Adaptively Secure 5 Round Threshold Signatures from $\mathsf{MLWE}/\mathsf{MSIS}$ and DL with Rewinding

Shuichi Katsumata[1,2], Michael Reichle[3], Kaoru Takemure*[1,2]

[1]PQShield

{shuichi.katsumata, kaoru.takemure}@pqshield.com

[2]AIST

[3]ETH Zürich

michael.reichle@inf.ethz.ch

June 26, 2024

Abstract

T-out-of-N threshold signatures have recently seen a renewed interest, with various types now available, each offering different tradeoffs. However, one property that has remained elusive is *adaptive* security. When we target thresholdizing existing efficient signatures schemes based on the Fiat-Shamir paradigm such as Schnorr, the elusive nature becomes clear. This class of signature schemes typically rely on the forking lemma to prove unforgeability. That is, an adversary is *rewound and run twice* within the security game. Such a proof is at odds with adaptive security, as the reduction must be ready to answer $2(T-1)$ secret key shares in total, implying that it can reconstruct the full secret key. Indeed, prior works either assumed strong idealized models such as the algebraic group model (AGM) or modified the underlying signature scheme so as not to rely on rewinding based proofs.

In this work, we propose a new proof technique to construct adaptively secure threshold signatures for existing rewinding-based Fiat-Shamir signatures. As a result, we obtain the following:

1. The first adaptively secure 5 round lattice-based threshold signature under the MLWE and MSIS assumptions in the ROM. The resulting signature is a standard signature of $\mathsf{Raccoon}$, a lattice-based signature scheme by del Pino et al., submitted to the additional NIST call for proposals.
2. The first adaptively secure 5 round threshold signature under the DL assumption in the ROM. The resulting signature is a standard Schnorr signature. To the best of our knowledge, this is the first adaptively secure threshold signature based on DL even assuming stronger models like AGM.

Our work is inspired by the recent statically secure lattice-based 3 round threshold signature by del Pino et al. (Eurocrypt 2024) based on $\mathsf{Raccoon}$. While they relied on so-called one-time additive masks to solve lattice specific issues, we notice that these masks can also be a useful tool to achieve adaptive security. At a very high level, we use these masks throughout the signing protocol to carefully control the information the adversary can learn from the signing transcripts. Intuitively, this allows the reduction to return a total of $2(T-1)$ *randomly sampled* secret key shares to the adversary consistently and without being detected, resolving the above paradoxical situation. Lastly, by allowing the parties to maintain a simple state, we can compress our 5 round schemes into 4 rounds.

*Most of this work was done while this author was a PhD student at The University of Electro-Communications, Japan.

Contents

1 Introduction **4**
1.1 Our Contribution . . . 5
1.2 Technical Overview . . . 7
1.3 Related Works . . . 11

2 Preliminary **13**
2.1 Notations . . . 13
2.2 Threshold Signatures . . . 13
2.3 Linear Secret Sharing . . . 15
2.4 Lattices, Gaussians, and Rounding . . . 16
2.5 Hardness Assumptions . . . 16

3 Construction of Our 3-Round Threshold Raccoon **17**
3.1 Parameters and Preparations . . . 18
3.2 Construction . . . 18
3.3 Correctness . . . 19

4 Selective Security of Our 3-Round Threshold Raccoon **21**
4.1 Proof Overview . . . 21

5 Construction of Our 5-Round Threshold Raccoon **25**
5.1 Parameters and Preparations . . . 25
5.2 Construction . . . 25
5.3 Correctness . . . 26
5.4 Our 4-Round Raccoon Threshold Signature . . . 27

6 Adaptive Security of Our 5 Round Threshold Raccoon **27**
6.1 Intuition . . . 29
6.2 Proof Overview . . . 31

7 Construction of Our 5-Round Threshold Schnorr **35**
7.1 Preparations . . . 36
7.2 Construction . . . 36
7.3 Security and Correctness . . . 36
7.4 Our 4-Round Schnorr Threshold Signature . . . 40

A Omitted Preliminaries **47**
A.1 Security Notions for Threshold Signature . . . 47
A.2 Rounding and Norms Modulo q . . . 48
A.3 Hardness of Lattice-Related Problems . . . 49
A.4 Hardness of DL-Related Problems . . . 50
A.5 Forking Lemmas . . . 50

B Details of Our 4-Round Threshold Raccoon **51**
B.1 Construction . . . 51
B.2 Security . . . 51

C Details of Our 4-Round Threshold Schnorr **53**
C.1 Construction . . . 53
C.2 Security . . . 53

D Candidate Parameters for Our Threshold Raccoon **53**

E Formal Security Proofs **55**

E.1 Formal Security Proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ 55

E.2 Formal Security Proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ 73

1 Introduction

A T-out-of-N threshold signature [Des90, DF90] allows to distribute a secret key to N parties such that a set of at least T parties can jointly generate a signature with respect to the verification key. In particular, even if an adversary corrupts up to $T-1$ parties, it should not be possible to forge a signature. This ability to distribute trust has seen a renewed interest in the blockchain ecosystem where secure and reliable storage of secret keys are critical. With the increase in real-world interest, governmental bodies such as the US agency NIST has announced a standardization effort for multi-party threshold schemes [PB23].

In this work, we focus on thresholdizing existing signature schemes based on the Fiat-Shamir paradigm [FS87], e.g., ECDSA, Schnorr [Sch91], Dilithium [DKL+18], Raccoon [dPEK+23], a wide class of efficient signature schemes that has been a popular target to thresholdize in the literature.

Static vs Adaptive Security. Threshold signatures being a multi-party protocol, we have two choices when defining unforgeability: static and adaptive security. Static security artificially restricts the adversary to commit to all the $T-1$ parties it corrupts at the beginning of the security game. In contrast, adaptive security allows the adversary to arbitrarily corrupt up to $T-1$ parties as the security game progresses. Specifically, it may dynamically choose which party to corrupt even after observing the verification key, partial signatures, and the corrupted secret key shares of the other parties.

Adaptive security captures much more closely the threat model in reality, and as such, the recent call for threshold schemes by NIST [PB23, BD22] has put a strong preference on schemes satisfying it. Indeed, there are simple schemes that are statically secure but trivially non-adaptively secure [CFGN96], highlighting a fundamental difference in these two security models.

Difficulty of Adaptive Security (in the ROM). While adaptive security is the sought after security requirement, most prior works on threshold signatures have only been proven statically secure. However, this is not a simple lack of interest to prove adaptive security but rather a demonstration of the limit of our current proof technique. Signature schemes based on the Fiat-Shamir paradigm [FS87] typically rely on the forking lemma [FS87, BN06] to prove security in the random oracle model (ROM). At a high level, a proof using the forking lemma proceeds as follows: The reduction embeds a hard problem into the verification key and simulates the security game to the adversary. Once the adversary outputs a forgery, it *rewinds* the adversary and runs it again from some specific point in the security game by programming the random oracle differently. If the adversary outputs a forgery in the second run, the reduction is able to extract the solution to the hard problem using the two forgeries.

Now, consider what happens when we try to use this for adaptive security. The reduction needs to simulate $T-1$ secret key shares of the corrupted parties in the first *and* second run. Since the set of corrupted parties may change after rewinding, the reduction may need to simulate up to $2(T-1)$ secret key shares. However, if a simulator knew T (or more) of the secret key shares, it can generate forgeries without the adversary's help, thus breaking the hard problem on its own. This seemingly contradicts the hardness of the problem, indicating that such a security proof does not work. Here, such an issue does not appear in the static setting since the set of corrupted parties remain unchanged in the two runs.

State-of-the-Art. To overcome this apparent issue, Crites, Komlo, and Maller [CKM23] considered a relaxed form of adaptive security where the adversary is limited to making either $T-1$ static or $(T-1)/2$ adaptive corruptions; the latter implies that the reduction only needs to simulate at most $T-1$ secret key shares in total. In this relaxed model, they proved security of their threshold Schnorr signature ($\mathsf{Sparkle}$) under an interactive algebraic one-more DL ($\mathsf{AOM\text{-}DL}$) assumption. More recently, Bacho et al. [BLT+24] proposed an adaptive threshold signature ($\mathsf{Twinkle}$) under the DDH assumption. Their novel insight was to base $\mathsf{Twinkle}$ on a specific class of signature schemes based on identification protocols that avoided rewinding to prove unforgeability, drawing inspiration from [Che05, KLP17]. A caveat though is that $\mathsf{Twinkle}$ no longer produces Schnorr signatures like $\mathsf{Sparkle}$ and requires a signature size that is twice as large.

It is also worth noting that another way to overcome the issue is to assume a stronger idealized model like the algebraic group model (AGM) [FKL18]. Indeed, Crites et al. [CKM23] showed that $\mathsf{Sparkle}$ is adaptively secure in the ROM and AGM under the $\mathsf{AOM\text{-}DL}$ assumption. However, this avenue of research seems quite grim for *post-quantum* threshold signatures like those based on lattices, since no such model is known to exist nor believed to hold in general.

This brings us to our main question of this work:

> *Can we construct an adaptively secure threshold signature scheme for existing rewinding-based Fiat-Shamir signatures? Moreover, can we base security under the same assumption?*

We believe the latter question is also an important point if we were to deploy threshold signatures based on existing signature schemes. For example, an ideal situation would be to prove threshold Schnorr signature under the DL assumption in the ROM, similarly to the non-thresholdized Schnorr signature.

1.1 Our Contribution

We answer the above question affirmatively and propose a new proof technique to construct adaptively secure threshold signatures. As a result, we obtain the following:

1. The first adaptively secure 5 round lattice-based threshold signature under the MLWE and MSIS assumptions in the ROM. The resulting signature is a standard signature of $\mathsf{Raccoon}$ [dPEK+23], a lattice-based signature scheme by del Pino et al., submitted to the additional NIST call for proposals [NIS22]. We can easily make this 4 round by assuming a (non-repeating) unique session identifier sid being broadcast to the signing parties.

2. The first adaptively secure 5 round threshold signature under the DL assumption in the ROM. The resulting signature is a standard Schnorr signature. Making the same assumption as above, we can turn it into a 4 round protocol. We note this is the first adaptively secure threshold signature based on the DL assumption.

As a byproduct of our new proof technique, we also obtain the following lattice-based threshold signature:

3. A selectively secure 3 round lattice-based threshold signature under the MLWE and MSIS assumptions in the ROM. The resulting signature is a standard signature of $\mathsf{Raccoon}$. This improves the very recent work by del Pino et al. [dPKM+24] in two metrics: it removes the need for a stateful signing algorithm and improves the communication cost. The signature size remains identical to [dPKM+24].

Importantly, all of our threshold signature is proven secure under the same assumptions and security model (i.e., ROM) as those of the underlying non-thresholdized signature. In addition, none of them require secure state erasures, a requirement often difficult to enforce in practice. We can also preprocess the first round of both of our 5 round threshold signatures, making the online phase four rounds [BD22, Section 5.3.5]. A comparison of prior related threshold signatures is given in Tables 1 and 2.

Table 1: Comparison of T-out-of-N lattice-based threshold signatures.

Schemes	Adaptive?	Assumptions	Rounds	Model	Corruptions	Stateless session id?
[dPKM+24]	✗	MLWE + $\mathsf{MSIS}^\dagger$	3	ROM	$< T$	✗
[EKT24]	✗	$\mathsf{AOM\text{-}MLWE}$	2	ROM	$< T$	✓
$\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$	✗	MLWE + MSIS	3	ROM	$< T$	✓
$\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$	✓	MLWE + MSIS	4	ROM	$< T$	✗
$\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$	✓	MLWE + MSIS	5	ROM	$< T$	✓

We omit schemes based on (linearly/fully) homomorphic encryption e.g.,[BGG+18, ASY22, GKS23]. MLWE and MSIS stand for the module LWE and SIS, respectively. $\mathsf{AOM\text{-}MLWE}$ stands for the algebraic one-more MLWE. The (✓) in the column "Stateless session id?" indicates that the parties can be stateless. Else (✗), the parties need to store the session id's so as not to reuse them.

† To be precise, they rely on the hint MLWE and self target MSIS problems, both of which are know to reduce from the MLWE and MSIS problem. The same can be said for our schemes.

Table 2: Comparison of T-out-of-N classical Schnorr-like threshold signatures.

Schemes	Adaptive?	Assumptions	Rounds	Model	Corruptions	Stateless session id?
[KG20, BCK+22] (Frost)	✗	AOM-DL	2	ROM	$< T$	✓
[Lin22]	✗	DL	3	ROM	$< T$	✓
[TZ23]	✗	DL	2	ROM	$< T$	✓
[CKM23] (Sparkle)	✗	DL	3	ROM	$< T$	✓
[CKM23] (Sparkle)	✓	AOM-DL	3	ROM	$< T/2$	✓
[CKM23] (Sparkle)	✓	AOM-DL	3	ROM + AGM	$< T$	✓
[BLT+24] (Twinkle)	✓	DDH	3	ROM	$< T$	✗
$\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ (Snap)	✓	DL	4	ROM	$< T$	✗
$\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ (Crackle)	✓	DL	5	ROM	$< T$	✓

We compare pairing-free Schnorr-like threshold signatures. ($< T/2$)-corruption indicates that adaptive security holds if only at most $T/2$ parties are corrupted. AOM-DL stands for the algebraic one-more DL. The (✓) in the column "Stateless session id?" indicates that the parties can be stateless. Else (✗), the parties need to store the session id's so as not to reuse them.

Before getting into the technical details, we clarify a downside of Crackle and Snap compared with other threshold Schnorr signatures. In order to prove adaptive security, we do not allow the parties to publish a partial verification key of the form $g^{a_i} \in \mathbb{G}$, where $a_i \in \mathbb{Z}_p$ is the secret key share. This specific partial verification key is typically used to achieve *non-interactive* identifiable abort; a property allowing the parties to non-interactively trace a malicious party in case the threshold signing protocol outputs an invalid signature. We note that static security of Crackle and Snap remains intact even if we publish g^{a_i}. For lattice-based threshold signatures, none of the schemes in Table 1 consider partial verification keys or non-interactive identifiable abort. See Section 1.3 for more details.

Technique in a Birds Eye's View. At a technical level, our work is inspired by the recent lattice-based 3 round threshold signature by del Pino et al. [dPKM+24] based on Raccoon. Their work can be seen as a lattice-based counterpart of the 3 round threshold Schnorr signature Sparkle by Crites et al. [CKM23] but with a unique twist. Due to lattice specific reasons, a natural adaptation of Sparkle to the lattice setting turns out to be insecure as the partial signature leaks too much information on the signing key shares. To overcome this issue, del Pino et al. relied on non-interactively shared *one-time additive masks*. At a high level, this allows each parties to output a *masked* partial signature, where the masks cancel out only when all T partial signatures are combined.

Our main technical contribution is noticing that this one-time additive mask not only solves lattice specific issues but is also a useful technique for achieving adaptive security. Recall that one of the reasons why rewinding proofs were at odds with adaptive security was that a reduction being able to simulate $T-1$ secret key shares in both of the runs can seemingly reconstruct the full secret key on its own. Specifically, there is no room left to embed a hard problem.

An idea to resolve this paradoxical situation is for the reduction to simply output *random* secret key shares in both runs. Looking through the lens of the adversary, this modification seems undetectable since *within each individual run*, the $T-1$ secret keys are indeed uniformly random when the secret key is being T-out-of-N secret shared. However, such a naive approach does not work as it stands. Throughout the game, an adversary can concurrently interact with the parties and observe the transcripts of any given signing session. Then, once corrupted a party, the adversary can check whether the secret key share is consistent with what it observed; if the secret key share was randomly simulated, this will certainly be detected.

This brings us to our main idea. We use one-time additive masks throughout the protocol to carefully control the information the adversary can learn from the transcripts. When a corruption occurs, we will generate randomness for the one-time additive masks so that it becomes consistent with the random secret key share and the transcript the adversary observed. While the proof strategy is intuitive, constructing a protocol that fits this intuition and proving it is *far* easier said than done. We refer the readers to the next section for a more technical overview. Lastly, while our technique is quite general, we choose not to phrase our scheme abstractly using linear function families (cf.,[HKL19, HKLN20]) as optimized lattice-based constructions like Raccoon do not neatly fit in this abstraction.

1.2 Technical Overview

We first recall the statically secure 3 round lattice-based threshold signature by del Pino et al. [dPKM+24]. We then show a simple improvement of their protocol, leading to our stateless 3 round threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. This scheme is only statically secure but our proof technique forms the basis of the more complex adaptively secure 5 round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$, which we then explain. While our overview focuses on the lattice setting, it will be clear that it trivially adapts to the classical Schnorr setting.

Lastly, while we try our best to keep the overview self-contained, we encourage the readers to look at [dPKM+24, Section 2] for an in depth overview on the original threshold signature by del Pino et al.

3 Round Threshold Raccoon by [dPKM+24]. We recall below a simplified variant of their scheme based on Lyubashevsky's lattice-based signature scheme [Lyu09, Lyu12]. The NIST submission Raccoon [dPEK+23] is a variant of Lyubashevsky's (and also Dilithium [DKL+18]), that is more susceptible to thresholdization. For the sake of simplicity, we ignore this lattice specific detail in the overview.

The verification key vk is an MLWE instance $(\mathbf{A}, \mathbf{b} = [\mathbf{A} \mid \mathbf{I}] \cdot \mathbf{s}) \in \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^k$, where $\mathbf{s} \in \mathcal{R}_q^{\ell+k}$ is a secret short vector.[1] The secret key $\mathbf{s}$ is distributed to each party using Shamir's secret sharing [Sha79]. Namely, each party $i \in [N]$ is given $\mathbf{s}_i$ such that for any $\mathsf{SS} \subseteq [N]$ and $|\mathsf{SS}| = T$, we have $\mathbf{s} = \sum_{j\in\mathsf{SS}} L_{\mathsf{SS},j}\mathbf{s}_i$ where $(L_{\mathsf{SS},j})_{j\in\mathsf{SS}}$ are the Lagrange coefficients. The novelty of [dPKM+24] is that, each party i is further distributed a random PRF seed $\vec{\mathsf{seed}}_i = (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]}$, where parties i and j share seeds $(\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})$. Lastly, each party i is also assumed to have their own set of keys $(\mathsf{vk}_{\mathsf{S},i}, \mathsf{sk}_{\mathsf{S},i})$ for a standard signature scheme. In summary, the secret key share for party i is $\mathsf{sk}_i = (\mathbf{s}_i, \vec{\mathsf{seed}}_i, \mathsf{sk}_{\mathsf{S},i})$.

To sign on a message M with a signer set SS, it proceeds as follows, where assume sid is a session identifier that has never been used.

Round 1. Signer i samples a commitment $\mathbf{w}_i = [\mathbf{A}|\mathbf{I}]\cdot\mathbf{r}_i \in \mathcal{R}_q^k$ for a short vector $\mathbf{r}_i \in \mathcal{R}_q^{\ell+k}$ and creates a hash commitment $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i)$. It also computes a so-called *row mask* $\mathbf{m}_i = \sum_{j\in\mathsf{SS}} \mathsf{PRF}(\mathsf{seed}_{i,j}, \mathsf{sid}) \in \mathcal{R}_q^{\ell+k}$ and outputs $(\mathsf{cmt}_i, \mathbf{m}_i)$.

Round 2. Signer i obtains $\mathsf{ctnt} = (\mathsf{cmt}_j, \mathbf{m}_j)_{j\in\mathsf{SS}}$, signs it $\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{sid}\|\mathsf{ctnt})$, and outputs the opening $\mathbf{w}_i$ and the signature $\sigma_{\mathsf{S},j}$.

Round 3. Signer i checks that for all $j \in \mathsf{SS}$, the hash commitment cmt_j are opened correctly by signer j, *i.e.*, $\mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \mathbf{w}_j)$, and the signature $\sigma_{\mathsf{S},j}$ verifies with respect to $\mathsf{sid}\|\mathsf{ctnt}$ it signed in Round 2. If the check passes, it computes the aggregate commitment $\mathbf{w} = \sum_{j\in\mathsf{SS}} \mathbf{w}_j$. It then computes the challenge $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$, the so-called *column mask* $\mathbf{m}_i^* = \sum_{j\in\mathsf{SS}} \mathsf{PRF}(\mathsf{seed}_{j,i}, \mathsf{sid}) \in \mathcal{R}_q^{\ell+k}$, and outputs the *masked* response $\tilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i - \mathbf{m}_i^* \in \mathcal{R}_q^{\ell+k}$.

Aggregate. Given the transcript $((\mathsf{cmt}_j, \mathbf{m}_j), (\mathbf{w}_j, \sigma_{\mathsf{S},j}), \tilde{\mathbf{z}}_j)_{j\in\mathsf{SS}}$, it outputs the aggregate signature $\mathsf{sig} = (c, \mathbf{z})$, where $\mathbf{z} = \sum_{j\in\mathsf{SS}}(\tilde{\mathbf{z}}_j - \mathbf{m}_j)$ and $(\mathbf{w}, c)$ are computed as above.

A signature sig is deemed valid if $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{Az} - c\cdot\mathbf{b})$ and $\mathbf{z}$ is short. Correctness is established by the equality $\sum_{i\in\mathsf{SS}} \mathbf{m}_i = \sum_{i\in\mathsf{SS}} \mathbf{m}_i^*$, i.e., the sum of row masks and column masks are identical. Concretely, we have $\mathbf{z} = c\cdot\sum_{j\in\mathsf{SS}}(L_{\mathsf{SS},i}\cdot\mathbf{s}_i + \mathbf{r}_i) = c\cdot\mathbf{s} + \sum_{j\in\mathsf{SS}}\mathbf{r}_i$. Since $\mathbf{w} = \sum_{j\in\mathsf{SS}}\mathbf{w}_j = [\mathbf{A}|\mathbf{I}]\cdot(\sum_{j\in\mathsf{SS}}\mathbf{r}_j)$, the signature

[1] In the main body, we write $\mathbf{b}$ as $\mathbf{As} + \mathbf{e} = [\mathbf{A} \mid \mathbf{I}] \cdot \begin{bmatrix}\mathbf{s}\\ \mathbf{e}\end{bmatrix}$. This reflects the standard optimization [BG14] performed by Dilithium and Raccoon where we ignore the noise $\mathbf{e}$ and only view the upper $\mathbf{s}$ as the secret key. Since this makes the protocol more complex, we opt using the simplest version in the overview.

sig is indeed a valid Lyubashevsky's signature.

Intuition of Security Proof. As opposed to classical cryptography, in lattice-based cryptography secrets are "short". Specifically, if parties instead output an unmasked partial response $\widetilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i \in \mathcal{R}_q^{\ell+k}$, there is a concrete attack on the scheme as the Lagrange coefficient $L_{\mathsf{SS},i}$ can arbitrarily amplify the size of the secret key share $\mathbf{s}_i$ (see [dPKM+24, Section 2] for the details). This is where the mask plays a critical role. Due to the pseudorandomness of the PRF, the masks are distributed uniformly random conditioned on the sum of row masks and column masks being identical. Informally, this implies that the partial response only leaks information of the final aggregate signature $\mathbf{z} = c \cdot \mathbf{s} + \sum_{j \in \mathsf{SS}} \mathbf{r}_i$. Turning this around, the partial response can be simulated only by using the full secret key $\mathbf{s}$. From a reductionist's point of view, this allows us to invoke honest-verifier zero-knowledge (HVZK) with respect to the verification key vk to simulate the signature $\mathsf{sig} = (c, \mathbf{z})$, followed by programming the random oracle so that $\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{A}\mathbf{z} - c \cdot \mathbf{b}) = c$.

While the intuition is clear, the concrete proof contains subtleties. First of all, the above argument hinges on the pseudorandomness of the PRF, and in particular, if the same input sid is used to derive the masks, the scheme becomes insecure. This is where del Pino et al. [dPKM+24] assumes the parties to maintain state of all the sid it signed. Furthermore, when we stated that the partial response only leaks information of the final aggregate signature $\mathbf{z} = c \cdot \mathbf{s} + \sum_{j \in \mathsf{SS}} \mathbf{r}_i$, we implicitly used the fact that all the parties agree on the *same* challenge c in Round 3. As an example of how things can go wrong, assume a malicious party 3 invokes honest parties 1 and 2 on the same sid and provides them $(\mathsf{cmt}_1, \mathsf{cmt}_2, \mathsf{cmt}_3)$ and $(\mathsf{cmt}_1, \mathsf{cmt}_2, \mathsf{cmt}'_3)$, respectively, in Round 2, where cmt_3 and cmt'_3 open to different commitments. Then, since the aggregate commitment differs, parties 1 and 2 will derive different challenges in Round 3. However, since the masks are only defined via sid, they will cancel out when combining the partial responses, and in particular, the adversary learns $c_1 \cdot L_{\mathsf{S},1} \cdot \mathbf{s}_1 + c_2 \cdot L_{\mathsf{S},2} \cdot \mathbf{s}_2 + \sum_{j \in [2]} \mathbf{r}_j$ for $c_1 \neq c_2$. Again, this leads to concrete attacks.[2] Thus, to thwart such an attack, del Pino et al. [dPKM+24] requires the parties to sign their entire view sid‖ctnt in Round 2. This effectively enforces that if all parties with sid finished Round 3, then they must have used the same unique challenge c. Piecing these arguments together, we can formally invoke HVZK to complete the above proof intuition.

