret for revealed halves by using Shr'. For (1), SimRefresh is run only when a step falls into the non-flipped case. This algorithm runs under a sub-secret $x^{(e)}_{\mathsf{scr}} + \hat{x}_{\mathsf{scr}}$ by regarding $x^{(e)}_{\mathsf{scr}}$ and $\hat{x}_{\mathsf{scr}}$ as a secret and a refresh vector at next level of recursion, respectively. Since the simulator honestly generates shares $(\hat{x}_{\mathsf{rvl}}, \hat{x}_{\mathsf{scr}})$ of $\mathbf{s}_r$, it knows a refresh vector $\hat{x}_{\mathsf{scr}}$ at the next level.

To show (2), it is sufficient to check that all bits under revealed halves are 0. We use induction. By construction of SimShares, it is easy to check this in the case $e = 1$. Now assume $b'^{(e)}$ satisfies this. During

recursion, values in $b^{(e+1)}$ might be set to 1 both in the non-flipped and flipped cases. For non-flipped cases, $b^{(e+1)}[u][\mathsf{idx}]$ remains assigned 0 for revealed halves. Furthermore, when running $\mathsf{SimRefresh}$ recursively under secure halves, $b^{(e+1)}[u][\mathsf{idx}]$ is set to 1 only when we reach the base case or a flipped case. Since the simulator recurses only on secure halves, bits involved in revealed halves are never assigned 1 via the base case. In the flipped case, $b^{(e+1)}[u][\mathsf{idx}]$ is assigned 1 when $b_w[u][\mathsf{idx}] = 1$. Since b_w is produced by $\mathsf{SimShares}$, $b_w[u][\mathsf{idx}]$ involved in revealed halves in this sub-sharing are 0. Thus this fact also holds.

(*Checking properties.*) To complete the proof, we show that $\mathsf{Shr}'^{(e+1)}$ and $b'^{(e+1)}$ satisfy the required properties. By construction, the number of hints suffices to run the algorithm. We follow similar arguments as for $\mathsf{SimShares}$: corrupt sets are empty when the induction reaches the base case. Also, $\mathsf{SimShares}$ does not assign 1 to shares for corrupted users as shown above. Thus they satisfy the property (a). $\mathsf{SimRefresh}$ determines secure halves in the same manner as $\mathsf{SimShares}$. Moreover, it runs $\mathsf{SimShares}$ and $\mathsf{SimRefresh}$ recursively only for secure halves. Thus property (b) and (c) also holds.

Finally, we check the property (d). For the base case, it is easy to check it by $\mathbf{s} - \mathbf{s}_r - \mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = \mathbf{s} - \mathbf{s}_r - 0$. We now focus on the non-flipped case. For a revealed half, we compute $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + z[u][\mathsf{idx}]$ and set $b^{(e+1)}[u][\mathsf{idx}] = 0$. Since $\mathsf{Shr}'^{(e)}[u][\mathsf{idx}]$ is an actual share of $\mathbf{s}$, and it honestly generates a share $z[u][\mathsf{idx}]$ of $\mathbf{s}_r$, this property holds. For a secure half, it generates $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ by using $\mathsf{SimRefresh}$ under a sub-secret $x_{\mathsf{scr}}^{(e+1)}$ using honestly generated $\hat{x}_{\mathsf{scr}}$. Then $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ for any $b^{(e+1)}[u][\mathsf{idx}] = 0$ is identically distributed to a real share of $x_{\mathsf{scr}}^{(e+1)}$. Recall that shares of a sub-sharing are shares of its parent sharing. Thus it also follows the same distribution of a share of $\mathbf{s}+\mathbf{s}_r$. For u and idx with $b^{(e+1)}[u][\mathsf{idx}] = 1$, $x_{\mathsf{scr}}^{(e+1)} - \mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ follows the same distribution of a real share of $x_{\mathsf{scr}}^{(e+1)}$, namely, a share of $\mathbf{s} + \mathbf{s}_r$. Then, because we have

$$\mathbf{s} + \mathbf{s}_r - (\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] + x_{\mathsf{rvl}}^{(e+1)}) = x_{\mathsf{scr}}^{(e+1)} - \mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}], \tag{5}$$

updated simulated shares $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ satisfy the property (d).

We turn to the flipped case. For a revealed half, due to Theorem 4.2, $\mathsf{ShareReveal}$ outputs correctly distributed shares of $x_{\mathsf{rvl}}^{(e+1)}$. Since we set $b^{(e+1)}$ for all path under this sub-sharing to 0, the property (d) holds. For a secure half, we assign $b^{(e+1)}[u][\mathsf{idx}]$ by using $b_w[u][\mathsf{idx}]$ produced by $\mathsf{SimShares}$ on secret $\hat{x}_{\mathsf{scr}}$. For u and idx s.t. $b^{(e+1)}[u][\mathsf{idx}] = 0$, we set $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] = \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + w[u][\mathsf{idx}]$. Since this secure half is a revealed half in epoch e, $\mathsf{Shr}'^{(e)}[u][\mathsf{idx}]$ has a share of $x_{\mathsf{rvl}}^{(e)}$. Since the distribution of $w[u][\mathsf{idx}]$ is identical to that of a real share of $\hat{x}_{\mathsf{scr}}$, $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ satisfies (d) in this case. For u and idx s.t. $b_w^{(e+1)}[u][\mathsf{idx}] = 1$, $\hat{x}_{\mathsf{scr}} - w[u][\mathsf{idx}]$ follows the distribution of the real sharing of $\hat{x}_{\mathsf{scr}}$. For $\mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ in this case, we have

$$\begin{aligned}
&\mathbf{s} + \mathbf{s}_r - \mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}] \\
&= \mathbf{s} + \mathbf{s}_r - (w[u][\mathsf{idx}] - \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + x_{\mathsf{rvl}}^{(e)} + y) \\
&= \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + (\mathbf{s} + \mathbf{s}_r - x_{\mathsf{rvl}}^{(e)} - x_{\mathsf{scr}}^{(e)} - \hat{x}_{\mathsf{rvl}}) - w[u][\mathsf{idx}] \\
&= \mathsf{Shr}'^{(e)}[u][\mathsf{idx}] + (\hat{x}_{\mathsf{scr}} - w[u][\mathsf{idx}])
\end{aligned}$$

where the second equation follows from the fact that y is used as $x_{\mathsf{rvl}}^{(e+1)} = x_{\mathsf{scr}}^{(e)} + \hat{x}_{\mathsf{rvl}}$ for the revealed half. As explained above, $\mathsf{Shr}'^{(e)}[u][\mathsf{idx}]$ is an actual share of $x_{\mathsf{rvl}}^{(e)}$ in epoch e, $\mathbf{s}+\mathbf{s}_r - \mathsf{Shr}'^{(e+1)}[u][\mathsf{idx}]$ is identically distributed to a real share of $x_{\mathsf{rvl}}^{(e)} + \hat{x}_{\mathsf{scr}}$, namely, a real share of $\mathbf{s}+\mathbf{s}_r$. Thus this property holds in this case. This completes the proof. □

5 Hermine: A FROST-like Two-round Threshold Signature

In this section, we propose $\mathsf{Hermine}$, a two-round lattice-based threshold signature scheme that is unforgeable and has NI-IA with proactive security. See Section 1.4 for the overview.

Parameter	Explanation
$\mathcal{R}_q$	Polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n+1)$
(k, ℓ)	Dimension of public matrix $\mathbf{A} \in \mathcal{R}_q^{k \times \ell}$
Q_{Sign}, Q_{rf}	Maximum number of signing sessions and refreshes allowed, respectively
$(\mathcal{D}_{\mathbf{s}}, \sigma_{\mathbf{s}})$	Gaussian distribution with width $\sigma_{\mathbf{s}}$ used for the secret key $\mathbf{s}$ and for the Vandermonde secret sharing, e.g., $\chi = \mathcal{D}_{\mathbf{s}}$
$\nu_{\mathbf{t}}$	Amount of bit dropping performed on verification key
$(\mathcal{D}_{\mathbf{r}}, \sigma_{\mathbf{r}})$	Gaussian distribution with width $\sigma_{\mathbf{r}}$ used for the commitment $\mathbf{w}$
$\nu_{\mathbf{w}}, q_{\nu_{\mathbf{w}}}, q_\perp$	Amount of bit dropping performed on aggregated commitment. Define $q_{\nu_{\mathbf{w}}} = \lfloor q/2^{\nu_{\mathbf{w}}} \rceil$ and $q_\perp = q - 2^{\nu_{\mathbf{w}}} \cdot q_{\nu_{\mathbf{w}}}$.
$\nu'_{\mathbf{w}}$	Amount of bit dropping performed on partial commitment
$\mathbb{T}$	Set of signed monomials
rep	An integer s.t. $\lvert\mathbb{T}\rvert^{\mathsf{rep}-1} \geqslant 2^\kappa$
$(\mathcal{C} \subset \mathcal{R}_q, \omega)$	Challenge set $\{c \in \mathcal{R}_q \mid \lVert c\rVert_\infty = 1 \wedge \lVert c\rVert_1 = \omega\}$ s.t. $\lvert\mathcal{C}\rvert \geq 2^\kappa$
B_2^{ind}	Two-norm bound for partial verification
B_2	Two-norm bound on the signature
$B_{\mathsf{rf}}^{\mathsf{ind}}$	Two-norm bounds for refresh verification

Table 2: Overview of parameters used in Hermine.

5.1 Construction

We provide the algorithms of Hermine in Figs. 6 and 7 and parameters in Table 2. We omit TS.Setup as this just outputs $\mathsf{tspar} = (N, T)$. Our scheme uses the two hash functions modeled as a random oracle: $H_c : \{0,1\}^* \to \mathcal{C}$ and $H_\beta : \{0,1\}^* \to \{1\} \times \mathbb{T}^{\mathsf{rep}-1}$. The former is used to generate the challenge polynomial, and the latter is used to generate the weights when aggregating the individual commitments. We note that the factor 2 in the verification key is inherited from [49, 88], which we believe to be an artifact of the proof technique.

6 Security of Hermine

In this subsection, we show that our proposed scheme is correct, provides IA, and is unforgeable. For a set of asymptotic parameters for which our scheme is secure and correct, see Section 6.1.

6.1 Asymptotic Parameters

Before showing the security of our scheme, we first provide a candidate of asymptotic parameters such that our scheme is secure and correct.

- $n, k, \ell = \mathsf{poly}(\kappa)$ such that $(k+\ell)n \geqslant 2\kappa$.
- $\nu_{\mathbf{t}}$, $\nu_{\mathbf{w}}$, $\nu'_{\mathbf{w}} < \log q$.
- $\sigma_{\mathbf{s}}, \sigma_{\mathbf{r}} \geqslant \max(\sqrt{2} \cdot \eta'_\varepsilon(\mathbb{Z}^{n(\ell+k)}), \sqrt{\log(6(k+\ell)n)/\pi})$ for $\varepsilon = \mathsf{negl}(\kappa)$.
- $\alpha \geqslant 1 + \frac{\kappa \log 2}{3n}$ or $\alpha \geqslant 1 + \sqrt{\frac{4\kappa \log 2}{n}}$.
- $B_{\mathsf{rf}}^{\mathsf{ind}} = \alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell) \cdot \lceil \log_2 N \rceil}$ for bounding the norm of refresh tokens (Theorem 6.1).

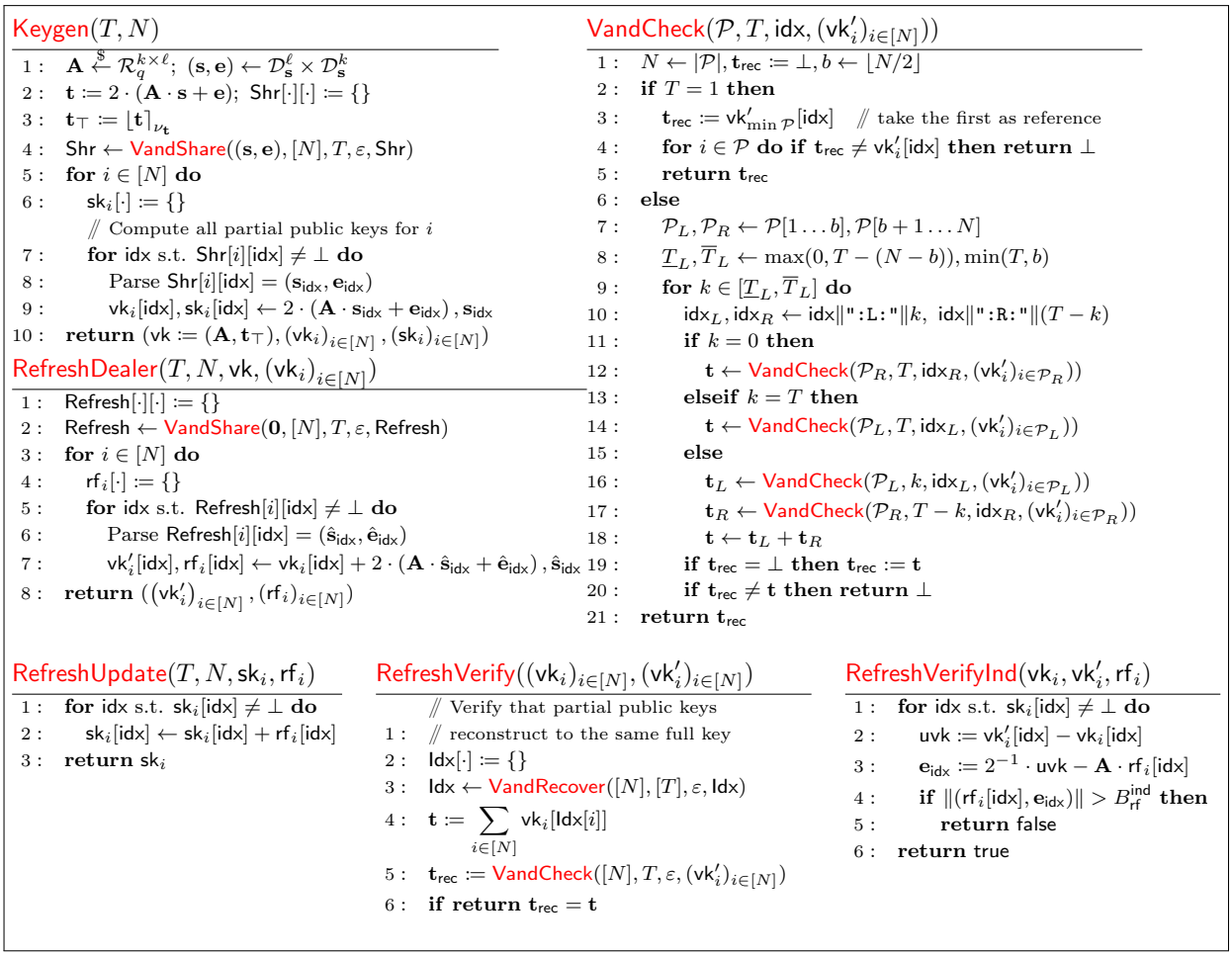


Figure 6: Key generation and Refresh algorithms. For conciseness, we omit index-related aborts in Refresh: on any unexpected message, it returns false.

- $B_2^{\mathsf{ind}} \geqslant 2\omega\cdot\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\cdot\left(1+(Q_{\mathsf{rf}}+1)\sqrt{\lceil\log_2 N\rceil}\right) + \alpha\cdot\sigma_{\mathbf{r}}\cdot\sqrt{n(\ell+k)\cdot\mathsf{rep}} + \mathsf{rep}\cdot 2^{\nu'_{\mathbf{w}}-1}\sqrt{n\cdot k}$. [10]
- $B_2 \geqslant T\cdot B_2^{\mathsf{ind}} + \left((3\cdot 2^{\nu_{\mathbf{w}}-1}+\max(0,q_\perp)) + \omega\cdot 2^{\nu_{\mathbf{t}}-1}\right)\cdot\sqrt{nk}$
- N, T such that $B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}}\cdot B_{\mathsf{rf}}(N,T) + \mathsf{rep}\cdot Q_{\mathsf{Sign}} + 1 = \mathsf{poly}(\kappa)$.
- $\beta_{\mathsf{sis}} = \beta_{\mathbf{s}} + \beta_{\mathbf{b}}\sqrt{(k+\ell)n}$ where $\beta_{\mathbf{s}} = 2B_2 + (2\cdot(2^{\nu_{\mathbf{w}}-1}+\max(0,q_\perp)) + \omega\cdot 2^{\nu_{\mathbf{t}}})\cdot\sqrt{kn}$ and $\beta_{\mathbf{b}} = 4\cdot\omega\cdot\sigma_{\mathbf{s}}$.
- $\beta_{\mathsf{lwe}} = 1/(\beta_{\mathbf{u}}\sqrt{(k+\ell)n\cdot\log 6/\pi})$ where $\beta_{\mathbf{u}} = \sqrt{(B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}}\cdot B_{\mathsf{rf}}(N,T)+1)/\sigma_{\mathbf{s}}^2 + (8\omega^2+8\omega+4)\cdot n\cdot Q_{\mathsf{Sign}}/\sigma_{\mathbf{r}}^2}$
- q such that $\mathsf{MSIS}_{q,k,\ell,\beta_{\mathsf{sis}}}$ and $\mathsf{MLWE}_{q,k,\ell-1,\beta_{\mathsf{lwe}}}$ are assumed to be exponentially hard in κ. [11]

6.2 Correctness and Identifiable Abort.

Recall that the standard correctness of threshold signatures is implied by the identifiable abort property (see Theorem 3.4). Before showing the identifiable abort property, we establish the correctness of refresh

[10] We use a heuristic bound in the concrete instantiation. See Theorem 7.1.
[11] We set $\varepsilon = 1/2$ and $\alpha = 2$ for Theorem 2.11 as in [88].

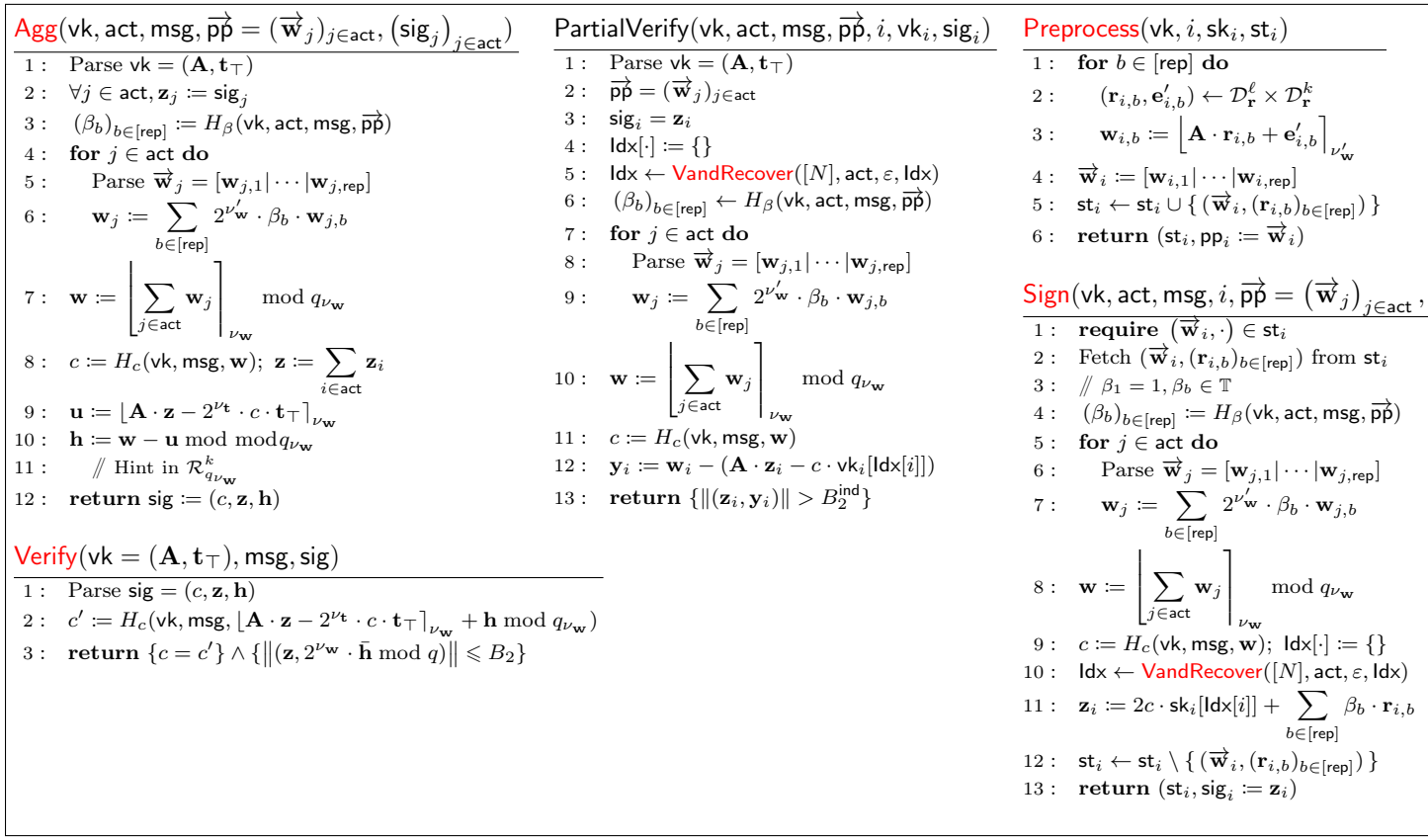


Figure 7: Signing, Verification, and Partial Verification algorithms. For conciseness, we omit aborts due to parsing errors: on any unexpected message, parties abort and report the sender as malicious.

verification, which relies on the fact that our secret sharing scheme distributes short secrets into short shares.