A Simple Tweak and a New Proof. Looking at the prior construction (and security proof) closely, it can be checked that there is no need to compute the masks in different rounds. In particular, the row and column masks can be generated together in Round 3. We also remove the signature in Round 2 and consider the following simplified 3-round threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$.

Round 1. Signer i samples a commitment $\mathbf{w}_i = [\mathbf{A}|\mathbf{I}] \cdot \mathbf{r}_i \in \mathcal{R}_q^k$ for a short vector $\mathbf{r}_i \in \mathcal{R}_q^{\ell+k}$ and outputs a hash commitment $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i)$.

Round 2. Signer i obtains $\mathsf{ctnt}_{\mathbf{w}} = (\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ and outputs the opening $\mathbf{w}_i$.

Round 3. Signer i checks that for all $j \in \mathsf{SS}$, the hash commitment cmt_j are opened correctly by signer j, *i.e.*, $\mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \mathbf{w}_j)$. If the check passes, it computes the aggregate commitment $\mathbf{w} = \sum_{j \in \mathsf{SS}} \mathbf{w}_j$ and sets $\mathsf{ctnt}_{\mathbf{z}} = (\mathsf{cmt}_j, \mathbf{w}_j)_{j \in \mathsf{SS}}$. It then computes the challenge $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$, a *zero-share* mask $\mathbf{\Delta}_i = \sum_{j \in \mathsf{SS}} (\mathsf{PRF}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}) - \mathsf{PRF}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})) \in \mathcal{R}_q^{\ell+k}$, and outputs the masked response $\widetilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell+k}$.[3]

Aggregate. Given the transcript $(\mathsf{cmt}_j, \mathbf{w}_j, \widetilde{\mathbf{z}}_j)_{j \in \mathsf{SS}}$, it outputs the aggregate signature $\mathsf{sig} = (c, \mathbf{z})$, where $\mathbf{z} = \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{z}}_j$ and $(\mathbf{w}, c)$ are computed as above.

[2]For those knowledgeable in classical threshold signatures like Sparkle and Twinkle, we note that such an attack cannot be used to break unforgeability. This is because unlike in the lattice setting, we can invoke HVZK with respect to the *partial* verification key, i.e., the partial response can be simulated individually. In the lattice setting, the unmasked partial response leaks too much information on the secret key share and thus HVZK must be applied to the full secret key.

[3]In the actual construction, we include the signer set SS and message M in $\mathsf{ctnt}_{\mathbf{z}}$, as otherwise, it opens the door to ROS attacks [BLL+21]. We gloss over this detail in the overview for simplicity as it can be handled using standard methods.

The verification algorithm is defined identically as before, where correctness follows immediately by the equality $\sum_{j\in\mathsf{SS}} \mathbf{\Delta}_j = \mathbf{0}$. Notice that this modification allows to remove the state since an honest party i is now guaranteed to always invoke the PRF on a distinct input; this is an immediate implication of including $\mathbf{w}_i$ in the input $\mathsf{ctnt}_{\mathbf{z}}$.

The most important part though, is whether the scheme remains secure even if we remove the standard signature in Round 2. As a sanity check, let us observe that the aforementioned attack will not work. Consider an adversary invoking parties 1 and 2 on different Round 2 hash commitments $(\mathsf{cmt}_1, \mathsf{cmt}_2, \mathsf{cmt}_3)$ and $(\mathsf{cmt}_1, \mathsf{cmt}_2, \mathsf{cmt}'_3)$, respectively. In our modified scheme, since both parties now compute a mask using different PRF inputs in Round 3, the masks no longer cancel out. Specifically, the adversary learns nothing by combining the the partial response as it will compute to $\sum_{j\in[2]}(c_j \cdot L_{\mathsf{S},j} \cdot \mathbf{s}_j + \mathbf{r}_j + \mathbf{\Delta}_j)$ for $\sum_{j\in[2]} \mathbf{\Delta}_j \neq \mathbf{0}$. Let us now formalize this below.

The key argument in the security proof of del Pino et al. [dPKM+24] was enforcing all parties with sid in Round 2, eventually finishing Round 3, to satisfy two properties: (i) the masks they use in Round 3 must be computed from the same PRF input and (ii) they must all be using the same unique challenge c. The first property guaranteed the equality $\sum_{i\in\mathsf{SS}} \mathbf{m}_i = \sum_{i\in\mathsf{SS}} \mathbf{m}^*_i$ and this held by virtue since the masks were computed only using sid. The second property guaranteed that the partial responses leak no more information than the final signature $\mathbf{z} = c \cdot \mathbf{s} + \sum_{j\in\mathsf{SS}} \mathbf{r}_i$. This was enforced by using standard signatures.

Translating their key argument to our scheme, we have to enforce the same two properties as above but now with respect to all parties with $\mathsf{ctnt}_{\mathbf{w}}$ in Round 2, eventually finishing Round 3. We show that these two properties combined allows us to invoke HVZK *before* Round 3 as desired. To prove the properties, we use the fact that $\mathsf{ctnt}_{\mathbf{w}} = (\mathsf{cmt}_j)_{j\in\mathsf{SS}}$ is a set of binding hash commitments, i.e., cmt_j can only be uniquely opened to a commitment $\mathbf{w}_j$. Using this, we can guarantee that there is only a unique $\mathsf{ctnt}_{\mathbf{z}}$ in Round 3 that can lead from $\mathsf{ctnt}_{\mathbf{w}}$ in Round 2, enforcing the first property. Moreover, using the same argument, there can only be one aggregate commitment $\mathbf{w} = \sum_{j\in\mathsf{SS}} \mathbf{w}_j$ in Round 3, enforcing the second property. It is worth noting that a similar argument appears in Bacho et al. [BLT+24], where they notice a slight gap in the security proof of Sparkle [CKM23]. They use a technical argument called *equivalence classes* to enforce the second property. Our proof is inherently more involved than theirs as we must also enforce the first property, stemming from the fact that we have to invoke HVZK on the full verification key vk (see also Footnote 2). Piecing the arguments together, we conclude static security of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$.

Why Adaptive Security Fails. As briefly explained in the previous section, to resolve the paradoxical situation, a natural proof strategy for adaptive security is for the reduction to simply output a random secret key share $\mathbf{s}_i \xleftarrow{\$} \mathcal{R}_q^{\ell+k}$. However, it is clear that such a proof strategy fails for $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. Once party i is corrupted, the adversary obtains $\mathsf{sk}_i = (\mathbf{s}_i, \vec{\mathsf{seed}}_i)$. Using $\vec{\mathsf{seed}}_i$, it can unmask the masked response $\widetilde{\mathbf{z}}_i$ and further recover the commitment randomness $\mathbf{r}_i = \widetilde{\mathbf{z}}_i - c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i - \mathbf{\Delta}_i$. From this, it can check if the commitment $\mathbf{w}_i$ equals $[\mathbf{A}|\mathbf{I}] \cdot \mathbf{r}_i$. If $\mathbf{s}_i$ was sampled uniformly, this equality will clearly not hold, rendering the simulation to be distinguishable from the real security game.

This example illustrates another difficulty of adaptive security. We have seen that if the adversary obtains sk_i, it can recover the randomness $\mathbf{r}_i$ used to generate the commitment $\mathbf{w}_i$. Let us consider the following situation: the adversary invokes all the parties up to Round 2 and obtains $(\mathbf{w}_j)_{j\in\mathsf{SS}}$. Assume the adversary corrupts half of the parties $\mathcal{Q} \subset \mathsf{SS}$ in the first run. The reduction then rewinds the adversary, and assume it corrupts the other half of the parties $\mathsf{SS}\backslash\mathcal{Q}$ in the second run. For this to work, the reduction must be ready to answer all the commitment randomness $(\mathbf{r}_j)_{j\in\mathsf{SS}}$. However, once again, this leaves the reduction no space to embed its hard problem! Indeed, if the reduction tries to invoke HVZK, there is no place to embed the simulated commitment $\mathbf{w}$.

More Masking Solves the Problem. Our key insight to solve the above problem is to add another layer of masking to the commitments $\mathbf{w}_i$, and moreover, to generate the mask using a random oracle $\mathsf{H}_{\mathsf{mask}}$ as opposed to using a PRF. The following is our 4-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$, where we again assume each party has their own set of keys for a standard signature scheme and assume an unused sid is provided to the parties.

Round 1. Signer i samples a commitment $\mathbf{w}_i = [\mathbf{A}|\mathbf{I}] \cdot \mathbf{r}_i \in \mathcal{R}_q^k$ for a short vector $\mathbf{r}_i \in \mathcal{R}_q^{\ell+k}$ and computes a zero-share mask $\widetilde{\mathbf{\Delta}}_i = \sum_{j \in \mathsf{SS}} (\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{sid}) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{sid})) \in \mathcal{R}_q^k$. It then computes a *masked* commitment $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$ and outputs a hash commitment $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(i, \widetilde{\mathbf{w}}_i)$.

Round 2. Signer i obtains $\mathsf{ctnt}_{\mathbf{w}} = (\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ and outputs a signature $\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{sid} \| \mathsf{ctnt}_{\mathbf{w}})$.

Round 3. Signer checks that for all $j \in \mathsf{SS}$ the signature $\sigma_{\mathsf{S},j}$ verifies with respect to $\mathsf{sid} \| \mathsf{ctnt}_{\mathbf{w}}$ it signed in Round 2. If so, it outputs the opening $\widetilde{\mathbf{w}}_i$.

Round 4. Signer i checks that for all $j \in \mathsf{SS}$, the hash commitment cmt_j are opened correctly by signer j, *i.e.*, $\mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)$. If the check passes, it computes the aggregate commitment $\mathbf{w} = \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j$ and sets $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{sid} \| (\mathsf{cmt}_j, \widetilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$. It then computes the challenge $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$, a zero-share mask $\mathbf{\Delta}_i = \sum_{j \in \mathsf{SS}} (\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})) \in \mathcal{R}_q^{\ell+k}$, and outputs the masked response $\widetilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell+k}$.

The aggregation algorithm is defined identically as in $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. Correctness holds by observing that the aggregate commitment $\mathbf{w}$ adds up to the same value as before using the fact $\sum_{j \in \mathsf{SS}} \widetilde{\mathbf{\Delta}}_j = \mathbf{0}$.

While it is not immediately clear why this scheme can be proven adaptively secure, it will be informative to see why the previously explained distinguishing attack no longer works. First, observe that before the adversary corrupts any party, the individual commitments $\widetilde{\mathbf{w}}_j$ and partial responses $\widetilde{\mathbf{z}}_j$ are distributed uniformly random thanks to the two masks $\widetilde{\mathbf{\Delta}}_j$ and $\mathbf{\Delta}_j$, conditioned on the resulting signature $\mathsf{sig} = (c, \mathbf{z})$ being valid. That is, $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ and $\mathbf{w} = \mathbf{A}\mathbf{z} - c \cdot \mathbf{b}$ where $\mathbf{w} = \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j$ and $\mathbf{z} = \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{z}}_j$. Now, assume party i is corrupted. Then, the reduction first samples a random secret key share $\mathbf{s}_i \xleftarrow{\$} \mathcal{R}_q^{\ell+k}$ and a random commitment randomness $\mathbf{r}_i$. It then computes a *fake* commitment $\mathbf{w}_i = [\mathbf{A}|\mathbf{I}] \cdot \mathbf{r}_i$ and a *fake* response $\mathbf{z}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i$. It further programs the random oracle $\mathsf{H}_{\mathsf{mask}}$ so that the two masks $(\widetilde{\mathbf{\Delta}}_i, \mathbf{\Delta}_i)$ compute to $(\widetilde{\mathbf{w}}_j - \mathbf{w}_j, \widetilde{\mathbf{z}}_j - \mathbf{z}_j)$. This is where we require to generate the masks using a random oracle as opposed to using a PRF. Lastly, the reduction outputs $\mathsf{sk}_i = (\mathbf{s}_i, \vec{\mathsf{seed}}_i, \mathsf{sk}_{\mathsf{S},i})$ to the adversary. Due to the way we program the masks, sk_i is consistent with $(\widetilde{\mathbf{w}}_i, \widetilde{\mathbf{z}}_i)$ observed by the adversary. It is worth noting that no secure state erasure is necessary since we can simulate all the randomness to the adversary.

The above reduction strategy tells us, at least intuitively, that the entire transcript only leaks the information on the full signature $\mathsf{sig} = (c, \mathbf{z})$. Turning this intuition into a formal proof consists the main technical contribution of our work. In particular, the above reduction only concerns how to randomly answer the adversary's corruption query, and tells us nothing about how to embed a hard problem in the reduction. As in the static setting, the goal will be to simulate sig by invoking HVZK with respect to the verification key vk through a careful chain of technical arguments.

At a very high level, our proof consists of three steps. We first enforce all parties with sid in Round2, eventually arriving at Round 3, to agree on the same $\mathsf{ctnt}_{\mathbf{w}}$ (call this property (i)). This allows us to argue that every masked commitment $\widetilde{\mathbf{w}}_i$ except for the last one is uniform random. To prove property (i), we use a similar argument to del Pino et al. [dPKM+24], relying on the masks generated with sid and standard signatures $\sigma_{\mathsf{S},i}$. We next enforce all parties with $\mathsf{ctnt}_{\mathbf{w}}$ in Round 3, eventually finishing Round 4, to satisfy two additional properties: (ii) the masks $\mathbf{\Delta}_j$ they use in Round 4 must be computed from the same $\mathsf{H}_{\mathsf{mask}}$ input and (iii) they must all be using the same unique challenge c. This allows us to argue that every masked response $\widetilde{\mathbf{z}}_i$ except for the last one is uniform random and that the aggregated response $\mathbf{z}$ can be expressed using the fully secret key $\mathbf{s}$, as opposed to using the secret key shares. To prove properties (ii) and (iii), we use a similar argument to our $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. The final step consists of carefully gluing these properties (i), (ii), and (iii) together to show that we can consistently embed $(\mathbf{w}, c, \mathbf{z})$ simulated by HVZK.

We emphasize that the above is a major simplification of our proof. The simplification arises when we loosely used the term "every masked commitment $\widetilde{\mathbf{w}}_i$ (and masked response $\widetilde{\mathbf{z}}_i$) *except the last one* is uniform random". Since the adversary is adaptive, the last masked commitment and response are not known in advance to the reduction. To make matters worse, we have to consider situations where the last commitment is $\widetilde{\mathbf{w}}_i$ while the last response is $\widetilde{\mathbf{z}}_j$ for a different party $i \neq j$. This highly non-trivializes the final step of consistently embedding $(\mathbf{w}, c, \mathbf{z})$ into the security game. We provide a more detailed proof overview in Section 6.2.

Removing States with One More Round. Our 4-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ required to maintain state so that the same sid is never reused. We remove this restriction and construct a stateless scheme by adding one more round, resulting in our 5-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. The idea is very simple: In the first round, each party i broadcasts a random string str_i. The parties then set $\mathsf{sid} = (\mathsf{str}_j)_{j \in \mathsf{SS}}$ and proceeds as in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$. The main observation is that since sid contains a uniform random string str_i sampled by party i, it is guaranteed to be distinct from prior sid's. Since the first round can be performed without knowing the signer set SS or the message M, it can be preprocessed, making the online phase of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ 4-round.

1.3 Related Works

We provide a brief overview of related works in the area of threshold signatures.

Threshold Signatures

We give an overview of threshold signatures from the literature. The below schemes are roughly classified by the underlying security assumptions. In particular, we will focus on adaptively secure schemes later.

Post-Quantum. Boneh et al. [BGG+18] and Agrawal, Stehlé, and Yadav [ASY22] proposed the one-round lattice-based threshold signature. Both rely on threshold fully homomorphic encryption (FHE). Gur, Katz, and Silde [GKS23] constructed a two-round threshold signature by using threshold linear homomorphic encryption and homomorphic trapdoor commitment [GVW15, DOTT21]. In a subsequent advancement, del Pino et al. [dPKM+24] proposed a three-round lattice-based threshold signature without using such heavy cryptographic tools. Recently, Espitau et al. [ENP24] proposed a lattice-based hash-and-sign robust threshold signature scheme with robust distributed key generation by constructing a verifiable (short) secret sharing scheme without relying on FHE. Bendlin et al. [BKP13] constructed a threshold signature scheme based on the GPV signature [GPV08] by relaying on generic multi-party computation (MPC). Khaburzaniya et al [KCLM22] proposed a threshold signature scheme based on hash-based signatures by using STARKs. Some works [CS20, DM20] proposed isogeny-based threshold signatures, which only achieve sequential aggregation.

DL-Based. Stinson and Strobl [SS01], and Gennaro et al. [GJKR07] constructed threshold Schnorr signatures with a signing protocol of at least 4 round. Both works rely on distributed key generation (DKG) to generate randomness within the signing protocol which impacts the number of rounds. Komlo and Goldberg [KG20] proposed FROST, a two round threshold signature scheme. Later, FROST was proven secure under the one-more DL (OMDL) assumption by Bellare et al. [BCK+22]. Tessaro and Zhu [TZ23] constructed a variant of FROST based on the DL assumption. Lindell [Lin22] proposed a three-round threshold Schnorr signature and proved its security in the UC model. Recent works [CKM23, BLT+24] constructed adaptively secure three round threshold signature schemes.

Others. Boldyreva [Bol03] constructed a pairing-based threshold signature based on BLS signatures [BLS01]. Later, it was proven to achieve a slightly stronger notion of security by [BTZ22]. The adaptive security for pairing-based threshold signatures are studied in [LJY14, BL22, DR23]. Several works proposed selectively secure RSA-based threshold signatures [DDFY94, Rab98, FMY98, Sho00, ADN06, GHKR08, TZ23] and threshold ECDSA signatures [GGN16, LN18, GG18, DKLs19, DJN+20, GG20, CGG+20, CCL+20, GKSŚ20].

Threshold Signatures with Adaptive Security

Adaptive security is a stronger notion of security for threshold signatures compared to selective security. Recall that selective security means that the adversary fixes the set of corrupted users before it can query signing oracles in the security game. In the adaptive setting, the signer can corrupt up to $T-1$ signers at an arbitrary point throughout the security game. The challenger then has to reveal the state of the corrupted signer. This signer state includes the randomness used throughout the signing sessions which makes proving security challenging. To show adaptive security, it is possible to guess the set of corrupted signers CS for N-out-of-N threshold signatures with a polynomial loss. However, guessing CS leads to an exponentially large loss in security for T-out-of-N threshold signatures. While adaptively secure RSA-based [ADN06] and ECDSA [CGG+20] threshold signatures were proposed, both adopt the N-out-of-N setting.

Some adaptively secure T-out-of-N threshold signatures were proposed so far. The lattice-based threshold signature proposed in [ASY22] achieves adaptive security. More precisely, in the random oracle model, [ASY22] achieves only partial adaptive security, *i.e.*, the adversary has to fix the set of corrupted users at once (but can do so adaptively). Without the random oracle model, [ASY22] achieves full adaptive security at the cost of additional preprocessing.

Libert, Joye, and Yung [LJY14] constructed an adaptively secure pairing-based threshold signature. Recently, Bacho and Loss [BL22] showed that threshold BLS is adaptively secure under the OMDL assumption in the AGM. Subsequently, Das and Ren [DR23] showed that threshold BLS is adaptively secure under the DDH and co-CDH assumptions in asymmetric pairing group.

Canetti et al. [CGJ+99] proposed an adaptively secure threshold signature scheme based on the digital signature standard (DSS) by assuming secure erasures. To prove the security, they used a technique for proving adaptive security called single-inconsistent-player (SIP). While this technique ensures that the simulation works well unless an inconsistent user is corrupted, it requires all N users to participate in the signing protocol. This requirement is often acceptable when considering robustness, which will be described below. Lysyanskaya and Peikert [LP01] constructed an erasure-free threshold Cramer-Shoup signature scheme, whose security is proved using the SIP technique. Abe and Fehr [AF04] proposed erasure-free threshold DSS and Schnorr signature schemes and proved their security in the relaxed UC framework cased the SIP UC model.

Crites, Komlo, and Maller [CKM23] proposed the three round adaptively secure threshold Schnorr signature scheme $\mathsf{Sparkle}$ based on the algebraic OMDL assumption in the AGM. Without the AGM, it is only half adaptively secure (*i.e.*, the number of corrupt signers is limited to $T/2$). Bacho et al. [BLT+24] constructed the adaptively secure scheme $\mathsf{Twinkle}$ based on the DDH assumption. Their result relies on adapting the underlying signature scheme to not rely on rewinding (and their scheme is not a classical Schnorr signature).

Robustness and Identifiable Abort

The robustness property ensures that a valid signature can be obtained from the signing protocol, even if a malicious signer participates. In [RRJ+22, BHK+23, Sho23, GS23], selective secure and robust threshold Schnorr threshold signature schemes are proposed. Another useful property is identifiable abort. Identifiable aborts enable the detection of malicious parties when the singing protocol aborts. Canetti et al. [CGG+20] constructed threshold ECDSA with identifiable abort.

The lattice-based schemes [BGG+18, ASY22] satisfy the robustness and identifiable abort properties by relying on homomorphic signatures. The adaptively secure Schnorr-like threshold signatures $\mathsf{Sparkle}$ and $\mathsf{Twinkle}$ do not achieve robustness but satisfy the identifiable abort property. The latter is possible because $\mathsf{Sparkle}$ and $\mathsf{Twinkle}$ publish a partial verification keys that allows to verify individual protocol messages. As discussed, publishing a partial verification key seems at odds with rewinding-based adaptive security proofs. Consequently, our constructions do not satisfy this property.

Distributed Key Generation

If we can ensure the existence of a trusted dealer, we can easily distribute secret key shares. On the other hand, instead of relying on a trusted dealer, we can use multi-party computation to generate key shares. Many works have studied DL-based distributed key generation (DKG) protocols [Ped92, CGJ+99, JL00, GJKR07, KMS20, DYX+22, KGS23]. In particular, adaptive security is considered in [CGJ+99, JL00, KMS20]. In [GKS23], a lattice-based DKG protocol to set up key shares with respect to a LWE-type verification key was proposed.

2 Preliminary

We provide a some backgrounds. Standard definitions and primitives are provided in Appendix A.

2.1 Notations

We use lower (resp. upper) case bold fonts $\mathbf{v}$ (resp. $\mathbf{M}$) for vectors (resp. matrices). We always view vectors in the column form. We use v_i (resp. $\mathbf{m}_i$) to indicate the i-th entry (resp. column) of $\mathbf{v}$ (resp. $\mathbf{M}$). For $(\mathbf{v}, \mathbf{M}) \in \mathcal{R}_q^\ell \times \mathcal{R}_q^{k\times\ell}$, $\mathbf{v}^\top \odot \mathbf{M}$ denotes the column-wise multiplication: $[v_1 \cdot \mathbf{M}_1 \mid \cdots \mid v_\ell \cdot \mathbf{M}_\ell]$. We denote by $\mathcal{U}_S$ the uniform distribution over some set S. We write $x \sim D$ if a random variable x follows the distribution D.

2.2 Threshold Signatures

We define R round threshold signatures. Let N be the number of total signers and T be a reconstruction threshold s.t. $T \leqslant N$. Also, let SS be a signer set such that $\mathsf{SS} \subseteq [N]$ with size T. Each signer $i \in [N]$ maintains a state st_i to retain short-lived session specific information.

$\mathsf{Setup}(1^\lambda, N, T) \to \mathsf{tspar}$**:** The setup algorithm takes as input a security parameter 1^λ, the number N of total signers, and a reconstruction threshold $T \leqslant N$ and outputs a public parameter tspar. We assume tspar includes N and T.

$\mathsf{KeyGen}(\mathsf{tspar}) \to (\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]})$**:** The key generation algorithm takes as input a public parameter tspar and outputs a verification key vk, and secret key shares $(\mathsf{sk}_i)_{i\in[N]}$. It implicitly sets up an empty state $\mathsf{st}_i := \varnothing$ for all N signers. We assume vk includes tspar.

$\mathsf{Sign} = (\mathsf{Sign}_1, \cdots, \mathsf{Sign}_R, \mathsf{Agg})$**:** The signing algorithms for the signing protocol of R round threshold signatures consist the following $(R+1)$ algorithms.

$\mathsf{Sign}_r(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i) \to (\mathsf{pm}_{r,i}, \mathsf{st}_i)$**:** The signing algorithm for the rth round for $r \in [R]$ takes as input a verification key vk, a signer set SS, a message M, an index i of a signer, a tuple of protocol messages of the $(r-1)$th round $(\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}}$, a secret key share sk_i, and a state st_i of the signer i and outputs a protocol message $\mathsf{pm}_{r,i}$ for the second round and an updated state st_i. Note that $\mathsf{pm}_{0,j}$ is $\perp$ for all $j \in \mathsf{SS}$. If the round r can be executed before deciding SS and/or M, Sign_r does not take as input them.

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{r,i})_{r\in[R],i\in\mathsf{SS}}) \to \mathsf{sig}$**:** The aggregation algorithm takes as input a verification key vk, a signer set SS, a message M, and a tuple of protocol messages $(\mathsf{pm}_{r,i})_{r\in[R],i\in\mathsf{SS}}$ and outputs a signature sig.

$\mathsf{Verify}(\mathsf{vk}, \mathsf{M}, \mathsf{sig}) \to 1$ or 0**:** The verification algorithm takes as input a verification key vk, a message M, and a signature sig, and outputs 1 if sig is valid and 0 otherwise.

Below, we define the correctness of a R round threshold signature scheme.

Definition 2.1 (Correctness). *We say that a R round threshold signature scheme TS satisfies correctness if, for all $\lambda \in \mathbb{N}$, $N, T \in \mathsf{poly}(\lambda)$ s.t. $T \leqslant N$, message M, and $\mathsf{SS} \subseteq [N]$ s.t. $|\mathsf{SS}| = T$, the following respectively hold:*

$$\Pr\left[\mathsf{Game}_{\mathsf{TS}}^{\mathsf{ts\text{-}cor}}(1^\lambda, N, T, \mathsf{M}, \mathsf{SS}) = 1\right] \geqslant 1 - \mathsf{negl}(\lambda),$$

where $\mathsf{Game}_{\mathsf{TS}_3}^{\mathsf{ts\text{-}cor}}$ are shown in Fig. 1.

$\mathsf{Game}^{\mathsf{ts\text{-}cor}}_{\mathsf{TS}}(1^\lambda, N, T, \mathsf{M}, \mathsf{SS})$

$\textbf{for } i \in \mathsf{SS} \textbf{ do } \mathsf{st}_i := \varnothing$
$\mathsf{tspar} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, N, T)$
$(\mathsf{vk}, (\mathsf{sk}_i)_{i \in [N]}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{tspar})$
$\textbf{for } i \in \mathsf{SS} \textbf{ do } \mathsf{pm}_{0,i} := \perp$
$\textbf{for } r \in [R] \textbf{ do}$
$\quad \textbf{for } i \in \mathsf{SS} \textbf{ do}$
$\quad\quad (\mathsf{pm}_{r,i}, \mathsf{st}_i) \xleftarrow{\$} \mathsf{Sign}_r(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$
$\mathsf{sig} \xleftarrow{\$} \mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{r,i})_{r \in [R], i \in \mathsf{SS}})$
$\textbf{return } \mathsf{Verify}(\mathsf{vk}, \mathsf{M}, \mathsf{sig})$

Figure 1: Correctness game for a R round threshold signature scheme.

2.2.1 Selective Security

In the selective setting, an adversary $\mathcal{A}$ determines the set CS of users to be corrupted at the beginning of the security game (after obtaining the parameters tspar). After this, it is not allowed to corrupt more honest user during the game. The challenger executes the key generation after CS is determined. It then provides $\mathcal{A}$ with the verification key and secret key shares of corrupted users as input. It also provides access to signing oracles for each round. In the end, $\mathcal{A}$ outputs a signature-message pair $(\mathsf{sig}^*, \mathsf{M}^*)$ that constitutes the forgery. The adversary $\mathcal{A}$ wins the game if $(\mathsf{sig}^*, \mathsf{M}^*)$ is deemed *non-trivial*. We refer to Appendix A.1.1 for a formal definition based on [CKM23]. Note that we use a stronger security model, *i.e.*, we classify more forgeries as non-trivial. We discuss this in more detail below.

2.2.2 Adaptive Security

We define the adaptive security of threshold signature schemes. In the adaptive setting, an adversary is allowed to corrupt a signer at any time via a corruption oracle $\mathcal{O}_{\mathsf{Corrupt}}$. The oracle $\mathcal{O}_{\mathsf{Corrupt}}$ receives an index i of a honest signer and returns the secret key share sk_i and the state st_i of the ith signer.

Our definition of adaptive security is based on the game-based definition for a three round scheme provided by [CKM23]. While the corruption rate τ is considered in the definition in [CKM23], in which an adversary allowed to corrupt at most $\lfloor (t-1)/\tau \rfloor$ honest signers, we do not consider this since we only consider full adaptive security, i.e., $\tau = 1$.

Also, we only consider a forgery $(\mathsf{sig}^*, \mathsf{M}^*)$ as *trivial* if at least $T - |\mathsf{CS}|$ honest users complete the signing protocol on M^*. Thus, even if $\mathcal{A}$ queried M^* in the last round for less than $T - |\mathsf{CS}|$ honest users, we consider a signature on M^* a valid forgery.

Now we define the adaptive security for a R round threshold signature scheme.

Definition 2.2 (TS-UF-1 **Adaptive Security).** *For a R round threshold signature scheme TS, the advantage of an adversary $\mathcal{A}$ (with oracle access to a random oracle H) against the adaptive security of TS is defined as*

$$\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T) = \Pr[\mathsf{Game}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T) = 1],$$

where $\mathsf{Game}^{\mathsf{ts\text{-}adp\text{-}uf}}$ is described in Fig. 2, respectively. We say that TS is adaptive secure in the random oracle model if, for all $\lambda \in \mathbb{N}$, $N, T \in \mathsf{poly}(\lambda)$ s.t. $T \leqslant N$ and PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}\mathsf{TS}, \mathcal{A}(1^\lambda, N, T) \leqslant \mathsf{negl}(\lambda)$ holds.

Remark 2.3 (Non-trivial forgeries). We consider a stronger notion of security than previous game-based definitions of adaptive security of threshold signatures [CKM23, BLT$^+$24]. In previous definitions, a forgery

$\mathsf{Game}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T)$

$\mathsf{Q_M}[\cdot] = \varnothing$ // No message was signed yet
$\mathsf{tspar} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, N, T)$
$\mathsf{HS} := [N], \mathsf{CS} := \varnothing$
for $i \in \mathsf{HS}$ **do** $\mathsf{st}_i := \varnothing$
$(\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{tspar})$
$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{(\mathcal{O}_{\mathsf{Sign}_r})_{r\in[R]}, \mathcal{O}_{\mathsf{Corrupt}}, \mathsf{H}}(\mathsf{vk})$
req $[\![|\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}| \leqslant T-1]\!]$
return $\mathsf{Verify}(\mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

req $[\![i \in \mathsf{HS}]\!] \wedge [\![|\mathsf{CS}| \leqslant T-1]\!]$
$\mathsf{HS} := \mathsf{HS}\backslash\{i\}$
$\mathsf{CS} := \mathsf{CS} \cup \{i\}$
return $(\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_r}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}})$

// $r \in [R]$, $\mathsf{pm}_{0,j} = \perp$ for all $j \in \mathsf{SS}$.
req $[\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{HS} \cap \mathsf{SS}]\!]$
$(\mathsf{pm}_{r,i}, \mathsf{st}_i) \xleftarrow{\$} \mathsf{Sign}_r(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$
if $[\![r = R]\!]$ **then** $\mathsf{Q_M}[\mathsf{M}] \leftarrow \mathsf{Q_M}[\mathsf{M}] \cup \{i\}$
return $\mathsf{pm}_{r,i}$

Figure 2: Adaptive security game for a R round threshold signature scheme, where H denotes the random oracle. In the above, the oracles return $\perp$ to $\mathcal{A}$ when Sign_r outputs $\perp$ for $r \in [R]$ (i.e., fail to output a protocol message or a partial signature).

$(\mathsf{sig}^*, \mathsf{M}^*)$ is considered *trivial* if *any* signing oracle was queried for message M^* at least once. If we loosely follow the classification of security definitions for threshold signatures given in [BCK+22], our definition corresponds to the notion TS-UF-1, but adapted to the adaptive setting with R rounds. For reference, the definition considered in [dPKM+24, BLT+24] corresponds to the weaker notion TS-UF-0 and we provide a definition in Appendix A.1.2

Finally, we note that it is straightforward to adapt our proofs to the strongest notion TS-UF-4 [BCK+22]. We discuss this briefly in the security proof (cf. Footnote 6). Nevertheless, we consider TS-UF-1 to for simplicity [4]. In the following, we refer with threshold signature security to TS-UF-1 (cf. Definition 2.2) unless specified otherwise.

Remark 2.4 (Security model with a stateful session identifier). Our security models above are stateless. We also consider a security model with session identifier sid. In such a security model, the adversary additionally provides sid when querying a signing oracle. If sid has already been used for honest user i in the queried round, the challenger returns $\perp$. As a consequence, all users are required to store the all used sid to avoid reuse of some sid, *i.e.*, the signers are stateful. Our 4 round threshold signature schemes $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ and $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ are proven adaptively secure in this security model with a stateful sid (see Sections 5.4 and 7.4, for the details). This stateful model is considered to show security of some threshold signature schemes, *e.g.*, the adaptively secure group-based threshold signature scheme Twinkle [BLT+24] and the selectively secure lattice-based threshold signature scheme [dPKM+24].

2.3 Linear Secret Sharing

We recall the *linear Shamir secret sharing* scheme [Sha79]. Let $N < q$ be an integer such that for distinct $i, j \in [N]$, $(i - j)$ is invertible over $\mathbb{Z}_q$. Let $S \subseteq [N]$ be a set of cardinality at least T. Then, given $i \in S$, we define the Lagrange coefficient $L_{S,i}$ as

$$L_{S,i} := \prod_{j\in S\backslash\{i\}} \frac{-j}{i-j}.$$

[4] The definition of TS-UF-4 in [BCK+22] relies on the notion of a leader request lr which is more tricky to define for R round schemes. The notion TS-UF-1 is simpler and allows us to avoid definitional subtleties in our involved proofs.

Let $s \in \mathcal{R}_q$ be a secret to be shared, $P \in \mathcal{R}_q[X]$ a degree $T-1$polynomial such that $P(0) = s$. Given any set of evaluation points $E = \{(i, y_i)\}_{i \in S}$ such that $y_i = P(i)$ for all $i \in S$, we note that

$$s = \sum_{i \in S} L_{S,i} \cdot y_i.$$

The notations naturally extend to secrets that are in vector form. With a slight abuse of notation, we say $\vec{P} \in \mathcal{R}_q^\ell[X]$ is of degree $T-1$ if each entry of $\vec{P}$ is a degree $T-1$ polynomial. Moreover, $\vec{P}(x)$ denotes the evaluation of each entry of $\vec{P}$ on the point x.

2.4 Lattices, Gaussians, and Rounding

For integers $n, q \in \mathbb{N}$ we define the ring $\mathcal{R}$ as $\mathbb{Z}[X]/(X^n+1)$ and $\mathcal{R}_q$ as $\mathcal{R}/q\mathcal{R}$. For a positive real σ, let $\rho_\sigma(\mathbf{z}) = \exp\left(-\frac{\|\mathbf{z}\|_2}{2\sigma^2}\right)$. The discrete Gaussian distribution over $\mathbb{Z}^n$ and standard deviation σ is defined by its probability distribution function: $\mathcal{D}_{S,\sigma}(\mathbf{z}) = \frac{\rho_\sigma(\mathbf{z})}{\sum_{\mathbf{z}' \in \mathbb{Z}^n} \rho_\sigma(\mathbf{z}')}$. We may simply note $\mathcal{D}_\sigma$. Lastly, we provide a useful bound on the norm of discrete Gaussians. Is due to [dPKM+24, Lemma 3.4], which is a standard tail-cut bound (see for example [MR04, Lyu12]) combined with the Minkowski's inequality.

Lemma 2.5. *For $\mathbf{s} \xleftarrow{\$} \mathcal{D}_\sigma^k$ and $v \in \mathcal{R}$, we have*

$$\Pr\left[\|v \cdot \mathbf{s}\|_2 \geqslant e^{1/4}\|v\|_1\sigma \cdot \sqrt{nk}\right] \leqslant 2^{-\frac{nk}{10}}.$$

Similarly to [dPEK+23, dPKM+24], we rely on *rounding* for efficiency purpose. Below, we provide a minimal preparation and omit the formal treatment to Appendix A.2. For a positive integer q and ν such that $q > 2^\nu$, we define $q_\nu = \lfloor q/2^\nu \rceil$. We then define the rounding function as follows:

$$\lfloor \cdot \rceil_\nu : \mathbb{Z}_q \mapsto \mathbb{Z}_{q_\nu} \quad \text{s.t.} \quad \lfloor x \rceil_\nu = \lfloor \bar{x}/2^\nu \rceil + q_\nu \mathbb{Z},$$

where $\bar{x} \in [0, 1, \cdots, q-1]$ denotes the canonical unsigned representation (or the so-called *lift*) of $x \in \mathbb{Z}_q$. The function $\lfloor \cdot \rceil_\nu$ naturally extends to vectors coefficient-wise.

2.5 Hardness Assumptions

2.5.1 Lattice-based Assumptions

We rely on two lattice-based assumptions: the *hint* MLWE (Hint-MLWE) and *self-target* MSIS (SelfTargetMSIS) assumptions. Both assumptions are reduced from the standard MLWE and MSIS assumptions, and in particular, are merely useful intermediate assumptions to aid the security proof (see Appendix A.3 for the formal statements). They have been used by the three-round selective threshold Raccoon by del Pino et al. [dPKM+24].

The Hint-MLWE problem, introduced in [KLSS23], is defined similarly to MLWE, except that the adversary also obtains some *noisy leakage* of the MLWE secrets. This is useful when invoking honest-verifier zero-knowledge, which unlike in the group setting, is not perfectly indistinguishable from the real transcript. The Hint-MLWE problem is known to be as hard as MLWE for the parameter settings we are interested in.

Definition 2.6 (Hint-MLWE). *Let ℓ, k, q, Q be integers, $\mathcal{D}, \mathcal{G}$ be probability distributions over $\mathcal{R}_q$, and $\mathcal{C}$ be a set over $\mathcal{R}_q$. The advantage of an adversary $\mathcal{A}$ against the* Hint Module Learning with Errors $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q,\mathcal{D},\mathcal{G},\mathcal{C}}$ *problem is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Hint\text{-}MLWE}}(\lambda) = \left|\Pr\left[1 \leftarrow \mathcal{A}\Big(\mathbf{A}, \mathbf{A}\cdot\mathbf{s}+\mathbf{e}, (c_i, \mathbf{z}_i, \mathbf{z}'_i)_{i\in[Q]}\Big)\right] - \Pr\left[1 \leftarrow \mathcal{A}\Big(\mathbf{A}, \mathbf{b}, (c_i, \mathbf{z}_i, \mathbf{z}'_i)_{i\in[Q]}\Big)\right]\right|$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e}) \leftarrow \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^k \times \mathcal{D}^\ell \times \mathcal{D}^k$, $c_i \leftarrow \mathcal{C}$ for $i \in [Q]$. Moreover, $(\mathbf{z}_i, \mathbf{z}'_i) = (c_i \cdot \mathbf{s} + \mathbf{r}_i, c_i \cdot \mathbf{e} + \mathbf{e}'_i)$ where $(\mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathcal{G}^\ell \times \mathcal{G}^k$ for $i \in [Q]$. The $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q,\mathcal{D},\mathcal{G},\mathcal{C}}$ *assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage. We may write* $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q,\sigma_\mathcal{D},\sigma_\mathcal{G},\mathcal{C}}$ *as a shorthand when $\mathcal{D}$ and $\mathcal{G}$ are the discrete Gaussian distributions of standard deviation $\sigma_\mathcal{D}$ and $\sigma_\mathcal{G}$, respectively.*

The *self-target* MSIS (SelfTargetMSIS) problem [DKL+18, KLS18] is a variant of the standard MSIS problem, where the problem is defined relative to some hash function modeled as a random oracle. Using the forking lemma [FS87, BN06], it is easily shown to be equivalent to the MSIS problem (see Appendix A.3). This has also been used by the signature scheme Dilithium [DKL+18], recently selected by NIST for standardisation.

Definition 2.7 (SelfTargetMSIS). *Let ℓ, k, q be integers and $B_{\mathsf{stmsis}} > 0$ be a real number. Let $\mathcal{C}$ be a subset of $\mathcal{R}_q$ and let $\mathsf{H} : \mathcal{R}_q^k \times \{0,1\}^{2\lambda} \to \mathcal{C}$ be a cryptographic hash function modeled as a random oracle. The advantage of an adversary $\mathcal{A}$ against the* Self Target MSIS *problem, noted* $\mathsf{SelfTargetMSIS}_{q,\ell,k,C,B_{\mathsf{stmsis}}}$*, is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetMSIS}}(\lambda) = \Pr\Big[\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}, (\mathsf{M}, \mathbf{z}) \xleftarrow{\$} \mathcal{A}^{\mathsf{H}}(\mathbf{A}) : (\mathsf{M}, \mathbf{z}) \in \{0,1\}^{2\lambda} \times \mathcal{R}_q^{\ell+k}$$
$$\wedge \left(\mathbf{z} = \begin{bmatrix} c \\ \mathbf{z}' \end{bmatrix}\right) \wedge (\|\mathbf{z}\|_2 \leqslant B_{\mathsf{stmsis}}) \wedge \ \ \mathsf{H}\left(\left[\mathbf{A} \mid \mathbf{I}\right] \cdot \mathbf{z},\ \mathsf{M}\right) = c\Big].$$

The $\mathsf{SelfTargetMSIS}_{q,\ell,k,C,B_{\mathsf{stmsis}}}$ *assumption states that any efficient adversary $\mathcal{A}$ has no more than negligible advantage.*

The following is an immediate application of the regularity lemma [LPR13]. [dPKM+24] provides a formal case for $\mathsf{rep} = 1$ but generalizes easily to any rep.

Lemma 2.8. *For any $\sigma > \sqrt{\frac{\log(2n\cdot\max\{\ell,k\})+\lambda}{\pi}}$ and $\sqrt{\mathsf{rep}} \cdot \sigma > 2n \cdot q^{\frac{1}{k+\ell}+\frac{2}{n\ell}}$ and $\nu < \log(q) - 2$, the following holds with all but probability $2^{-\lambda}$:*

$$\Pr_{\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}}\left[H_\infty\left(\mathcal{D}_{q,\ell,k,\sigma,\mathsf{rep},\nu}^{\mathsf{bd\text{-}MLWE}}(\mathbf{A})\right) \geqslant n-1\right] \geqslant 1 - 2^{-n+1}.$$

2.5.2 Group-Based Assumption

We use the variant of the discrete logarithm (DL) assumption: the *self-target* DL (SelfTargetDL) assumption. The SelfTargetDL problem is an interactive discrete logarithm assumption introduced in [KMP16], which we renamed for consistency with the SelfTargetMSIS problem. Originally, the hardness of the SelfTargetDL problem was analyzed in the generic group model [KMP16]. Bellare and Dai later proved that this assumption is reduced from the standard DL assumption in [BD21] (see Appendix A.4).

Let GenG be an algorithm that on input 1^λ, outputs a tuple $(\mathbb{G}, p, G)$, where G is a generator of cyclic group $\mathbb{G}$ of prime order p.

Definition 2.9. *Let $(\mathbb{G}, p, G) \leftarrow \mathsf{GenG}(1^\lambda)$. Let $\mathsf{H} : \mathbb{G}^2 \times \{0,1\}^{2\lambda} \to \mathbb{Z}_p$ be a cryptographic function modeled as a random oracle. The advantage of an adversary $\mathcal{A}$ against the* Self Target DL *problem, noted* SelfTargetDL*, is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetDL}}(\lambda) = \Pr\left[\begin{array}{c} x \xleftarrow{\$} \mathbb{Z}_p \\ X := x \cdot G, \end{array} (\mathsf{M}, z) \xleftarrow{\$} \mathcal{A}^{\mathsf{H}}(X) : \begin{array}{c} (\mathsf{M}, z) \in \{0,1\}^{2\lambda} \times \mathbb{Z}_p \\ \wedge\ \mathsf{H}(X, z\cdot G - c\cdot X, \mathsf{M}) = c \end{array}\right].$$

The SelfTargetDL *assumption states that any efficient adversary $\mathcal{A}$ has no more than negligible advantage.*

3 Construction of Our 3-Round Threshold Raccoon

In this section, we present our 3-round threshold signature scheme $\mathsf{TRaccoon}_{\mathsf{3\text{-}rnd}}^{\mathsf{sel}}$, a thresholdized version of the NIST submission Raccoon by del Pino et al. [dPEK+23]. We show in Section 4 that $\mathsf{TRaccoon}_{\mathsf{3\text{-}rnd}}^{\mathsf{sel}}$ is selectively secure under the Hint-MLWE and MSIS assumptions. Our protocol is only a slight adaptation of the 3-round threshold Raccoon by del Pino et al. [dPKM+24], and notably, the hardness assumptions and the concrete parameters we rely on are exactly the same as theirs. The main novelty is the new security analysis due to the modification in the scheme.

3.1 Parameters and Preparations

For reference, we provide the parameters of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ in Table 3. Our protocol relies on the same parameters as those by del Pino et al. [dPKM+24, Section 7.1]. For completeness, we provide a candidate parameter selection in Appendix D.

Parameter	Explanation
$\mathcal{R}_q$	Polynomial ring $\mathcal{R}_q = \mathbb{Z}[X]/(q, X^n + 1)$
(k, ℓ)	Dimension of public matrix $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$
$(\mathcal{D}_{\mathbf{t}}, \sigma_{\mathbf{t}})$	Gaussian distribution with width $\sigma_{\mathbf{t}}$ used for the verification key $\mathbf{t}$
$(\mathcal{D}_{\mathbf{w}}, \sigma_{\mathbf{w}})$	Gaussian distribution with width $\sigma_{\mathbf{w}}$ used for the commitment $\mathbf{w}$
$\nu_{\mathbf{t}}$	Amount of bit dropping performed on verification key
$\nu_{\mathbf{w}}$	Amount of bit dropping performed on (aggregated) commitment
$(q_{\nu_{\mathbf{t}}}, q_{\nu_{\mathbf{w}}})$	Rounded moduli satisfying $(q_{\nu_{\mathbf{t}}}, q_{\nu_{\mathbf{w}}}) := (\lfloor q/2^{\nu_{\mathbf{t}}} \rceil, \lfloor q/2^{\nu_{\mathbf{w}}} \rceil) = (\lceil q/2^{\nu_{\mathbf{t}}} \rceil, \lceil q/2^{\nu_{\mathbf{w}}} \rceil)$
$(\mathcal{C} \subset R_q, W)$	Challenge set $\{c \in \mathcal{R}_q \mid \lVert c \rVert_\infty = 1 \wedge \lVert c \rVert_1 = W\}$ s.t. $\lvert\mathcal{C}\rvert \geqslant 2^\lambda$
B	Two-norm bound on the signature

Table 3: Overview of parameters used in $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$, and $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$.