Lemma 6.1 (Bounded Norm of Shares). *Let α be a rejection sampling parameter as defined in Section 6.1. If $\sigma_{\mathbf{s}}$ satisfies the conditions in Section 6.1, then for any $i \in [N]$ and idx, the norm of $(\mathbf{s}_{\mathsf{idx}}, \mathbf{e}_{\mathsf{idx}})$ stored in $\mathsf{Shr}[i][\mathsf{idx}]$ outputted by VandShare is upper bounded by $\mu_0 = \alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)\cdot\lceil\log_2 N\rceil}$ (if a secret to be shared is $\mathbf{0}$), except with negligible probability.*

Proof. To prove this lemma, we can see that our sharing algorithm works by adding up to $\lceil\log_2 N\rceil$ Gaussian noises sampled from χ to the input secret value. Hence, in the end, by the convolution theorem Theorem 2.3, all shares are bigger Gaussian samples. Applying Theorem 2.4, we deduce the result.

Let us count formally the number of samples added to a share. We show by induction on the number of parties N that the maximal number of samples that compose a share (except for the input secret) is bounded by $\lceil\log_2 N\rceil$.

1. In the case $T = 1$, the shares are equal to the secret, and no additional sample is added.
2. In the general case, let k be as in line 9 of VandShare. For $k = 0$ or $k = T$ no sample is added and we conclude by induction. Otherwise, we add at most one new Gaussian sample per share before sharing recursively. Each of the two recursive calls is made with at most $\lceil N/2\rceil$ parties. Hence, by the induction hypothesis, we can conclude that the number of samples added to a share is at most $1 + \lceil\log_2(N/2)\rceil = \lceil\log_2 N\rceil$.

□

Theorem 6.2 (Correctness of Refresh Verification). *The refresh verification algorithms* RefreshVerify *and* RefreshVerifyInd *output* true *with overwhelming probability when the inputs are honestly generated.*

Proof. We prove Theorem 6.2 as follows:

- For RefreshVerify, it is clear that the updated public keys vk'_i generated by an honest dealer satisfy $\sum_{i\in\mathsf{act}} \mathsf{vk}'_i[\mathsf{Idx}[i]] = \mathbf{t}$ as we only add shares of zero to the original public keys, and by the correctness of our sharing.
- For RefreshVerifyInd, similarly to the case of RefreshVerify, it is clear that the updated public keys vk'_i generated by an honest dealer satisfy $\mathsf{vk}'_i[\mathsf{idx}] = \mathsf{vk}_i[\mathsf{idx}] + 2\cdot(\mathbf{A}\cdot\hat{\mathbf{s}}_{\mathsf{idx}} + \hat{\mathbf{e}}_{\mathsf{idx}})$. The norm bound follows from Theorem 6.1 as we sample the refresh tokens as a share of zero.

□

We show that Hermine provides IA, implying standard correctness.

Theorem 6.3. Hermine *provides IA (Fig. 4), except with negligible probability.*

Proof. Let us denote i^* the index of the honest party to frame in the game of Fig. 4. At the end of the game, the adversary outputs the transcript of a signing session, consistent with the view of party i^*, and tries to have i^* fail its partial verification.

Partial verification. Let us first show that the partial verification of i^* passes except with negligible probability.

By definition of the game, the view of party i^* is the same as the one that is used to perform the partial verification. In particular, the challenge c and the coefficients β_b used in PartialVerify are the same as those used in $\mathsf{ShareSign}_2$. It suffices to show that the honestly generated $(\mathbf{w}_{i^*}, \mathbf{z}_{i^*})$ passes the check of $\{\|(\mathbf{z}_{i^*}, \mathbf{y}_{i^*})\| \leqslant B_2^{\mathsf{ind}}\}$ except with negligible probability. By the triangle inequality,

$$\begin{aligned}\|(\mathbf{z}_{i^*}, \mathbf{y}_{i^*})\| &\leqslant \left\| \left(\mathbf{z}_{i^*}, 2c\cdot\mathbf{e}_{i^*} + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{e}'_{i^*,b} \right) \right\| + \left\| \left(0, \mathbf{y}_{i^*} - 2c\cdot\mathbf{e}_{i^*} - \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{e}'_{i^*,b} \right) \right\| \\ &= \left\| 2c\cdot(\mathbf{s}_{i^*}, \mathbf{e}_{i^*}) + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot (\mathbf{r}_{i^*,b}, \mathbf{e}_{i^*,b}) \right\| \\ &\quad + \left\| \mathbf{w}_{i^*} - (\mathbf{A}\cdot\mathbf{z}_{i^*} - c\cdot\mathbf{t}_{i^*}) - 2c\cdot\mathbf{e}_{i^*} - \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{e}'_{i^*,b} \right\|\end{aligned}$$

where $\mathsf{idx} = \mathsf{Idx}[i^*]$, $\mathbf{s}_{i^*} = \mathsf{sk}_{i^*}[\mathsf{idx}]$, $\mathbf{t}_{i^*} = \mathsf{vk}_{i^*}[\mathsf{idx}]$, and $\mathbf{e}_{i^*}$ satisfies $\mathbf{t}_{i^*} = \mathbf{A}\cdot\mathbf{s}_{i^*} + \mathbf{e}_{i^*}$.

We first focus on the first term. From the triangle inequality, we obtain

$$\left\| 2c\cdot(\mathbf{s}_{i^*}, \mathbf{e}_{i^*}) + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot (\mathbf{r}_{i^*,b}, \mathbf{e}_{i^*,b}) \right\| \leqslant \|2c\cdot(\mathbf{s}_{i^*}, \mathbf{e}_{i^*})\| + \left\| \sum_{b\in[\mathsf{rep}]} \beta_b \cdot (\mathbf{r}_{i^*,b}, \mathbf{e}_{i^*,b}) \right\|.$$

We can bound $\|(\mathbf{s}_{i^*}, \mathbf{e}_{i^*})\|$ by $\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\cdot\left(1+(Q_{\mathsf{rf}}+1)\sqrt{\lceil\log_2 N\rceil}\right)$ using Theorems 2.3, 2.4 and 6.1. Indeed, the key generation produces secret shares with norm bounded by $\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)\cdot(1+\lceil\log_2 N\rceil)}$

to which we add at most Q_{rf} times shares of zero with norm bounded by $\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)\cdot\lceil\log_2 N\rceil}$ since they pass the check RefreshVerifyInd. Since $\|c\|_1 = \omega$, we have $\|2c\cdot(\mathbf{s}_{i^*},\mathbf{e}_{i^*})\| \leqslant 2\omega\cdot\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\cdot\left(1+(Q_{\mathsf{rf}}+1)\sqrt{\lceil\log_2 N\rceil}\right)$. Because β_b is a signed monomial, $\beta_b\cdot(\mathbf{r}_{i^*,b},\mathbf{e}_{i^*,b})$ also follows the distribution $\mathcal{D}_{\mathbf{r}}^{\ell+k}$. Thus, due to Theorems 2.3 and 2.4, $\left\|\sum_{b\in[\mathsf{rep}]}\beta_b\cdot(\mathbf{r}_{i^*,b},\mathbf{e}_{i^*,b})\right\|$ is upper bounded by $\alpha\cdot\sigma_{\mathbf{r}}\cdot\sqrt{n(\ell+k)\cdot\mathsf{rep}}$ except with negligible probability.

We move to the second term. From the procedure of the key generation and signing protocol, we have

$$\begin{aligned}&\left\|\mathbf{w}_{i^*}-(\mathbf{A}\cdot\mathbf{z}_{i^*}-c\cdot\mathsf{vk}_{i^*}[\mathsf{Idx}[i^*]])-2c\cdot\mathbf{e}_{i^*}-\sum_{b\in[\mathsf{rep}]}\beta_b\cdot\mathbf{e}'_{i^*,b}\right\|\\ &=\left\|\sum_{b\in[\mathsf{rep}]}2^{\nu'_{\mathbf{w}}}\cdot\beta_b\cdot\left\lfloor\mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}\right\rceil_{\nu'_{\mathbf{w}}}-\sum_{b\in[\mathsf{rep}]}\beta_b\cdot\left(\mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}\right)\right\| \qquad (6)\end{aligned}$$

Following Theorem 2.6, there is a vector $\boldsymbol{\delta}_b$ s.t. $2^{\nu'_{\mathbf{w}}}\cdot\left\lfloor\mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}\right\rceil_{\nu'_{\mathbf{w}}} = \mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}+\boldsymbol{\delta}_b$ and $\|\boldsymbol{\delta}_b\|_\infty \leqslant 2^{\nu'_{\mathbf{w}}-1}$. By plugging this into the above equation, we get

$$\begin{aligned}Eq.\ (6) &= \left\|\sum_{b\in[\mathsf{rep}]}\beta_b\cdot\left(\mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}+\boldsymbol{\delta}_b\right)-\sum_{b\in[\mathsf{rep}]}\beta_b\cdot\left(\mathbf{A}\cdot\mathbf{r}_{i^*,b}+\mathbf{e}'_{i^*,b}\right)\right\|\\ &= \left\|\sum_{b\in[\mathsf{rep}]}\beta_b\cdot\boldsymbol{\delta}_b\right\| \leqslant \mathsf{rep}\cdot 2^{\nu'_{\mathbf{w}}-1}\sqrt{n\cdot k}\end{aligned}$$

where the final inequality follows the facts that β_b is a signed monomial and $\|\boldsymbol{\delta}_b\|_\infty \leqslant 2^{\nu'_{\mathbf{w}}-1}$.

Combining all arguments, the following inequality holds with overwhelming probability.

$$\begin{aligned}\|(\mathbf{z}_{i^*},\mathbf{y}_{i^*})\| \leqslant& 2\omega\cdot\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\cdot\left(1+(Q_{\mathsf{rf}}+1)\sqrt{\lceil\log_2 N\rceil}\right)\\ &+\alpha\cdot\sigma_{\mathbf{r}}\cdot\sqrt{n(\ell+k)\cdot\mathsf{rep}}+\mathsf{rep}\cdot 2^{\nu'_{\mathbf{w}}-1}\sqrt{n\cdot k}\leqslant B_2^{\mathsf{ind}}\end{aligned}$$

Final signature validity. We now show that the final signature $\mathsf{sig} := (c,\mathbf{z},\mathbf{h})$ outputted by Agg is valid if all partial signatures pass the partial verification.

By construction, c, $(\beta_b)_{b\in[\mathsf{rep}]}$, and inputs of H_c used in Agg, PartialVerify and Verify are identical. Thus it is sufficient to show that $\mathsf{sig} := (c,\mathbf{z},\mathbf{h})$ computed from $(\mathbf{w}_i,\mathbf{z}_i)_{i\in\mathsf{act}}$ s.t. $\|(\mathbf{z}_i,\mathbf{y}_i)\| \leqslant B_2^{\mathsf{ind}}$ satisfies $\|(\mathbf{z},2^{\nu_{\mathbf{w}}}\cdot\mathbf{h})\| \leqslant B_2$ with probability 1. From the triangle inequality and the partial verification of signatures, we have

$$\begin{aligned}\left\|(\mathbf{z},2^{\nu_{\mathbf{w}}}\cdot\bar{\mathbf{h}}\bmod q)\right\| &\leqslant \sum_{i\in\mathsf{act}}\|(\mathbf{z}_i,\mathbf{y}_i)\|+\left\|2^{\nu_{\mathbf{w}}}\cdot\bar{\mathbf{h}}-\sum_{i\in\mathsf{act}}\mathbf{y}_i\right\|\\ &\leqslant T\cdot B_2^{\mathsf{ind}}+\left\|2^{\nu_{\mathbf{w}}}\cdot\bar{\mathbf{h}}-\sum_{i\in\mathsf{act}}\mathbf{y}_i\right\|.\end{aligned}$$

We now evaluate the second term. Recall that $\mathbf{h} = \left\lfloor\sum_{i\in\mathsf{act}}\mathbf{w}_i\right\rceil_{\nu_{\mathbf{w}}}-\left\lfloor\mathbf{A}\cdot\mathbf{z}-2^{\nu_{\mathbf{t}}}\cdot c\cdot\mathbf{t}_\top\right\rceil_{\nu_{\mathbf{w}}}\bmod q_{\nu_{\mathbf{w}}}$. Let idx_i be $\mathsf{Idx}[i]$ for the list Idx outputted by VandRecover$([N],\mathsf{act},\varepsilon,\mathsf{Idx})$ and $\mathbf{t}_i$ be $\mathsf{vk}_i[\mathsf{idx}_i]$ during the epoch considered. Then, from the procedure of algorithms, $\mathbf{A}\cdot\mathbf{z} = \sum_{i\in\mathsf{act}}(\mathbf{w}_i-\mathbf{y}_i+c\cdot\mathbf{t}_i)$ holds. Replacing these equations into the above term, we obtain

$$\left\|2^{\nu_{\mathbf{w}}}\cdot\bar{\mathbf{h}}-\sum_{i\in\mathsf{act}}\mathbf{y}_i\right\|$$

$$= \left\| 2^{\nu_{\mathbf{w}}} \cdot \overline{\left\lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \right\rceil_{\nu_{\mathbf{w}}} - \left\lfloor \sum_{i \in \mathsf{act}} (\mathbf{w}_i - \mathbf{y}_i + c \cdot \mathbf{t}_i) - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t}_\top \right\rceil_{\nu_{\mathbf{w}}}} \bmod q_{\nu_{\mathbf{w}}} - \sum_{i \in \mathsf{act}} \mathbf{y}_i \right\|. \tag{7}$$

By the correctness of our sharing, we are initially guaranteed that the partial keys produced by Keygen verify $\sum_{i \in \mathsf{act}} \mathbf{t}_i = \mathbf{t}$ (as the corresponding partial secrets must sum to the overall secret). This property is then preserved even after refreshes as the partial public keys pass RefreshVerify during every refresh protocol. Also we recall that $\mathbf{t}_\top = \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}$. Substituting these equations into the above equation, we get

$$\begin{aligned} \textit{Eq. (7)} &= \left\| 2^{\nu_{\mathbf{w}}} \cdot \overline{\left\lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i \right\rceil_{\nu_{\mathbf{w}}} - \left\lfloor \sum_{i \in \mathsf{act}} \mathbf{w}_i - \sum_{i \in \mathsf{act}} \mathbf{y}_i + c \cdot \left(\mathbf{t} - 2^{\nu_{\mathbf{t}}} \cdot \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}\right) \right\rceil_{\nu_{\mathbf{w}}}} \bmod q_{\nu_{\mathbf{w}}} - \sum_{i \in \mathsf{act}} \mathbf{y}_i \right\| \\ &\leqslant 3 \cdot (2^{\nu_{\mathbf{w}}-1} + \max(0, q_\perp)) \cdot \sqrt{nk} + \left\| -c \cdot \left(\mathbf{t} - 2^{\nu_{\mathbf{t}}} \cdot \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}\right) \right\| \end{aligned}$$

where the second equality follows from Theorem 2.7 and the triangle inequality. Moreover, from Theorem 2.6, the infinity norm of $\mathbf{t} - 2^{\nu_{\mathbf{t}}} \cdot \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}$ is upper bounded by $2^{\nu_{\mathbf{t}}-1}$. Then we obtain

$$\left\| -c \cdot \left(\mathbf{t} - 2^{\nu_{\mathbf{t}}} \cdot \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}\right) \right\| \leqslant \omega \cdot 2^{\nu_{\mathbf{t}}-1} \cdot \sqrt{nk}.$$

Therefore, we have

$$\|(\mathbf{z}, 2^{\nu_{\mathbf{w}}} \cdot \mathbf{h})\| \leqslant T \cdot B_2^{\mathsf{ind}} + \left(3 \cdot (2^{\nu_{\mathbf{w}}-1} + \max(0, q_\perp)) + \omega \cdot 2^{\nu_{\mathbf{t}}-1}\right) \cdot \sqrt{nk} \leqslant B_2.$$

Combining all arguments $(c, \mathbf{z}, \mathbf{h})$ outputted by Agg always passes the verification with probability 1. This completes the proof.

□

6.3 Unforgeability

We show that Hermine is ts-suf-2 under MSIS and AOM-MISIS. Before stating the formal theorem, we explain how to set the parameters of the assumptions in the security theorem. The parameters q, k, ℓ are set to the values used in the scheme. For MSIS, we set $\beta = 2 \cdot B_2 + 2^{\nu_{\mathbf{w}}} \cdot \sqrt{nk}$. For AOM-MISIS, we set

- $Q = B_{\mathsf{shr}}(N, T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N, T) + \mathsf{rep} \cdot Q_{\mathsf{Sign}} + 1$
- $\beta_{\mathbf{s}} = 2B_2 + (2^{\nu_{\mathbf{w}}} + \omega \cdot 2^{\nu_{\mathbf{t}}}) \cdot \sqrt{kn}$
- $\beta_{\mathbf{b}} = 4 \cdot \omega \cdot \sigma_{\mathbf{s}}$
- $\beta_{\mathbf{u}} = \sqrt{(B_{\mathsf{shr}}(N, T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N, T) + 1)/\sigma_{\mathbf{s}}^2 + (8\omega^2 + 8\omega + 4) \cdot n \cdot Q_{\mathsf{Sign}}/\sigma_{\mathbf{r}}^2}$
- $\sigma_i = \sigma_{\mathbf{s}}$ for $i \in [1, B_{\mathsf{shr}}(N, T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N, T) + 1]$
- $\sigma_i = \sigma_{\mathbf{r}}$ for $i \in [B_{\mathsf{shr}}(N, T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N, T) + 2, Q]$

Below, we provide the formal security theorem and the security proof.

Theorem 6.4. Hermine *is* ts-suf-2 *under* MSIS *and* AOM-MISIS. *Formally, for any N, T such that $T < N$ and an adversary $\mathcal{A}$ against $\mathsf{Game}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{Hermine},\mathcal{A}}(1^\kappa, N, T)$ making at most Q_{H_c}, Q_{H_β}, Q_{Sign}, and Q_{rf} queries to the random oracles H_c and H_β, the signing oracle, and the refresh oracle, respectively, there are adversaries $\mathcal{B}_1$ against $\mathsf{MSIS}_{q,k,\ell,\beta}$ and $\mathcal{B}_2$ against $\mathsf{AOM\text{-}MISIS}_{q,k,\ell,Q,\beta_{\mathbf{s}},\beta_{\mathbf{b}},\beta_{\mathbf{u}},(\sigma_i)_{i\in[Q]}}$ such that*

$$\mathsf{Adv}^{\mathsf{ts\text{-}suf\text{-}2}}_{\mathsf{Hermine},\mathcal{A}}(1^\kappa, N, T) \leqslant \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}_1}(1^\kappa) + \frac{(Q_{H_c} + Q_{H_\beta})^2}{|\mathcal{C}|}$$

$$+ N \cdot \sqrt{Q_h \cdot \left(\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{B}_2}(\kappa) + \frac{2}{|\mathcal{C}|} + \frac{Q_{\mathsf{Sign}} + Q_{H_\beta}^2}{|\mathbb{T}|^{\mathsf{rep}-1}} \right)}$$

where $Q_h = Q_{H_c} + 2Q_{H_\beta} + 2Q_{\mathsf{Sign}} + 1$, $\mathsf{Time}(\mathcal{B}_1) \approx \mathsf{Time}(\mathcal{A})$, $\mathsf{Time}(\mathcal{B}_2) \approx 2 \cdot \mathsf{Time}(\mathcal{A})$.

Before showing this formally, we provide the proof sketch. The high-level proof strategy is similar to that of the two-round masking schemes [49, 88], which rely on AOM-MISIS. Specifically, the reduction embeds the problem instance $\mathbf{A}$ and $(\mathbf{t}_i)_{i\in[Q]}$ of AOM-MISIS into vk and the honest signers' commitments, simulates all signing queries by exploiting access to the PI oracle and extracts a valid solution to AOM-MISIS from two valid forgeries obtained through rewinding. The main differences are the following three points: (i) how the reduction simulates keys for corrupted signers and deals with malicious key refreshes. (ii) how it simulates all signing queries without masking, and (iii) how it computes a valid solution from forgeries satisfying the winning condition of ts-suf-2.