We also prepare a helper algorithm called *zero share* ($\mathsf{ZeroShare}$) to simplify the presentation of the protocol. While the underlying property of $\mathsf{ZeroShare}$ has been implicitly used in prior threshold signatures based on $\mathsf{Raccoon}$ [dPKM+24, EKT24][5], we make this explicit. We believe this abstraction fosters a more intuitive understanding of our protocol, particularly for our adaptively secure 5-round variant. Concretely, each user is given a tuple of random strings of the form $\vec{\mathsf{seed}}_i = (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]}$ at the setup. For any set $\mathsf{SS} \subseteq [N]$, we denote $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ as the tuple $(\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in\mathsf{SS}}$. The helper algorithm $\mathsf{ZeroShare}$ is defined with respect to a random oracle $\mathsf{H}_{\mathsf{mask}}$ with range $\mathcal{R}_q^\ell$. For any $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ and string $x \in \{0,1\}^*$, it is defined as follows:

$$\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) := \sum_{j\in\mathsf{SS}\setminus\{i\}} \big(\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, x) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, x)\big).$$

Looking ahead, we use $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x)$ to mask the response $\mathbf{z}_i \in \mathcal{R}_q^\ell$. We will extensively use the following easy to check fact:

$$\sum_{i\in\mathsf{SS}} \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) = \sum_{i\in\mathsf{SS}} \mathbf{\Delta}_i = \mathbf{0}. \tag{1}$$

Moreover, observe that from an adversary without knowledge of $(\vec{\mathsf{seed}}_i[\mathsf{SS}])_{i\in\mathsf{SS}}$, each $\mathbf{\Delta}_i$ is distributed uniformly over $\mathcal{R}_q^\ell$ conditioned on their sum being $\mathbf{0}$. In the remainder of the document, we may call $\mathbf{\Delta}_i$ as a mask or zero share interchangeably.

3.2 Construction

The construction of our 3-round threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ is provide in Fig. 3. Our scheme uses three hash functions modeled as a random oracle in the security proof. $\mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ is used to generate the hash commitment. $\mathsf{H}_c : \{0,1\}^* \to \mathcal{C}$ is used to generate the random challenge polynomial for which the users reply with a response. $\mathsf{H}_{\mathsf{mask}} : \{0,1\}^* \to \mathcal{R}_q^\ell$ is used to generate the random vectors to mask the individual response. We give a brief overview of the protocol below.

The setup algorithm outputs system parameters $\mathsf{tspar} = (\mathbf{A}, N, T)$ for some random $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}$. The verification key is tspar and a (rounded) MLWE instance $\mathbf{t} = \lfloor \mathbf{A}\mathbf{s} + \mathbf{e} \rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}_{q_{\nu_{\mathbf{t}}}}^k$. The secret keys are of the

[5] To be precise, the way del Pino et al. [dPKM+24] implements the $\mathsf{ZeroShare}$ algorithm is slightly different from our abstraction. However, this is a superficial difference and our formalization allows for a slightly better communication cost as we remove broadcasting one element in $\mathcal{R}_q^\ell$.

form $\mathsf{sk}_i = (\mathbf{s}_i, \vec{\mathsf{seed}}_i)$, where $\mathbf{s}_i$ is a share of $\mathbf{s}$ and $\vec{\mathsf{seed}}_i$ are seeds for ZeroShare. Importantly, the verification key and the verification algorithm are identical to Raccoon [dPEK+23]. The signing protocol proceeds in 3 rounds as follows:

Round 1. Signer i samples a commitment $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$, where $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$, and outputs a hash commitment $\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i)$.

Round 2. Signer i obtains the hash commitments $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ and opens cmt_i by sending $\mathbf{w}_i$.

Round 3. Signer i checks that for all $j \in \mathsf{SS}$, the hash commitments cmt_j are opened correctly by signer j, *i.e.*, $\mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \mathbf{w}_j)$. If the check passes, it computes the aggregate commitment $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$, else it aborts. Then, it sets $\mathsf{ctnt}_{\mathbf{z}} := \mathsf{SS}||\mathsf{M}||(\mathsf{cmt}_j, \mathbf{w}_j)_{j \in \mathsf{SS}}$, computes the challenge $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ and outputs the *masked* response $\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$, where $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}^{\ell}_q$ and $L_{\mathsf{SS},i}$ is the Lagrange coefficient.

The aggregate algorithm computes the response $\mathbf{z} := \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{z}}_j$, $\mathbf{y} := \lfloor \mathbf{A}\mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}}$, *hint* $\mathbf{h} := \mathbf{w} - \mathbf{y}$, and outputs $\mathsf{sig} = (c, \mathbf{z}, \mathbf{h})$, where $\mathbf{w}$ and c are computed as above. Importantly, the final signature is a valid Raccoon signature. While it looks quite complicating on first glance, the hint $\mathbf{h}$ and multiplying of $c \cdot \mathbf{t}$ by $2^{\nu_{\mathbf{w}}}$ are simply there to compensate for the error induced by the rounding in the verification key $\mathbf{t}$ and the aggregate commitment $\mathbf{w}$. Specifically, these are simply used for optimization purpose, inherited by Raccoon [dPEK+23], and will not appear for instance in the threshold Schnorr signatures (see Section 7).

Lastly, let us summarize the main differences between the 3-round selective threshold signature by del Pino et al. [dPKM+24].

- They compute "half" of the zero share (i.e., $\sum_{j \in \mathsf{SS} \setminus \{i\}} \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, x)$) in the first round and the other half in the third round. By altering the input to the algorithm ZeroShare, we are able to push everything into the third round. This allows us to reduce the first round communication cost by $\mathcal{R}^{\ell}_q$ per user.

- They require a unique session identifier sid to be broadcast at the beginning of first round to derive the zero share, and importantly, parties must maintain state so as not to reuse sid. Our protocol replaces sid with $(\mathbf{w}_j)_{j \in \mathsf{SS}}$ defined in the third round. Using the fact that a signer i never samples the same $\mathbf{w}_i$, the scheme can be made stateless.

- They require a standard signature scheme or a MAC to sign the second round message. This is used to argue consistency of the users' view in the security proof; indeed, without it the scheme becomes insecure. We are able to remove this by relying on the above modification of the zero shares and arguing through a new security proof.

We emphasize that the first two tricks have been used in a recent two-round (stateless) selective threshold signature by [EKT24]. What is new to this work is our security proof, which is non-trivialized by the fact that we have one extra round; [EKT24] relies on a new *algebraic one-more* MLWE assumption and can be viewed as a lattice-variant of the two-round threshold Schnorr protocol Frost [KG20].

3.3 Correctness

The following establishes the correctness of our protocol.

Lemma 3.1 (Correctness). *The 3-round threshold signature* $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ *in Fig. 3 is correct if* $\nu_{\mathbf{w}} \geqslant 4$ *and:*

$$B = e^{1/4} \cdot (W\,\sigma_{\mathbf{t}} + \sqrt{T}\,\sigma_{\mathbf{w}})\sqrt{n(k+\ell)} + (W \cdot 2^{\nu_{\mathbf{t}}} + 2^{\nu_{\mathbf{w}}+1}) \cdot \sqrt{nk}.$$

Proof. The proof is almost identical to those provided in del Pino et al. [dPKM+24, Lemma 7.1] as we use the same parameters. The only difference between their protocol and ours is how the shares are generated.

$\mathsf{Setup}(1^\lambda, N, T)$

$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}$
$\mathsf{tspar} := (\mathbf{A}, N, T)$
return tspar

$\mathsf{KeyGen}(\mathsf{tspar})$

$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$
$\mathbf{t} := \lfloor \mathbf{A}\mathbf{s} + \mathbf{e} \rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}_{q_{\nu_{\mathbf{t}}}}^{k}$
for $i \in [N]$ **do**
  for $j \in [N]$ **do**
    $\mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$
    $\mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$
$(\vec{\mathsf{seed}}_i)_{i\in[N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]}\right)_{i\in[N]}$
$\vec{P} \xleftarrow{\$} \mathcal{R}_q^{\ell}[X]$ with $\deg(\vec{P}) = T - 1, \vec{P}(0) = \mathbf{s}$
$(\mathbf{s}_i)_{i\in[N]} := (\vec{P}(i))_{i\in[N]}$
$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$
$(\mathsf{sk}_i)_{i\in[N]} := \left(\mathbf{s}_i, \vec{\mathsf{seed}}_i\right)_{i\in[N]}$
return $(\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]})$

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{b,j})_{(b,j)\in[5]\times\mathsf{SS}})$

parse $(\mathbf{w}_j, \tilde{\mathbf{z}}_j)_{j\in\mathsf{SS}} \leftarrow (\mathsf{pm}_{2,j}, \mathsf{pm}_{3,j})_{j\in\mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$
$\mathbf{z} := \sum_{j\in\mathsf{SS}} \tilde{\mathbf{z}}_j \in \mathcal{R}_q^{\ell}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$
$\mathbf{y} := \lfloor \mathbf{A}\mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^{k}$
$\mathbf{h} := \mathbf{w} - \mathbf{y} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^{k}$
return $\mathsf{sig} := (c, \mathbf{z}, \mathbf{h})$

$\mathsf{Verify}(\mathsf{vk}, \mathsf{M}, \mathsf{sig})$

parse $(c, \mathbf{z}, \mathbf{h}) \leftarrow \mathsf{sig}$
$c' := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \lfloor \mathbf{A}\mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}} + \mathbf{h})$
if $[\![c = c']\!] \wedge [\![\|(\mathbf{z}, 2^{\nu_{\mathbf{w}}} \cdot \mathbf{h})\|_2 \leqslant B]\!]$ **then**
  return 1
return 0

$\mathsf{Sign}_1(\mathsf{vk}, i, \mathsf{sk}_i, \mathsf{st}_i)$

$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$
$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^{k}$
$\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)\}$
return $(\mathsf{pm}_{1,i} := \mathsf{cmt}_i, \mathsf{st}_i)$

$\mathsf{Sign}_2(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

req $[\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
req $[\![(\mathsf{pm}_{1,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
pick $(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{1,i} = \mathsf{cmt}_i$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
return $(\mathsf{pm}_{2,i} := \mathbf{w}_i, \mathsf{st}_i)$

$\mathsf{Sign}_3(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

req $[\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{2,i}, \cdot) \in \mathsf{st}_i]\!]$
parse $(\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$
parse $(\mathbf{w}_j)_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}\backslash\{i\}}$
pick $(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{2,i} = \mathbf{w}_i$
req $[\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \mathbf{w}_j)]\!]$
$\mathsf{ctnt}_{\mathbf{z}} := \mathsf{SS} || \mathsf{M} || (\mathsf{cmt}_j, \mathbf{w}_j)_{j\in\mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^{k}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ // $c \in \mathcal{C}$
$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$
$\tilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
return $(\mathsf{pm}_{3,i} := \mathbf{z}_i, \mathsf{st}_i)$

Figure 3: Our 3-round selective secure threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. The differences between the 3-round selective threshold signature by del Pino et al. [dPKM+24] is highlighted in blue.

By using Eq. (1) and using the correctness of the Shamir secret sharing scheme, the response can be written as follows:

$$\mathbf{z} := \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{z}}_j = \sum_{j \in \mathsf{SS}} c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i = c \cdot \mathbf{s} + \sum_{j \in \mathsf{SS}} \mathbf{r}_i$$

Since this is exactly the same as those computed in [dPKM+24], correctness is satisfied under the same parameters as theirs. □

4 Selective Security of Our 3-Round Threshold Raccoon

In this section we provide the proof of our 3-round threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ in Fig. 3. Due to the proof being quite long and involved, we first provide a proof overview in Section 4.1. The formal security proof appears in Appendix E.1. Below, we provide the main theorem statement of this section establishing the security of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. The parameters for which the following theorem hold is provided in Appendix D.

Theorem 4.1. *The 3-round threshold signature* $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ *in Fig. 3 is selectively secure under the* $\mathsf{Hint\text{-}MLWE}$ *and* $\mathsf{SelfTargetMSIS}$ *assumptions.*

Formally, for any N and T with $T \leqslant N$ and an adversary $\mathcal{A}$ against the selective security game making at most Q_{H_c}, $Q_{\mathsf{H}_{\mathsf{com}}}$, $Q_{\mathsf{H}_{\mathsf{mask}}}$, and Q_{S} queries to the random oracles H_c, $\mathsf{H}_{\mathsf{com}}$, $\mathsf{H}_{\mathsf{mask}}$, and the signing oracle, respectively, there exists adversaries $\mathcal{B}$ and $\mathcal{B}'$ against the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$ *and* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B_{\mathsf{stmsis}}}$ *problems, respectively, such that*

$$\mathsf{Adv}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T) \leqslant Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(1^\lambda) + \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(1^\lambda) + \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{2^{n-1}} + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda),$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ and $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$. From Lemma A.10, we can replace $\mathcal{B}'$ by an adversary $\mathcal{B}''$ against the $\mathsf{MSIS}_{q,\ell+1,k,2B}$ *problem with $\mathsf{Time}(\mathcal{B}'') \approx 2 \cdot \mathsf{Time}(\mathcal{B}')$ such that*

$$\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}''}(\lambda)} + \frac{Q_{\mathsf{H}_c}}{|\mathcal{C}|}.$$

4.1 Proof Overview

Let us provide the proof overview. Our strategy is to use a hybrid argument to transition to a game, where the challenger simulates the signing oracles without the secret key shares $(\mathbf{s}_i)_{i \in \mathsf{HS}}$. We then embed an $\mathsf{SelfTargetMSIS}$ problem into the verification key and extract a solution from the forgery. We denote by $\mathcal{A}$ the adversary, and by sHS (resp. sCS) the subset of honest users $\mathsf{sHS} = \mathsf{SS} \cap \mathsf{HS}$ (resp. corrupt users $\mathsf{sCS} = \mathsf{SS} \cap \mathsf{CS}$) queried to the signing oracle. We describe the hybrids below.

Game_1 **to** Game_3**: postpone sampling** $\mathbf{w}_i$**.** In Game_1 to Game_3, the challenger delays sampling $\mathbf{w}_i$ until the 2nd round. Instead of committing to $\mathbf{w}_i$ in $\mathcal{O}_{\mathsf{Sign}_1}$, the challenger samples a random $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$. In $\mathcal{O}_{\mathsf{Sign}_2}$, it samples $\mathbf{w}_i = \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$ as before and programs $\mathsf{H}_{\mathsf{com}}$ such that $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(\mathbf{w}_i, i)$. Also, the challenger aborts in case there is a collision in $\mathsf{H}_{\mathsf{com}}$ and prepares some tables for bookkeeping in $\mathcal{O}_{\mathsf{Sign}_2}$. In more detail:

Game_1**:** This game is identical to the real game.

Game_2 : In this game, the challenger outputs a fresh $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_1}$ and delays sampling $\mathbf{w}_i$ until $\mathcal{O}_{\mathsf{Sign}_2}$, where $\mathsf{H}_{\mathsf{com}}$ is programmed such that $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(\mathbf{w}_i, i)$. Since $\mathbf{w}_i$ has high min-entropy, this change is unnoticeable.

Game_3: In this game, the challenger aborts if there is a collision in $\mathsf{H}_{\mathsf{com}}$. Note that the output of $\mathsf{H}_{\mathsf{com}}$ is of bit-size 2λ, so we can conclude that the abort probability is negligible by a birthday bound argument.

Game_4: In this game, the challenger introduces tables $\mathsf{UnOpenedHS}$ and $\mathsf{SumComRnd}$ in $\mathcal{O}_{\mathsf{Sign}_2}$, indexed by $\mathsf{ctnt}_{\mathbf{w}} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$. Note that $\mathsf{ctnt}_{\mathbf{w}}$ represents the signer's view in $\mathcal{O}_{\mathsf{Sign}_2}$. None of these tables are accessed, so the view of $\mathcal{A}$ remains identical, but we describe their meaning below. If $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$, then some honest user started round 2 with $\mathsf{ctnt}_{\mathbf{w}}$ and $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ stores the set of honest users $\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ that have not passed round 2 with $\mathsf{ctnt}_{\mathbf{w}}$. The table $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the sum of the commitment $\mathbf{w}_i$'s randomness $\mathbf{r}_i$ of honest users $i \in \mathsf{sHS}\backslash\widetilde{\mathsf{sHS}}_{\mathbf{w}}$, *i.e.*, honest users that have opened their commitment cmt_i via $\mathcal{O}_{\mathsf{Sign}_2}$ with respect to $\mathsf{ctnt}_{\mathbf{w}}$.

Before proceeding, let us remark that the adversary cannot invoke $\mathcal{O}_{\mathsf{Sign}_2}$ (resp. $\mathcal{O}_{\mathsf{Sign}_3}$) twice with the same value $\mathsf{ctnt}_{\mathbf{w}}$ for a honest user $i \in \mathsf{sHS}$ except with negligible probability. This is because user i samples cmt_i with high min-entropy in $\mathcal{O}_{\mathsf{Sign}_1}$ at random and cmt_i is part of $\mathsf{ctnt}_{\mathbf{w}}$. In some sense, this notion of uniqueness of $\mathsf{ctnt}_{\mathbf{w}}$ is a core reason we can omit the requirement of a unique session identifier (which was required in [dPKM+24]). This is captured in the following remark.

Remark 4.2. The adversary cannot invoke $\mathcal{O}_{\mathsf{Sign}_2}$ (resp. $\mathcal{O}_{\mathsf{Sign}_3}$) twice with the same value $\mathsf{ctnt}_{\mathbf{w}}$.

Game_5 **to** Game_9**: sample** $\widetilde{\mathbf{z}}_i$ **at random.** In Game_5 to Game_9, the challenger transitions to a game where $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ is sampled at random. Roughly, this is possible because $\widetilde{\mathbf{z}}_i = \mathbf{z}_i + \mathbf{\Delta}_i$ is masked by $\mathbf{\Delta}_i = \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$. Note that not all responses $\widetilde{\mathbf{z}}_i$ are random in the view of $\mathcal{A}$: the last mask $\mathbf{\Delta}_i$ is fully determined by $(\mathbf{\Delta}_j)_{j\in\mathsf{SS}\backslash\{i\}}$ since all masks sum up to $\mathbf{0}$ (cf. Eq. (1)). Thus, when i is the last user to sign with respect to $\mathsf{ctnt}_{\mathbf{w}}$, then the response $\widetilde{\mathbf{z}}_i$ is setup *consistently* in $\mathcal{O}_{\mathsf{Sign}_3}$, *i.e.*, it respects the constraint $\mathbf{\Delta}_i = -\sum_{j\in\mathsf{SS}\backslash\{i\}} \mathbf{\Delta}_j$.

Note that while in the protocol, the value $\mathsf{ctnt}_{\mathbf{z}}$ serves as input to $\mathsf{ZeroShare}$, the value $\mathsf{ctnt}_{\mathbf{w}}$ uniquely defines $\mathsf{ctnt}_{\mathbf{z}}$ implicitly due to the binding of the hash commitments. This allows us to interchange $\mathsf{ctnt}_{\mathbf{w}}$ and $\mathsf{ctnt}_{\mathbf{z}}$ within the security proof when analyzing the distribution of $\mathbf{\Delta}_i$.

Also, observe that if the views of honest users $\mathsf{ctnt}_{\mathbf{w}}$ were distinct in round 2, then the value $\mathsf{ctnt}_{\mathbf{z}}$ is distinct in round 3, so all $\mathbf{\Delta}_i$ are distributed at random. This observation is the core reason we can simulate later: if the view $\mathsf{ctnt}_{\mathbf{w}}$ is identical amongst honest users in round 2, then we can invoke HVZK with respect to the verification key $\mathbf{t}$ and program H_c accordingly. If the view is distinct in round 2, then the reduction cannot simulate, but since all responses in round 3 are random, this is not required. Our proof structure handles this by only sampling $\widetilde{\mathbf{z}}_i$ consistently if i is the last user in round 3 with respect to $\mathsf{ctnt}_{\mathbf{w}}$. Below, we show that the last masked response is distributed as follows:

$$\widetilde{\mathbf{z}}_i := c\cdot\mathbf{s} - c\sum_{j\in\mathsf{sCS}} L_{\mathsf{SS},j}\cdot\mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sHS}\backslash\{i\}} \widetilde{\mathbf{z}}_j - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j, \tag{2}$$

where $\widetilde{\mathbf{z}}_j$ is the masked response of user i with $\mathsf{ctnt}_{\mathbf{w}}$. Recall that $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j\in\mathsf{sHS}} \mathbf{r}_j$ stores the sum of the honest commitment $\mathbf{w}_j$'s randomness.

Game_5: In this game, the challenger introduces tables $\mathsf{UnSignedHS}$, $\mathsf{Mask}_{\mathbf{z}}$ and $\mathsf{MaskedResp}$ indexed by $\mathsf{ctnt}_{\mathbf{w}}$. None of the tables impact the view of $\mathcal{A}$ but we detail their meaning. If $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$, then it stores the set of honest users $\widetilde{\mathsf{sHS}}_{\mathbf{z}}$ that have not passed round 3 with $\mathsf{ctnt}_{\mathbf{w}}$. The tables $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ store the mask $\mathbf{\Delta}_i$ and the masked response $\widetilde{\mathbf{z}}_i$ of user i in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$.

Game_6: In this game, we expand the definition of $\mathsf{ZeroShare}$ in $\mathcal{O}_{\mathsf{Sign}_3}$. The challenger samples partial masks $\mathbf{m}_{i,j} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$ and $\mathbf{m}_{j,i} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$ for $j \in \mathsf{SS}\backslash\{i\}$, then sets $\mathbf{\Delta}_i = \sum_{j\in\mathsf{SS}\backslash\{i\}}(\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$. This change is purely conceptual.

Game_7: In this game, the challenger samples the partial masks $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ at random for $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$ (and programs $\mathsf{H}_{\mathsf{mask}}$ accordingly). Both games are identically distributed in the view of $\mathcal{A}$ because seeds $\mathsf{seed}_{i,j}$ and $\mathsf{seed}_{j,i}$ are hidden from from $\mathcal{A}$ and the partial masks have not yet been evaluated for users in $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}$. In the detailed proof, we argue this formally via Remark 4.2.

Game_8**:** In this game, the challenger samples the mask $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ in $\mathcal{O}_{\mathsf{Sign}_3}$, except if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$, *i.e.*, user i is the last signer with respect to $\mathsf{ctnt}_{\mathbf{w}}$. If i is the last signer, it sets

$$\mathbf{\Delta}_i = - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j. \tag{3}$$

Both games are identically distributed because: (1) If i is not the last signer for $\mathsf{ctnt}_{\mathbf{w}}$, then $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \setminus \{i\}$ contains at least another honest signer j, so $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ are sampled at random from the previous game. in particular, $\mathbf{\Delta}_i = \sum_{j \in \mathsf{SS} \setminus \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$ is distributed uniform random over $\mathcal{R}_q^\ell$. (2) If i is the last signer for $\mathsf{ctnt}_{\mathbf{w}}$, then all partial masks are fully determined. Since we have that $\sum_{j \in \mathsf{SS}} \mathbf{\Delta}_j = \mathbf{0}$ (cf. Eq. (1)), we can reorder the expression to obtain Eq. (3) via the identity $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] = \mathbf{\Delta}_j$. Lastly, note that the masked response for each signer i is still defined as in the real game:

$$\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i. \tag{4}$$

Game_9**:** In this game, the challenger aborts if in $\mathcal{O}_{\mathsf{Sign}_3}$ the value of challenge c is not unique amongst invocations with $\mathsf{ctnt}_{\mathbf{w}}$. The view of $\mathcal{A}$ remains identical conditioned on no abort. Note that the hash commitments $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ in $\mathsf{ctnt}_{\mathbf{w}}$ fix the commitments $\mathbf{w}_i$ due to binding. Since $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$, where $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$, the value of c is fixed by $\mathsf{ctnt}_{\mathbf{w}}$ and the abort probability is negligible.

Game_{10} : In this game, the challenger instead samples $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ in $\mathcal{O}_{\mathsf{Sign}_3}$, except if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$. If $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$, then it sets $\widetilde{\mathbf{z}}_i$ according to Eq. (2). We can show that Game_{10} and Game_9 are identically distributed by looking at an intermediate game $\mathsf{Game}_{9,*}$, where instead of sampling $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^\ell$, we sample $\mathbf{\Delta}_i^* \xleftarrow{\$} \mathcal{R}_q^\ell$ and set $\mathbf{\Delta}_i := \mathbf{\Delta}_i^* - (c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i)$. This intermediate game is identically distributed to Game_9 as both $\mathbf{\Delta}_i$ and $\mathbf{\Delta}_i^*$ are uniform random. Also, observe that in $\mathsf{Game}_{9,*}$, we have that $\widetilde{\mathbf{z}}_i = \mathbf{\Delta}_i^*$ if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$ due to Eq. (4), and $\widetilde{\mathbf{z}}_i$ as in Eq. (2) otherwise. To see the latter, first substitute $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] = \mathbf{\Delta}_j = \widetilde{\mathbf{z}}_j - (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j)$ for all $j \in \mathsf{sHS} \setminus \{i\}$ in Eq. (3), then substitute the resulting identity for $\mathbf{\Delta}_i$ in the identity of $\widetilde{\mathbf{z}}_i$ in Eq. (4). Finally, using the equality $\mathbf{s} = \sum_{j \in \mathsf{SS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j$ yields Eq. (2). We point out that for the last step, it is crucial that the value c is identical for all users in round 3 with $\mathsf{ctnt}_{\mathbf{w}}$. This is guaranteed by the abort condition added in the previous game.

Game_{11} **to** Game_{14}**: Invoke HVZK.** In games Game_{11} to Game_{14}, we invoke HVZK to simulate the commitment $\mathbf{w}_i$ for the last signer i in round 2 with respect to the verification key $\mathbf{t}$. This later allows to compute the response $\widetilde{\mathbf{z}}_h$ of the last signer h in round 3 without secret key $\mathbf{s}$. At the end of Game_{14}, the challenger no longer requires the secret key $\mathbf{s}$ to simulate the signing oracles.

Game_{11}: In this game, the challenger chooses a random challenge $c \xleftarrow{\$} \mathcal{C}$ before sampling the commitment $\mathbf{w}_i$ for the last signer i in round 2 with $\mathsf{ctnt}_{\mathbf{w}}$. Before outputting $\mathbf{w}_i$, the challenger retrieves the other commitments $\mathbf{w}_j$ for $j \in \mathsf{SS} \setminus \{i\}$ from cmt_j by searching through all the random oracle queries made to $\mathsf{H}_{\mathsf{com}}$. If $\mathbf{w}_j$ are found, it programs H_c such that $\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) = c$, where $\mathbf{w} = \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$. Further, the challenger aborts if some $\mathbf{w}_j$ was not found in round 2, but $\mathcal{O}_{\mathsf{Sign}_3}$ is invoked for all honest users with $\mathsf{ctnt}_{\mathbf{w}}$.

Note that since $\mathbf{w}_i$ has high min-entropy, H_c was never queried with $(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ before it is programmed, so the view of $\mathcal{A}$ is identically distributed. Let us analyze the probability of an abort. Since the challenger checks in $\mathcal{O}_{\mathsf{Sign}_3}$ whether each $\mathbf{w}_j$ is committed in cmt_j, the adversary must have found a preimage for cmt_j between the last call to $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ and the first call to $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$. This happens with negligible probability.

Game_{12}: In this game, the challenger invokes HVZK with respect to the verification key $\mathbf{t}$ to simulate the commitment $\mathbf{w}_i$ of the last honest signer i in round 2, and computes the response $\widetilde{\mathbf{z}}_h$ of the last honest user h in round 3 in a different manner. Note that the last signers in round $\mathcal{O}_{\mathsf{Sign}_2}$ and $\mathcal{O}_{\mathsf{Sign}_3}$ are not required to be the same, *i.e.*, it can be that $h \neq i$. In more detail, in $\mathcal{O}_{\mathsf{Sign}_2}$ if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, after sampling the challenge c, the challenger simulates the commitment-response pair $(\mathbf{w}_i, \mathbf{z}_*)$, where $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_i$. Also, $\mathbf{r}_i$ is *not* added to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$. Instead, the challenger computes the last honest response, *i.e.*, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{h\}$ in $\mathcal{O}_{\mathsf{Sign}_3}$, via the simulated response $\mathbf{z}_*$

$$\widetilde{\mathbf{z}}_h := \mathbf{z}_* - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{h\}} \widetilde{\mathbf{z}}_j - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j,$$

where the simulated response is chosen as above.

The above identity for $\widetilde{\mathbf{z}}_h$ is obtained by rewriting Eq. (2) using $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_i$. Note that it is crucial that the challenge c—precomputed when the last user i opens its commitment in round 2 to define $\mathbf{z}_*$—must be identical to the challenge c in round 3 for the last user h. This is guaranteed by the abort condition in the previous game.

Game_{13}: In this game, the challenger replaces $\mathbf{t}$ in the verification key with $\mathbf{t} := \left[\hat{\mathbf{t}}\right]_{\nu_{\mathbf{t}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{t}}}}$, where $\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}^k_q$. Also, when setting up the secret key shares, it samples $\mathbf{s}_i$ for $j \in \mathsf{CS}$ at random, and omits the honest secret key share in $\mathsf{vk}_i = (\perp, \vec{\mathsf{seed}}_i)$ for $j \in \mathsf{HS}$.

Observe that the challenger in Game_{11} uses the secret key $\mathbf{s}$ only when computing the simulated response $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_i$ in $\mathcal{O}_{\mathsf{Sign}_2}$ for a challenge c randomly chosen by the challenger. Under Hint-MLWE, Game_{12} and Game_{13} are indistinguishable. Note that simulated responses $\mathbf{z}_*$ correspond to the provided hints in Hint-MLWE.

Reduction from SelfTargetMSIS. In Game_{13}, the challenger no longer requires the secret key $\mathbf{s}$ to run the game. When considering the same unforgeability notion as [dPKM+24] (cf. Appendix A.1.2), the rest of the proof is identical to theirs. For this notion, the reduction is guaranteed that the forgery's message M^* is never queried to any signing oracle. This allows the reduction to simulate H_c in such a way that it is consistent with the SelfTargetMSIS oracle H for the forgery's challenge c^*, and consequently we can recompute a SelfTargetMSIS solution. We refer to the proof by del Pino et al. [dPKM+24, Lemma 7.4] for more details.

However, if we target the stronger notion of security (c.f. Section 2.2.1), there remains a subtlety. Observe that H_c is also programmed by the challenger with a random value c in Game_{13} when simulating a commitment. This happens *before* the last round and there is no more guarantee that for $\mathcal{A}$'s forgery, the associated hash value c^* is consistent with H. The proof fails.

Instead, we prepare a last intermediate game, where the challenger guesses the hash query associated to the forgery's output. Since there are H_c queries in total, this guess is correct with probability $1/Q_{\mathsf{H}_c}$. Further, before simulating a commitment $\mathbf{w}_i$ and programming H_c with a random challenge c, the challenger checks if the corresponding H_c query is identical to the guessed query. If that is the case, the challenger uses the provided oracle H to sample the challenge c instead. As a result, the simulated response $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is *not* of the correct form. But because the adversary never queries the third round for the last honest user, the value $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is never used and the view of $\mathcal{A}$ remains unchanged [6].

With this modification, it is straightforward to adapt the proof by del Pino et al. [dPKM+24, Lemma 7.4] as our scheme has the same verification algorithm as theirs and the final step merely consists of extracting a solution from the forgery. From Lemma A.10, such an adversary can be used to construct an adversary against the more standard MSIS problem via the forking lemma [FS87, BN06].

Lastly, we remark that we incur a loss of $1/Q_{\mathsf{H}_c}$ in the advantage of solving the SelfTargetMSIS problem when considering the stronger notion of security. However, this does not affect the concrete parameters of

[6]Observe that for this argument, it is sufficient that forgeries are considered trivial iff $\mathsf{ctnt}_{\mathbf{w}}$ including the message M^* was queried for all honest users in sHS in the last round within a single signing session.

del Pino et al. [dPKM+24] as they consider the *working factor* to deduce the bit-security. That is, they set the parameters so that the probability of success of an adversary (i.e., advantage of solving $\mathsf{SelfTargetMSIS}$) divided by the running time, which is larger than Q_{H_c}, is less than $2^{-\lambda}$ for λ-bits security.

5 Construction of Our 5-Round Threshold Raccoon

In this section, we present our 5-round threshold signature scheme $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. We prove that $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is adaptive secure under the $\mathsf{Hint\text{-}MLWE}$ and MSIS assumptions.

5.1 Parameters and Preparations

The used parameters are identical to $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ (cf. Section 3) and the threshold protocol by del Pino et al. [dPKM+24]. We refer the readers to Table 3 for the parameters. Moreover, the correctness and security proof relies on the same parameter selection as well, which are provided in Appendix D for completeness.

As in $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, we rely on masking via the helper algorithm $\mathsf{ZeroShare}$. We slightly adapt the definition to our needs. For any set $\mathsf{SS} \subseteq [N]$, we denote $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ as the tuple $(\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in \mathsf{SS}}$. As before, these seeds are given to the users during key generation. The helper algorithm $\mathsf{ZeroShare}$ defined with respect to a random oracle $\mathsf{H}_{\mathsf{mask}}$. Here, we require that $\mathsf{H}_{\mathsf{mask}}$ has variable range. Then, for any $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ and string $x \in \{0,1\}^*$, $\mathsf{ProgramZeroShare}$ is defined as follows:

$$\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) := \sum_{j \in \mathsf{SS} \setminus \{i\}} \big(\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, x) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, x)\big),$$

where $\mathsf{H}_{\mathsf{mask}}$ outputs vectors over $\mathcal{R}_q^k$ and $\mathcal{R}_q^\ell$ when the first bit of x is 0 and 1, respectively. Looking ahead, we use $\mathsf{ZeroShare}$ to mask the commitment $\mathbf{w}_i \in \mathcal{R}_q^k$ and response $\mathbf{z}_i \in \mathcal{R}_q^\ell$. Note that Eq. (1) still holds (*i.e.*, $\sum_{i \in \mathsf{SS}} \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) = \mathbf{0}$) with this minor modification.

In addition, we also require an EUF-CMA secure signature scheme $\mathsf{S} = (\mathsf{KeyGen}_\mathsf{S}, \mathsf{Sign}_\mathsf{S}, \mathsf{Verify}_\mathsf{S})$. Looking ahead, we use S in the security proof to ensure that the view of all honest users is consistent in the round where the commitment cmt_i is revealed.

5.2 Construction

The construction of our 5-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is provided in Figs. 4 and 5 in detail. Our scheme uses three hash functions modeled as a random oracle in the security proof. $\mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ is used to generate the hash commitment. $\mathsf{H}_c : \{0,1\}^* \to \mathcal{C}$ is used to generate the random challenge polynomial for which the users reply with a response. $\mathsf{H}_{\mathsf{mask}} : \{0,1\}^* \to \mathcal{R}_q^k \cup \mathcal{R}_q^\ell$ is used to generate the random vectors to mask the individual commitment or response via $\mathsf{ZeroShare}$. We give a brief overview below.

The setup algorithm outputs system parameters $\mathsf{tspar} = (\mathbf{A}, N, T)$ for some random $\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times \ell}$. The public key is identical to a Raccoon public $\mathsf{vk} = (\mathsf{tspar}, \mathbf{t})$ with Raccoon secret key $\mathbf{s}$, and the secret keys are of the form $\mathsf{sk}_i = (\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},j})_{j \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i)$, where $\mathbf{s}_i$ is a share of $\mathbf{s}$, $(\mathsf{vk}_{\mathsf{S},j})_{j \in [N]}$ are S verification keys with secret keys $(\mathsf{sk}_{\mathsf{S},j})_{j \in [N]}$, and $\vec{\mathsf{seed}}_i$ are seeds for $\mathsf{ZeroShare}$. Verification is identical to Raccoon verification. The signing protocol proceeds in 5 rounds as follows:

Round 1. Signer i outputs a random string $\mathsf{str}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$.

Round 2. Signer i samples a commitment $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$, where $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^\ell \times \mathcal{D}_{\mathbf{w}}^k$. Then, it sets $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$ and computes a mask $\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$. Signer i uses $\widetilde{\mathbf{\Delta}}_i$ to compute a masked commitment $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$. It outputs a hash commitment $\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \widetilde{\mathbf{w}}_i)$.

Round 3. Signer i sets $\mathsf{M_S} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$ and outputs a signature $\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign_S}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M_S})$ on $\mathsf{M_S}$.

Round 4. Signer i checks that for $j \in \mathsf{SS}$, all the signatures $\sigma_{\mathsf{S},j}$ are valid with respect to verification key $\mathsf{vk}_{\mathsf{S},j}$ of signer j and message $\mathsf{M_S}$. This check guarantees that the view of all honest signers is consistent at this point. If the check succeeds, signer i opens cmt_i by outputting $\widetilde{\mathbf{w}}_i$, else it aborts the signing session.

Round 5. Signer i checks that for all $j \in \mathsf{SS}$, the hash commitments cmt_j are opened correctly by signer j, *i.e.*, $\mathsf{cmt}_j = \mathsf{H_{com}}(j, \widetilde{\mathbf{w}}_j)$. If the check passes, it computes the sum $\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_\mathbf{w}}$, else it aborts. Then, it sets $\mathsf{ctnt}_\mathbf{z} := 1\|\mathsf{SS}||\mathsf{M}||(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}\|(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$, computes the challenge $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ and outputs the masked response $\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$, where $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_\mathbf{z}) \in \mathcal{R}_q^\ell$.

An aggregated signature is computed via $\mathbf{z} := \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{z}}_j$, $\mathbf{y} := \lfloor \mathbf{Az} - 2^{\nu_\mathbf{t}} \cdot c \cdot \mathbf{t} \rceil_{\nu_\mathbf{w}}$ and $\mathbf{h} := \mathbf{w} - \mathbf{y}$, where as above $\mathbf{w} = \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_i \right\rceil_{\nu_\mathbf{w}}$ and $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$. The Raccoon signature $(c, \mathbf{z}, \mathbf{h})$ is output.

Let us highlight the main differences to our 3-round selective threshold signature $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ from Section 3. These changes are made to prove adaptive security. We provide some intuition for our choices.

Masking the commitments. In $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, the signer sends the commitments $\mathbf{w}_i$ in clear. In security proof, one $\mathbf{w}_i$ per session is simulated via HVZK. In that case, the reduction does *not* know its randomness $\mathbf{r}_i$ and $\mathbf{e}'_i$. But in the adaptive setting, the adversary $\mathcal{A}$ is allowed to corrupt honest users after $\mathbf{w}_i$ is output. Then, we have to provide the randomness $(\mathbf{r}_i, \mathbf{e}'_i)$ to $\mathcal{A}$, so the reduction fails in the adaptive setting.

In $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$, the signer masks the commitment $\mathbf{w}_i$ with a fresh mask $\widetilde{\mathbf{\Delta}}_i = \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_\mathbf{w})$ and sends $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$ instead of $\mathbf{w}_i$, where $\mathsf{ctnt}_\mathbf{w} = 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$. Here, str_j are random strings exchanged in the additional initial round. Note that the entropy of str_i ensures that each mask $\widetilde{\mathbf{\Delta}}_i$ is random in each signing session [7]. With our modification, the values $(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$ output in round 4 only reveal the sum $\mathbf{w} = \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_i \right\rceil_{\nu_\mathbf{w}}$. This allows the reduction to simulate a single commitment $\mathbf{w}_*$ via HVZK and sample the other commitments $\mathbf{w}_j = \mathbf{A} \cdot \mathbf{r}_j + \mathbf{e}'_j$ honestly with known $(\mathbf{r}_j, \mathbf{e}'_j)$ for honest users. When some user i is corrupted, we can choose a honest commitment $\mathbf{w}_j$ and program $\mathsf{H_{mask}}$ in such a way that $\widetilde{\mathbf{w}}_i = \mathbf{w}_j + \widetilde{\mathbf{\Delta}}_i$. Since at most $T-1$ honest users are corrupted, we never have to reveal the randomness of the simulated $\mathbf{w}_*$. Formalizing this vague argument is a core technical challenge in the security proof.

Authenticating the views. In $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, the security proof crucially relies on the fact that $\mathsf{ctnt}_\mathbf{w}$ in round 2 fixes the value of $\mathsf{ctnt}_\mathbf{z}$ used in round 3. The security proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ also requires this to hold, but here, $\mathsf{ctnt}_\mathbf{w}$ does not contain the commitments $(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$. Thus, $\mathsf{ctnt}_\mathbf{w}$ itself does not determine $\mathsf{ctnt}_\mathbf{z}$ yet. Instead, we ensure that for each $\mathsf{ctnt}_\mathbf{w}$, there is a unique $\mathsf{ctnt}_\mathbf{z}$ in round 5 via signature-based authentication of the views in round 4.

5.3 Correctness

We establish correctness of our protocol.

Lemma 5.1 (Correctness). *The 5-round threshold signature* $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ *in Figs. 4 and 5 is correct if* $\nu_\mathbf{w} \geqslant 4$ *and:*

$$B = e^{1/4} \cdot (W\,\sigma_\mathbf{t} + \sqrt{T}\,\sigma_\mathbf{w})\sqrt{n(k+\ell)} + (W \cdot 2^{\nu_\mathbf{t}} + 2^{\nu_\mathbf{w}+1}) \cdot \sqrt{nk}.$$

Proof. The only difference between $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ and $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is that the commitment $\mathbf{w}$ is computed in a different manner and the additional signature verification step. By using Eq. (1), the commitment $\mathbf{w}$ can be rewritten as

$$\mathbf{w} = \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_i \right\rceil_{\nu_\mathbf{w}} = \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \right\rceil_{\nu_\mathbf{w}} = \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_i \right\rceil_{\nu_\mathbf{w}}.$$

[7] More concretely, it is guaranteed that $\mathsf{ZeroShare}$ is never invoked more than once with the same input $\mathsf{ctnt}_\mathbf{w}$ for each honest user. This allows the reduction to program $\mathsf{H_{mask}}$ freely in the security proof.

$\mathsf{Setup}(1^\lambda, N, T)$

$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}$

$\mathsf{tspar} := (\mathbf{A}, N, T)$

$\textbf{return } \mathsf{tspar}$

$\mathsf{Verify}(\mathsf{vk}, \mathsf{M}, \mathsf{sig})$

$\textbf{parse } (c, \mathbf{z}, \mathbf{h}) \leftarrow \mathsf{sig}$

$c' := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \lfloor \mathbf{Az} - 2^{\nu_\mathbf{t}} \cdot c \cdot \mathbf{t} \rceil_{\nu_\mathbf{w}} + \mathbf{h})$

$\textbf{if } [\![c = c']\!] \wedge [\![\|(\mathbf{z}, 2^{\nu_\mathbf{w}} \cdot \mathbf{h})\|_2 \leqslant B]\!] \textbf{ then}$

$\quad \textbf{return } 1$

$\textbf{return } 0$

$\mathsf{KeyGen}(\mathsf{tspar})$

$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_\mathbf{t}^\ell \times \mathcal{D}_\mathbf{t}^k$

$\mathbf{t} := \lfloor \mathbf{As} + \mathbf{e} \rceil_{\nu_\mathbf{t}} \in \mathcal{R}_{q_{\nu_\mathbf{t}}}^k$

$\textbf{for } i \in [N] \textbf{ do}$

$\quad (\mathsf{vk}_{\mathsf{S},i}, \mathsf{sk}_{\mathsf{S},i}) \xleftarrow{\$} \mathsf{KeyGen}_\mathsf{S}(1^\lambda)$

$\quad \textbf{for } j \in [N] \textbf{ do}$

$\quad\quad \mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$

$\quad\quad \mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$

$(\vec{\mathsf{seed}}_i)_{i\in[N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]} \right)_{i\in[N]}$

$\vec{P} \xleftarrow{\$} \mathcal{R}_q^\ell[X] \text{ with } \deg(\vec{P}) = T-1, \vec{P}(0) = \mathbf{s}$

$(\mathbf{s}_i)_{i\in[N]} := (\vec{P}(i))_{i\in[N]}$

$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$

$(\mathsf{sk}_i)_{i\in[N]} := \left(\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},i})_{i\in[N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i \right)_{i\in[N]}$

$\textbf{return } (\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]})$

Figure 4: Setup, KeyGen, and Verify for our five round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. The differences to $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ are highlighted in blue (except changes with respect to the signer states).

This is exactly how $\mathbf{w}$ is computed in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{3\text{-}rnd}}$. Also, by correctness of the signature scheme S, the signatures $\sigma_{\mathsf{S},i}$ on M_S verify correctly in Sign_4 when computed as in Sign_3. We remark that $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ uses the parameters of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ and while the value of $\mathsf{ctnt}_\mathbf{z}$ differs in Sign_5 compared to $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, it still holds that $\sum_{i\in\mathsf{SS}} \mathbf{\Delta}_i = \mathbf{0}$ due to Eq. (1). Combining the above arguments with correctness of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ established in Lemma 3.1, the statement follows. □

5.4 Our 4-Round Raccoon Threshold Signature

We can easily transform the *stateless* 5 round scheme in Figs. 4 and 5 into the *stateful* 4 round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ by using the session identifier sid that is never reused. Our 5 round scheme needs to share the string str_i in the first round, that is used to generate the mask $\tilde{\mathbf{\Delta}}$ for the commitment. Importantly, the same str_i is never reused except with negligible probability. The idea of the transform is simply to use non-reusable session identifier sid, instead of sharing str_i. Specifically, the first round is no longer executed, and users take sid as input instead of $(\mathsf{str}_j)_{j\in\mathsf{SS}}$ and proceed as in 5 round scheme by replacing $(\mathsf{str}_j)_{j\in\mathsf{SS}}$ with sid. Note that users need to maintain state so that the same sid is never reused. This transformation preserves security since sid provides the same non-reuseability as $(\mathsf{str}_j)_{j\in\mathsf{SS}}$. We provide more details in Appendix B.

6 Adaptive Security of Our 5 Round Threshold Raccoon

In this section, we provide the proof of our 5-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ in Figs. 4 and 5. The proof is involved and technical, so we first provide a proof overview in Section 6.2. The formal security proof is provided in Appendix E.2. Below, we state the main theorem establishing adaptive security of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. The parameters for which the following theorem hold is provided in Appendix D.

Theorem 6.1. *The 5-round threshold signature* $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ *in Figs. 4 and 5 is adaptive secure under the* $\mathsf{Hint\text{-}MLWE}$ *and* MSIS *assumptions.*

$\mathsf{Sign}_1(\mathsf{vk}, i, \mathsf{sk}_i, \mathsf{st}_i)$

$\mathsf{str}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{\mathsf{str}_i\}$
$\textbf{return } (\mathsf{pm}_{1,i} := \mathsf{str}_i, \mathsf{st}_i)$

$\mathsf{Sign}_2(\mathsf{vk}, \mathsf{SS}, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
$\textbf{req } [\![\mathsf{pm}_{1,i} \in \mathsf{st}_i]\!]$
$\textbf{pick } \mathsf{str}_i \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{1,i} = \mathsf{str}_i$
$\textbf{parse } (\mathsf{str}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\textbf{parse } (\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$
$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$
$\tilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$
$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$
$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$
$\tilde{\mathbf{w}}_i := \mathbf{w}_i + \tilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$
$\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \tilde{\mathbf{w}}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{\mathsf{str}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } (\mathsf{pm}_{2,i} := \mathsf{cmt}_i, \mathsf{st}_i)$

$\mathsf{Sign}_3(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \cdot, \mathsf{pm}_{2,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i$
$\textbf{parse } (\mathsf{cmt}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\qquad \text{with } \mathsf{pm}_{2,i} = \mathsf{cmt}_i$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_{\mathsf{S}}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_{\mathsf{S}})$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } (\mathsf{pm}_{3,i} := \sigma_{\mathsf{S},i}, \mathsf{st}_i)$

$\mathsf{Sign}_4(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i$
$\qquad \text{with } \mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$
$\textbf{parse } (\sigma_{\mathsf{S},j})_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\textbf{req } [\![\forall j \in \mathsf{SS}\setminus\{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } (\mathsf{pm}_{3,i} := \tilde{\mathbf{w}}_i, \mathsf{st}_i)$

$\mathsf{Sign}_5(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{parse } (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i$
$\qquad \text{with } \mathsf{pm}_{4,i} = \tilde{\mathbf{w}}_i$
$\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \tilde{\mathbf{w}}_j)]\!]$
$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \tilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) \quad /\!/\ c \in \mathcal{C}$
$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$
$\tilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } (\mathsf{pm}_{5,i} := \tilde{\mathbf{z}}_i, \mathsf{st}_i)$

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{b,j})_{(b,j) \in [5] \times \mathsf{SS}})$

$\textbf{parse } (\tilde{\mathbf{w}}_j, \tilde{\mathbf{z}}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{4,j}, \mathsf{pm}_{5,j})_{j \in \mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \tilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}}$
$\mathbf{z} := \sum_{j \in \mathsf{SS}} \tilde{\mathbf{z}}_j \in \mathcal{R}_q^{\ell}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$
$\mathbf{y} := \lfloor \mathbf{A}\mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
$\mathbf{h} := \mathbf{w} - \mathbf{y} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
$\textbf{return } \mathsf{sig} := (c, \mathbf{z}, \mathbf{h})$

Figure 5: The Signing protocol of our five round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. In the above, $L_{\mathsf{SS},i}$ denotes the Lagrange coefficient of user i in the set $\mathsf{SS} \subseteq [N]$ (see Section 2.3 for the definition). **pick** X **from** Y denotes the process of picking an element X from the set Y. The differences to $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ are highlighted in blue (except changes with respect to the signer states).