For point (i), to simulate the corrupted signers' keys, we rely on the simulators SimShares and SimRefresh from Theorem 4.1. To run these simulators, we need to provide hints on the secret $(\mathbf{s}, \mathbf{e})$ as input. We prepare these hints by relying on the PI oracle access. Concretely, the reduction receives $\mathbf{t}_i$ not only for vk and the honest signers' commitments but also for these hints. For clarity, we denote the $\mathbf{t}_i$ involved in vk, an individual commitment, and a hint by $\mathbf{t}^*$, $\mathbf{t}'$, and $\mathbf{t}''$, respectively. Then, the reduction obtains a hint h_i by making a query $\mathbf{b}$ corresponding to $\mathbf{t}^* + \mathbf{t}''_i$. Thanks to these simulators, we perfectly simulate the honest component of the keys for corrupted signers. Moreover, it is easy to deal with malicious key refreshes because we can track the honest component $\mathsf{sk}_i^{(\mathsf{hon})}$ and the malicious component $\mathsf{sk}_i^{(\mathsf{mal})}$ such that $\mathsf{sk}_i = \mathsf{sk}_i^{(\mathsf{hon})} + \mathsf{sk}_i^{(\mathsf{mal})}$. In fact, for corrupted signers, the reduction can derive malicious components from the simulated honest components and the malicious sk_i provided as a query to $\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}$. Also, for the honest signers' keys, since the adversary $\mathcal{A}$ queries rf_i, the reduction can simply store it as the malicious components.

Regarding the point (ii), we again exploit the simulators. According to Theorem 4.1, for the honest signers, the simulators provide two types of simulated values $\mathsf{Shr}'[i][\mathsf{idx}]$: (1) a value distributed identically to the real share $\mathsf{sk}_i[\mathsf{idx}] = \mathsf{Shr}'[i][\mathsf{idx}]$, or (2) a value distributed as the secret *minus the share* $\mathsf{sk}_i[\mathsf{idx}] = \mathbf{s} - \mathsf{Shr}'[i][\mathsf{idx}]$. We note that there exists exactly one signer holding values of type (2) due to property (b). In the preprocessing, we embed $\mathbf{t}'$ into an individual commitment as $\mathbf{w}_{i,b} := \left\lfloor \mathbf{t}'_{i,b} \right\rceil_{\nu'_{\mathbf{w}}}$. Then, we can simulate $\mathbf{z}_i$ using the PI oracle access. Concretely, for a signer corresponding to type (1), we make a PI query for $\sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{t}'_{i,b}$ and compute $\mathbf{z}_i$ from $\mathsf{Shr}'[i][\mathsf{idx}]$ and the response from the PI oracle. For type (2), we instead make a query for $2c \cdot \mathbf{t}^* + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{t}'_{i,b}$, obtain the response $(\mathbf{s}', \mathbf{e}')$, and compute $\mathbf{z}_i$ as $\mathbf{s}' - 2c \cdot \mathsf{Shr}'[i][\mathsf{idx}]$. This perfectly simulates the real $\mathbf{z}_i$ due to $\mathsf{sk}_i[\mathsf{idx}] = \mathbf{s} - \mathsf{Shr}'[i][\mathsf{idx}]$. To ensure consistency with the malicious key refresh, we add $2c \cdot \mathsf{sk}_i^{(\mathsf{mal})}$ in both cases.

Finally, we describe how the final point (iii) is addressed. For simplicity, we first consider standard unforgeability rather than the strong one. Due to special soundness, we can derive a valid MISIS solution for $\mathbf{t}^*$ from two valid forgeries produced under distinct challenges via rewinding. The non-trivial point is whether the reduction can compute a valid $\mathbf{u}$ satisfying $\mathbf{b}^\top \cdot \mathbf{u} = 0$ for all PI queries $\mathbf{b}$. Recall that this condition is to ensure that the AOM-MISIS adversary does not make queries from which a valid MISIS solution could be trivially derived from the PI oracle responses. In our reduction, such trivial queries might occur if it makes two queries involving vk and $\overrightarrow{\mathbf{w}}_i$ with the same weights $(\beta_b)_{b\in[\mathsf{rep}]}$ but distinct c_1 and c_2 before and after rewinding, respectively, noting the reduction does not reuse $\overrightarrow{\mathbf{w}}_i$ within a single run. Indeed, responses $(\mathbf{s}'_1, \mathbf{e}'_1)$ and $(\mathbf{s}'_2, \mathbf{e}'_2)$ to such queries satisfy $\mathbf{A} \cdot (\mathbf{s}'_1 - \mathbf{s}'_2) + (\mathbf{e}'_1 - \mathbf{e}'_2) = 2(c_1 - c_2) \cdot \mathbf{t}^*$. Thus, we need to show that $(\star)$ the reduction does not make such trivial queries.

Due to how rewinding works, the rewinding point occurs at the forgery. Then, assuming no collisions in the random oracle, c_1 and c_2 are challenges used in the forgery, and must be defined after $(\beta_b)_{b\in[\mathsf{rep}]}$ is defined.[12] Then it is easy to check that $(\star)$ holds in the case of TS-UF-0 because $\mathcal{A}$ cannot query the

[12] A more careful argument is required to account for key refreshing. See the formal proof for more details.

to-be-forged message msg^* to the signing oracle. In contrast, in the case of TS-UF-2, there may be signing sessions associated with c^* since $\mathcal{A}$ is allowed to make signing queries related to c^* as long as fewer than $T - |\mathsf{CS}|$ honest signers complete the signing session.

To overcome this, we rely on property (c) of the simulators. This property allows us to generate Shr' such that a designated signer i^* has type (2) values whenever $i^* \in \mathsf{HS}$. Thus, the reduction makes no PI query involving c^* if this signer is honest in the epoch in which such a signing session occurs, and does not complete it. To this end, we guess such a signer i^* at the beginning of the game and run the simulators with the designated signer i^*. After $\mathcal{A}$ terminates, we run SecureUser to obtain the actual honest signer j^* who did not complete such a session, and the reduction aborts if $j^* \neq i^*$. Then, we ensure that $(\star)$ holds unless the reduction aborts.

The aforementioned strategy for (iii) cannot be directly applied to the case of strong unforgeability. Recall that $\mathcal{A}$ can win the strong unforgeability game as long as it outputs $t+1$ valid distinct forgeries associated with c^*, even if there are t signing sessions involving c^* in which $T - |\mathsf{CS}|$ honest signers completed the sessions. To address this issue, we rely on the MSIS assumption because such multiple forgeries yield a valid solution to MSIS for the matrix $\mathbf{A}$. By combining all arguments, we can construct a reduction from AOM-MISIS to ts-suf-2.

Proof. We prove the theorem by a sequence of games. The first game is the original security game defined in Fig. 3, and we will construct a reduction from AOM-MISIS by using an adversary who wins the final game. We denote the winning probability of an adversary $\mathcal{A}$ in Game_i by ε_i.

Game_0: This is the original ts-suf-2, which is depicted in Fig. 8. Thus we have

$$\varepsilon_0 = \mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{Hermine},\mathcal{A}}(1^\kappa, N, T).$$

Game_1: In this game, the challenger defines the corresponding challenge c when a random oracle query to H_β is made. The game is depicted in Fig. 9. Concretely, after defining $H_\beta(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$, it also defines $H_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$ where $\mathbf{w}$ is computed from a queried $\overrightarrow{\mathsf{pp}}$ if it has not been defined. Also, it keeps the corresponding $\overline{\mathsf{ctnt}} = (\mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$ in the set ChallengeSet along with each challenge c. If c is defined within H_c, we store $(c, \perp)$ in ChallengeSet. If it is defined via $H_\beta(\overline{\mathsf{ctnt}})$, we store $\overline{\mathsf{ctnt}}$ with c in ChallengeSet. Because this is a conceptual change, we obtain

$$\varepsilon_0 = \varepsilon_1.$$

Game_2: In this game, the challenger aborts the game if there is a collision in the hash function H_c. Specifically, when it defines H_c, it checks whether there exists $(c, \cdot)$ in ChallengeSet. If so, it aborts. This is depicted in Fig. 10. Since $\mathcal{A}$ can make at most $Q_{H_c} + Q_{H_\beta}$ random oracle queries, we have

$$|\varepsilon_1 - \varepsilon_2| \leqslant \frac{(Q_{H_c} + Q_{H_\beta})^2}{|\mathcal{C}|}.$$

Game_3: In this game, the challenger aborts if there are two forgeries by the adversary with the same challenge c. This is depicted in Fig. 11. Concretely, after $\mathcal{A}$ outputs forgeries $(\mathsf{msg}^*, (c_j, \mathbf{z}_j, \mathbf{h}_j)_{j\in[\ell]})$, it first checks the winning condition as in the previous game. After that, it checks whether there exist two distinct forgeries involved in the same challenge c. If so, it aborts the game, otherwise, it returns 0.

We show that we can construct an algorithm solving the MSIS problem if $\mathcal{A}$ outputs two such forgeries. The reduction $\mathcal{B}_1$ is given a matrix $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$ as input and embeds $\mathbf{A}$ into a matrix in vk. It simulates the rest of this game by behaving as the challenger. Note that it can perfectly simulate the game since it honestly generates secret key shares.

Below we prove that, if $\mathcal{A}$ produces two forgeries that trigger the abort condition, then $\mathcal{B}_1$ can compute a solution to the MSIS problem as follows: Let $(c_i, \mathbf{z}_i, \mathbf{h}_i)$ and $(c_j, \mathbf{z}_j, \mathbf{h}_j)$ be the corresponding two forgeries satisfying the abort condition such that $c_i = c_j$ and $(\mathbf{z}_i, \mathbf{h}_i) \neq (\mathbf{z}_j, \mathbf{h}_j)$. Since there is no collision in H_c due to the modification made in Game_2, c_i and c_j are the output of H_c with the same input. Then, we have

$$\lfloor \mathbf{A}\cdot\mathbf{z}_i - 2^{\nu_\mathbf{t}}\cdot c_i\cdot\mathbf{t}_\top \rceil_{\nu_\mathbf{w}} + \mathbf{h}_i = \lfloor \mathbf{A}\cdot\mathbf{z}_j - 2^{\nu_\mathbf{t}}\cdot c_j\cdot\mathbf{t}_\top \rceil_{\nu_\mathbf{w}} + \mathbf{h}_j \bmod q_{\nu_\mathbf{w}}.$$

$\mathsf{Game}^{\text{ts-suf-2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$

$Q := \emptyset$; $\mathsf{id_{rf}} := 0$; $\mathsf{Signed}[\cdot] := \{\}$
$\mathsf{tspar} \leftarrow \mathsf{TS.Setup}(1^\kappa, N, T)$
$(\mathsf{CS}^{(0)}, \mathsf{st}_\mathcal{A}) \leftarrow \mathcal{A}^H(\mathsf{tspar})$
$\mathsf{HS}^{(0)} := [N] \backslash \mathsf{CS}^{(0)}$
for $i \in \mathsf{HS}^{(0)}$ **do** $\mathsf{st}_i := \emptyset$
$(\mathsf{vk}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[N]}) \leftarrow \mathsf{TS.Keygen}(\mathsf{tspar})$
$O := H, \mathsf{O}_{\mathsf{vk}_i}, \mathsf{O}_{\mathsf{TS.Preprocess}}, \mathsf{O}_{\mathsf{TS.Sign}}, \mathsf{O}_{\mathsf{TS.RefreshUpdate}}, \mathsf{O}_{\mathsf{TS.RefreshDealer}}$
$\mathsf{msg}^*, (\mathsf{sig}^*_j)_{j\in[\ell]} \leftarrow \mathcal{A}^O(\mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(0)}}, \mathsf{st}_\mathcal{A})$
require $(\mathsf{sig}^*_j)_{j\in[\ell]}$ pairwise distinct
$s := 0$ // Count signatures on msg^*
for $\mathsf{ctnt} \in Q$ s.t. $\mathsf{ctnt} = (\cdot, \cdot, \mathsf{msg}^*, \cdot)$ **do**
  Parse $\mathsf{ctnt} = (\mathsf{id}^*, \mathsf{act}^*, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}^*)$
  if $|\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$ **then**
    $s \leftarrow s + 1$
return $\forall j \in [\ell], \mathsf{TS.Verify}(\mathsf{vk}, \mathsf{msg}^*, \mathsf{sig}^*_j) \wedge \ell > s$

$\mathsf{O}_{\mathsf{vk}_i}(i, \mathsf{vk}'_i)$

for idx s.t. $\mathsf{vk}'_i[\mathsf{idx}] \neq \perp$ **do**
  $\mathsf{vk}_i[\mathsf{idx}] \leftarrow \mathsf{vk}'_i[\mathsf{idx}]$

$\mathsf{O}_{\mathsf{TS.RefreshDealer}}((\mathsf{vk}_i)_{i\in[N]}, \mathsf{CS}', (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id_{rf}})}})$

require $\mathsf{CS}' \subset [N] \wedge |\mathsf{CS}'| < T$
// Dealer refresh
$\mathsf{Refresh}[\cdot][\cdot] := \{\}$
Parse $\mathsf{vk} = (\mathbf{A}, \mathbf{t}_\top)$
$\mathsf{Refresh} \leftarrow \mathsf{VandShare}(\mathbf{0}, [N], T, \varepsilon, \mathsf{Refresh})$
for $i \in [N]$ **do**
  $\mathsf{rf}_i[\cdot] := \{\}$
  for idx s.t. $\mathsf{Refresh}[i][\mathsf{idx}] \neq \perp$ **do**
    Parse $\mathsf{Shr}[i][\mathsf{idx}] = (\hat{\mathbf{s}}_{\mathsf{idx}}, \hat{\mathbf{e}}_{\mathsf{idx}})$
    $\mathsf{vk}_i[\mathsf{idx}] \leftarrow \mathsf{vk}_i[\mathsf{idx}] + 2 \cdot (\mathbf{A} \cdot \hat{\mathbf{s}}_{\mathsf{idx}} + \hat{\mathbf{e}}_{\mathsf{idx}})$
    $\mathsf{rf}_i[\mathsf{idx}] \leftarrow \hat{\mathbf{s}}_{\mathsf{idx}}$
// Signer refresh
for $i \in [N]$ **do**
  for idx s.t. $\mathsf{sk}_i[\mathsf{idx}] \neq \perp$ **do**
    $\mathsf{sk}_i[\mathsf{idx}] \leftarrow \mathsf{sk}_i[\mathsf{idx}] + \mathsf{rf}_i[\mathsf{idx}]$
  $\mathsf{st}_i \leftarrow \{\}$ // Reset the state of participants
$\mathsf{CS}^{(\mathsf{id_{rf}}+1)} := \mathsf{CS}'; \mathsf{HS}^{(\mathsf{id_{rf}}+1)} := [N] \backslash \mathsf{CS}'$
$\mathsf{id_{rf}} \leftarrow \mathsf{id_{rf}} + 1$
return $((\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id_{rf}})}})$

$\mathsf{O}_{\mathsf{TS.Preprocess}}(i)$

require $i \in \mathsf{HS}^{(\mathsf{id_{rf}})}$
for $b \in [\mathsf{rep}]$ **do**
  $(\mathbf{r}_{i,b}, \mathbf{e}'_{i,b}) \leftarrow \mathcal{D}^\ell_\mathbf{r} \times \mathcal{D}^k_\mathbf{r}$
  $\mathbf{w}_{i,b} := \left\lfloor \mathbf{A} \cdot \mathbf{r}_{i,b} + \mathbf{e}'_{i,b} \right\rceil_{\nu'_\mathbf{w}}$
$\overrightarrow{\mathbf{w}}_i := [\mathbf{w}_{i,1} | \cdots | \mathbf{w}_{i,\mathsf{rep}}]$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{ (\overrightarrow{\mathbf{w}}_i, (\mathbf{r}_{i,b})_{b\in[\mathsf{rep}]}) \}$
$\mathsf{pp}_i := \overrightarrow{\mathbf{w}}_i$
return pp_i

$\mathsf{O}_{\mathsf{TS.Sign}}(\mathsf{act}, \mathsf{msg}, i, \overrightarrow{\mathsf{pp}} = (\overrightarrow{\mathbf{w}}_j)_{j\in\mathsf{act}})$

require $\mathsf{act} \subset [N] \wedge |\mathsf{act}| = T \wedge i \in \mathsf{HS}^{(\mathsf{id_{rf}})} \cap \mathsf{act}$
require $(\overrightarrow{\mathbf{w}}_i, \cdot) \in \mathsf{st}_i$
Fetch $(\overrightarrow{\mathbf{w}}_i, (\mathbf{r}_{i,b})_{b\in[\mathsf{rep}]})$ from st_i
// $\beta_1 = 1, \beta_b \in \mathbb{T}$
$(\beta_b)_{b\in[\mathsf{rep}]} := H_\beta(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$
for $j \in \mathsf{act}$ **do**
  Parse $\overrightarrow{\mathbf{w}}_j = [\mathbf{w}_{j,1} | \cdots | \mathbf{w}_{j,\mathsf{rep}}]$
  $\mathbf{w}_j := \sum_{b\in[\mathsf{rep}]} 2^{\nu'_\mathbf{w}} \cdot \beta_b \cdot \mathbf{w}_{j,b}$
$\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{act}} \mathbf{w}_j \right\rceil_{\nu_\mathbf{w}} \mod q_{\nu_\mathbf{w}}$
$c := H_c(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$
$\mathsf{Idx}[\cdot] := \{\}$ // Find idx corresponding to act
$\mathsf{Idx} \leftarrow \mathsf{VandRecover}([N], \mathsf{act}, \varepsilon, \mathsf{Idx})$
$\mathbf{z}_i := 2c \cdot \mathsf{sk}_i[\mathsf{Idx}[i]] + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{ (\overrightarrow{\mathbf{w}}_i, (\mathbf{r}_{i,b})_{b\in[\mathsf{rep}]}) \}$
// Tracks completed signing session
$\mathsf{ctnt} \leftarrow (\mathsf{id_{rf}}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$
$Q \leftarrow Q \cup \{ \mathsf{ctnt} \}$
$\mathsf{Signed}[\mathsf{ctnt}] \leftarrow \mathsf{Signed}[\mathsf{ctnt}] \cup \{i\}$
return $\mathsf{sig}_i := \mathbf{z}_i$

$\mathsf{O}_{\mathsf{TS.RefreshUpdate}}(i, \mathsf{rf}_i)$

require $i \in \mathsf{HS}^{(\mathsf{id_{rf}})}$
for idx s.t. $\mathsf{sk}_i[\mathsf{idx}] \neq \perp$ **do**
  $\mathsf{sk}_i[\mathsf{idx}] \leftarrow \mathsf{sk}_i[\mathsf{idx}] + \mathsf{rf}_i[\mathsf{idx}]$

Figure 8: The ts-suf-2 game for Hermine.

Since $\mathbf{h}_i = \mathbf{h}_j$ holds if $\mathbf{z}_i = \mathbf{z}_j$, which contradicts with $(\mathbf{z}_i, \mathbf{h}_i) \neq (\mathbf{z}_j, \mathbf{h}_j)$, $\mathbf{z}_i \neq \mathbf{z}_j$ holds. Let $\boldsymbol{\delta}$ be

$$\begin{aligned}\boldsymbol{\delta} = &(\mathbf{A} \cdot (\mathbf{z}_i - \mathbf{z}_j) - 2^{\nu_\mathbf{t}} \cdot (c_i - c_j) \cdot \mathbf{t}_\top) \\ &+ 2^{\nu_\mathbf{w}} \cdot \overline{\lfloor \mathbf{A} \cdot \mathbf{z}_j - 2^{\nu_\mathbf{t}} \cdot c_i \cdot \mathbf{t}_\top \rceil_{\nu_\mathbf{w}} - \lfloor \mathbf{A} \cdot \mathbf{z}_i - 2^{\nu_\mathbf{t}} \cdot c_j \cdot \mathbf{t}_\top \rceil_{\nu_\mathbf{w}}} \mod q_{\nu_\mathbf{w}}.\end{aligned}$$

Substituting $c_i = c_j$, we get

$$\mathbf{A} \cdot (\mathbf{z}_i - \mathbf{z}_j) + 2^{\nu_\mathbf{w}} \cdot (\overline{\mathbf{h}_i - \mathbf{h}_j}) - \boldsymbol{\delta} = 0 \pmod q.$$

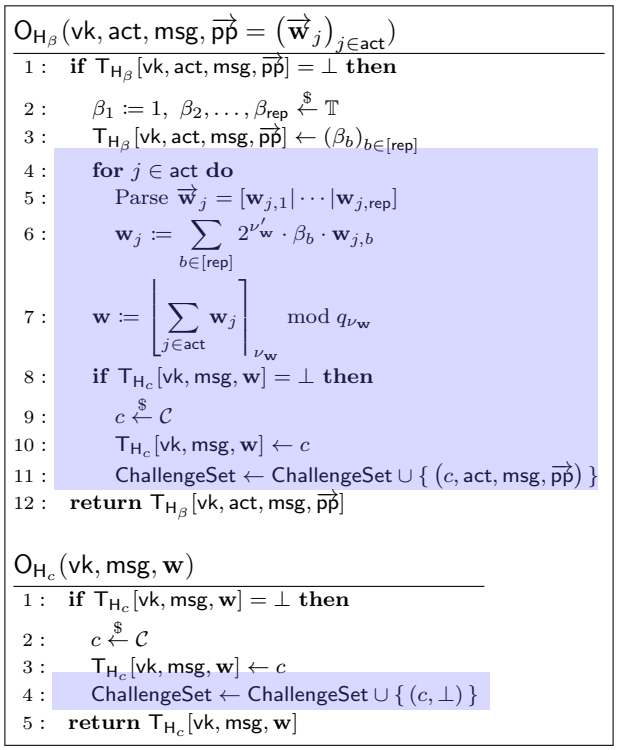


Figure 9: The first game Game_1. $\mathsf{T}_{\mathsf{H}_c}$ and $\mathsf{T}_{\mathsf{H}_\beta}$ are random oracle tables which are initialized to $\bot$. This game initializes the set $\mathsf{ChallengeSet}$ to empty. Differences from Game_0 are highlighted in blue.