Formally, for any N and T with $T \leqslant N$ and an adversary $\mathcal{A}$ against the adaptive security game making at most Q_{H_c}, $Q_{\mathsf{H}_{\mathsf{com}}}$, $Q_{\mathsf{H}_{\mathsf{mask}}}$, and Q_{S} queries to the random oracles H_c, $\mathsf{H}_{\mathsf{com}}$, and $\mathsf{H}_{\mathsf{mask}}$ and the signing oracle, respectively, there exists adversaries $\mathcal{B}$, $\mathcal{B}'$, and $\mathcal{B}_{\mathsf{S}}$ against the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$, $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B_{\mathsf{stmsis}}}$ problems, and the unforgeability of signatures, respectively, such that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T) \leqslant{}& Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(1^\lambda) + \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(1^\lambda) + N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S},\mathcal{B}_{\mathsf{S}}}(\lambda)\\ &+ \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{2^{n-1}} + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{Q_{\mathsf{S}}^2 + (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda),\end{aligned}$$

where $\mathsf{Time}(\mathcal{B}), \mathsf{Time}(\mathcal{B}'), \mathsf{Time}(\mathcal{B}_{\mathsf{S}}) \approx \mathsf{Time}(\mathcal{A})$. *From Lemma A.10, we can replace $\mathcal{B}'$ by an adversary $\mathcal{B}''$ against the* $\mathsf{MSIS}_{q,\ell+1,k,2B}$ *problem with* $\mathsf{Time}(\mathcal{B}'') \approx 2 \cdot \mathsf{Time}(\mathcal{B}')$ *such that*

$$\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}''}(\lambda)} + \frac{Q_{\mathsf{H}_c}}{|\mathcal{C}|}.$$

6.1 Intuition

The security proof is involved. Before we provide details, let us discuss our simulation strategy on a high level. Roughly, our goal is to construct a simulator for the unforgeability game such that:

(1) The simulator simulates the signing oracles without knowing the secret key $\mathbf{s}$.

(2) When signer i is corrupted, the simulator provides a partial secret key $\mathbf{s}_i$ and signer's state st_i to the adversary $\mathcal{A}$ that is *consistent* with the public key and all previous signing oracle queries.

Given such a simulator, it is possible to reduce security to MSIS via rewinding by embedding an MSIS challenge into the public key $\mathbf{t}$ [8].

For (1), we proceed similarly to the proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ (cf. Section 3). That is, the simulator invokes HVZK on the *full* public key $\mathbf{t}$ to simulate a commitment $\mathbf{w}_*$ with valid response $\mathbf{z}_*$ for some challenge c_*. Then, $\mathbf{w}_*$ is embedded in some honest signer's masked commitment $\widetilde{\mathbf{w}}_i$, the challenge c_* is embedded in H_c, and $\mathbf{z}_*$ is embedded into some honest signer's response. Note that because adaptive corruptions are allowed, the embedding of $\mathbf{w}_*$ and $\mathbf{z}_*$ is left *implicit* which is possible due to masking.

For (2), we use the fact that the masks $\widetilde{\mathbf{\Delta}}_i$ (resp. $\mathbf{\Delta}_i$) are distributed uniform if at most $T-1$ mask values are known. This allows the simulator to sample the signer's secret share $\mathbf{s}_i$ and state st_i when the corruption is made. Only then the random oracle $\mathsf{H}_{\mathsf{mask}}$ is programmed by the simulator for consistency with previous signing queries.

We elaborate below. Note that the overview is heavily simplified for the sake of readability.

Notation. For this exposition, we by denote $\mathbb{V}[x]$ a variable x that is statically hidden and we simply write x if x is already statistically determined. For instance, at the beginning of the unforgeability game, we denote by $\mathbb{V}[\mathbf{s}_i]$ the variable that represents the partial secret shares of user i, and $\mathbf{s}$ the (determined) full secret key. Let us also note that for any subset $\mathcal{S} := \{\mathbb{V}[\mathbf{s}_i]\}_{i \in [N]}$ of partial shares of $\mathbf{s}$ with $|\mathcal{S}| \leqslant T-1$, the shares $\mathbb{V}[s_i] \in \mathcal{S}$ follow a uniform distribution over $\mathcal{R}_q^\ell$.

Preparation. Before we dive into our techniques, let us make some observations. First of all, let us assume that the public key $\mathbf{t}$ fixes the secret key $\mathbf{s}$ statistically. Notice that the partial secret keys $\mathbb{V}[\mathbf{s}_i]$ must always satisfy the following constraint for any signer set SS of size T:

$$\mathbf{s} = \sum_{i \in \mathsf{SS}} L_{\mathsf{SS},i} \mathbb{V}[\mathbf{s}_i]. \tag{5}$$

[8] An attentive reader might observe that this is *not* immediate if the reduction relies on oracle queries that influence the winning condition as, *e.g.*, in *one-more* assumptions. Our simulator has no such dependencies.

In the above, notice that each of the partial secret keys $\mathbb{V}[\mathbf{s}_i]$ are information-theoretically hidden from the adversary at the beginning of the game. However, the adversary knows that the observed secret shares $\mathbf{s}_i$ (due to later corruption queries) must satisfy Eq. (5). As recalled above, since the adversary $\mathcal{A}$ observes at most $T-1$ values $\mathbf{s}_i$ throughout the game, these are distributed uniformly at random over $\mathcal{R}_q^\ell$ in the view of $\mathcal{A}$.

Moreover, the adversary $\mathcal{A}$ learns partial signing transcripts $\mathsf{trans}_i := (\mathsf{cmt}_i, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \widetilde{\mathbf{z}}_i)$ of some honest user $i \in \mathsf{HS}$ in a signer set SS throughout the game. The simulator needs to ensure that trans_i follows the distribution of the real game. Observe that the adversary $\mathcal{A}$ knows that the following equations hold:

$$\widetilde{\mathbf{w}}_i = \mathbb{V}[\mathbf{w}_i] + \mathbb{V}[\widetilde{\mathbf{\Delta}}_i], \tag{6}$$

$$\mathbb{V}[\mathbf{w}_i] = \mathbf{A}\mathbb{V}[\mathbf{r}_i] + \mathbb{V}[\mathbf{e}'_i], \tag{7}$$

$$\widetilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbb{V}[\mathbf{s}_i] + \mathbb{V}[\mathbf{r}_i] + \mathbb{V}[\mathbf{\Delta}_i], \tag{8}$$

$$0 = \sum_{j \in \mathsf{SS}} \mathbb{V}[\widetilde{\mathbf{\Delta}}_j] = \sum_{j \in \mathsf{SS}} \mathbb{V}[\mathbf{\Delta}_j]. \tag{9}$$

Notice that the partial secret $\mathbb{V}[\mathbf{s}_i]$, commitment $\mathbb{V}[\mathbf{w}_i]$, its randomness $(\mathbb{V}[\mathbf{r}_i], \mathbb{V}[\mathbf{e}'_i])$, the mask $\mathbb{V}[\widetilde{\mathbf{\Delta}}_i]$ for the commitment and the mask $\mathbb{V}[\mathbf{\Delta}_i]$ for the response are information-theoretically hidden at this point, except for the fact that they satisfy Eqs. (5) to (9).

Before we discuss the simulator, let us briefly discuss the core challenge. The simulator needs to provide responses $\widetilde{\mathbf{z}}_i$ that follow Eq. (8) without fixing the values of $\mathbb{V}[\mathbf{s}_i]$ and $\mathbb{V}[\mathbf{r}_i]$. (Else, we cannot embed a hard problem into the public key $\mathbf{t}$ later.) But on corruption queries, the adversary expects (amongst other values) $\mathbf{r}_i$ and $\mathbf{s}_i$ which are consistent with the above equations. Roughly, this is possible because each signing query introduces a fresh variables $\mathbb{V}[\widetilde{\mathbf{\Delta}}_i], \mathbb{V}[\mathbf{\Delta}_i]$. By reprogramming the random oracle $\mathsf{H}_{\mathsf{mask}}$, the simulator can recompute a well-distributed state st_i and secret key sk_i. Importantly, this is done *only* when a user is corrupted.

Simulating the Signing Oracle. Let us sketch how the simulator answers the signing oracles. In round 1, 2 and 3, the simulator computes appropriate $\mathsf{str}_i, \mathsf{cmt}_i$ and $\sigma_{\mathsf{S},i}$, respectively, that follow the correct distribution. That is:

Round 1: The simulator outputs a string str_i sampled at random as in the real game.

Round 2: The simulator outputs a commitment cmt_i sampled at random. (Later, in round 4, the oracle $\mathsf{H}_{\mathsf{com}}$ is programmed such that cmt_i commits to $\widetilde{\mathbf{w}}_i$ which is sampled only in round 4.)

Round 3: In round 3, the simulator computes a signature on its public view as in the real game.

Let us briefly comment on the consequence of the first three rounds. The entropy of str_i ensures that the masks $\mathbb{V}[\widetilde{\mathbf{\Delta}}]$ and $\mathbb{V}[\mathbf{\Delta}]$ are computed by evaluating $\mathsf{H}_{\mathsf{mask}}$ on fresh inputs, and thus the masks are distributed uniformly and independently at random conditioned on Eq. (9). The hash commitment cmt_i ensures that the simulator knows the malicious commitments $\widetilde{\mathbf{w}}_i$ chosen by the adversary $\mathcal{A}$ before the simulator chooses its own commitments (by observing $\mathcal{A}$'s $\mathsf{H}_{\mathsf{com}}$ queries). The signature $\sigma_{\mathsf{S},i}$ ensures that the (public) view of each honest signer is consistent with the view of other signers within a signing session (through the check of signature validity in round 4).

Before we continue, notice that in round 4, the simulator knows adversary $\mathcal{A}$'s masked commitments $\widetilde{\mathbf{w}}_i$ and since the public view of all honest signers is identical, so in particular $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$, these values $\widetilde{\mathbf{w}}_i$ are identical for each honest user. In round 4 and round 5, the simulator needs to ensure that Eqs. (5) to (9) are satisfied. As mentioned above, $\mathbb{V}[\widetilde{\mathbf{\Delta}}]$ and $\mathbb{V}[\mathbf{\Delta}]$ are distributed randomly conditioned on Eq. (9). Thus, by reordering Eqs. (6) and (8), we know that $\widetilde{\mathbf{w}}_i$ and $\widetilde{\mathbf{z}}_i$ are distributed at random conditioned on:

$$\sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j = \mathbf{w}, \tag{10}$$

$$\sum_{j \in \mathsf{SS}} \widetilde{\mathbf{z}}_j = c \cdot \mathbf{s} + \mathbf{r}, \tag{11}$$

where $\mathbf{r} := \sum_{j \in \mathsf{SS}} \mathbb{V}[\mathbf{r}_j]$ and $\mathbf{w} := \sum_{j \in \mathsf{SS}} \mathbb{V}[\mathbf{w}_i]$ are determined after all honest users finished round 4. Note that above, we used correctness of Shamir's secret sharing and the fact that c is identical for all honest users.

Let us briefly remark that Eqs. (10) and (11) are heavily simplified. Notably, the simulator does not know the randomness $\mathbf{r}_i$ of malicious users or whether a malicious hash commitment cmt_i even commits to a valid commitment $\widetilde{\mathbf{w}}_i$. In the proof overview (cf. Section 6.2), we provide exact equations that the simulator can evaluate to sample the last $\widetilde{\mathbf{w}}_i$ and $\widetilde{\mathbf{z}}_i$ accordingly. For now, let us ignore this technicality.

Let us finally describe how the simulator proceeds in round 4 and 5. Note that the simulator does not know the partial secret key $\mathbf{s}$. Let H be the number of honest signers sHS in the signer set SS. Roughly:

Round 4: For all but the last honest signer, the simulator outputs $\widetilde{\mathbf{w}}_i$ sampled at random. For the last honest signer, the simulator samples $\widetilde{\mathbf{w}}_i$ according to Eq. (10). That is, the simulator honestly samples $H-1$ commitments $\mathbf{w}_i = \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$, and simulates one commitment $\mathbf{w}_*$ with challenge c and response $\mathbf{z}_*$ via HVZK on the public key $\mathbf{t}$. In particular, the response is (implicitly) of the form $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_*$, where $\mathbf{w}_* = \mathbf{A}\mathbf{r}_* + \mathbf{e}^*_i$, but the randomness of $\mathbf{w}_*$ is unknown to the simulator. Note that with these choices, Eq. (7) is satisfied—but the commitments $\mathbf{w}_i$ are *not* yet attributed to a honest user. Also, note that this fixes the determined values

$$\mathbf{r} = \mathbf{r}_* + \sum_{j \in [H-1]} \mathbf{r}_j + \sum_{j \in \mathsf{sCS}} \mathbf{r}_j,$$
$$\mathbf{w} = \mathbf{w}_* + \sum_{j \in [H-1]} \mathbf{w}_j + \sum_{j \in \mathsf{sCS}} \mathbf{w}_j,$$

where (as mentioned above) we assume for simplicity that $\mathbf{w}_j$ and $\mathbf{r}_j$ of dishonest users $j \in \mathsf{sCS}$ of the signing session are known. Then, the simulator then embeds c into H_c at the right location [9]. Finally, the simulator and outputs $\widetilde{\mathbf{w}}_i := \mathbf{w} - \sum_{j \in \mathsf{SS} \setminus \{i\}} \widetilde{\mathbf{w}}_j$.

Round 5: For all but the last honest signer, the simulator outputs $\widetilde{\mathbf{z}}_i$ sampled at random. For the last honest signer, the simulator samples $\widetilde{\mathbf{z}}_i$ according to Eq. (11). That is, the signer outputs $\widetilde{\mathbf{z}}_i = \mathbf{z}_* + \sum_{j \in \mathsf{SS} \setminus \{i\}} \mathbf{r}_i$.

Note that the simulator does *not* program $\mathsf{H}_{\mathsf{mask}}$ in the signing oracle. Also, we stress that the commitments $\mathbf{w}_i$ are *not* yet attributed to a honest user.

Simulating the Corruption Oracle. When a signer i is corrupted, the simulator first picks a random partial share $\mathbf{s}_i$. As discussed above, this is sufficient for Eq. (5). Then, the simulator needs to compute a state st_i such that all signing sessions are consistent with $\mathbf{s}_i$ and st_i. For each session, the simulator picks one of the (not yet chosen) commitments $\mathbf{w}_i$ sampled with randomness $(\mathbf{r}_i, \mathbf{e}'_i)$ in round 4. As mentioned above, $\mathbf{w}_i$ satisfies Eq. (7). But since after corruption of signer i, the adversary can compute $\widetilde{\boldsymbol{\Delta}}$ and $\mathbf{w}_i$ and $\widetilde{\mathbf{w}}_i$ is fixed, Eq. (6) is *not* yet satisfied. For this, the simulator crucially programs $\mathsf{H}_{\mathsf{mask}}$. That is, by Eq. (6), we have $\widetilde{\boldsymbol{\Delta}}_i = \widetilde{\mathbf{w}}_i - \mathbf{w}_i$, and the simulator programs $\mathsf{H}_{\mathsf{mask}}$ such that $\widetilde{\boldsymbol{\Delta}}_i = \mathsf{ZeroShare}(\overrightarrow{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ holds. Here, it is important to note that at most $T-1$ users are corrupted. Thus, the simulator never has to reveal the randomness of the simulated commitment $\mathbf{w}_*$ and more subtly, $\widetilde{\boldsymbol{\Delta}}_i$ is distributed uniform, which allows to reprogram $\mathsf{H}_{\mathsf{mask}}$ accordingly. Finally, it remains to ensure that Eq. (8) holds. Again, this is possible by programming $\mathsf{H}_{\mathsf{mask}}$ accordingly, that is such that $\boldsymbol{\Delta}_i = \widetilde{\mathbf{z}}_i - c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i - \mathbf{r}_i$.

6.2 Proof Overview

Let us provide the proof overview. As in the proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, our strategy is to use a hybrid argument to transition to a game, where the challenger simulates the signing oracles without the secret key $\mathbf{s}$. We then

[9] This is possible because $\mathbf{w}$ is determined at this point. We omit details.

embed an $\mathsf{SelfTargetMSIS}$ problem into the verification key and extract a solution from the forgery. The core difference to the security proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ is that the challenger provides a corruption oracle to the adversary $\mathcal{A}$. This means we have to setup the signer states st_i in accordance with the adversaries view when user i is corrupted. As before, we denote by sHS (resp. sCS) the subset of honest users $\mathsf{sHS} = \mathsf{SS} \cap \mathsf{HS}$ (resp. corrupt users $\mathsf{sCS} = \mathsf{SS} \cap \mathsf{CS}$) queried to the signing or corruption oracle. We describe the hybrids below. Since the proof is involved, the arguments and hybrids are simplified for the sake of readability. We encourage the reader to first look at the proof overview for $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ in Section 4.1 since the techniques are related—but simpler in the selective setting.

Game_1 **to** Game_5**:** In Game_1 to Game_5, the challenger delays sampling $\widetilde{\mathbf{w}}_i$ until the 4th round or when a user is corrupted. That is, the challenger outputs a random $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_2}$. In $\mathcal{O}_{\mathsf{Sign}_4}$, it samples $\mathbf{w}_i = \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$ and sets $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$. Then, it programs $\mathsf{H}_{\mathsf{com}}$ such that $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(\mathbf{w}_i, i)$ and outputs $\widetilde{\mathbf{w}}_i$. This is also done if i is corrupted for all signer states before round 4. Further, the challenger aborts in case there is a collision in $\mathsf{H}_{\mathsf{com}}$ and ensures that all sampled str_i are unique.

Game_1**:** This game is identical to the real game.

Game_2**:** In this game, the challenger aborts in $\mathcal{O}_{\mathsf{Sign}_1}$ if str_i was previously sampled. The abort probability is negligible because str_i has high min-entropy.

Game_3**:** In this game, the challenger outputs a fresh $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_2}$. The preimage for cmt_i is computed either in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$ as described above. Since $\mathbf{w}_i$ has high min-entropy, this change is not noticable.

Game_4**:** In this game, the challenger aborts if there is a collision in $\mathsf{H}_{\mathsf{com}}$. We can show with as birthday bound argument that this happens with negligible probability.

Game_5**:** In this game, the challenger aborts if it the adversary invokes $\mathcal{O}_{\mathsf{Sign}_4}$ but it did not sign M_{S} in $\mathcal{O}_{\mathsf{Sign}_3}$ for all honest users. Under EUF-CMA security of S, this happens with negligible probability.

Before we proceed, let us discuss the implication of Game_5. Roughly, M_{S} corresponds to the view of each honest user in round 4 before the commitments are opened. The consistency check ensures that $\mathcal{O}_{\mathsf{Sign}_5}$ is not invoked unless all honest users share an identical view in round 4 with respect to $\mathsf{ctnt}_{\mathbf{w}}$ *before* their commitments are opened. This is essential for simulation later.

As in the selective proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, the adversary cannot invoke each signing oracle twice with the same value $\mathsf{ctnt}_{\mathbf{w}}$ for a honest user $i \in \mathsf{sHS}$ except with negligible probability. Here, this is because user i samples str_i with high min-entropy in $\mathcal{O}_{\mathsf{Sign}_1}$ at random and str_i is part of $\mathsf{ctnt}_{\mathbf{w}}$. This is captured in the following remark.

Remark 6.2. The adversary cannot invoke $\mathcal{O}_{\mathsf{Sign}_r}$ twice with the same value $\mathsf{ctnt}_{\mathbf{w}}$ for $r \in [5]$.

Game_6 **to** Game_{11}**:** In Game_6 to Game_{11}, the challenger transitions to a game where $\widetilde{\mathbf{w}}_j \xleftarrow{\$} \mathcal{R}_q^k$ is sampled at random, except the last revealed commitment $\widetilde{\mathbf{w}}_i$ is sampled *consistently*. Note that the adversary $\mathcal{A}$ can request the opening $\widetilde{\mathbf{w}}_i$ of hash commitment cmt_i either via a call to $\mathcal{O}_{\mathsf{Sign}_4}$ by following the protocol *or* via a corruption query [10]. Again, consistently means that $\widetilde{\mathbf{w}}_i$ respects the constraint $\widetilde{\mathbf{\Delta}}_i = -\sum_{j \in \mathsf{SS} \setminus \{i\}} \widetilde{\mathbf{\Delta}}_j$. Below, we show that the last masked commitment $\widetilde{\mathbf{w}}_i$ is distributed as follows:

$$\widetilde{\mathbf{w}}_i = \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \widetilde{\mathbf{w}}_j, \tag{12}$$

where $\widetilde{\mathbf{w}}_j$ is the masked commitment of user i with $\mathsf{ctnt}_{\mathbf{w}}$ and $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}} \mathbf{w}_j$ stores the sum of the honest commitments $(\mathbf{w}_j)_{j \in \mathsf{sHS}}$. Further, since all $\widetilde{\mathbf{w}}_j$ but the last are random, the challenger

[10] The challenger identifies that user i is the last user to open cmt_i via $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, where $\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ is introduced in Game_6.

can delay sampling the honest commitments $(\mathbf{w}_j)_{j\in\mathsf{sHS}}$ until the last signer opens its commitment cmt_i. Also, observe that the protocol messages $(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{sHS}}$ of round 4 reveal only the sum of the commitments $\mathbf{w}_j$ but *not* their attribution to users, *i.e.*, which user sampled which commitment $\mathbf{w}_j$. Thus, when the last cmt_i is opened to $\widetilde{\mathbf{w}}_i$, the challenger generates $|\mathsf{sHS}|$-many honest commitments at once, stores them in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] = \{\widetilde{\mathbf{w}}_j\}_{j\in\mathsf{SS}}$ and their sum in $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$. The challenger then carefully attributes commitments from the set $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ in round 5 or when a user between round 4 and 5 is corrupted. In the latter case, the reduction also programs the oracle $\mathsf{H}_{\mathsf{mask}}$ so that the users state is consistent with the choice. Finally, the challenger also sets up a table $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j\in\mathsf{sHS}} \mathbf{r}_j$ that the sum of the honest commitments $\mathbf{w}_j$'s randomness for later.

Game_6 : In this game, we introduce some tables $\mathsf{InitializeOpen}$, $\mathsf{UnOpenedHS}$, $\mathsf{Mask}_{\mathbf{w}}$ and $\mathsf{MaskedCom}$ indexed by $\mathsf{ctnt}_{\mathbf{w}}$. None of the tables impact the view of $\mathcal{A}$ but we detail their meaning. If $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$, then $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ stores the set of honest users $\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ that have not passed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$, *i.e.*, the hash commitment cmt_i is not yet opened. The tables $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ store the mask $\widetilde{\mathbf{\Delta}}_i$ and masked commitment $\widetilde{\mathbf{w}}_i$ of user i in $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$, respectively.

Game_7**:** In this game, we expand the definition of $\mathsf{ZeroShare}$ in $\mathcal{O}_{\mathsf{Sign}_4}$ and $\mathcal{O}_{\mathsf{Corrupt}}$. The challenger samples partial masks $\widetilde{\mathbf{m}}_{i,j} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}})$ and $\widetilde{\mathbf{m}}_{j,i} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}})$ for $j \in \mathsf{SS}\backslash\{i\}$, then sets $\mathbf{\Delta}_i = \sum_{j\in\mathsf{SS}\backslash\{i\}}(\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j})$. This change is purely conceptual.

Game_8**:** In this game, the challenger samples the partial masks $\widetilde{\mathbf{m}}_{i,j}$ and $\widetilde{\mathbf{m}}_{j,i}$ at random for $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}\backslash\{i\}$ (and programs $\mathsf{H}_{\mathsf{mask}}$ accordingly). Both games are identically distributed in the view of $\mathcal{A}$ because seeds $\mathsf{seed}_{i,j}$ and $\mathsf{seed}_{j,i}$ are hidden from $\mathcal{A}$ and the partial masks have not yet been evaluated for users in $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$. In the formal proof, this is argued via Remark 6.2

Game_9**:** In this game, the challenger samples the mask $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$ in $\mathcal{O}_{\mathsf{Sign}_4}$ and $\mathcal{O}_{\mathsf{Corrupt}}$, except if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, *i.e.*, user i is the last to open its cmt_i with respect to $\mathsf{ctnt}_{\mathbf{w}}$. If i is the last signer to open cmt_i, it sets

$$\widetilde{\mathbf{\Delta}}_i = -\sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j. \tag{13}$$

Note that when user i is corrupted, it also obtains $\vec{\mathsf{seed}}_i$. Since we sample the masks $\widetilde{\mathbf{\Delta}}_i$ *without* consulting the oracle $\mathsf{H}_{\mathsf{mask}}$ in this game, the challenger needs to ensure that $\mathsf{H}_{\mathsf{mask}}$ respects the identity

$$\widetilde{\mathbf{\Delta}}_i = \sum_{j\in\mathsf{SS}\backslash\{i\}} (\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}))$$

when user i is corrupted. It does so via an additional helper algorithm $\mathsf{ProgramZeroShare}$ that sets up $\mathsf{H}_{\mathsf{mask}}$ in accordance with $\widetilde{\mathbf{\Delta}}_i$ stored in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$. We refer to the formal proof for more information on $\mathsf{ProgramZeroShare}$.

Both games are identically distributed because: (1) If i is not the last signer for $\mathsf{ctnt}_{\mathbf{w}}$ in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, then $\widetilde{\mathsf{sHS}}_{\mathbf{w}}\backslash\{i\}$ contains at least another honest signer j, so $\widetilde{\mathbf{m}}_{i,j}$ and $\widetilde{\mathbf{m}}_{j,i}$ are sampled at random from the previous game. In particular, $\widetilde{\mathbf{\Delta}}_i = \sum_{j\in\mathsf{SS}\backslash\{i\}}(\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j})$ is distributed uniform random over $\mathcal{R}_q^k$. (2) If i is the last signer for $\mathsf{ctnt}_{\mathbf{w}}$, then all partial masks are fully determined and it holds that $\sum_{j\in\mathsf{SS}} \widetilde{\mathbf{\Delta}}_j = \mathbf{0}$. The latter yields Eq. (13) via the identity $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] = \mathbf{\Delta}_j$. Note that the masked commitment for each signer i is still defined as in the real game:

$$\widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i. \tag{14}$$

Game_{10} : In this game, the challenger samples the masked commitment $\widetilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$ at random when cmt_i is opened, except if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, then it sets $\widetilde{\mathbf{w}}_i$ according to Eq. (16) consistently. Also, it manages tables

$\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}} \mathbf{w}_j$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}} \mathbf{r}_j$ that store the sum of commitments $\mathbf{w}_j$ and $\mathbf{w}_j$'s randomenes $\mathbf{r}_j$, respectively. Also, the table $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] := \tilde{\mathbf{w}}_j - \mathbf{w}_i$ is initialized via $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] = \tilde{\mathbf{w}}_j$ only when a user is corrupted. This is required to setup $\mathsf{H}_{\mathsf{mask}}$ consistently when a user is corrupted, but not in $\mathcal{O}_{\mathsf{Sign}_4}$ anymore.

We can show that Game_{10} and Game_9 are identically distributed by looking at an intermediate game $\mathsf{Game}_{9,*}$, where instead of sampling $\tilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$, we sample $\tilde{\mathbf{\Delta}}_i^* \xleftarrow{\$} \mathcal{R}_q^k$ and set $\tilde{\mathbf{\Delta}}_i := \tilde{\mathbf{\Delta}}_i^* - \mathbf{w}_i$. This intermediate game is identically distributed to Game_9 as both $\tilde{\mathbf{\Delta}}_i$ and $\tilde{\mathbf{\Delta}}_i^*$ are uniform random. Also, observe that in $\mathsf{Game}_{9,*}$, we have that $\tilde{\mathbf{w}}_i = \tilde{\mathbf{\Delta}}_i^*$ if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}$ due to Eq. (14), and $\tilde{\mathbf{w}}_i$ as in Eq. (16) otherwise. To see the latter, first substitute $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] = \tilde{\mathbf{w}}_j - \mathbf{w}_i$ for all $j \in \mathsf{sHS} \backslash \{i\}$ in Eq. (13), then substitute the resulting identity for $\tilde{\mathbf{\Delta}}_i$ in the identity of $\tilde{\mathbf{w}}_i$ in Eq. (14).

Game_{11} : In this game, the challenger samples the honest commitments $\mathbf{w}_j$ in round 4 when the last cmt_i is opened to $\tilde{\mathbf{w}}_i$. That is, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger outputs random masked commitments $\tilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$ as before but does *not* sample $\mathbf{w}_i$ yet. Only when the last commitment cmt_i is opened (either via a corruption or $\mathcal{O}_{\mathsf{Sign}_4}$ query), the challenger generates $|\mathsf{sHS}|$-many honest commitments at once and stores them in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] = \{\mathbf{w}_j\}_{j \in [|\mathsf{sHS}|]}$. Also, tables $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in [|\mathsf{sHS}|]} \mathbf{w}_j$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in [|\mathsf{sHS}|]} \mathbf{r}_j$ are initialized, where $\mathbf{r}_j$ is $\mathbf{w}_j$'s randomness. In round 5 or when a user i is corrupted between round 4 and 5, then the challenger chooses one of the commitments $\mathbf{w}_i$ from the set $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and removes it from the set. In the latter case, the reduction also sets $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \tilde{\mathbf{w}}_i - \mathbf{w}_i$ and programs the oracle $\mathsf{H}_{\mathsf{mask}}$ via $\mathsf{ProgramZeroShare}$ for consistency.

We can show that Game_{10} and Game_{11} are identically distributed. For this, observe that the protocol messages $(\tilde{\mathbf{w}}_j)_{j \in \mathsf{sHS}}$ of round 4 reveal only the sum of the commitments $\mathbf{w}_j$ but *not* their attribution to users, *i.e.*, which user sampled which commitment $\mathbf{w}_j$. For now, this attribution is leaked implicitly in round 5 (since the challenger uses $\mathbf{r}_i$ in the computation of the masked response $\tilde{\mathbf{z}}_i$) or explicitly when a user is corrupted. In those cases, since the challenger chooses a fresh commitment via $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, the view of adversary $\mathcal{A}$ remains consistent. We refer to the formal proof for details.

This key step allows us to later simulate one of the commitments and attribute non-simulated honest commitments $\mathbf{w}_j$ to users *on-the-fly* in corruption queries.

Game_{12} to Game_{17}: In Game_{12} to Game_{17}, the challenger transitions to a game where $\tilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ is sampled at random, except that the last response $\tilde{\mathbf{z}}_i$ with $\mathsf{ctnt}_{\mathbf{w}}$ is setup consistently, *i.e.*, it respects the constraint $\mathbf{\Delta}_i = -\sum_{j \in \mathsf{SS} \backslash \{i\}} \mathbf{\Delta}_j$. Again, adversary $\mathcal{A}$ can obtain this response either via $\mathcal{O}_{\mathsf{Sign}_5}$ or $\mathcal{O}_{\mathsf{Corrupt}}$. While in the protocol, the value $\mathsf{ctnt}_{\mathbf{z}}$ serves as input to $\mathsf{ZeroShare}$, we can show that the value $\mathsf{ctnt}_{\mathbf{w}}$ uniquely determines $\mathsf{ctnt}_{\mathbf{z}}$ in round 5. This allows us to interchange $\mathsf{ctnt}_{\mathbf{w}}$ and $\mathsf{ctnt}_{\mathbf{z}}$ within the security proof when analyzing the distribution of $\mathbf{\Delta}_i$. We can show that the last masked response is distributed as follows:

$$\tilde{\mathbf{z}}_i := c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \tilde{\mathbf{z}}_j - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j, \tag{15}$$

where $\tilde{\mathbf{z}}_j$ is the masked response of user i with $\mathsf{ctnt}_{\mathbf{w}}$. Recall that $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in [|\mathsf{sHS}|]} \mathbf{r}_j$ stores the sum of the honest commitment $\mathbf{w}_j$'s randomness.

The transition to Game_{17} is similar to the transition from Game_5 to Game_{10} in the proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$. A crucial difference is that $\mathsf{ctnt}_{\mathbf{w}}$ does *not* contain the commitments $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ here. The authentication of the signer views in round 4 via the signatures $\sigma_{\mathsf{S},i}$ on M_{S} binds $\mathsf{ctnt}_{\mathbf{w}}$ to a unique set of commitments $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$. This allows us to argue that $\mathsf{ctnt}_{\mathbf{z}}$—and thus the challenge c—in round 5 is identical for all honest users. The identity in Eq. (15) follows as in the selective proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, but due to the corruption oracle, we also need to ensure that $\mathsf{H}_{\mathsf{mask}}$ is consistent with the masks $\tilde{\mathbf{\Delta}}_i$. This can be ensured via $\mathsf{ProgramZeroShare}$ as above. We refer to the formal proof for more details.

Finally, note that since $\mathbf{r}_i$ is not required anymore in $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger can exclusively attribute commitments from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ to users during corruption.

Game_{18} **to** Game_{20}**:** In games Game_{18} to Game_{20}, we invoke HVZK with respect to the verification key $\mathbf{t}$ to simulate one of the honest commitments in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ when the last commitment cmt_i with $\mathsf{ctnt}_{\mathbf{w}}$ is opened. This later allows to compute the response $\widetilde{\mathbf{z}}_h$ of the last signer h in round 5 without secret key $\mathbf{s}$. At the end of Game_{20}, the challenger no longer requires the secret key $\mathbf{s}$ to simulate the signing oracles.

Game_{18}: In this game, the challenger chooses a random challenge $c \xleftarrow{\$} \mathcal{C}$ before sampling the honest commitments $\{\mathbf{w}_j\}_{j \in |\mathsf{sHS}|}$ for $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$. Before outputting $\widetilde{\mathbf{w}}_i$ or the state st_i, the challenger retrieves the corrupt commitments $\mathbf{w}_j$ for $j \in \mathsf{CS}$ from cmt_j by searching through all the random oracle queries made to $\mathsf{H}_{\mathsf{com}}$. If all $\mathbf{w}_j$ are found, it programs H_c such that $\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) = c$, where $\mathbf{w} = \left\lfloor \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \sum_{j \in \mathsf{CS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$. Further, the challenger aborts if some $\mathbf{w}_j$ was not found in round 4 or $\mathcal{O}_{\mathsf{Corrupt}}$ with $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, but $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked with $\mathsf{ctnt}_{\mathbf{w}}$.

Since $|\mathsf{sHS}| \geqslant 1$ and $\mathbf{w}_j$ has high min-entropy, H_c was never queried with $(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ before it is programmed, so the view of $\mathcal{A}$ is identically distributed. Since the challenger checks in $\mathcal{O}_{\mathsf{Sign}_5}$ whether each $\widetilde{\mathbf{w}}_j$ is committed in cmt_j, the adversary must have found a preimage for all cmt_j. This happens with negligible probability. To argue the above, we use that due to signature-based authentication, we know that $\mathsf{ctnt}_{\mathbf{w}}$ fixes $(\mathsf{cmt}_j)_{j \in \mathsf{SS}}$ implicitly and for $j \in \mathsf{sHS}$, the hash commitments cmt_j are honest.

Game_{19}: In this game, the challenger invokes HVZK with respect to the verification key $\mathbf{t}$ to simulate one of the commitments $(\mathbf{w}_j)_{j \in |\mathsf{sHS}|}$ when setting up $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, and computes the consistent response $\widetilde{\mathbf{z}}_h$ for $\mathsf{ctnt}_{\mathbf{w}}$ in a different manner. In more detail, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ in $\mathcal{O}_{\mathsf{Corrupt}}$ or $\mathcal{O}_{\mathsf{Sign}_4}$, after sampling the challenge c, the challenger simulates the commitment-response pair $(\mathbf{w}_*, \mathbf{z}_*)$, where $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_*$. The commitment $\mathbf{w}_*$ is added to $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ but *not* stored in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ to avoid attributing it to a user in $\mathcal{O}_{\mathsf{Corrupt}}$. Also, $\mathbf{r}_*$ is *not* added to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$. Instead, the challenger computes the last consistent response $\widetilde{\mathbf{z}}_h$, *i.e.*, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_5}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, via the simulated response $\mathbf{z}_*$ as follows:

$$\widetilde{\mathbf{z}}_h := \mathbf{z}_* - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{h\}} \widetilde{\mathbf{z}}_j - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j.$$

The above identity for $\widetilde{\mathbf{z}}_h$ is obtained by rewriting Eq. (15) using $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_*$. Note that it is crucial that the challenge c—precomputed when the last user i opens its commitment to define $\mathbf{z}_*$—must be identical to the challenge c in Eq. (15). This is guaranteed by the abort condition in the previous game. Also, note that since at least one user $i \in \mathsf{sHS}$ remains uncorrupted, so we never have to attribute the simulated commitment to a user.

Game_{20}: In this game, the challenger replaces $\mathbf{t}$ in the verification key with $\mathbf{t} := \left\lfloor \hat{\mathbf{t}} \right\rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{t}}}}$, where $\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}^k_q$. Also, when a user i is corrupted, it samples $\mathbf{s}_i$ at random.

Observe that the challenger in Game_{19} uses the secret key $\mathbf{s}$ only when computing the simulated response $\mathbf{z}_* = c \cdot \mathbf{s} + \mathbf{r}_i$. Under $\mathsf{Hint\text{-}MLWE}$, Game_{19} and Game_{20} are indistinguishable. Note that simulated responses $\mathbf{z}_*$ correspond to the provided hints in $\mathsf{Hint\text{-}MLWE}$.

Reduction from $\mathsf{SelfTargetMSIS}$**.** In Game_{20}, the challenger can simulate the signing oracles without knowing $\mathbf{s}$. At this point, we are finally read to construct an adversary against $\mathsf{SelfTargetMSIS}$. This step is identical to the last step in selective proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$ and we omit details.

7 Construction of Our 5-Round Threshold Schnorr

In this section, we present our 5-round threshold signature scheme $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$, a thresholdized version of the classical Schnorr signature. We show in Section 7.3 that $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is adaptively secure under the DL assumption. Our protocol is an adaption of our 5-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ to the group setting.

7.1 Preparations

Let GenG be an algorithm that on input 1^λ, outputs a tuple $(\mathbb{G}, p, G)$, where G is a generator of group $\mathbb{G}$ of prime order p.

In our scheme, each user is given a tuple of random strings of the form $\vec{\mathsf{seed}}_i = (\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in[N]}$. For any set $\mathsf{SS} \subseteq [N]$, we denote $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ as the tuple $(\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j\in\mathsf{SS}}$. As in our other schemes, we further prepare a deterministic helper algorithm named $\mathsf{ZeroShare}$ defined with respect to a random oracle $\mathsf{H}_{\mathsf{mask}}$ with variable range. For any $\vec{\mathsf{seed}}_i[\mathsf{SS}]$ and string $x \in \{0,1\}^*$, it is defined as follows:

$$\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) := \sum_{j\in\mathsf{SS}\setminus\{i\}} \left(\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, x) - \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, x)\right),$$

where $\mathsf{H}_{\mathsf{mask}}$ outputs vectors over $\mathbb{G}$ and $\mathbb{Z}_p$ when the first bit of x is 0 and 1, respectively. Looking ahead, we use $\mathsf{ZeroShare}$ to mask the commitment $R \in \mathbb{G}$ and response $z \in \mathbb{Z}_p$. We will extensively use the following easy to check fact:

$$\sum_{i\in\mathsf{SS}} \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], x) = \mathbf{0}.$$

We also require an EUF-CMA secure signature scheme $\mathsf{S} = (\mathsf{KeyGen}_{\mathsf{S}}, \mathsf{Sign}_{\mathsf{S}}, \mathsf{Verify}_{\mathsf{S}})$.

7.2 Construction

The construction of our 5-round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is provided in Figs. 6 and 7. Our scheme uses three hash functions modeled as a random oracle in the security proof. $\mathsf{H}_{\mathsf{com}} : \{0,1\}^* \to \{0,1\}^{2\lambda}$ is used to generate the hash commitment. $\mathsf{H}_c : \{0,1\}^* \to \mathbb{Z}_p$ is used to generate the random challenge for which the users reply with a response. $\mathsf{H}_{\mathsf{mask}} : \{0,1\}^* \to \mathbb{G} \cup \mathbb{Z}_p$ is used to generate the random vectors to mask the individual commitment or response.

Essentially, we obtain $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ from $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ by replacing the Raccoon-related elements with their Schnorr-counterparts. We give a brief summary of the modifications:

- We replace the matrix $\mathbf{A}$ in tspar with $(\mathbb{G}, p, G) \xleftarrow{\$} \mathsf{GenG}(1^\lambda)$, and the verification key with (tspar, X), where $X = x \cdot G$ and $x \xleftarrow{\$} \mathbb{Z}_p$ is a Schnorr public and secret key, respectively. The secret key x is shared into partial secrets $(x_i)_{i\in[N]}$ via Shamir's secret sharing as before.
- We replace verification—previously identical to Raccoon verification—with the classical Schnorr verification.
- In Sign_2, we replace the Raccoon commitments $\mathbf{w}_i = \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$ with commitments Schnorr commitments $R_i = r_i \cdot G$, where $r_i \xleftarrow{\$} \mathbb{Z}_p$. The commitments are masked via $\widetilde{R}_i = R_i + \widetilde{\mathbf{\Delta}}_i$ as before and the masked commitments $\widetilde{R}_i$ are committed in cmt_i as before.
- In Sign_5, we still sum up the the masked commitments $\widetilde{R}_i$ to obtain an aggregated commitment $R = \sum_{j\in\mathsf{SS}} \widetilde{R}_j$. Also, the masked response is computed as before via $\widetilde{z}_i := c \cdot L_{\mathsf{SS},i} \cdot x_i + r_i + \mathbf{\Delta}_i$, except that $\widetilde{z}_i \in \mathbb{Z}_p$.

7.3 Security and Correctness

Correctness follows immediately using the correctness of $\mathsf{ZeroShare}$ as in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. Also, we have the following.

Theorem 7.1. *The 5-round threshold signature* $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ *in Figs. 6 and 7 is adaptive secure under the* DL *assumption.*

Formally, for any N and T with $T \leqslant N$ and an adversary $\mathcal{A}$ against the adaptive security game making at most Q_{H_c}, $Q_{\mathsf{H}_{\mathsf{com}}}$, $Q_{\mathsf{H}_{\mathsf{mask}}}$, and Q_{S} queries to the random oracles H_c, $\mathsf{H}_{\mathsf{com}}$, and $\mathsf{H}_{\mathsf{mask}}$ and the signing oracle,

$\mathsf{Setup}(1^\lambda, N, T)$

$$
\begin{array}{l}
(\mathbb{G}, p, G) \leftarrow \mathsf{GenG}(1^\lambda) \\
\mathsf{tspar} := ((\mathbb{G}, p, G), N, T) \\
\textbf{return } \mathsf{tspar}
\end{array}
$$

$\mathsf{Verify}(\mathsf{vk}, \mathsf{M}, \mathsf{sig})$

$$
\begin{array}{l}
\textbf{parse } (c, z) \leftarrow \mathsf{sig} \\
c' := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, z \cdot G - c \cdot X) \\
\textbf{if } [\![c = c']\!] \textbf{ then} \\
\quad \textbf{return } 1 \\
\textbf{return } 0
\end{array}
$$

$\mathsf{KeyGen}(\mathsf{tspar})$

$$
\begin{array}{l}
x \xleftarrow{\$} \mathbb{Z}_p; X := x \cdot G \\
\textbf{for } i \in [N] \textbf{ do} \\
\quad (\mathsf{vk}_{\mathsf{S},i}, \mathsf{sk}_{\mathsf{S},i}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{S}}(1^\lambda) \\
\quad \textbf{for } j \in [N] \textbf{ do} \\
\quad\quad \mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda \\
\quad\quad \mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j} \\
(\vec{\mathsf{seed}}_i)_{i \in [N]} := \Big((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in [N]} \Big)_{i \in [N]} \\
P \xleftarrow{\$} \mathbb{Z}_p[Y] \text{ with } \deg(P) = T - 1, P(0) = x \\
(x_i)_{i \in [N]} := (P(i))_{i \in [N]} \\
\mathsf{vk} := (\mathsf{tspar}, X) \\
(\mathsf{sk}_i)_{i \in [N]} := \big(x_i, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i\big)_{i \in [N]} \\
\textbf{return } (\mathsf{vk}, (\mathsf{sk}_i)_{i \in [N]})
\end{array}
$$

Figure 6: Setup, KeyGen, and Verify for our five round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. The differences to $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ are marked in blue.

respectively, there exists adversaries $\mathcal{B}$ *and* $\mathcal{B}_{\mathsf{S}}$ *against the* $\mathsf{SelfTargetDL}$ *problem and the unforgeability of signatures, respectively, such that*

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}, \mathcal{A}}(1^\lambda, N, T, 1) \leqslant{}& Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetDL}}_{\mathcal{B}}(1^\lambda) + N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S}, \mathcal{B}_{\mathsf{S}}}(\lambda) \\
&+ \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{p} + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{Q_{\mathsf{S}}^2 + (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}},
\end{aligned}
$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ *and* $\mathsf{Time}(\mathcal{B}_{\mathsf{S}}) \approx \mathsf{Time}(\mathcal{A})$*. From Lemma A.12, we can replace* $\mathcal{B}$ *by an adversary* $\mathcal{B}'$ *against the* DL *problem with* $\mathsf{Time}(\mathcal{B}') \approx 2 \cdot \mathsf{Time}(\mathcal{B})$ *such that*

$$
\mathsf{Adv}^{\mathsf{SelfTargetDL}}_{\mathcal{B}}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}'}(\lambda)} + \frac{Q_{\mathsf{H}_c}}{p}.
$$

Proof. Let us prove the statement. Note that this proof is *not* self-contained and we encourage the reader to first read the security proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ (cf. Section 6). As noted above, the structure of our protocol $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ is almost identical to $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ except for the underlying signature. This is reflected in the proof structure. Since until Game_{18} the proof is *almost* oblivious to the underlying algebraic structure, these games are almost identical. The only required property for the transitions from Game_1 to Game_{17} is that the commitments R_i have high min-entropy (which holds). The last step, *i.e.*, Game_{18} to Game_{20} and extraction, is more tailored to the underlying signature. We sketch security [11] and highlight the differences below.

Our strategy remains to use a hybrid argument to transition to a game, where the challenger simulates the signing oracles without the secret key x. We then embed an $\mathsf{SelfTargetDL}$ problem into the verification key X and extract a solution from the forgery. As before, we denote by sHS (resp. sCS) the subset of honest users $\mathsf{sHS} = \mathsf{SS} \cap \mathsf{HS}$ (resp. corrupt users $\mathsf{sCS} = \mathsf{SS} \cap \mathsf{CS}$) queried to the signing or corruption oracle. We sketch the hybrids below.

[11] It is straightforward but tedious to formalize the proof following the security proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ (cf. Section 6).

$\mathsf{Sign}_1(\mathsf{vk}, i, \mathsf{sk}_i, \mathsf{st}_i)$

$\mathsf{str}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{\mathsf{str}_i\}$
$\textbf{return } (\mathsf{pm}_{1,i} := \mathsf{str}_i, \mathsf{st}_i)$

$\mathsf{Sign}_2(\mathsf{vk}, \mathsf{SS}, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
$\textbf{req } [\![\mathsf{pm}_{1,i} \in \mathsf{st}_i]\!]$
$\textbf{pick } \mathsf{str}_i \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{1,i} = \mathsf{str}_i$
$\textbf{parse } (\mathsf{str}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS} \setminus \{i\}}$
$\textbf{parse } \mathsf{sk}_i \leftarrow (x_i, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i)$
$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$
$\tilde{\boldsymbol{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathbb{G}$
$r_i \xleftarrow{\$} \mathbb{Z}_p; R_i := r_i \cdot G$
$\widetilde{R}_i := R_i + \tilde{\boldsymbol{\Delta}}_i \in \mathbb{G}$
$\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \widetilde{R}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{\mathsf{str}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{2,i} := \mathsf{cmt}_i, \mathsf{st}_i)$

$\mathsf{Sign}_3(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \cdot, \mathsf{pm}_{2,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\textbf{parse } (\mathsf{cmt}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS} \setminus \{i\}}$
$\text{with } \mathsf{pm}_{2,i} = \mathsf{cmt}_i$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_{\mathsf{S}}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_{\mathsf{S}})$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \widetilde{R}_i, r_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{3,i} := \sigma_{\mathsf{S},i}, \mathsf{st}_i)$

$\mathsf{Sign}_4(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\text{with } \mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$
$\textbf{parse } (\sigma_{\mathsf{S},j})_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS} \setminus \{i\}}$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\textbf{req } [\![\forall j \in \mathsf{SS} \setminus \{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{3,i} := \widetilde{R}_i, \mathsf{st}_i)$

$\mathsf{Sign}_5(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{parse } (\widetilde{R}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS} \setminus \{i\}}$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\text{with } \mathsf{pm}_{4,i} = \widetilde{R}_i$
$\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{R}_j)]\!]$
$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\widetilde{R}_j)_{j \in \mathsf{SS}}$
$R := \sum_{j \in \mathsf{SS}} \widetilde{R}_j \in \mathbb{G}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, R) \quad /\!/\ c \in \mathbb{Z}_p$
$\boldsymbol{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathbb{Z}_p$
$\tilde{z}_i := c \cdot L_{\mathsf{SS},i} \cdot x_i + r_i + \boldsymbol{\Delta}_i \in \mathbb{Z}_p$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{5,i} := \tilde{z}_i, \mathsf{st}_i)$

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{b,j})_{(b,j) \in [5] \times \mathsf{SS}})$

$\textbf{parse } (\widetilde{R}_j, \tilde{z}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{4,j}, \mathsf{pm}_{5,j})_{j \in \mathsf{SS}}$
$R := \sum_{j \in \mathsf{SS}} \widetilde{R}_j$
$z := \sum_{j \in \mathsf{SS}} \tilde{\mathbf{z}}_j$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, R)$
$\textbf{return } \mathsf{sig} := (c, z)$

Figure 7: The Signing protocol of our five round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. In the above, $L_{\mathsf{SS},i}$ denotes the Lagrange coefficient of user i in the set $\mathsf{SS} \subseteq [N]$ (see Section 2.3 for the definition). **pick** X **from** Y denotes the process of picking an element X from the set Y. The differences to $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ are marked in blue.