Thus $\mathbf{z}^* = (\mathbf{z}_i - \mathbf{z}_j, 2^{\nu_{\mathbf{w}}} \cdot (\mathbf{h}_i - \mathbf{h}_j) - \boldsymbol{\delta})$ satisfies $[\mathbf{A}\ \mathbf{I}] \cdot \mathbf{z}^* = 0$ and is a non-zero vector thanks to $\mathbf{z}_i \neq \mathbf{z}_j$. Also, we can see that

$$\begin{aligned}\|\boldsymbol{\delta}\| &\leqslant \left\| \mathbf{A} \cdot \mathbf{z}_i - 2^{\nu_{\mathbf{w}}} \cdot \overline{\lfloor \mathbf{A} \cdot \mathbf{z}_i - 2^{\nu_{\mathbf{t}}} \cdot c_i \cdot \mathbf{t}_\top \rceil_{\nu_{\mathbf{w}}}} \mod q_{\nu_{\mathbf{w}}} \right\| \\ &\quad + \left\| \mathbf{A} \cdot \mathbf{z}_j - 2^{\nu_{\mathbf{w}}} \cdot \overline{\lfloor \mathbf{A} \cdot \mathbf{z}_j - 2^{\nu_{\mathbf{t}}} \cdot c_j \cdot \mathbf{t}_\top \rceil_{\nu_{\mathbf{w}}}} \mod q_{\nu_{\mathbf{w}}} \right\| \\ &\leqslant 2 \cdot (2^{\nu_{\mathbf{w}}-1} + \max(0, q_\perp)) \cdot \sqrt{nk}.\end{aligned}$$

where the last inequality follows from Theorem 2.7.

From above arguments and construction of Verify, we have $\|\mathbf{z}^*\| \leqslant 2 \cdot B_2 + 2 \cdot (2^{\nu_{\mathbf{w}}-1} + \max(0, q_\perp)) \cdot \sqrt{nk} = \beta$. Since vectors $\boldsymbol{\delta}_1$ and $\boldsymbol{\delta}_2$ are efficiently computable, $\mathcal{B}_1$ can compute a solution of MSIS. Therefore, we have $\mathsf{Time}(\mathcal{B}_1) \approx \mathsf{Time}(\mathcal{A})$ and

$$|\varepsilon_2 - \varepsilon_3| \leqslant \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}_1}(1^\kappa).$$

Game_4: In this game, the challenger guesses a secure signer j^* who does not participate in the signing session corresponding to a non-trivial forgery. This is depicted in Fig. 12. Specifically, in the initial phase, it initializes a set Triv to empty and chooses a signer index j^* from $[N]$ uniformly at random. After the adversary terminates with its output, when the challenger counts the number of ctnt such that $|\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$, it computes the challenge c^* from ctnt and stores (c^*, ctnt) in Triv if ctnt satisfies that condition, i.e., the signing session with ctnt has been completed. Moreover, after checking the winning condition made in the previous game, it searches for a forgery $(c_i, \mathbf{z}_i, \mathbf{h}_i)$ such that c_i is not in Triv. If there is no such forgery, the game returns 0. If such a forgery is found, which is called a non-trivial forgery, it checks if there exist $\overline{\mathsf{ctnt}}_i$ such that $(c_i, \overline{\mathsf{ctnt}}_i) \in \mathsf{ChallengeSet}$, $\overline{\mathsf{ctnt}}_i \neq \bot$, and $\overline{\mathsf{ctnt}}_i$ is contained in Q, which means that

$\mathsf{O}_{\mathsf{H}_\beta}(\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}} = (\overrightarrow{\mathbf{w}}_j)_{j\in\mathsf{act}})$

1 : **if** $\mathsf{T}_{\mathsf{H}_\beta}[\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}}] = \bot$ **then**
2 : $\quad \beta_1 \coloneqq 1,\ \beta_2, \ldots, \beta_{\mathsf{rep}} \xleftarrow{\$} \mathbb{T}$
3 : $\quad \mathsf{T}_{\mathsf{H}_\beta}[\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}}] \leftarrow (\beta_b)_{b\in[\mathsf{rep}]}$
4 : $\quad$ **for** $j \in \mathsf{act}$ **do**
5 : $\quad\quad$ Parse $\overrightarrow{\mathbf{w}}_j = [\mathbf{w}_{j,1}| \cdots |\mathbf{w}_{j,\mathsf{rep}}]$
6 : $\quad\quad \mathbf{w}_j \coloneqq \sum_{b\in[\mathsf{rep}]} 2^{\nu'_{\mathbf{w}}} \cdot \beta_b \cdot \mathbf{w}_{j,b}$
7 : $\quad \mathbf{w} \coloneqq \left\lfloor \sum_{j\in\mathsf{act}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}} \bmod q_{\nu_{\mathbf{w}}}$
8 : $\quad$ **if** $\mathsf{T}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{msg}, \mathbf{w}] = \bot$ **then**
9 : $\quad\quad c \xleftarrow{\$} \mathcal{C}$
10 : $\quad\quad$ **abort if** $(c, \cdots) \in \mathsf{ChallengeSet}$
11 : $\quad\quad \mathsf{T}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{msg}, \mathbf{w}] \leftarrow c$
12 : $\quad\quad \mathsf{ChallengeSet} \leftarrow \mathsf{ChallengeSet} \cup \{ (c, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}}) \}$
13 : $\quad$ **return** $\mathsf{T}_{\mathsf{H}_\beta}[\mathsf{vk}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}}]$

$\mathsf{O}_{\mathsf{H}_c}(\mathsf{vk}, \mathsf{msg}, \mathbf{w})$

1 : **if** $\mathsf{T}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{msg}, \mathbf{w}] = \bot$ **then**
2 : $\quad c \xleftarrow{\$} \mathcal{C}$
3 : $\quad$ **abort if** $(c, \cdots) \in \mathsf{ChallengeSet}$
4 : $\quad \mathsf{T}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{msg}, \mathbf{w}] \leftarrow c$
5 : $\quad \mathsf{ChallengeSet} \leftarrow \mathsf{ChallengeSet} \cup \{ (c, \bot) \}$
6 : **return** $\mathsf{T}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{msg}, \mathbf{w}]$

Figure 10: The second game Game_2. Differences from Game_1 are highlighted in blue.

$\mathsf{Game}^{\text{ts-suf-2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$

1 : $Q \coloneqq \emptyset;\ \mathsf{id}_{\mathsf{rf}} \coloneqq 0;\ \mathsf{Signed}[\cdot] \coloneqq \{\}$
2 : $\mathsf{tspar} \leftarrow \mathsf{TS.Setup}(1^\kappa, N, T)$
3 : $(\mathsf{CS}^{(0)}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}^H(\mathsf{tspar})$
4 : $\mathsf{HS}^{(0)} \coloneqq [N] \backslash \mathsf{CS}^{(0)}$
5 : **for** $i \in \mathsf{HS}^{(0)}$ **do** $\mathsf{st}_i \coloneqq \emptyset$
6 : $(\mathsf{vk}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[N]}) \leftarrow \mathsf{TS.Keygen}(\mathsf{tspar})$
7 : $O \coloneqq (H, \mathsf{O}_{\mathsf{vk}_i}, \mathsf{O}_{\mathsf{TS.Preprocess}}, \mathsf{O}_{\mathsf{TS.Sign}}, \mathsf{O}_{\mathsf{TS.RefreshUpdate}}, \mathsf{O}_{\mathsf{TS.RefreshDealer}})$
8 : $\mathsf{msg}^*, (\mathsf{sig}^*_j)_{j\in[\ell]} \leftarrow \mathcal{A}^O(\mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(0)}}, \mathsf{st}_{\mathcal{A}})$
9 : **require** $(\mathsf{sig}^*_j)_{j\in[\ell]}$ pairwise distinct
10 : $s \coloneqq 0$ // Count signatures on msg^*
11 : **for** $\mathsf{ctnt} \in Q$ s.t. $\mathsf{ctnt} = (\cdot, \cdot, \mathsf{msg}^*, \cdot)$ **do**
12 : $\quad$ Parse $\mathsf{ctnt} = (\mathsf{id}^*, \mathsf{act}^*, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}^*)$
13 : $\quad$ **if** $|\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$ **then** $s \leftarrow s + 1$
14 : **require** $\forall j \in [\ell], \mathsf{TS.Verify}(\mathsf{vk}, \mathsf{msg}^*, \mathsf{sig}^*_j) \wedge \ell > s$ // Check winning condition
15 : Parse $(c_j, \mathbf{z}_j, \mathbf{h}_j)_{j\in[\ell]} = (\mathsf{sig}^*_j)_{j\in[\ell]}$
16 : **if** $\exists i, j \in [\ell]$ s.t. $i \neq j \wedge c_i = c_j$ **then**
17 : $\quad$ **return** 0
18 : **return** 1

Figure 11: The third game Game_3. Differences from Game_2 are highlighted in blue.

there exist honest signers who complete the signing session corresponding to $\overline{\mathsf{ctnt}}_i$. We note that this c_i was defined just after $\mathsf{O}_{H_\beta}(\mathsf{vk}, \mathsf{act}_i, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}_i)$ was defined where $\overline{\mathsf{ctnt}}_i = (\mathsf{id}_i, \mathsf{act}_i, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}_i)$. If so, it derives j by running $\mathsf{SecureUser}(\mathsf{CS}, \mathsf{act}_i, \mathsf{act}'_i)$ where $\mathsf{act}'_i = \mathsf{Signed}[(\mathsf{id}_i, \overline{\mathsf{ctnt}}_i)]$ is the set of honest participants who contributed to the signing session under $\overline{\mathsf{ctnt}}_i$. If $j^* \neq j$, it returns 0. Otherwise, it returns 1. We note that $\mathsf{act}'_i \subset \mathsf{act}_i \cap \mathsf{HS}$ and $|\mathsf{act}'_i| < T - |\mathsf{CS}^{(\mathsf{id}_i)}|$ holds since the adversary does not complete the signing session involved in $\overline{\mathsf{ctnt}}_i$ due to the winning condition.

The advantage of $\mathcal{A}$ remains the same unless (i) $j^* \neq \mathsf{SecureUser}(\mathsf{CS}, \mathsf{act}_i, \mathsf{act}'_i)$ holds or (ii) all c_i output by $\mathcal{A}$ are contained in Triv. Since j^* is uniformly chosen from $[N]$, $j^* = \mathsf{SecureUser}(\mathsf{CS}, \mathsf{act}_i, \mathsf{act}'_i)$ holds with probability $1/N$. Thus the $\mathcal{A}$'s winning probability decreases by at most a factor N in case (i). For the case (ii), since all challenges are distinct due to the abort condition added in the previous game, all corresponding ctnt satisfy $|\mathsf{act}'_i| \geqslant |\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$. This contradicts the winning condition $\ell > s$. Thus the event (ii) never happens. Thus it is easy to check that the following holds:

$$\varepsilon_3 \leqslant N \cdot \varepsilon_4.$$

$\mathsf{Game}^{\text{ts-suf-2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$

$Q := \emptyset$; $\mathsf{id}_{\mathsf{rf}} := 0$; $\mathsf{Signed}[\cdot] := \{\}$
$\mathsf{tspar} \leftarrow \mathsf{TS.Setup}(1^\kappa, N, T)$
$\mathsf{Triv} := \emptyset$
$j^* \xleftarrow{\$} [N]$
$(\mathsf{CS}^{(0)}, \mathsf{st}_{\mathcal{A}}) \leftarrow \mathcal{A}^H(\mathsf{tspar})$
$\mathsf{HS}^{(0)} \leftarrow [N] \backslash \mathsf{CS}^{(0)}$
for $i \in \mathsf{HS}^{(0)}$ **do** $\mathsf{st}_i \leftarrow \emptyset$
$(\mathsf{vk}, (\mathsf{vk}_i, \mathsf{sk}_i)_{i\in[N]}) \leftarrow \mathsf{TS.Keygen}(\mathsf{tspar})$
$O := (H, \mathsf{O}_{\mathsf{vk}_i}, \mathsf{O}_{\mathsf{TS.Preprocess}}, \mathsf{O}_{\mathsf{TS.Sign}}, \mathsf{O}_{\mathsf{TS.RefreshUpdate}}, \mathsf{O}_{\mathsf{TS.RefreshDealer}})$
$\mathsf{msg}^*, (\mathsf{sig}^*_j)_{j\in[\ell]} \leftarrow \mathcal{A}^O(\mathsf{vk}, (\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(0)}}, \mathsf{st}_{\mathcal{A}})$
require $(\mathsf{sig}^*_j)_{j\in[\ell]}$ pairwise distinct
$s := 0$ // Count signatures on msg^*
for $\mathsf{ctnt} \in Q$ s.t. $\mathsf{ctnt} = (\cdot, \cdot, \mathsf{msg}^*, \cdot)$ **do**
  Parse $\mathsf{ctnt} = (\mathsf{id}^*, \mathsf{act}^*, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}^*)$
  if $|\mathsf{Signed}[\mathsf{ctnt}]| \geqslant T - |\mathsf{CS}^{(\mathsf{id})}|$ **then**
    $s \leftarrow s + 1$
    Compute c^* from $(\mathsf{act}^*, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}^*)$
    $\mathsf{Triv} \leftarrow \mathsf{Triv} \cup \{ c^*, \mathsf{ctnt} \}$
require $\forall j \in [\ell], \mathsf{TS.Verify}(\mathsf{vk}, \mathsf{msg}^*, \mathsf{sig}^*_j) \wedge \ell > s$ // Check winning condition
Parse $(c_j, \mathbf{z}_j, \mathbf{h}_j)_{j\in[\ell]} = (\mathsf{sig}^*_j)_{j\in[\ell]}$
if $\exists i, j \in [\ell]$ s.t. $i \neq j, c_i = c_j$ **then**
  return 0
if $\exists i \in [\ell]$ s.t. $(c_i, \cdot) \notin \mathsf{Triv}$ **then**
  Fetch $(c_i, \overline{\mathsf{ctnt}}_i)$ from $\mathsf{ChallengeSet}$
  if $\overline{\mathsf{ctnt}}_i \neq \bot \wedge (\cdot, \overline{\mathsf{ctnt}}_i) \in Q$ **then**
    Fetch $(\mathsf{id}_i, \overline{\mathsf{ctnt}}_i)$ from Q
    Parse $\overline{\mathsf{ctnt}}_i = (\mathsf{act}_i, \mathsf{msg}^*, \overrightarrow{\mathsf{pp}}_i)$
    $\mathsf{act}'_i := \mathsf{Signed}[(\mathsf{id}_i, \overline{\mathsf{ctnt}}_i)]$
    if $j^* \neq \mathsf{SecureUser}(\mathsf{CS}^{(\mathsf{id}_i)}, \mathsf{act}_i, \mathsf{act}'_i)$ **then**
      return 0
  return 1
else
  return 0

Figure 12: The fourth game Game_4. Differences from Game_3 are highlighted in blue.

Game_5: In this game, the challenger keeps track of what malicious key updates $\mathcal{A}$ performs. This is depicted in Fig. 13. Concretely, we make the following changes.

Initial phase: The challenger additionally initializes lists $\mathsf{vk}_i^{(\mathsf{mal})}$ and $\mathsf{sk}_i^{(\mathsf{mal})}$ for all $i \in [N]$ to $\{\}$, to keep track of the malicious parts of partial verification keys and secret key shares, respectively. After completing the key generation, it sets $(\mathsf{vk}_i, \mathsf{sk}_i)$ to $(\mathsf{vk}_i^{(\mathsf{hon})}, \mathsf{sk}_i^{(\mathsf{hon})})$ for all $i \in [N]$ and idx, which is to keep track of the honest parts.

$\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshUpdate}}$**:** After updating sk_i, it also updates $\mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}] \leftarrow \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}] + \mathsf{rf}_i[\mathsf{idx}]$.

$\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}$**:** It first derives the malicious component of vk_i by $\mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}] \leftarrow \mathsf{vk}_i[\mathsf{idx}] - \mathsf{vk}_i^{(\mathsf{hon})}[\mathsf{idx}]$ and that of sk_i by $\mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}] \leftarrow \mathsf{sk}_i[\mathsf{idx}] - \mathsf{sk}_i^{(\mathsf{hon})}[\mathsf{idx}]$ for all idx and $i \in \mathsf{CS}$ where vk_i and sk_i are given as a part of a query. After generating shares of $\mathbf{0}$, it appropriately updates vk_i using $\mathsf{vk}_i^{(\mathsf{hon})}$, $\mathsf{vk}_i^{(\mathsf{mal})}$ and those shares, and after that, it updates $\mathsf{sk}_i^{(\mathsf{hon})}$ by $\mathsf{sk}_i^{(\mathsf{hon})}[\mathsf{idx}] \leftarrow \mathsf{sk}_i^{(\mathsf{hon})}[\mathsf{idx}] + \mathsf{rf}_i[\mathsf{idx}]$.

$\mathsf{O}_{\mathsf{TS}.\mathsf{Sign}}$**:** When it computes $\mathbf{z}_i$, it uses $\mathsf{sk}_i^{(\mathsf{hon})}$ and $\mathsf{sk}_i^{(\mathsf{mal})}$ instead of sk_i.

By construction, the challenger updates lists for honest signers so that $\mathsf{vk}_i[\mathsf{idx}] = \mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}] + \mathsf{vk}_i^{(\mathsf{hon})}[\mathsf{idx}]$ and $\mathsf{sk}_i[\mathsf{idx}] = \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}] + \mathsf{sk}_i^{(\mathsf{hon})}[\mathsf{idx}]$ throughout the entire game. Also, it updates them for corrupted signers so that the same two conditions hold during the key refresh. Thus, the modifications do not affect the view of $\mathcal{A}$. Therefore we have

$$\varepsilon_4 = \varepsilon_5.$$

Game_6: In this game, we generate shares using simulators defined in Theorem 4.1 during the key generation and refresh and computes $\mathbf{z}_i$ using simulated shares. This game is depicted in Fig. 14. Specifically, the modifications are as follows:

Initial phase: The challenger first prepares hints on the secret $(\mathbf{s}, \mathbf{e})$, runs $\mathsf{SimShares}$ using them, and obtains lists Shr' and b. After that, for all i in $[N]$ and idx such that $b[i][\mathsf{idx}] = 0$, it computes $\mathsf{vk}_i[\mathsf{idx}]$ by $2 \cdot (\mathbf{A} \cdot \mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}})$ and sets $\mathbf{s}_{i,\mathsf{idx}}$ to $\mathsf{sk}_i[\mathsf{idx}]$. For all act, it finds i such that $b[i][\mathsf{idx}] = 1$ and computes $\mathsf{vk}_i[\mathsf{idx}]$ by $\mathbf{t} - \sum_{j \in \mathsf{act} \setminus \{i\}} \mathsf{vk}_j[\mathsf{idx}_j]$. It no longer uses $\mathsf{sk}_i^{(\mathsf{hon})}$ since Shr' keeps the simulated secret shares.

$\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}$**:** It computes $\mathsf{sk}^{(\mathsf{mal})} \leftarrow \mathsf{sk}_i[\mathsf{idx}] - \mathbf{s}_{i,\mathsf{idx}}$ using the simulated secret shares Shr' instead of $\mathsf{sk}_i^{(\mathsf{hon})}$ where $\mathbf{s}_{i,\mathsf{idx}}$ is stored in $\mathsf{Shr}'[i][\mathsf{idx}]$. Instead of honestly generating shares of $\mathbf{0}$ using the sharing algorithm $\mathsf{VandShare}$, it updates Shr' and b using $\mathsf{SimRefresh}$. Then Shr' contains simulated shares $(\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$ for the next epoch. It also updates vk_i using Shr' and b instead of $\mathsf{vk}_i^{(\mathsf{hon})}$. If $b[i][\mathsf{idx}] = 0$, it computes $\mathsf{vk}_i[\mathsf{idx}] \leftarrow 2 \cdot (\mathbf{A} \cdot \mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}}) + \mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}]$, otherwise, it computes $\mathsf{vk}_i[\mathsf{idx}] \leftarrow \mathbf{t} - 2 \cdot (\mathbf{A} \cdot \mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}}) + \mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}]$. Finally, it updates sk_i for corrupted signers in the next epoch by $\mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}]$.

$\mathsf{O}_{\mathsf{TS}.\mathsf{Sign}}$**:** It computes $\mathbf{z}_i$ using $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$ as follows:

$$\mathbf{z}_i = \begin{cases} 2c \cdot (\mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{Idx}[i]]) + \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b} & \text{if } b[i][\mathsf{idx}] = 0 \\ 2c \cdot (\mathbf{s} - \mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{Idx}[i]]) + \sum_{b \in [\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b} & \text{if } b[i][\mathsf{idx}] = 1 \end{cases}.$$

Below we check that the modification does not change the view of $\mathcal{A}$ relying on Theorem 4.1. We first focus on the initial phase. Properties (a) and (d) ensure that the distribution of $\mathsf{vk}_i[\mathsf{idx}]$ for i and idx such that $b[i][\mathsf{idx}] = 0$ and sk_i for corrupted signers remains unchanged. Also, due to the property (b), there is only one honest signer i such that $b[i][\mathsf{Idx}[i]] = 1$ for each act where $\mathsf{Idx} \leftarrow \mathsf{VandRecover}([N], \mathsf{act}, \varepsilon, \mathsf{Idx})$. Then $\mathsf{vk}_i[\mathsf{Idx}[i]]$ is uniquely determined from vk and $\mathsf{vk}_j[\mathsf{Idx}[j]]$ for $j \in \mathsf{act} \setminus \{i\}$. Since vk is generated as in the previous game, the distribution of $\mathsf{vk}_i[\mathsf{Idx}[i]]$ for i and idx such that $b[i][\mathsf{idx}] = 1$ remains the same. For

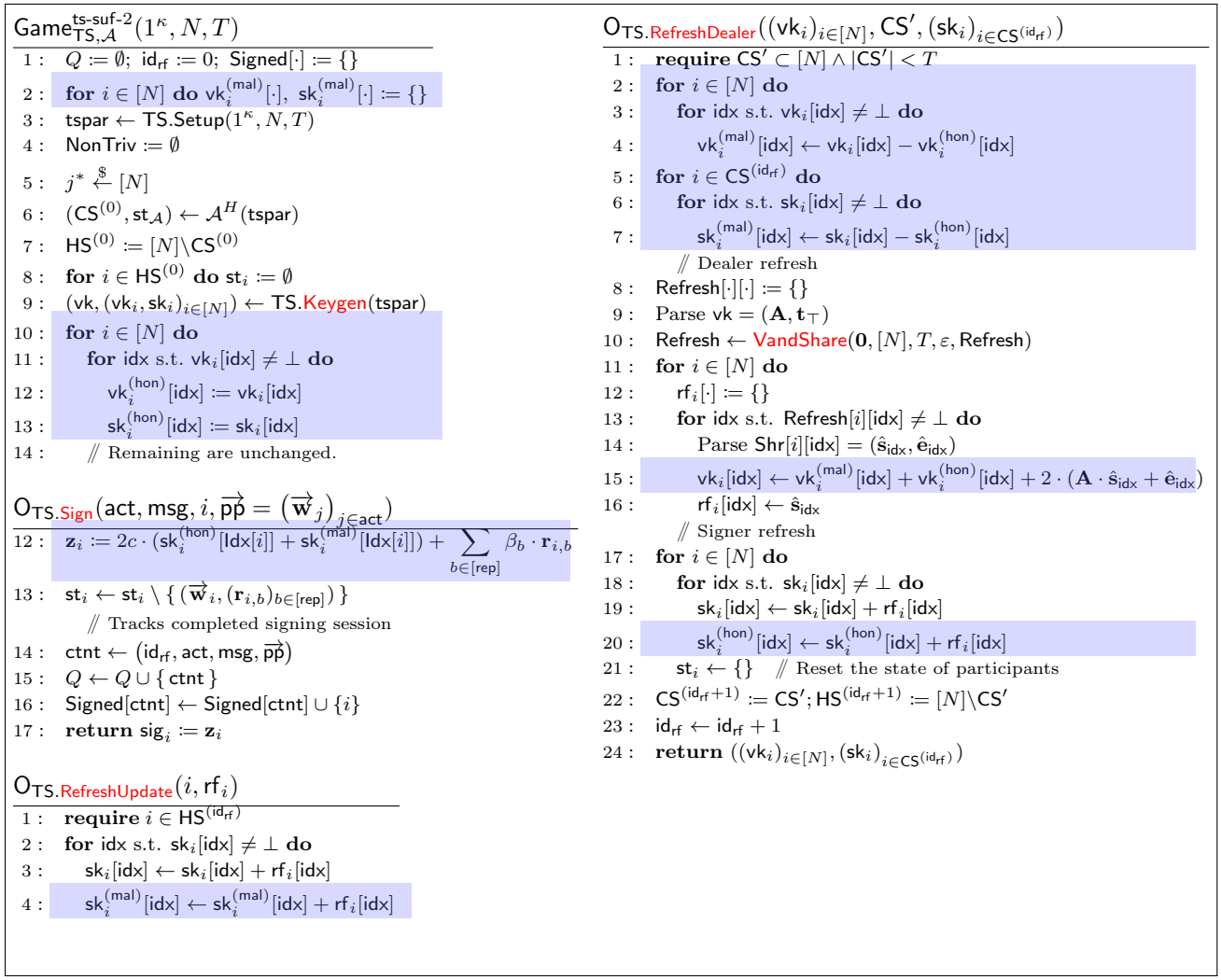


Figure 13: The fifth game Game_5. Differences from Game_4 are highlighted in blue.

$\mathsf{O}_{\mathsf{TS.RefreshDealer}}$, following the properties (a) and (d), it is easy to check the distribution of $\mathsf{sk}_i^{(\mathsf{mal})}$ and sk_i for corrupted signers and vk_i for all signers is identical to that in the previous game. Finally, for $\mathsf{O}_{\mathsf{TS.Sign}}$, $\mathbf{z}_i$ is identically distributed as in the previous game thanks to the property (d).

Combining all arguments, we conclude that $\mathcal{A}$'s view remains identical. Thus we have

$$\varepsilon_5 = \varepsilon_6.$$

By Theorem 6.5, which we will show later, we can construct an adversary $\mathcal{B}_2$ that solves the AOM-MISIS problem from $\mathcal{A}$ against Game_6 such that

$$\varepsilon_6 \leqslant \sqrt{Q_h \cdot \left(\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{B}_2}(\kappa) + \frac{2}{|\mathcal{C}|} + \frac{Q_{\mathsf{Sign}} + Q^2_{H_\beta}}{|\mathbb{T}|^{\mathsf{rep}-1}}\right)}$$

where $Q_h := Q_{H_c} + 2Q_{H_\beta} + 2Q_{\mathsf{Sign}} + 1$ and $\mathsf{Time}(\mathcal{B}_2) \approx 2 \cdot \mathsf{Time}(\mathcal{A})$. Collecting all bounds, we can complete the proof. □

$\mathsf{Game}^{\text{ts-suf-2}}_{\mathsf{TS},\mathcal{A}}(1^\kappa, N, T)$

// Key generation
$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}$
$(\mathbf{s},\mathbf{e}) \leftarrow \mathcal{D}_{\mathbf{s}}^{\ell} \times \mathcal{D}_{\mathbf{s}}^{k}$
$\mathbf{t} := 2\cdot(\mathbf{A}\cdot\mathbf{s}+\mathbf{e})$
$\mathbf{t}_\top := \lfloor \mathbf{t} \rceil_{\nu_{\mathbf{t}}}$
$\mathsf{vk} := (\mathbf{A}, \mathbf{t}_\top)$
for $i \in [B_{\mathsf{shr}}(N,T)]$ **do**
  $(\mathbf{s}'_i, \mathbf{e}'_i) \leftarrow \mathcal{D}_{\mathbf{s}}^{\ell} \times \mathcal{D}_{\mathbf{s}}^{k}$
  $h_i \leftarrow (\mathbf{s},\mathbf{e}) + (\mathbf{s}'_i, \mathbf{e}'_i)$
$(\mathsf{Shr}', b) \leftarrow \mathsf{SimShares}(T, \mathsf{CS}^{(0)}, j^*, (h_i)_{i\in[B_{\mathsf{shr}}(N,T)]})$
for $i \in [N]$ **do**
  for idx s.t. $b[i][\mathsf{idx}] = 0$ **do**
    Parse $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$
    $\mathsf{vk}_i[\mathsf{idx}] \leftarrow 2\cdot(\mathbf{A}\cdot\mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}})$
    if $i \in \mathsf{CS}^{(0)}$ **then**
      $\mathsf{sk}_i[\mathsf{idx}] \leftarrow \mathbf{s}_{i,\mathsf{idx}}$ // $b[i][\mathsf{idx}] = 0$ holds for all idx and $i \in \mathsf{CS}^{(0)}$
for act $\subseteq [N]$ **do**
  $\mathsf{Idx} \leftarrow \mathsf{VandRecover}([N], \mathsf{act}, \varepsilon, \mathsf{Idx})$
  Let i be $i \in \mathsf{act}$ s.t. $b[i][\mathsf{Idx}[i]] = 1$
  $\mathsf{vk}_i[\mathsf{Idx}[i]] \leftarrow \mathbf{t} - \sum_{j\in\mathsf{act}\setminus\{i\}} \mathsf{vk}_j[\mathsf{Idx}[j]]$
for $i \in [N]$ **do**
  for idx s.t. $\mathsf{vk}_i[\mathsf{idx}] \neq \perp$ **do**
    $\mathsf{vk}_i^{(\mathsf{hon})}[\mathsf{idx}] := \mathsf{vk}_i[\mathsf{idx}]$
// Remaining are unchanged.

$\mathsf{O}_{\mathsf{TS}.\mathsf{Sign}}(\mathsf{act}, \mathsf{msg}, i, \overrightarrow{\mathsf{pp}} = (\overrightarrow{\mathbf{w}}_j)_{j\in\mathsf{act}})$

Parse $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$
if $b[i][\mathsf{idx}] = 0$ **then**
  $\mathbf{z}_i := 2c\cdot(\mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{Idx}[i]]) + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$
else // $b[i][\mathsf{idx}] = 1$
  $\mathbf{z}_i := 2c\cdot(\mathbf{s} - \mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{Idx}[i]]) + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\overrightarrow{\mathbf{w}}_i, (\mathbf{r}_{i,b})_{b\in[\mathsf{rep}]})\}$
// Tracks completed signing session
$\mathsf{ctnt} \leftarrow (\mathsf{id}_{\mathsf{rf}}, \mathsf{act}, \mathsf{msg}, \overrightarrow{\mathsf{pp}})$
$Q \leftarrow Q \cup \{\mathsf{ctnt}\}$
$\mathsf{Signed}[\mathsf{ctnt}] \leftarrow \mathsf{Signed}[\mathsf{ctnt}] \cup \{i\}$
return $\mathsf{sig}_i := \mathbf{z}_i$

$\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}((\mathsf{vk}_i)_{i\in[N]}, \mathsf{CS}', (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}})}})$

require $\mathsf{CS}' \subset [N] \wedge |\mathsf{CS}'| < T$
for $i \in [N]$ **do**
  for idx s.t. $\mathsf{vk}_i[\mathsf{idx}] \neq \perp$ **do**
    $\mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}] \leftarrow \mathsf{vk}_i[\mathsf{idx}] - \mathsf{vk}_i^{(\mathsf{hon})}[\mathsf{idx}]$
for $i \in \mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}})}$ **do**
  for idx s.t. $\mathsf{sk}_i[\mathsf{idx}] \neq \perp$ **do**
    Parse $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$
    $\mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}] \leftarrow \mathsf{sk}_i[\mathsf{idx}] - \mathbf{s}_{i,\mathsf{idx}}$ // $b[i][\mathsf{idx}] = 0$
// Dealer refresh
$\mathsf{Refresh}[\cdot][\cdot] := \{\}$
Parse $\mathsf{vk} = (\mathbf{A}, \mathbf{t}_\top)$
for $i \in [B_{\mathsf{rf}}(N,T)]$ **do**
  $(\mathbf{s}'_i, \mathbf{e}'_i) \leftarrow \mathcal{D}_{\mathbf{s}}^{\ell} \times \mathcal{D}_{\mathbf{s}}^{k}$
  $h_i \leftarrow (\mathbf{s},\mathbf{e}) + (\mathbf{s}'_i, \mathbf{e}'_i)$
$(\mathsf{Shr}', b) \leftarrow \mathsf{SimRefresh}(T, \mathsf{CS}', j^*, (h_i)_{i\in[B_{\mathsf{rf}}(N,T)]}, \mathsf{Shr}', b)$
for $i \in [N]$ **do** // Updates partial verification keys
  for idx s.t. $\mathsf{sk}'[i][\mathsf{idx}] \neq \perp$ **do**
    Parse $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$
    if $b[i][\mathsf{idx}] = 0$ **then**
      $\mathsf{vk}_i[\mathsf{idx}] \leftarrow 2\cdot(\mathbf{A}\cdot\mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}}) + \mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}]$
    else
      $\mathsf{vk}_i[\mathsf{idx}] \leftarrow \mathbf{t} - 2\cdot(\mathbf{A}\cdot\mathbf{s}_{i,\mathsf{idx}} + \mathbf{e}_{i,\mathsf{idx}}) + \mathsf{vk}_i^{(\mathsf{mal})}[\mathsf{idx}]$
for $i \in \mathsf{CS}'$ **do** // Updates revealed secret shares
  for idx s.t. $\mathsf{Shr}'[i][\mathsf{idx}] \neq \perp$ **do**
    Parse $\mathsf{Shr}'[i][\mathsf{idx}] = (\mathbf{s}_{i,\mathsf{idx}}, \mathbf{e}_{i,\mathsf{idx}})$
    $\mathsf{sk}_i[\mathsf{idx}] \leftarrow \mathbf{s}_{i,\mathsf{idx}} + \mathsf{sk}_i^{(\mathsf{mal})}[\mathsf{idx}]$
for $i \in [N]$ **do**
  $\mathsf{st}_i \leftarrow \{\}$ // Reset the state of participants
$\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}}+1)} := \mathsf{CS}'; \mathsf{HS}^{(\mathsf{id}_{\mathsf{rf}}+1)} := [N]\setminus\mathsf{CS}'$
$\mathsf{id}_{\mathsf{rf}} \leftarrow \mathsf{id}_{\mathsf{rf}} + 1$
return $((\mathsf{vk}_i)_{i\in[N]}, (\mathsf{sk}_i)_{i\in\mathsf{CS}^{(\mathsf{id}_{\mathsf{rf}})}})$

Figure 14: The sixth game Game_6. Differences from Game_5 are highlighted in blue.

Lemma 6.5. *If there exists an adversary $\mathcal{A}$ who wins Game_6 with advantage ε_6, then we can construct an adversary $\mathcal{B}_2$ against the* AOM-MISIS *problem such that*

$$\varepsilon_6 \leqslant \sqrt{Q_h \cdot \left(\mathsf{Adv}^{\text{AOM-MISIS}}_{\mathcal{B}_2}(\kappa) + \frac{2}{|\mathcal{C}|} + \frac{Q_{\mathsf{Sign}} + Q_{H_\beta}^2}{|\mathbb{T}|^{\mathsf{rep}-1}}\right)}$$

where $Q_h := Q_{H_c} + 2Q_{H_\beta} + 2Q_{\mathsf{Sign}} + 1$ *and* $\mathsf{Time}(\mathcal{B}_2) \approx 2\cdot\mathsf{Time}(\mathcal{A})$.

Proof. We construct an adversary $\mathcal{B}_2$ that solves the AOM-MISIS problem using an adversary against Game_6.

$\mathcal{B}_2$ receives an AOM-MISIS challenge $(\mathbf{A}, (\mathbf{t}_i)_{i\in[Q]})$ for $Q = B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N,T) + \mathsf{rep} \cdot Q_{\mathsf{Sign}} + 1$ and has access to the PI oracle O_{PI}. Roughly speaking, $\mathcal{B}_2$ simulates Game_6 using the oracle access to O_{PI}, obtains two valid forgeries satisfying the winning condition and derives a solution of AOM-MISIS from them. We formally prove this in the following three steps: (1) we explain a wrapper algorithm $\mathcal{B}'$ and invoke the forking lemma Theorem 2.12, (2) describe the necessary conditions to extract a solution, and (3) show how $\mathcal{B}_2$ computes a solution.

Defining $\mathcal{B}'$ to invoke the Forking Lemma. Before defining the wrapper algorithm $\mathcal{B}'$ to be rewound, we appropriately define IG, $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$, and $\mathcal{H}$ for the forking lemma. IG generates $(\mathbf{A}, (\mathbf{t}_i)_{i\in[Q]})$ as in $\mathsf{Game}_{\mathcal{A}}^{\text{AOM-MISIS}}(1^\kappa)$ and outputs $\mathsf{par} = (\mathbf{A}, (\mathbf{t}_i)_{i\in[Q]})$ and $\overline{\mathsf{par}} = (\mathbf{s}_i)_{i\in[Q]}$. Also, $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$ responds as O_{PI}. Moreover, we define $\mathcal{H} \coloneqq \mathcal{C} \times \{1\} \times \mathbb{T}^{\mathsf{rep}-1}$.

We now define $\mathcal{B}'$. This algorithm is given $\mathsf{par} = (\mathbf{A}, (\mathbf{t}_i)_{i\in[Q]})$ and $\overrightarrow{h}$ as input, where $\overrightarrow{h} \in \mathcal{H}^{Q_h}$ and $Q_h = Q_{H_c} + 2Q_{H_\beta} + 2Q_{\mathsf{Sign}} + 1$. For the sake of clarity, we partition $(\mathbf{t}_i)_{i\in[Q]}$ into three parts: the first element $\mathbf{t}^*$ for the challenge verification key, the next $Q_1 = B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N,T)$ elements $(\mathbf{t}'_i)_{i\in[Q_1]}$ for key generation and refresh, and the final $Q_2 = \mathsf{rep} \cdot Q_{\mathsf{Sign}}$ elements $(\mathbf{t}''_i)_{i\in[Q_2]}$ for signing. Also, we introduce the following notation. For a query vector $\mathbf{b}$ to $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$, we denote the components corresponding to $\mathbf{t}^*$, $\mathbf{t}'_i$, and $\mathbf{t}''_i$ by b^*, b'_i, and b''_i, respectively. Namely, $\mathbf{b} = (b^*, b'_1, \ldots, b'_{Q_1}, b''_1, \ldots, b''_{Q_2})$. For a response $\mathbf{s}' \in \mathcal{R}_q^{k+\ell}$ from $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$, we often split it into the first k components $\mathbf{s}'_1 \in \mathcal{R}_q^k$ and the remaining ℓ components $\mathbf{s}'_2 \in \mathcal{R}_q^\ell$ and write $\mathbf{s}' = (\mathbf{s}'_1, \mathbf{s}'_2)$.

$\mathcal{B}'$ internally runs $\mathcal{A}$ by behaving as the challenger in Game_6, except for the following modifications:

Initial phase: Instead of honestly generating $(\mathbf{A}, \mathbf{t})$, it uses $\mathbf{A}$ given as input and computes $\mathbf{t} = 2 \cdot \mathbf{t}^*$. Also, it generates each hint h_i by using the i-th instance $\mathbf{t}'_i$ and accessing $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$. Concretely, it makes a query vector $\mathbf{b}$ such that $b^* = b_i = 1$ and other components are 0 and uses a response $(\mathbf{s}'_1, \mathbf{s}'_2)$ as h_i. Moreover, it additionally initializes counters ctr and ctr_s to 1.

$\mathsf{O}_{\mathsf{TS}.\mathsf{Preprocess}}$**:** Instead of honestly generating $(\mathbf{w}_{i,b})_{b\in[\mathsf{rep}]}$, it uses $(\mathbf{t}''_\kappa)_{\kappa\in[(\mathsf{ctr}_s-1)\cdot\mathsf{rep}+1, \mathsf{ctr}_s\cdot\mathsf{rep}]}$. Finally, it stores ctr_s in st_i and increments ctr_s.

$\mathsf{O}_{\mathsf{TS}.\mathsf{Sign}}$**:** It computes $\mathbf{z}_i$ by accessing $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$. Let J be a value of the counter ctr_s stored in st_i. Specifically, if $b[i][\mathsf{idx}] = 0$, it queries $\mathbf{b}$ such that $b''_{(\mathsf{ctr}_s-1)\cdot\mathsf{rep}+b} = \beta_b$ for $b \in [\mathsf{rep}]$ and other components are set to 0 to $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$ and uses $\mathbf{s}'_1$ in place of $\sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$. In the case of $b[i][\mathsf{idx}] = 1$, it makes a query $\mathbf{b}$ such that $b^* = 2c$ and $b''_{(\mathsf{ctr}_s-1)\cdot\mathsf{rep}+b} = \beta_b$ for $b \in [\mathsf{rep}]$ to $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$ and uses $\mathbf{s}'_1$ instead of $2c\cdot\mathbf{s} + \sum_{b\in[\mathsf{rep}]} \beta_b \cdot \mathbf{r}_{i,b}$.

$\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}$**:** For the j-th refresh query, it queries $\mathbf{b}$ such that $b^* = b_{(j-1)\cdot B_{\mathsf{rf}}(N,T)+i} = 1$ and other components are 0 to $\mathsf{O}(\overline{\mathsf{par}}, \cdot)$ and uses $(\mathbf{s}'_1, \mathbf{s}'_2)$ as h_i.

Random oracles: When it defines a cell in the table T_{H_c} (resp. T_{H_β}), it uses c (resp. $(\beta_b)_{b\in\mathsf{rep}}$) in ctr and increments ctr. Moreover, for H_c, it additionally stores ctr in $\mathsf{ChallengeSet}$ before incrementing it.

Final phase: It outputs a non-trivial forgery $\mathsf{sig}_i^* = (c_i, \mathbf{z}_i, \mathbf{h}_i)$ such that $(c_i, \cdot) \notin \mathsf{Triv}$ along with an index I^* where $(c_i, \overline{\mathsf{ctnt}}_i, I^*) \in \mathsf{ChallengeSet}$.

Due to our parameter setting, it is easy to check that $\mathcal{B}'$ perfectly simulates Game_6. Thus we have

$$\mathsf{acc} = \Pr\left[(I^*, \mathsf{sig}^*) \xleftarrow{\$} \mathcal{B}'^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par}, \overrightarrow{h}) : I^* \leqslant 1\right] = \varepsilon_6.$$

We now invoke the forking lemma. $\mathcal{B}$ runs a forking algorithm $\mathsf{Fork}_{\mathcal{B}'}^{\mathsf{O}(\overline{\mathsf{par}}, \cdot)}(\mathsf{par})$. Specifically, it sets the challenge instance $(\mathbf{A}, (\mathbf{t}_i)_{i\in[Q]})$ to par, forwards queries from $\mathcal{B}'$ to O_{PI} and returns the corresponding responses. We note that $\mathcal{B}$ stores all queries to compute the solution after terminating the forking algorithm. By Theorem 2.12, it obtains two non-trivial forgeries $\mathsf{sig}_1^* = (c_1, \mathbf{z}_1, \mathbf{h}_1)$ and $\mathsf{sig}_2^* = (c_2, \mathbf{z}_2, \mathbf{h}_2)$ such that $\lfloor \mathbf{A} \cdot \mathbf{z}_1 - 2^{\nu_{\mathbf{t}}} \cdot c_1 \cdot \mathbf{t}_\top \rceil_{\nu_{\mathbf{w}}} + \mathbf{h}_1 = \lfloor \mathbf{A} \cdot \mathbf{z}_2 - 2^{\nu_{\mathbf{t}}} \cdot c_2 \cdot \mathbf{t}_\top \rceil_{\nu_{\mathbf{w}}} + \mathbf{h}_2 \bmod q_{\nu_{\mathbf{w}}}$ with probability

$$\mathsf{frk} \geqslant \mathsf{acc} \cdot \left(\frac{\mathsf{acc}}{Q_h} - \frac{1}{|\mathcal{C} \times \{1\} \times \mathbb{T}^{\mathsf{rep}-1}|}\right) \geqslant \frac{\varepsilon_6^2}{Q_h} - \frac{1}{|\mathcal{C}| \cdot |\mathbb{T}|^{\mathsf{rep}-1}}.$$

Before moving to the next step, we observe the structure of the PI queries. Due to the construction of $\mathcal{B}'$, there are only the following two types of queries, involving $\mathbf{t}'_i$ or $\mathbf{t}''_i$.

Type 1: Queries involving $\mathbf{t}'_i$. For each $i \in [Q_1]$, there exists at most one query such that $b^* = b'_i = 1$ and all other components are 0.

Type 2: Queries involving $\left(\mathbf{t}''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}$. For each $i \in [Q_{\mathsf{Sign}}]$, there are at most two queries of the following form:

(1). $\left(b''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]} \in \{1\} \times \mathbb{T}^{\mathsf{rep}-1}$ and all other components are 0; or

(2). $b^* \in 2\mathcal{C}$ and the other components satisfy condition (1).

Conditions to extract a solution. Here we define two bad events under which $\mathcal{B}$ fails to compute the solution. The first event E_1 is the event $c_1 = c_2$. Since h_{I^*} and h'_{I^*} are uniformly chosen from $\mathcal{H}$, we have $\Pr[E_1] = 1/|\mathcal{C}|$.

The second event E_2 is the event where there exist two distinct PI oracle queries $\mathbf{b}_1$ and $\mathbf{b}_2$ such that $b^*_1 \neq b^*_2$ but all other components are identical. This event can occur only for Type 2-(2) queries. In particular, this occurs only if $\mathcal{B}$ uses the same $(\beta_b)_{b\in[\mathsf{rep}]}$ with distinct c for the same $(\mathbf{w}_{i,b})_{b\in[\mathsf{rep}]}$ before and after rewinding. Notice that these signing sessions must belong to the same epoch because honest signers reset their state in $\mathsf{O}_{\mathsf{TS}.\mathsf{RefreshDealer}}$. Below we show that $\Pr[E_2]$ is negligible by following a similar argument to [49, Proof of Lemma 6.2].

Let I_β and I'_β be the values of the counter ctr_h when $(\beta_b)_{b\in[\mathsf{rep}]}$ are defined before and after rewinding, respectively. Recall that I^* is the rewinding point, i.e., c_1 and c_2 are defined when $\mathsf{ctr}_h = I^*$. Note that $I_\beta, I'_\beta \neq I^*$. If $I^* < I_\beta$ or $I^* < I'_\beta$, then at least one of them is sampled uniformly after rewinding. Thus $\Pr[E_2]$ is at most $Q_{\mathsf{Sign}}/|\mathbb{T}|^{\mathsf{rep}-1}$ in this case. In the case of $I_\beta, I'_\beta < I^*$ and $I_\beta \neq I'_\beta$, E_2 cannot occur unless there is a collision in H_β. Thus this event occurs with probability at most $Q^2_{H_\beta}/|\mathbb{T}|^{\mathsf{rep}-1}$.

We now assume $I_\beta = I'_\beta < I^*$, which is the remaining case. Let I_c and I'_c be the values of ctr_h when the challenge c used in the corresponding signing query is defined before and after rewinding, respectively. Due to the modification in Game_1, $I_c = I'_c$ and $I_c \leqslant I^*$ hold. If $I_c < I^*$, E_2 does not occur since c is not changed. To complete this part, we show that there is no Type 2-(2) query involving c_i for $i \in \{1, 2\}$ in the case $I_c = I^*$, where c_1 and c_2 are the challenges in the non-trivial forgeries. Due to the abort condition added in Game_4, each challenge c_i has no corresponding $\overline{\mathsf{ctnt}}$ in $\mathsf{ChallengeSet}$, or signer j^* has not completed the signing session under $\overline{\mathsf{ctnt}}^*$. If the first case happens, $I_c < I_\beta, I'_\beta$ hold since the input of H_c that yields c_i is computed from the input of H_β outputting $(\beta_b)_{b\in[\mathsf{rep}]}$ corresponding to I_β. This contradicts the assumption that $I_\beta = I'_\beta < I^*$. Thus the first case never occurs. In the second case, since j^* has not completed the signing session both before and after rewinding, $\mathcal{B}'$ makes no Type 2-(2) query due to the property (c) in Theorem 4.1. Consequently, E_2 cannot occur in the case of $I_\beta = I'_\beta < I^*$.

As we will show in the final step, $\mathcal{B}$ can compute the solution of the $\mathsf{AOM\text{-}MISIS}$ problem if the bad events E_1 and E_2 do not occur. Therefore, combining all bounds above and applying the union bound, we obtain

$$\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{B}_2}(\kappa) \geqslant \mathsf{frk} - \Pr[E_1 \vee E_2] \geqslant \frac{\varepsilon_6^2}{Q_h} - \frac{2}{|\mathcal{C}|} - \frac{Q_{\mathsf{Sign}} + Q^2_{H_\beta}}{|\mathbb{T}|^{\mathsf{rep}-1}}$$

and $\mathsf{Time}(\mathcal{B}_2) \approx 2 \cdot \mathsf{Time}(\mathcal{A})$. We can rewrite the above inequality as follows:

$$\varepsilon_6 \leqslant \sqrt{Q_h \cdot \left(\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{B}_2}(\kappa) + \frac{2}{|\mathcal{C}|} + \frac{Q_{\mathsf{Sign}} + Q^2_{H_\beta}}{|\mathbb{T}|^{\mathsf{rep}-1}}\right)}.$$

How to extract a solution of $\mathsf{AOM\text{-}MISIS}$. Finally, we explain how $\mathcal{B}$ extracts a solution to the $\mathsf{AOM\text{-}MISIS}$ problem after $\mathsf{Fork}^{\mathsf{O}(\overline{\mathsf{par}},\cdot)}_{\mathcal{B}'}(\mathsf{par})$ successfully terminates. We assume that the bad events do not occur. Then,

following similar arguments to those we made in Game_3, we have

$$\mathbf{A}\cdot(\mathbf{z}_1-\mathbf{z}_2)+2^{\nu_{\mathbf{w}}}\cdot(\mathbf{h}_1-\mathbf{h}_2)-\boldsymbol{\delta}_1+\boldsymbol{\delta}_2=(c_1-c_2)\cdot 2^{\nu_{\mathbf{t}}}\cdot\mathbf{t}_\top \pmod q$$

where $\boldsymbol{\delta}_1,\boldsymbol{\delta}_2\in\mathcal{R}_q^k$ such that $\|\boldsymbol{\delta}\|_\infty\leqslant 2\cdot(2^{\nu_{\mathbf{w}}-1}+\max(0,q_\perp))$. Due to Theorem 2.6, there exists a vector $\boldsymbol{\delta}'$ such that $\boldsymbol{\delta}'=2\cdot\mathbf{t}^*-2^{\nu_{\mathbf{t}}}\cdot\mathbf{t}_\top$ and $\|\boldsymbol{\delta}'\|_\infty\leqslant 2^{\nu_{\mathbf{t}}-1}$. Substituting this into the above equation yields

$$\mathbf{A}\cdot(\mathbf{z}_1-\mathbf{z}_2)+2^{\nu_{\mathbf{w}}}\cdot(\mathbf{h}_1-\mathbf{h}_2)-\boldsymbol{\delta}+(c_1-c_2)\cdot\boldsymbol{\delta}'=2(c_1-c_2)\cdot\mathbf{t}^* \pmod q.$$

Thus we set $\mathbf{s}^*=(\mathbf{z}_1-\mathbf{z}_2,2^{\nu_{\mathbf{w}}}\cdot(\mathbf{h}_1-\mathbf{h}_2)-\boldsymbol{\delta}_1+\boldsymbol{\delta}_2+(c_1-c_2)\cdot\boldsymbol{\delta}_3)$ and $\mathbf{b}^{*\top}=(2(c_1-c_2),0,\ldots,0)$. The equation $[\mathbf{t}_1\cdots\mathbf{t}_Q]\cdot\mathbf{b}^{*\top}=[\mathbf{A}\ \mathbf{I}]\cdot\mathbf{s}^*$ obviously holds. Also, due to $\|\boldsymbol{\delta}\|_\infty\leqslant 2\cdot(2^{\nu_{\mathbf{w}}-1}+\max(0,q_\perp))$, $\|\boldsymbol{\delta}'\|_\infty\leqslant 2^{\nu_{\mathbf{t}}-1}$, and the validity of the forgeries, we have

$$\begin{aligned}\|\mathbf{s}^*\|_2&\leqslant\|(\mathbf{z}_1-\mathbf{z}_2,2^{\nu_{\mathbf{w}}}\cdot(\mathbf{h}_1-\mathbf{h}_2))\|_2+\|-\boldsymbol{\delta}+(c_1-c_2)\cdot\boldsymbol{\delta}'\|_2\\&\leqslant 2B_2+(2\cdot(2^{\nu_{\mathbf{w}}-1}+\max(0,q_\perp))+\omega\cdot 2^{\nu_{\mathbf{t}}})\cdot\sqrt{kn}=\beta_{\mathbf{s}}\\\text{and }\ \left\|\mathbf{b}^{*\top}\cdot[\sigma_1,\ldots,\sigma_Q]\right\|_1&=\|2(c_1-c_2)\cdot\sigma_{\mathbf{s}}\|_1\leqslant 4\cdot\omega\cdot\sigma_{\mathbf{s}}=\beta_{\mathbf{b}}.\end{aligned}$$

Thus $\mathbf{s}^*$ and $\mathbf{b}^{*\top}$ satisfy the winning condition.

It remains to define $\mathbf{u}$ such that $\mathbf{b}^{*\top}\cdot\mathbf{u}\neq 0$, $\mathbf{b}\cdot\mathbf{u}=0$ for all PI queries made by $\mathcal{B}$, and $\|(u_1/\sigma_1,\ldots,u_Q/\sigma_Q)\|_2\leqslant\beta_{\mathbf{u}}$. We do this by following similar arguments to [88, Proof of Lemma 11]. In the same manner as the notation of a PI query $\mathbf{b}$, we denote the components of $\mathbf{u}$ corresponding to $\mathbf{t}^*$, $\mathbf{t}'_i$, and $\mathbf{t}''_i$ by u^*, u'_i, and u''_i, respectively. Namely, $\mathbf{u}=(u^*,u'_1,\ldots,u'_{Q_1},u''_1,\ldots,u''_{Q_2})$. Similarly, we partition $(\sigma_1,\ldots,\sigma_Q)$ as $(\sigma^*,\sigma'_1,\ldots,\sigma'_{Q_1},\sigma''_1,\ldots,\sigma''_{Q_2})$.

First, setting $u^*=1$ ensures $\mathbf{b}^{*\top}\cdot\mathbf{u}\neq 0$. For each u'_i, $\mathbf{t}'_i$ is involved in only at most one Type 1 query. Thus we set $u'_i=-1$ if such a query exists, otherwise we set $u'_i=0$ so that, for any Type 1 query $\mathbf{b}_i$ such that $b'_i=1$, $\mathbf{b}_i\cdot\mathbf{u}=b^*_i\cdot u^*+b'_i\cdot u'_i=1-1=0$ holds.

Lastly, we define $\left(u''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}$ for each $i\in[Q_{\mathsf{Sign}}]$. $\left(\mathbf{t}''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}$ is involved in only at most two Type 2 queries. We can set $\left(u''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}=\mathbf{0}$ except for the case where there is at least one Type 2-(2) query since $\mathbf{b}_i\cdot\mathbf{u}=\sum_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}b''_j\cdot u''_j=0$ holds for any Type 2-(1) query $\mathbf{b}_i$. If there is only one Type 2-(2) query, we set $u''_{(i-1)\cdot\mathsf{rep}+1}=-b^*_i$ and the other components to 0. Then $\mathbf{b}_i\cdot\mathbf{u}=b^*_i\cdot u^*+b''_{(i-1)\cdot\mathsf{rep}+1}\cdot u''_{(i-1)\cdot\mathsf{rep}+1}=b^*_i-b^*_i=0$ holds because $b''_{(i-1)\cdot\mathsf{rep}+1}=1$.

We now focus on the remaining case where there are two Type 2 queries including at least one Type 2-(2) query. Let $\mathbf{b}_{i,1}$ be a Type 2-(2) query and $\mathbf{b}_{i,2}$ be another query. When the event E_2 does not occur, there is at least one pair of components such that $b''_{i,1,\alpha_i}\neq b''_{i,2,\alpha_i}$ for $\alpha_i\in[(i-1)\cdot\mathsf{rep}+2,i\cdot\mathsf{rep}]$. We note that $b''_{i,1,\alpha_i},b''_{i,2,\alpha_i}\in\mathbb{T}$. Due to Theorem 2.1, there exists $\gamma:=(b''_{i,1,\alpha_i}-b''_{i,2,\alpha_i})^{-1}$. Then we define $\left(u''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1,i\cdot\mathsf{rep}]}$ so that

$$u''_j=\begin{cases}-b^*_{i,1}+b''_{i,1,\alpha_i}\cdot\gamma\cdot(b^*_{i,1}-b^*_{i,2}) & j=(i-1)\cdot\mathsf{rep}+1\\-\gamma\cdot(b^*_{i,1}-b^*_{i,2}) & j=\alpha_i\\0 & \text{otherwise}\end{cases}.$$

We can check that it satisfies the requirement for both $\mathbf{b}_{i,1}$ and $\mathbf{b}_{i,2}$ by

$$\begin{aligned}\mathbf{b}_{i,1}\cdot\mathbf{u}&=b^*_{i,1}\cdot u^*+b''_{i,1,(i-1)\cdot\mathsf{rep}+1}\cdot u''_{(i-1)\cdot\mathsf{rep}+1}+b''_{i,1,\alpha_i}\cdot u''_{\alpha_i}\\&=b^*_{i,1}+(-b^*_{i,1}+b''_{i,1,\alpha_i}\cdot\gamma\cdot(b^*_{i,1}-b^*_{i,2}))-b''_{i,1,\alpha_i}\cdot\gamma\cdot(b^*_{i,1}-b^*_{i,2})=0\end{aligned}$$

and

$$\begin{aligned}\mathbf{b}_{i,2}\cdot\mathbf{u}&=b^*_{i,2}\cdot u^*+b''_{i,2,(i-1)\cdot\mathsf{rep}+1}\cdot u''_{(i-1)\cdot\mathsf{rep}+1}+b''_{i,2,\alpha_i}\cdot u''_{\alpha_i}\\&=b^*_{i,2}+(-b^*_{i,1}+b''_{i,1,\alpha_i}\cdot\gamma\cdot(b^*_{i,1}-b^*_{i,2}))-b''_{i,2,\alpha_i}\cdot\gamma\cdot(b^*_{i,1}-b^*_{i,2})\end{aligned}$$

$$= -(b^*_{i,1} - b^*_{i,2}) + (b''_{i,1,\alpha_i} - b''_{i,2,\alpha_i}) \cdot (b''_{i,1,\alpha_i} - b''_{i,2,\alpha_i})^{-1} \cdot (b^*_{i,1} - b^*_{i,2}) = 0$$

To complete this proof, we check that the above $\mathbf{u}$ satisfies $\left\|(u^*/\sigma^*, u'_1/\sigma'_1, \ldots, u'_{Q_1}/\sigma'_{Q_1}, u''_1/\sigma''_1, \ldots, u''_{Q_2}/\sigma''_{Q_2})\right\|_2 \leqslant \beta_{\mathbf{u}}$. Notice that this norm is maximized when $\mathcal{B}$ makes Type 1 queries for all $\mathbf{t}'_i$ and two queries including Type 2-(2) for all tuples $\left(\mathbf{t}''_j\right)_{j\in[(i-1)\cdot\mathsf{rep}+1, i\cdot\mathsf{rep}]}$. To this end, we evaluate $\left\|u''_{(i-1)\cdot\mathsf{rep}+1}\right\|_\infty$ and $\left\|u''_{\alpha_i}\right\|_\infty$. Recall that $b^*_{i,1}, b^*_{i,2} \in 2\cdot\mathcal{C}\cup\{0\}$. Thus we can write $b^*_{i,1} = 2\cdot c_{i,1}$ and $b^*_{i,2} = 2\cdot c_{i,2}$ for some $c_{i,1}, c_{i,2} \in \mathcal{C}$. Also, due to Theorem 2.1, we have $\|2\cdot\gamma\|_\infty \leqslant 1$. Applying them and the fact that multiplying an element in $\mathbb{T}$ does not affect the norm, we have $\left\|u''_{(i-1)\cdot\mathsf{rep}+1}\right\|_\infty = \|-2\cdot c_{i,1} + 2\cdot\gamma\cdot(c_{i,1}-c_{i,2})\|_\infty \leqslant 2\omega+2$ and $\left\|u''_{\alpha_i}\right\|_\infty = \|2\cdot\gamma\cdot(c_{i,1}-c_{i,2})\|_\infty \leqslant 2\omega$. Therefore, we obtain

$$\begin{aligned}&\left\|(u^*/\sigma^*, u'_1/\sigma'_1, \ldots, u'_{Q_1}/\sigma'_{Q_1}, u''_1/\sigma''_1, \ldots, u''_{Q_2}/\sigma''_{Q_2})\right\|_2\\ &\leqslant \sqrt{\frac{Q_1+1}{\sigma_{\mathbf{s}}^2} + \frac{(8\omega^2+8\omega+4)\cdot n\cdot Q_{\mathsf{Sign}}}{\sigma_{\mathbf{r}}^2}} = \beta_{\mathbf{u}}.\end{aligned}$$

This completes the proof. □

Remark 6.6. The proof reveals that the shape of the queries allowed in the resulting AOM-MISIS problem are of the following shape:

$$L = \left\{ \begin{bmatrix} \mathbf{1}^\top & c_1^\top & \cdots & c_{Q_{\mathsf{Sign}}}^\top \\ \mathrm{Id}_\tau & & & \\ & B_1 & & \\ & & \ddots & \\ & & & B_{Q_{\mathsf{Sign}}} \end{bmatrix} \subset \mathcal{R}_q^{(\mathsf{rep}Q_{\mathsf{Sign}}+\tau+1)\times(2Q_{\mathsf{Sign}}+\tau)} \;\middle|\; \forall i \in [Q_{\mathsf{Sign}}], (c_i, B_i) \in \mathcal{C}_{\mathsf{TS}} \times B_{\mathsf{TS}}, \right\}.$$

with

$$B_{\mathsf{TS}} = \left\{ \begin{bmatrix} 1 & 1 \\ b_1 & b'_1 \\ b_2 & b'_2 \\ \vdots & \vdots \\ b_{\mathsf{rep}-1} & b'_{\mathsf{rep}-1} \end{bmatrix} \in \mathcal{R}_q^{\mathsf{rep}\times 2} \;\middle|\; \begin{array}{c}\forall i \in [\tau-1], (b_i, b'_i) \in \mathbb{T}^2 \\ \wedge\ b_1 \neq b'_1\end{array} \right\},$$

and $\mathcal{C}_{\mathsf{TS}} = \left\{[c, c']^\top \in \mathcal{R}_q^2 \mid c, c' \in \mathcal{C}\cup\{0\}\right\}$ and setting $\tau = B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}}\cdot B_{\mathsf{rf}}(N,T)$.

However, when examining carefully the solution $\mathbf{u}$ extracted, we can see that it additionally satisfies the conditions $(0,\cdots,0,1,0,\cdots,0)^T u = 0$ where the 1 are at indices $4+\tau+i\mathsf{rep}+j$ for $0 \leqslant i < Q_{\mathsf{Sign}}$ and $0 \leqslant j < \mathsf{rep}-1$. This means that we can actually solve AOM-MISIS for *extended queries* of the shape:

$$L' = \left\{ \begin{bmatrix} \mathbf{1}^\top & d_1^\top & \cdots & d_{Q_{\mathsf{Sign}}}^\top \\ \mathrm{Id}_\tau & & & \\ & B'_1 & & \\ & & \ddots & \\ & & & B'_{Q_{\mathsf{Sign}}} \end{bmatrix} \subset \mathcal{R}_q^{Q\times(Q-1)} \;\middle|\; \forall i \in [Q_{\mathsf{Sign}}], (d_i, B'_i) \in \mathcal{C}'_{\mathsf{TS}} \times B'_{\mathsf{TS}}, \right\}.$$

with

$$B'_{\mathsf{TS}} = \left\{ \begin{bmatrix} 1 & 1 & & \\ b_1 & b'_1 & & \\ b_2 & b'_2 & 1 & \\ \vdots & \vdots & & \ddots \\ b_{\mathsf{rep}-1} & b'_{\mathsf{rep}-1} & & 1 \end{bmatrix} \in \mathcal{R}_q^{\mathsf{rep}\times\mathsf{rep}} \;\middle|\; \begin{array}{c}\forall i \in [\tau-1], (b_i, b'_i) \in \mathbb{T}^2 \\ \wedge\ b_1 \neq b'_1\end{array} \right\},$$

and $\mathcal{C}'_{\mathsf{TS}} = \{[c, c', 0, \cdots, 0]^\top \in \mathcal{R}_q^{\mathsf{rep}} \mid c, c' \in \mathcal{C} \cup \{0\}\}$, as the solution extracted for the queries in L will also be valid for the corresponding queries padded to belong to L'.

But now, the set of possible queries is almost the one used in the AOM-MLWE problem defined in [49] and we can readily rewrite the AOM-MISIS instance to rely on the cryptanalysis of [49, sec 8] to evaluate its hardness, thanks to the very similar shape of the AOM-MLWE/AOM-MISIS games for the queries in L'.

7 Instantiation

We follow the methodology of [49] to derive practical parameters for Hermine.

Unforgeability. Recall that the advantage in the unforgeability game satisfies

$$\mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{Hermine},\mathcal{A}}(1^\kappa, N, T) \leqslant \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}_1}(1^\kappa) + (Q_{H_c} + Q_{H_\beta})^2/|\mathcal{C}| + N \cdot (Q_{H_c} + 2Q_{H_\beta} + 2Q_{\mathsf{Sign}} + 1)^{\frac{1}{2}} \cdot \left(\mathsf{Adv}^{\mathsf{AOM\text{-}MISIS}}_{\mathcal{B}_2}(1^\kappa) + \frac{2}{|\mathcal{C}|} + \frac{Q_{\mathsf{Sign}} + Q_{H_\beta}^2}{|\mathbb{T}|^{\mathsf{rep}-1}}\right)^{1/2},$$

As is standard, we normalize the reduction loss by the total number of random-oracle queries Q_{RO} when evaluating bit security, and define $n_{\text{REAL}} = \log_2\left(\frac{T_{\mathcal{A}}}{\mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{Hermine},\mathcal{A}}(1^\kappa, N, T)}\right)$, where the adversary's running time satisfies $T_{\mathcal{A}} \geq Q_{\mathsf{RO}}$. Our objective is to ensure $n_{\text{REAL}} \geq \kappa + O(1)$. After this normalization, the dominant terms are the MSIS and AOM-MISIS advantages.

As is common in practice-oriented designs (e.g., [82, 52, 14, 49]), we ignore the square-root loss introduced by the forking lemma. As observed in Theorem 6.6, the structure of the admissible queries in AOM-MISIS allows us to reuse the worst-case analysis of [49, Section 8] and reduce security to a single MLWE instance. In this reduction, the secret and error vectors are scaled down by a factor of the order of

$$\sqrt{B_{\mathsf{shr}}(N,T) + Q_{\mathsf{rf}} \cdot B_{\mathsf{rf}}(N,T) + Q_{\mathsf{Sign}}\omega}.$$

We therefore rely on existing cryptanalytic analyses of MLWE. The current standard tool for estimating concrete hardness is the *lattice estimator*[13], introduced in [4]. According to this estimator, the most effective known attacks are the primal uSVP attack of [5] and the dual/hybrid attack of [48]. We additionally apply Ducas' *dimensions-for-free* optimization [44] when using sieve-based BKZ, which yields a small additional security margin.

Identifiable aborts. Using Theorem 6.3, we set the partial verification bound B_2^{ind} as $B_2^{\mathsf{ind}} = 2\omega\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\left(1 + Q_{\mathsf{rf}}\sqrt{\log N}\right) + \alpha\sigma_{\mathbf{r}}\sqrt{n(k+\ell)\mathsf{rep}} + \mathsf{rep}\cdot 2^{\nu'_{\mathbf{w}}-1}\sqrt{nk}$. Honestly generated partial signatures satisfy PartialVerify with overwhelming probability.

Remark 7.1 (Bounding rounding errors). Unconditionally, we have $\|\boldsymbol{\delta}'_i\| \leq \mathsf{rep} \cdot 2^{\nu'_{\mathbf{w}}-1}\sqrt{nk}$. However, a sharper heuristic bound can be obtained. We model each coefficient of $\boldsymbol{\delta}'_i$ as the sum of rep independent uniform variables over $\left[-2^{\nu'_{\mathbf{w}}-1}, 2^{\nu'_{\mathbf{w}}-1}\right]$. The resulting variance is $\mathbb{V} = \frac{\mathsf{rep}\cdot\left((2^{\nu'_{\mathbf{w}}}-1)^2-1\right)}{12} \sim \frac{\mathsf{rep}\cdot 2^{2\nu'_{\mathbf{w}}}}{12}$. Since $\mathbb{E}[\|\boldsymbol{\delta}'_i\|^2] = nk\mathbb{V}$, Hoeffding's inequality implies $\left|\|\boldsymbol{\delta}'_i\|^2 - \mathbb{E}[\|\boldsymbol{\delta}'_i\|^2]\right| = O\left(\sqrt{\kappa\, nk\, 2^{2\nu'_{\mathbf{w}}}\,\mathsf{rep}}\right)$ with overwhelming probability. Consequently, heuristically, $\|\boldsymbol{\delta}'_i\| = (1+o(1))\sqrt{nk}\,2^{\nu'_{\mathbf{w}}}\sqrt{\frac{\mathsf{rep}}{12}}$, which improves over the unconditional bound by a factor $\sqrt{3\mathsf{rep}}$. We therefore set parameters using the tighter bound $B_2^{\mathsf{ind}} = 2\omega\alpha\sigma_{\mathbf{s}}\sqrt{n(k+\ell)}\left(1 + Q_{\mathsf{rf}}\sqrt{\log N}\right) + \alpha\sigma_{\mathbf{r}}\sqrt{n(k+\ell)\mathsf{rep}} + (1+o(1))\sqrt{nk}\,2^{\nu'_{\mathbf{w}}}\sqrt{\frac{\mathsf{rep}}{12}}$.

Wrapping up. The resulting parameter sets are summarized in Table 3. Security levels are expressed as $\log_2$ of the number of ring operations estimated by the lattice estimator [4]. Sizes are given in bytes. Across all parameter sets, we fix $n = 512$, $\omega = \min\left\{\omega' \in \mathbb{Z} \mid \binom{n}{\omega'}2^{\omega'} \geq 2^\kappa\right\}$, $\mathsf{rep} = \min\left\{\mathsf{rep}' \in \mathbb{Z} \mid (2n)^{\mathsf{rep}'} \geq 2^\kappa\right\}$, and restrict $N \leq 16$ and $T \leq 8$.

[13] https://github.com/malb/lattice-estimator

Table 3: Parameter sets. Security metrics are $\log_2$ of the number of ring operations, as estimated by the lattice estimator [4]. Sizes are in bytes.

Targets			Scheme parameters									Security			Sizes			
κ	Q_{Sign}	Q_{rf}	$\lceil \log q \rceil$	k	ℓ	ω	$u_{\mathbf{s}}$	$u_{\mathbf{r}}$	$\nu_{\mathbf{t}}$	$\nu_{\mathbf{w}}$	$\nu_{\mathbf{w}'}$	κ_{pvk}	κ_{vk}	κ_{sig}	$\lvert\mathsf{vk}\rvert$	$\lvert\mathsf{sig}\rvert$	$\lvert\mathsf{pp}_i\rvert$	$\lvert\mathsf{sig}_i\rvert$
128	2^{60}	2^{10}	48	4	4	19	13	36	37	40	32	133	129	124	2,816	11,296	53,248	19,968
192	2^{64}	2^{14}	48	6	6	31	13	36	37	40	32	202	195	211	4,224	16,944	122,880	29,952
256	2^{64}	2^{14}	48	7	8	44	13	35	37	40	32	271	268	261	4,928	21,952	186,368	36,480
128	2^{28}	2^{5}	30	3	3	19	7	17	20	23	16	138	126	162	1,920	4,832	34,944	7,680
192	2^{27}	2^{4}	30	4	4	31	7	17	20	22	15	208	193	245	2,560	6,448	76,800	10,240
256	2^{27}	2^{3}	30	5	5	44	7	17	20	22	14	273	261	325	3,200	8,064	133,120	12,800

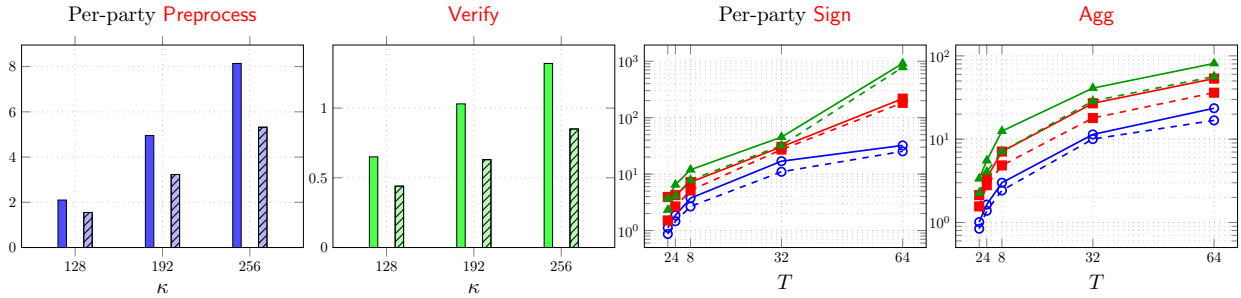


Figure 15: Performance costs of Hermine. Left: per-party preprocessing and verification times (independent of (N,T)), grouped by security level κ with solid bars for $\lceil \log q \rceil = 48$ and patterned bars for $\lceil \log q \rceil = 30$. Right: per-party signing and combine times as functions of T for $(N,T) \in \{(4,2),(8,4),(16,8),(32,32),(64,64)\}$, with line styles distinguishing modulus sizes (— $\lceil \log q \rceil = 48$, -- $\lceil \log q \rceil = 30$) and colors indicating κ (○ 128, ■ 192, ▲ 256). Times in ms (log-scale for Sign and Combine).

8 Implementation and Evaluation

We implement Hermine in Go, for all parameter sets defined in Table 3. We leverage `Lattigo` (v5) [1] for optimized arithmetic in power-of-two cyclotomic rings. Our implementation uses `Lattigo`'s NTT-friendly ring operations for polynomial arithmetic modulo a single RNS modulus $\mathcal{R}_q$, including forward/inverse NTT and Montgomery representations for fast coefficient-wise multiplication.

Performance evaluation.

We evaluate performance locally using Go's benchmarking functionality on a consumer-grade MacBook M3 with 24 GB RAM. We measure (i) key generation, (ii) per-party offline preprocessing, (iii) per-party online sign, and (iv) end-to-end verification (combine and verify): the results are shown in Table 5 and, visually, in Figure 15. To study scaling, we benchmark combinations of threshold parameters $T \in \{2,4,8,16,32,64\}, N \in \{2,4,8,16,32,64\}$, subject to $T \leq N$, and we run each benchmark for each parameter set (e.g., ($\kappa \in \{128,192,256\}$) in our two parameter families). To reduce run-to-run variance and improve comparability across (N,T) and security levels, we run all benchmarks in a single-OS-thread configuration by setting `GOMAXPROCS=1`. This avoids scheduler- and contention-induced noise from parallel execution (e.g., goroutine scheduling, GC assistance, and runtime work stealing) and makes the measured wall-clock time more representative of the algorithmic cost, rather than of multicore availability. We further use `-benchtime=100x`, which executes each benchmark exactly 100 times and reports the aggregated timing. Using a fixed iteration count avoids the adaptive behavior of time-based benchmarking, where fast parameter sets would be executed many more times than slower ones. This is particularly important in our setting, since

the dominant costs (NTT-based polynomial arithmetic and large matrix–vector multiplications) scale with the threshold and security parameters: fixing the iteration count ensures that the reported per-operation costs are directly comparable across all configurations while averaging out transient runtime and system noise. The detailed performance metrics in Section C, Table 5 validate the theoretical efficiency of Hermine. We also provide a comparison with state-of-the-art schemes Ringtail [20] and EKT [49] in Table 4. These results highlight two key takeaways.

First, the absence of masking in the signature generation translates into significant performance gains over prior works. Comparing with the masked threshold signature scheme EKT, we achieve consistently lower latencies, with savings of 20% to 40% for larger thresholds. Gains are even more pronounced when compared to Ringtail, though the comparison is less direct as its signature generation follows a different approach. This highlights an additional benefit of our approach beyond the support for non-interactive IA; by avoiding masking, we remove a significant overhead of the signing process present in prior lattice-based threshold signatures.

Second, our framework supports a flexible range of parameters, allowing for fine-grained trade-offs between number of signatures supported, signature size, and computation. Our parameter sets supporting a smaller number of signatures $Q_{\mathsf{Sign}} \approx 2^{27}$ ($\lceil \log q \rceil = 30$) yield a reduction of 15%–40% in computation time while producing more compact signatures compared to our settings supporting a large number of signatures ($\lceil \log q \rceil = 48$).

Finally, the signing costs exhibit a predictable linear increase with the threshold, explained by the aggregation of the contributions from the other parties.

Acknowledgement. This presentation is based on results obtained from a project, *JPNP24003*, commissioned by the New Energy and Industrial Technology Development Organization (NEDO).

Rafael del Pino, Thomas Espitau, Guilhem Niot and Thomas Prest are supported by the French National Research Agency (ANR), through the project *RELATE* (reference: ANR-25- CE39-4214-01).

Giacomo Borin is supported by *CryptonIs, SNSF Consolidator Grant 213766*, (https://data.snf.ch/grants/grant/213766).

References

[1] Lattigo v6. Online: https://github.com/tuneinsight/lattigo, August 2024. EPFL-LDS, Tune Insight SA.

[2] Shweta Agrawal, Damien Stehlé, and Anshu Yadav. Round-optimal lattice-based threshold signatures, revisited. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *ICALP 2022*, volume 229 of *LIPIcs*, pages 8:1–8:20. Schloss Dagstuhl, July 2022.

[3] Gorjan Alagic, Lily Chen, Quynh Dang, Dustin Moody, Christos Pappas, Angela Robinson, and Daniel Smith-Tone. Status report on the third round of the NIST post-quantum cryptography standardization process. NIST IR 8413-upd1, National Institute of Standards and Technology, 2022.

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

[5] Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 327–343. USENIX Association, August 2016.

[6] Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Berlin, Heidelberg, May / June 2006.

[7] Benny Applebaum, Oded Nir, and Benny Pinkas. How to recover a secret with $o(n)$ additions. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 236–262. Springer, Cham, August 2023.

[8] Michele Battagliola, Giacomo Borin, Giovanni Di Crescenzo, Alessio Meneghetti, and Edoardo Persichetti. Enhancing threshold group action signature schemes: Adaptive security and scalability improvements. *IACR Cryptol. ePrint Arch.*, page 85, 2025.

[9] Michele Battagliola, Giacomo Borin, Giovanni Di Crescenzo, Alessio Meneghetti, and Edoardo Persichetti. Enhancing threshold group action signature schemes: Adaptive security and scalability improvements. In Ruben Niederhagen and Markku-Juhani O. Saarinen, editors, *Post-Quantum Cryptography - 16th International Workshop, PQCrypto 2025, Part I*, pages 129–161. Springer, Cham, April 2025.

[10] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Cham, August 2022.

[11] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

[12] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 27–35. Springer, New York, August 1990.

[13] Fabrice Benhamouda, Jan Camenisch, Stephan Krenn, Vadim Lyubashevsky, and Gregory Neven. Better zero-knowledge proofs for lattice encryption and their application to group signatures. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part I*, volume 8873 of *LNCS*, pages 551–572. Springer, Berlin, Heidelberg, December 2014.

[14] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 227–247. Springer, Cham, December 2019.

[15] Alexander Bienstock, Leo de Castro, Daniel Escudero, Antigoni Polychroniadou, and Akira Takahashi. Efficient, scalable threshold ml-dsa signatures: An mpc approach. *To Appear in USENIX 2026. Available at https://eprint.iacr.org/2025/1163*, 2026.

[16] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003.

[17] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Cham, August 2018.

[18] Dan Boneh, Aditi Partap, and Lior Rotem. Proactive refresh for accountable threshold signatures. In Jeremy Clark and Elaine Shi, editors, *FC 2024, Part II*, volume 14745 of *LNCS*, pages 140–159. Springer, Cham, March 2024.

[19] Ravi B. Boppana. Amplification of probabilistic Boolean formulas. In *26th FOCS*, pages 20–29. IEEE Computer Society Press, October 1985.

[20] Cecilia Boschini, Darya Kaviani, Russell W. F. Lai, Giulio Malavolta, Akira Takahashi, and Mehdi Tibouchi. Ringtail: Practical two-round threshold signatures from learning with errors. In Marina Blanton, William Enck, and Cristina Nita-Rotaru, editors, *2025 IEEE Symposium on Security and Privacy*, pages 149–164. IEEE Computer Society Press, May 2025.

[21] Katharina Boudgoust, Oleksandra Lapiha, Rafaël del Pino, and Thomas Prest. IND-CCA lattice threshold KEM under 30 KiB. Cryptology ePrint Archive, Paper 2026/021, 2026.

[22] LTAN Brandão and Michael Davidson. Notes on threshold eddsa/schnorr signatures. National Institute of Standards and Technology, 2022. https://doi.org/10.6028/NIST.IR.8214B.ipd.

[23] Luís T. A. N. Brandão and Rene Peralta. Nist first call for multi-party threshold schemes. NIST Internal Report (IR) 8214C, National Institute of Standards and Technology, January 2023. Initial Public Draft.

[24] Lorenz Breidenbach, Christian Cachin, Benedict Chan, Brendan Magauran, Ari Juels, and Sergey Nazarov. Off-chain reporting (ocr3) protocol. Technical report, Chainlink Labs, August 2023. Version 3.0.

[25] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. Asynchronous verifiable secret sharing and proactive cryptosystems. In Vijayalakshmi Atluri, editor, *ACM CCS 2002*, pages 88–97. ACM Press, November 2002.

[26] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

[27] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA revisited: Online/offline extensions, identifiable aborts, proactivity and adaptive security. Cryptology ePrint Archive, Report 2021/291, 2021.

[28] Sofia Celi, Rafael del Pino, Thomas Espitau, Guilhem Niot, and Thomas Prest. Efficient threshold ml-dsa. *To Appear in USENIX 2026. Available at https://eprint.iacr.org/2026/013*, 2026.

[29] Chainflip Labs. FROST signature scheme – chainflip protocol documentation. https://docs.chainflip.io/protocol/frost-signature-scheme, 2025. Accessed: February 6, 2026.

[30] Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Partially non-interactive two-round lattice-based threshold signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part IV*, volume 15487 of *LNCS*, pages 268–302. Springer, Singapore, December 2024.

[31] Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from iterative shamir secret sharing. Cryptology ePrint Archive, Report 2023/545, 2023.

[32] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023.

[33] Annalisa Cimatti, Francesco De Sclavis, Giuseppe Galano, Sara Giammusso, Michela Iezzi, Antonio Muci, Matteo Nardelli, and Marco Pedicini. Dynamic-FROST: Schnorr threshold signatures with a flexible committee. Cryptology ePrint Archive, Report 2024/896, 2024.

[34] Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The Flexible Round-Optimized Schnorr Threshold (FROST) protocol for Two-Round Schnorr signatures. RFC 9591, June 2024. Accessed: February 6, 2026.

[35] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 75–90. Springer, Berlin, Heidelberg, April 2006.

[36] Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[37] Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.

[38] Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 409–444. Springer, Cham, August 2024.

[39] Rafaël del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarinen. Raccoon. Technical report, National Institute of Standards and Technology, 2023. Available at https://csrc.nist.gov/Projects/pqc-dig-sig/round-1-additional-signatures.

[40] Yvo Desmedt, Giovanni Di Crescenzo, and Mike Burmester. Multiplicative non-abelian sharing schemes and their application to threshold cryptography. In Josef Pieprzyk and Reihaneh Safavi-Naini, editors, *ASIACRYPT'94*, volume 917 of *LNCS*, pages 21–32. Springer, Berlin, Heidelberg, November / December 1995.

[41] Julien Devevey, Benoît Libert, Khoa Nguyen, Thomas Peters, and Moti Yung. Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 659–690. Springer, Cham, May 2021.

[42] DFINITY Foundation. Threshold signatures | Internet Computer. https://docs.internetcomputer.org/references/t-sigs-how-it-works, December 2025. Accessed: February 5, 2026.

[43] Justin Drake. Today marks an inflection in the Ethereum Foundation's long-term quantum strategy (@drakefjustin). https://x.com/drakefjustin/status/2014791629408784816, February 2026. Accessed: February 5, 2026.

[44] Léo Ducas. Shortest vector from lattice sieving: A few dimensions for free. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 125–145. Springer, Cham, April / May 2018.

[45] Ali El Kaafarani and Shuichi Katsumata. Attribute-based signatures for unbounded circuits in the ROM and efficient instantiations from lattices. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part II*, volume 10770 of *LNCS*, pages 89–119. Springer, Cham, March 2018.

[46] Karim Eldefrawy, Tancrède Lepoint, and Antonin Leroux. Communication-efficient proactive secret sharing for dynamic groups with dishonest majorities. In Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi, editors, *ACNS 2020, Part I*, volume 12146 of *LNCS*, pages 3–23. Springer, Cham, October 2020.

[47] Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. Plover: Masking-friendly hash-and-sign lattice signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 316–345. Springer, Cham, May 2024.

[48] Thomas Espitau, Antoine Joux, and Natalia Kharchenko. On a dual/hybrid approach to small secret LWE - A dual/enumeration technique for learning with errors and application to security estimates of FHE schemes. In Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran, editors, *INDOCRYPT 2020*, volume 12578 of *LNCS*, pages 440–462. Springer, Cham, December 2020.

[49] Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 387–424. Springer, Cham, August 2024.

[50] Thomas Espitau, Guilhem Niot, and Thomas Prest. Flood and submerse: Distributed key generation and robust threshold signature from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 425–458. Springer, Cham, August 2024.

[51] Brett Hemenway Falk, Daniel Noble, and Tal Rabin. Proactive secret sharing with constant communication. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part II*, volume 14370 of *LNCS*, pages 337–373. Springer, Cham, November / December 2023.

[52] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 1651–1662. ACM Press, October 2016.

[53] Yair Frankel, Peter Gemmell, Philip D. MacKenzie, and Moti Yung. Proactive RSA. In Burton S. Kaliski, Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 440–454. Springer, Berlin, Heidelberg, August 1997.

[54] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In Kwok-Yan Lam, Eiji Okamoto, and Chaoping Xing, editors, *ASIACRYPT'99*, volume 1716 of *LNCS*, pages 180–194. Springer, Berlin, Heidelberg, November 1999.

[55] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptive security for the additive-sharing based proactive RSA. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 240–263. Springer, Berlin, Heidelberg, February 2001.

[56] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy*, pages 3034–3052. IEEE Computer Society Press, May 2024.

[57] Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Robustness of the learning with errors assumption. In *Innovations in Computer Science - ICS 2010, Tsinghua University, Beijing, China, January 5-7, 2010. Proceedings*, pages 230–240. Tsinghua University Press, 2010.

[58] Kamil Doruk Gur, Patrick Hough, Jonathan Katz, Caroline Sandsbråten, and Tjerand Silde. Olingo: Threshold lattice signatures with DKG and identifiable abort. Cryptology ePrint Archive, Report 2025/1789, 2025.

[59] Kamil Doruk Gür, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice-based signatures from threshold homomorphic encryption. In Markku-Juhani Saarinen and Daniel Smith-Tone, editors, *Post-Quantum Cryptography - 15th International Workshop, PQCrypto 2024, Part II*, pages 266–300. Springer, Cham, June 2024.

[60] Amir Herzberg, Markus Jakobsson, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public key and signature systems. In Richard Graveman, Philippe A. Janson, Clifford Neuman, and Li Gong, editors, *ACM CCS 97*, pages 100–110. ACM Press, April 1997.

[61] Amir Herzberg, Stanislaw Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer, Berlin, Heidelberg, August 1995.

[62] Stanislaw Jarecki and Josh Olsen. Proactive RSA with non-interactive signing. In Gene Tsudik, editor, *FC 2008*, volume 5143 of *LNCS*, pages 215–230. Springer, Berlin, Heidelberg, January 2008.

[63] Stanislaw Jarecki and Nitesh Saxena. Further simplifications in proactive RSA signatures. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 510–528. Springer, Berlin, Heidelberg, February 2005.

[64] Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024.

[65] Chelsea Komlo. Updates on the FROST NIST submission. In *MPTS 2026: NIST Workshop on Multi-Party Threshold Schemes*, January 2026. Joint work with Elizabeth Crites, Conrado Gouvea, Jack Grigg, Ian Goldberg, Jonathan Katz, Mary Maller, Simon Rastikian, Stefano Tessaro, Nikita Sorokovikov, Denis Varlakov, and Chenzhi Zhu.

[66] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.

[67] Yashvanth Kondi, Bernardo Magri, Claudio Orlandi, and Omer Shlomovits. Refresh when you wake up: Proactive threshold wallets with offline devices. In *2021 IEEE Symposium on Security and Privacy*, pages 608–625. IEEE Computer Society Press, May 2021.

[68] Brian Koziel, S. Dov Gordon, and Craig Gentry. Fast two-party threshold ECDSA with proactive security. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1567–1580. ACM Press, October 2024.

[69] Benoît Libert, Damien Stehlé, and Radu Titiu. Adaptively secure distributed PRFs from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 391–421. Springer, Cham, November 2018.

[70] Vadim Lyubashevsky. Fiat-Shamir with aborts: Applications to lattice and factoring-based signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 598–616. Springer, Berlin, Heidelberg, December 2009.

[71] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Berlin, Heidelberg, April 2012.

[72] Sai Krishna Deepak Maram, Fan Zhang, Lun Wang, Andrew Low, Yupeng Zhang, Ari Juels, and Dawn Song. CHURP: Dynamic-committee proactive secret sharing. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2369–2386. ACM Press, November 2019.

[73] Sela Navot and Stefano Tessaro. One-more unforgeability for multi - and threshold signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 429–462. Springer, Singapore, December 2024.

[74] NIST. Call for additional digital signature schemes for the post-quantum cryptography standardization process. https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/call-for-proposals-dig-sig-sept-2022.pdf, 2022.

[75] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *10th ACM PODC*, pages 51–59. ACM, August 1991.

[76] Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 80–97. Springer, Berlin, Heidelberg, August 2010.

[77] Penumbra Labs. Threshold custody and signing in Penumbra. https://guide.penumbra.zone/usage/pcli/wallet/threshold, 2025. Accessed: February 6, 2026.

[78] Rafaël Del Pino, Shuichi Katsumata, Guilhem Niot, Michael Reichle, and Kaoru Takemure. Unmasking TRaccoon: A lattice-based threshold signature with an efficient identifiable abort protocol. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 423–456. Springer, Cham, August 2025.

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

[80] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 89–104. Springer, Berlin, Heidelberg, August 1998.

[81] Tim Ruffing, Viktoria Ronge, Elliott Jin, Jonas Schneider-Bensch, and Dominique Schröder. ROAST: Robust asynchronous Schnorr threshold signatures. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2551–2564. ACM Press, November 2022.

[82] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, New York, August 1990.

[83] David A. Schultz, Barbara Liskov, and Moses Liskov. Mobile proactive secret sharing. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM PODC*, page 458. ACM, August 2008.

[84] Solv Protocol Team. Solv drives FROST threshold signatures for Bitcoin mainnet execution. https://solv.finance/blog/frost-bitcoin-mainnet, January 2026. Accessed: February 6, 2026.

[85] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. COBRA: Dynamic proactive secret sharing for confidential BFT services. In *2022 IEEE Symposium on Security and Privacy*, pages 1335–1353. IEEE Computer Society Press, May 2022.

[86] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. Long live the honey badger: Robust asynchronous DPSS and its applications. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 5413–5430. USENIX Association, August 2023.

[87] Zcash Foundation. The state of FROST for Zcash. https://zfnd.org/the-state-of-frost-for-zcash/, May 2025. Accessed: February 5, 2026.

[88] Chenzhi Zhu and Stefano Tessaro. The algebraic one-more MISIS problem and applications to threshold signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part I*, volume 16000 of *LNCS*, pages 548–581. Springer, Cham, August 2025.

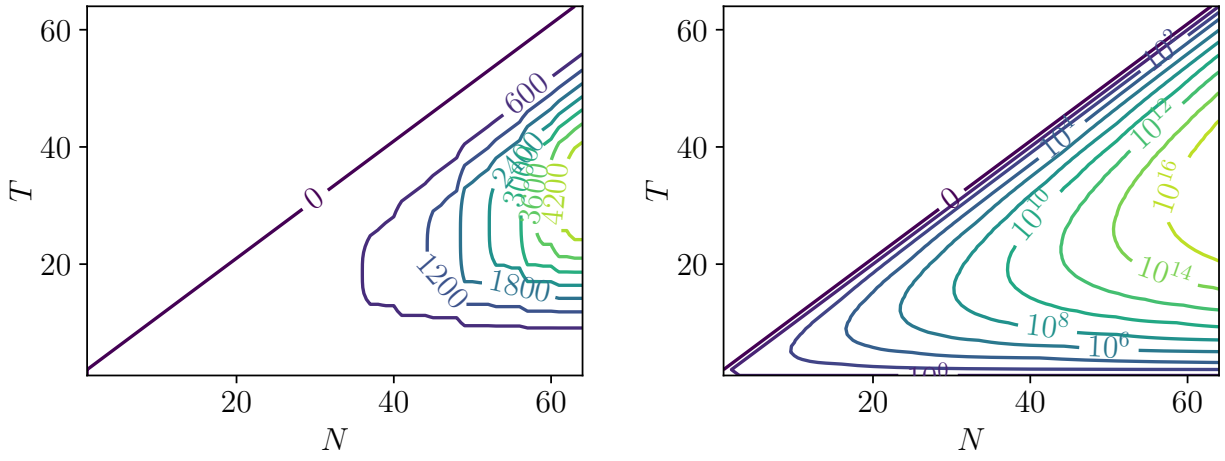
A Comparison with Other Secret Sharing

In [8], the authors provide estimates on the total and per party number of shares, showing a superpolynomial growth in T and $\log N$. However, Figure 16 shows that the scheme remains practical for all concrete choices of $T \leqslant N \leqslant 64$, with the individual number of shares staying below 4500. Note that only one share per user is used during the distributed signing since all coefficients are zero but one. Experimental results from our implementation corroborate these claims.

Other schemes with similar features have been proposed; however, for the range we consider, Vandermonde secret sharing is the most efficient. In the range of interest ($N \leqslant 64$), it outperforms:

- The Replicated Secret Sharing approach, in which the number of shares grow as $\binom{N}{T}$. See Figure 16 for the concrete number of shares for $T \leqslant N \leqslant 64$.
- The generic construction from [12] to Boppana's monotone formula [19] (see also [35]), used in [30], in which the number of shares grow as $O(\min(T, N-T)^{4.3} N \log N)$. While this provides the only asymptotically polynomial growth (in T and $\log N$), for the concrete range of $T \leqslant N \leqslant 64$, it is less efficient than both Vandermonde and Replicated secret sharing due to the $\approx N^{5.3}$ dependency.

Another potential alternative is provided by "near-threshold" or "ramp" secret sharing schemes, which are sharings for which the privacy threshold T_p is different from the reconstruction threshold T_r. For a large number of parties, these schemes can provide better efficiency than the perfect threshold schemes. In [7], Applebaum et al. show how to instantiate a ramp secret achieving any gap T_r/T_p arbitrarily close to 1. Moreover, the sharing and reconstruction can be performed with $O(N)$ additions. However, this does not seem compatible with lattice-based constructions (see [30, Section 3.3] for more detail).



(a) Vandermonde

(b) Replicated

Figure 16: Contour plots of the maximum number of shares per party, as a function of N and T (undefined for $T > N$). It can be checked that for $N \approx 20$, the number of shares can reach $\approx 10^6$ for replicated secret sharing.

B Optional algorithms of Vandermonde Short Secret Sharing

In Figure 17 we present optional algorithms for the Vandermonde short secret sharing as used in VandShare.

The idea of $\mathsf{VandShare}_N$ (with $\mathsf{VandRecover}_N$) is to take a shortcut when $T = N$ and directly perform an N-out-of-N sharing. This neither increases or decreases the number of required shares, but allows to increase the number of shares sampled uniformly, that are independent of the secret, and can be compressed to the seed used for their generation.

Instead $\mathsf{VandShare}_2$ (with $\mathsf{VandRecover}_2$) is a 2-out-of-N sharing that slightly reduces the total number of shares and relies on the simple idea of applying a binary sharing to the secret. The core idea is that, given two different users $u, v \in [N]$[14] their binary representations need to differ in at least one position. Thus, for each $i = 0, \ldots, n-1$, where n is the bit length of N, we can perform a 2-out-of-2 sharing of secret $x = x_0 + x_1$ and then send x_0 to all the users with i-th bit equal to 0 and x_1 to all the users with i-th bit equal to 1.

We leave as an open problem to find more efficient sharings for specific values of T and N.

$\mathsf{VandShare}_N(x, \mathcal{P}, \mathsf{idx}, \mathsf{Shr})$

1 : $N \leftarrow |\mathcal{P}|$
2 : $\mathsf{idx} \leftarrow \mathsf{idx} \,\|\, \texttt{":N:"}$
3 : **for** $i = 1, \ldots, N-1$ **do**
4 : $\quad x_i \leftarrow \chi$
5 : $x_N \leftarrow x - \sum_{i \in [N-1]} x_i$
6 : **for** $i \in [N]$ **do**
7 : $\quad \mathsf{Shr}[\mathcal{P}[i]][\mathsf{idx}] \leftarrow x_i$
8 : **return** Shr

$\mathsf{VandShare}_2(x, \mathcal{P}, \mathsf{idx}, \mathsf{Shr})$

1 : $N \leftarrow |\mathcal{P}|$
2 : $n \leftarrow \lceil \log(N) \rceil$
3 : $\mathsf{idx} \leftarrow \mathsf{idx} \,\|\, \texttt{":B:"}$
4 : **for** $i \in [n]$ **do**
5 : $\quad x_0 \leftarrow \chi; x_1 \leftarrow x - x_0$
6 : $\quad$ **for** $u \in [N]$ **do**
7 : $\quad\quad bit \leftarrow (u >> (i-1)) \,\%2$ // i-th bit of u
8 : $\quad\quad \mathsf{Shr}[\mathcal{P}[u]][\mathsf{idx}\|i] \leftarrow x_{bit}$
9 : **return** Shr

$\mathsf{VandRecover}_N(\mathcal{P}, \mathsf{act}, \mathsf{idx}, \mathsf{Idx})$

1 : $\mathsf{idx} \leftarrow \mathsf{idx} \,\|\, \texttt{":N:"}$
2 : **for** $i \in \mathcal{P}$ **do**
3 : $\quad \mathsf{Idx}[i] \leftarrow \mathsf{idx}$
4 : **return** Idx

$\mathsf{VandRecover}_2(\mathcal{P}, \mathsf{act}, \mathsf{idx}, \mathsf{Idx})$

1 : $N \leftarrow |\mathcal{P}|$
2 : $\mathsf{idx} \leftarrow \mathsf{idx} \,\|\, \texttt{":B:"}$
3 : $bit_1 \leftarrow \mathsf{act}[1] \,\%2$; $bit_2 \leftarrow \mathsf{act}[2] \,\%2$
4 : $i \leftarrow 0$
5 : **while** $bit_1 = bit_2$ **do**
6 : $\quad i \leftarrow i + 1$
7 : $\quad bit_1 \leftarrow (\mathsf{act}[1] >> i) \,\%2$
8 : $\quad bit_2 \leftarrow (\mathsf{act}[2] >> i) \,\%2$
9 : $\mathsf{Idx}[\mathsf{act}[1]] \leftarrow \mathsf{idx}\|(i+1)$
10 : $\mathsf{Idx}[\mathsf{act}[2]] \leftarrow \mathsf{idx}\|(i+1)$
11 : **return** Idx

Figure 17: Optional algorithms for $T \in \{2, N\}$ applying to the short Vandermonde secret sharing. % denotes the modulo operation, and >> denotes the right bitwise shift.

C Detailed Performance Data

This section includes the detailed Table 5 of performance data for Hermine, as plotted in Fig. 15. We also present a comparison table in Table 4: we compare with Ringtail and EKT at $\kappa = 128$. Note that we implement a variant of EKT that captures its performance characteristics, but we do not explicitly use Shamir secret sharing with Lagrange reconstruction. Instead, we omit the Shamir/Lagrange layer, which would only introduce a small overhead. As a result, the measured performance closely reflects that of full EKT, while highlighting the cost of using masks.

[14] We are identifying the users with their index in $\mathcal{P}$.

Table 4: Comparison at $\kappa = 128$. For each (N, T), we report (i) Sign & Preprocess (Preproc.) per party and (ii) Verify for all parties in ms. We highlight in green the best case.

Scheme	**Metric**	$(\mathbf{N}, \mathbf{T})$					
		$(2,2)$	$(4,4)$	$(8,8)$	$(16,16)$	$(32,32)$	$(64,64)$
Hermine: $\lceil \log q \rceil = 48$	Sign & Preproc.	3.26	3.98	5.43	8.81	19.04	34.56
	Verify				0.61		
Hermine: $\lceil \log q \rceil = 30$	Sign & Preproc.	2.38	2.96	4.05	6.13	13.11	27.47
	Verify				0.41		
Ringtail	Sign & Preproc.	54.06	59.09	72.88	101.19	165.19	287.79
	Verify				0.61		
EKT	Sign & Preproc.	3.54	4.69	6.75	10.98	32.30	44.67
	Verify				0.71		

$\lceil \log q \rceil$	κ	(N,T)	Per-party times		Setup & verification		
			Preprocess	Sign	Keygen	Agg	Verify
48	**128**	(4,2)	2.10	1.12	1.59	1.01	0.65
		(8,4)		1.86	5.55	1.64	
		(16,8)		3.75	48.52	3.00	
		(32,32)		16.92	14.22	11.40	
		(64,64)		32.22	14.23	23.51	
	192	(4,2)	4.95	3.92	2.50	2.13	1.03
		(8,4)		4.24	8.67	3.43	
		(16,8)		7.20	75.50	7.10	
		(32,32)		30.38	10.43	26.94	
		(64,64)		217.16	29.59	53.37	
	256	(4,2)	8.14	3.69	3.31	3.36	1.32
		(8,4)		6.49	11.23	5.54	
		(16,8)		11.96	96.23	12.50	
		(32,32)		45.22	12.17	40.98	
		(64,64)		913.30	29.59	81.14	
30	**128**	(4,2)	1.55	0.87	1.28	0.84	0.44
		(8,4)		1.46	4.51	1.38	
		(16,8)		2.67	34.71	2.42	
		(32,32)		10.98	6.41	10.03	
		(64,64)		25.32	11.62	16.81	
	192	(4,2)	3.23	1.51	1.67	1.56	0.63
		(8,4)		2.67	5.71	2.80	
		(16,8)		5.13	49.92	4.84	
		(32,32)		27.19	6.48	17.94	
		(64,64)		183.09	12.99	36.13	
	256	(4,2)	5.32	2.32	2.06	2.27	0.85
		(8,4)		4.09	7.07	4.06	
		(16,8)		7.85	62.93	7.12	
		(32,32)		32.12	7.76	28.73	
		(64,64)		781.20	17.91	56.55	

Table 5: Per-party preprocessing and signing time, plus key generation, combine and verification time for Hermine. Times are reported in milliseconds (ms). All costs are computed on a MacOS M3 machine and grouped by κ.