Game_1 **to** Game_5**:** In Game_1 to Game_4, the challenger delays sampling the commitment R_i until the 4th round or when a user is corrupted. That is, the challenger outputs a random $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_2}$. In $\mathcal{O}_{\mathsf{Sign}_4}$, it samples $R_i = r_i \cdot G$ and sets $\widetilde{R}_i = R_i + \widetilde{\mathbf{\Delta}}_i$. Then, it programs $\mathsf{H}_{\mathsf{com}}$ such that $\mathsf{cmt}_i = \mathsf{H}_{\mathsf{com}}(\widetilde{R}_i, i)$ and outputs $\widetilde{R}_i$. This is also done if i is corrupted for all signer states before round 4. Further, the challenger aborts in case there is a collision in $\mathsf{H}_{\mathsf{com}}$ and ensures that all sampled str_i are unique.

In Game_5, the challenger aborts if M_S was not signed by some honest user. As before, the implication of this is as follows. Roughly, M_S corresponds to the view of each honest user in round 4 before the commitments are opened. The consistency check ensures that $\mathcal{O}_{\mathsf{Sign}_5}$ is not invoked unless all honest users share an identical view in round 4 with respect to $\mathsf{ctnt}_\mathbf{w}$ *before* their commitments are opened. Again, this is essential for simulation later all the above steps can be argued as in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. Similarly, we have the following.

Remark 7.2. The adversary cannot invoke $\mathcal{O}_{\mathsf{Sign}_r}$ twice with the same value $\mathsf{ctnt}_\mathbf{w}$ for $r \in [5]$.

Game_6 **to** Game_{11}**:** In Game_6 to Game_{11}, the challenger transitions to a game where $\widetilde{R}_j \xleftarrow{\$} \mathbb{G}$ is sampled at random, except the last revealed commitment $\widetilde{R}_i$ is sampled *consistently*. Again, consistently means that $\widetilde{R}_i$ respects the constraint $\widetilde{\mathbf{\Delta}}_i = -\sum_{j \in \mathsf{SS}\setminus\{i\}} \widetilde{\mathbf{\Delta}}_j$, and we can show as before that the last masked commitment $\widetilde{R}_i$ is distributed as follows:

$$\widetilde{R}_i = \mathsf{SumCom}[\mathsf{ctnt}_\mathbf{w}] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS}\setminus\{i\}} \widetilde{R}_j, \tag{16}$$

where $\widetilde{R}_j$ is the masked commitment of user i with $\mathsf{ctnt}_\mathbf{w}$ and $\mathsf{SumCom}[\mathsf{ctnt}_\mathbf{w}] = \sum_{j \in \mathsf{sHS}} R_j$ stores the sum of the honest commitments $(R_j)_{j \in \mathsf{sHS}}$. Further, since all $\widetilde{R}_j$ but the last are random, the challenger can delay sampling the honest commitments $(R_j)_{j \in \mathsf{sHS}}$ until the last signer opens its commitment cmt_i. Also, observe that the protocol messages $(\widetilde{R}_j)_{j \in \mathsf{sHS}}$ of round 4 reveal only the sum of the commitments R_j but *not* their attribution to users, *i.e.*, which user sampled which commitment R_j. Thus, when the last cmt_i is opened to $\widetilde{R}_i$, the challenger generates $|\mathsf{sHS}|$-many honest commitments at once, stores them in $\mathsf{UnUsedCom}[\mathsf{ctnt}_\mathbf{w}] = \{R_j\}_{j \in \mathsf{SS}}$ and their sum in $\mathsf{SumCom}[\mathsf{ctnt}_\mathbf{w}]$. The challenger then attributes these commitments as before from the set $\mathsf{UnUsedCom}[\mathsf{ctnt}_\mathbf{w}]$ in round 5 or when a user between round 4 and 5 is corrupted. In the latter case, the reduction also programs the oracle $\mathsf{H}_{\mathsf{mask}}$ so that the users state is consistent with the choice. Finally, the challenger also sets up a table $\mathsf{SumComRnd}[\mathsf{ctnt}_\mathbf{w}] = \sum_{j \in \mathsf{sHS}} r_j$ that the sum of the honest commitments R_j's randomness for later. We can argue as in the security proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$, as the game transitions information theoretic and independent of the underlying algebraic structure.

Game_{12} **to** Game_{17}**:** In Game_{12} to Game_{17}, the challenger transitions to a game where $\widetilde{z}_i \xleftarrow{\$} \mathbb{Z}_p$ is sampled at random, except that the last response $\widetilde{z}_i$ with $\mathsf{ctnt}_\mathbf{w}$ is setup consistently, *i.e.*, it respects the constraint $\mathbf{\Delta}_i = -\sum_{j \in \mathsf{SS}\setminus\{i\}} \mathbf{\Delta}_j$. Again, adversary $\mathcal{A}$ can obtain this response either via $\mathcal{O}_{\mathsf{Sign}_5}$ or $\mathcal{O}_{\mathsf{Corrupt}}$. Note that we can interchange $\mathsf{ctnt}_\mathbf{w}$ and $\mathsf{ctnt}_\mathbf{z}$ within the security proof freely when analyzing the distribution of $\mathbf{\Delta}_i$ as before, and it follows that the last masked response is distributed as follows:

$$\widetilde{z}_i := c \cdot x - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot x_j + \mathsf{SumComRnd}[\mathsf{ctnt}_\mathbf{w}] - \sum_{j \in \mathsf{sHS}\setminus\{i\}} \widetilde{z}_j - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j, \tag{17}$$

where $\widetilde{z}_j$ is the masked response of user i with $\mathsf{ctnt}_\mathbf{w}$. Again, this follows the game transitions information theoretic and independent of the underlying algebraic structure.

Finally, note that since r_i is not required anymore in $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger can exclusively attribute commitments from $\mathsf{UnUsedCom}[\mathsf{ctnt}_\mathbf{w}]$ to users during corruption.

Game_{18} **to** Game_{20}**: Invoke HVZK** In games Game_{18} to Game_{20}, we invoke HVZK with respect to the verification key x to simulate one of the honest commitments in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ when the last commitment cmt_i with $\mathsf{ctnt}_{\mathbf{w}}$ is opened. This later allows to compute the response $\tilde{z}_h$ of the last signer h in round 5 without secret key x. At the end of Game_{20}, the challenger no longer requires the secret key x to simulate the signing oracles. Note that here, we need to argue using the algebraic structure underlying Schnorr signatures. We elaborate.

Game_{18}: In this game, the challenger chooses a random challenge $c \xleftarrow{\$} \mathbb{Z}_p$ before sampling the honest commitments $\{R_j\}_{j\in|\mathsf{sHS}|}$ for $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ if i is the last signer to open its hash commitment cmt_i to $\widetilde{R}_i$ in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$. Before outputting $\widetilde{R}_i$ or the state st_i, the challenger retrieves the corrupt commitments R_j for $j \in \mathsf{CS}$ from cmt_j by searching through all the random oracle queries made to $\mathsf{H}_{\mathsf{com}}$. If all R_j are found, it programs H_c such that $\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, R) = c$, where $R = \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \sum_{j\in\mathsf{CS}} R_j$. Further, the challenger aborts if some R_j was not found for $\mathsf{ctnt}_{\mathbf{w}}$, but $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked with $\mathsf{ctnt}_{\mathbf{w}}$.

Since $|\mathsf{sHS}| \geqslant 1$ and R_j has high min-entropy, H_c was never queried with $(\mathsf{vk}, \mathsf{M}, R)$ before it is programmed, so the view of $\mathcal{A}$ is identically distributed. Since the challenger checks in $\mathcal{O}_{\mathsf{Sign}_5}$ whether each $\widetilde{R}_j$ is committed in cmt_j, the adversary must have found a preimage for all cmt_j. This happens with negligible probability. To argue the above, we use that due to signature-based authentication, we know that $\mathsf{ctnt}_{\mathbf{w}}$ fixes $(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$ implicitly and for $j \in \mathsf{sHS}$, the hash commitments cmt_j are honest.

Game_{19}: In this game, the challenger invokes HVZK with respect to the verification key x to simulate one of the commitments $(R_j)_{j\in|\mathsf{sHS}|}$ when setting up $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, and computes the consistent response $\tilde{z}_h$ for $\mathsf{ctnt}_{\mathbf{w}}$ in a different manner. In more detail, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ in $\mathcal{O}_{\mathsf{Corrupt}}$ or $\mathcal{O}_{\mathsf{Sign}_4}$, after sampling the challenge c, the challenger simulates the commitment-response pair (R_*, z_*), where $z_* = c\cdot x + r_*$. The commitment R_* is added to $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ but *not* stored in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ to avoid attributing it to a user in $\mathcal{O}_{\mathsf{Corrupt}}$. Also, r_* is *not* added to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$. Instead, the challenger computes the last consistent response $\tilde{z}_h$, *i.e.*, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_5}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, via the simulated response z_* as follows:

$$\tilde{z}_h := z_* - c \sum_{j\in\mathsf{sCS}} L_{\mathsf{SS},j} \cdot x_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sHS}\setminus\{h\}} \tilde{z}_j - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j.$$

The above identity for $\tilde{z}_h$ is obtained by rewriting Eq. (17) using $z_* = c \cdot x + r_*$.

Game_{20}: This is the first game that requires the algebraic structure of $\mathbb{G}$. In this game, the challenger replaces $X \xleftarrow{\$} \mathbb{G}$ in the verification key with a random group element. When simulating the pair (R_*, z_*) with challenge c, the challenger samples $z_* \xleftarrow{\$} \mathbb{Z}_p$ and $R := z_* \cdot G - c \cdot X$. Also, when a user i is corrupted, it samples x_i at random.

Observe that the challenger in Game_{19} uses the secret key x only when computing the simulated response $z_* = c \cdot x + r_i$. It is straightforward to check that both games are identically distributed.

Reduction from $\mathsf{SelfTargetDL}$**.** In Game_{20}, the challenger can simulate the signing oracles without knowing $\mathbf{s}$. At this point, we are finally read to construct an adversary against $\mathsf{SelfTargetDL}$. Again, we can adapt the last step of the proof for $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ given in Section 6 to obtain the result. □

7.4 Our 4-Round Schnorr Threshold Signature

As $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$, we can construct the *stateful* 4 round threshold signature scheme $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ from $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ by using the session identifier sid that is never reused. For the detail of this transformation, see Section 5.4. The construct and the security theorem are provided in Appendix C.

Acknowledgement. This work has been supported in part by JST CREST Grant Number JPMJCR22M1, JST-AIP Acceleration Research JPMJCR22U5, JSPS KAKENHI Grant Numbers JP22KJ1366.

References

[ADN06] 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, Heidelberg, May / June 2006.

[AF04] Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Heidelberg, August 2004.

[ASY22] 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.

[BCK+22] 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, Heidelberg, August 2022.

[BD21] Mihir Bellare and Wei Dai. Chain reductions for multi-signatures and the HBMS scheme. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 650–678. Springer, Heidelberg, December 2021.

[BD22] 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.

[BG14] Shi Bai and Steven D. Galbraith. An improved compression technique for signatures based on learning with errors. In Josh Benaloh, editor, *CT-RSA 2014*, volume 8366 of *LNCS*, pages 28–47. Springer, Heidelberg, February 2014.

[BGG+18] 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, Heidelberg, August 2018.

[BHK+23] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Tal Rabin, and Yiping Ma. SPRINT: high-throughput robust distributed schnorr signatures. *IACR Cryptol. ePrint Arch.*, page 427, 2023.

[BKP13] Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to share a lattice trapdoor: Threshold protocols for signatures and (H)IBE. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 218–236. Springer, Heidelberg, June 2013.

[BL22] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 193–207. ACM Press, November 2022.

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Heidelberg, October 2021.

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

[BLT+24] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from ddh with full adaptive security. *To Appear in EUROCRYPT 2024. Available at https://eprint.iacr.org/2023/1482*, 2024.

[BN06] 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.

[Bol03] 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, Heidelberg, January 2003.

[BTZ22] Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: BLS and FROST. Cryptology ePrint Archive, Report 2022/833, 2022. https://eprint.iacr.org/2022/833.

[CCL+20] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 266–296. Springer, Heidelberg, May 2020.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[CGG+20] 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.

[CGJ+99] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 98–115. Springer, Heidelberg, August 1999.

[Che05] Benoît Chevallier-Mames. An efficient CDH-based signature scheme with a tight security reduction. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 511–526. Springer, Heidelberg, August 2005.

[CKM23] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. Fully adaptive Schnorr threshold signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 678–709. Springer, Heidelberg, August 2023.

[CS20] Daniele Cozzo and Nigel P. Smart. Sashimi: Cutting up CSI-FiSh secret keys to produce an actively secure distributed signing protocol. In Jintai Ding and Jean-Pierre Tillich, editors, *Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020*, pages 169–186. Springer, Heidelberg, 2020.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *26th ACM STOC*, pages 522–533. ACM Press, May 1994.

[Des90] Yvo Desmedt. Abuses in cryptography and how to fight them. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 375–389. Springer, Heidelberg, August 1990.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, August 1990.

[DJN+20] Ivan Damgård, Thomas Pelle Jakobsen, Jesper Buus Nielsen, Jakob Illeborg Pagter, and Michael Bæksvang Østergaard. Fast threshold ECDSA with honest majority. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 382–400. Springer, Heidelberg, September 2020.

[DKL+18] Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. CRYSTALS-Dilithium: A lattice-based digital signature scheme. *IACR TCHES*, 2018(1):238–268, 2018. https://tches.iacr.org/index.php/TCHES/article/view/839.

[DKLs19] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

[DM20] Luca De Feo and Michael Meyer. Threshold schemes from isogeny assumptions. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 187–212. Springer, Heidelberg, May 2020.

[DOTT21] Ivan Damgård, Claudio Orlandi, Akira Takahashi, and Mehdi Tibouchi. Two-round n-out-of-n and multi-signatures and trapdoor commitment from lattices. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 99–130. Springer, Heidelberg, May 2021.

[dPEK+23] 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.

[dPKM+24] Rafael del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions, 2024. To Appear in EUROCRYPT 2024. Available at https://eprint.iacr.org/2024/184.

[DR23] Sourav Das and Ling Ren. Adaptively secure BLS threshold signatures from DDH and co-cdh. *IACR Cryptol. ePrint Arch.*, page 1553, 2023.

[DYX+22] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy*, pages 2518–2534. IEEE Computer Society Press, May 2022.

[EKT24] Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors, 2024. To Appear in CRYPTO 2024. Available at https://eprint.iacr.org/2024/496.

[ENP24] Thomas Espitau, Guilhem Niot, , and Thomas Prest. Flood and submerse: Verifiable short secret sharing and application to robust threshold signatures on lattices, 2024. To Appear in CRYPTO 2024.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2018.

[FMY98] Yair Frankel, Philip D. MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, page 320. ACM, June / July 1998.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GG18] Rosario Gennaro and Steven Goldfeder. Fast multiparty threshold ECDSA with fast trustless setup. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1179–1194. ACM Press, October 2018.

[GG20] Rosario Gennaro and Steven Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 16*, volume 9696 of *LNCS*, pages 156–174. Springer, Heidelberg, June 2016.

[GHKR08] Rosario Gennaro, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Threshold RSA for dynamic and ad-hoc groups. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 88–107. Springer, Heidelberg, April 2008.

[GJKR07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, January 2007.

[GKS23] Kamil Doruk Gur, Jonathan Katz, and Tjerand Silde. Two-round threshold lattice signatures from threshold homomorphic encryption, 2023. To Appear in PQCrypto 2024. Available at https://eprint.iacr.org/2023/1318.

[GKSŚ20] Adam Gągol, Jędrzej Kula, Damian Straszak, and Michał Świętek. Threshold ECDSA for decentralized asset custody. Cryptology ePrint Archive, Report 2020/498, 2020. https://eprint.iacr.org/2020/498.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GS23] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. *IACR Cryptol. ePrint Arch.*, page 1175, 2023.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015.

[HKL19] Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375. Springer, Heidelberg, May 2019.

[HKLN20] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 500–529. Springer, Heidelberg, August 2020.

[JL00] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 221–242. Springer, Heidelberg, May 2000.

[KCLM22] Irakliy Khaburzaniya, Konstantinos Chalkias, Kevin Lewi, and Harjasleen Malvai. Aggregating and thresholdizing hash-based signatures using STARKs. In Yuji Suga, Kouichi Sakurai, Xuhua Ding, and Kazue Sako, editors, *ASIACCS 22*, pages 393–407. ACM Press, May / June 2022.

[KG20] 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, Heidelberg, October 2020.

[KGS23] Chelsea Komlo, Ian Goldberg, and Douglas Stebila. A formal treatment of distributed key generation, and new constructions. Cryptology ePrint Archive, Report 2023/292, 2023. https://eprint.iacr.org/2023/292.

[KLP17] Eike Kiltz, Julian Loss, and Jiaxin Pan. Tightly-secure signatures from five-move identification protocols. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 68–94. Springer, Heidelberg, December 2017.

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Heidelberg, April / May 2018.

[KLSS23] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-mlwe. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 549–580, Cham, 2023. Springer Nature Switzerland.

[KMP16] Eike Kiltz, Daniel Masny, and Jiaxin Pan. Optimal security proofs for signatures from identification schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 33–61. Springer, Heidelberg, August 2016.

[KMS20] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1751–1767. ACM Press, November 2020.

[Lin22] Yehuda Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

[LJY14] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014.

[LN18] Yehuda Lindell and Ariel Nof. Fast secure multiparty ECDSA with practical distributed key generation and applications to cryptocurrency custody. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1837–1854. ACM Press, October 2018.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Heidelberg, December 2001.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, Heidelberg, May 2013.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[Lyu09] 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, Heidelberg, December 2009.

[Lyu12] Vadim Lyubashevsky. Lattice signatures without trapdoors. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 738–755. Springer, Heidelberg, April 2012.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[NIS22] 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.

[PB23] René Peralta and Luís T.A.N. Brandão. Nist first call for multi-party threshold schemes. National Institute of Standards and Technology, 2023. https://doi.org/10.6028/NIST.IR.8214C.ipd.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Heidelberg, August 1992.

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

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In Hugo Krawczyk, editor, *CRYPTO'98*, volume 1462 of *LNCS*, pages 89–104. Springer, Heidelberg, August 1998.

[RRJ+22] 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.

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

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

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

[Sho23] Victor Shoup. The many faces of schnorr. *IACR Cryptol. ePrint Arch.*, page 1019, 2023.

[SS01] Douglas R. Stinson and Reto Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In Vijay Varadharajan and Yi Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001.

[TZ23] Stefano Tessaro and Chenzhi Zhu. Threshold and multi-signature schemes from linear hash functions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 628–658. Springer, Heidelberg, April 2023.

A Omitted Preliminaries

A.1 Security Notions for Threshold Signature

A.1.1 Selective Security

In this section, we formally define the selective security of threshold signature schemes. Our definition is based on the game-based one for a three round scheme provided by [CKM23].

In the selective setting, an adversary $\mathcal{A}$ determines the set CS of users to be corrupted at the beginning of the security game (after obtaining the parameters tspar). After this, it is not allowed to corrupt more honest user during the game. The challenger executes the key generation after CS is determined. It then provides $\mathcal{A}$ with the verification key and secret key shares of corrupted users as input. It also provides access to signing oracles for each round. In the end, $\mathcal{A}$ outputs a signature-message pair $(\mathsf{sig}^*, \mathsf{M}^*)$ that constitutes the forgery. The adversary $\mathcal{A}$ wins the game if $(\mathsf{sig}^*, \mathsf{M}^*)$ is deemed *non-trivial*. Note that we use a stronger security model, *i.e.*, we classify more forgeries as non-trivial (cf. Remark 2.3).

$\mathsf{Game}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T)$

1 : $\mathsf{Q_M}[\cdot] = \varnothing$ // No message was signed yet
2 : $\mathsf{tspar} \xleftarrow{\$} \mathsf{Setup}(1^\lambda, N, T)$
3 : $(\mathsf{CS}, \mathsf{st}_\mathcal{A}) \xleftarrow{\$} \mathcal{A}^{\mathsf{H}}(\mathsf{tspar})$
4 : **req** $[\![\mathsf{CS} \subseteq [N]]\!] \wedge [\![|\mathsf{CS}| \leqslant T-1]\!]$
5 : $\mathsf{HS} := [N] \backslash \mathsf{CS}$
6 : **for** $i \in \mathsf{HS}$ **do** $\mathsf{st}_i := \varnothing$
7 : $(\mathsf{vk}, (\mathsf{sk}_i)_{i\in[N]}) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{tspar})$
8 : $(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{(\mathcal{O}_{\mathsf{Sign}_r})_{r\in[R]}, \mathsf{H}}(\mathsf{vk}, (\mathsf{sk}_i)_{i\in\mathsf{CS}}, \mathsf{st}_\mathcal{A})$
9 : **req** $[\![\,|\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}| \leqslant T-1]\!]$
10 : **return** $\mathsf{Verify}(\mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathcal{O}_{\mathsf{Sign}_r}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}})$

// $r \in [R]$, $\mathsf{pm}_{0,j} = \perp$ for all $j \in \mathsf{SS}$.
1 : **req** $[\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{HS} \cap \mathsf{SS}]\!]$
2 : $(\mathsf{pm}_{r,i}, \mathsf{st}_i) \xleftarrow{\$} \mathsf{Sign}_r(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{r-1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$
3 : **if** $[\![r = R]\!]$ **then** $\mathsf{Q_M}[\mathsf{M}] \leftarrow \mathsf{Q_M}[\mathsf{M}] \cup \{i\}$
4 : **return** $\mathsf{pm}_{r,i}$

Figure 8: Selective security game for a R round threshold signature scheme, where H denotes the random oracle. In the above, the oracles return $\perp$ to $\mathcal{A}$ when Sign_r outputs $\perp$ for $r \in [R]$ (i.e., fail to output a protocol message or a partial signature).

Now we define selective security for a R round threshold signature scheme.

Definition A.1 ($\mathsf{TS\text{-}UF\text{-}1}$ Selective Security). *For a R round threshold signature scheme TS, the advantage of an adversary $\mathcal{A}$ (with oracle access to a random oracle H) against the selective security of TS is defined as*

$$\mathsf{Adv}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T) = \Pr[\mathsf{Game}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TS},\mathcal{A}}(1^\lambda, N, T) = 1],$$

where $\mathsf{Game}^{\mathsf{ts\text{-}sel\text{-}uf}}$ is described in Fig. 8, respectively. We say that TS is adaptive secure in the random oracle model if, for all $\lambda \in \mathbb{N}$, $N, T \in \mathsf{poly}(\lambda)$ s.t. $T \leqslant N$ and PPT adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ts\text{-}sel\text{-}uf}}\mathsf{TS}, \mathcal{A}(1^\lambda, N, T) \leqslant \mathsf{negl}(\lambda)$ holds.

A.1.2 $\mathsf{TS\text{-}UF\text{-}0}$ Security

For completeness, we define the $\mathsf{TS\text{-}UF\text{-}0}$ security notion of selective and adaptive security considered by [CKM23]. For selective security, replace line 3 in $\mathcal{O}_{\mathsf{Sign}_r}$ and line 9 in Fig. 8 by $\mathsf{Q_M} \leftarrow \mathsf{Q_M} \cup \{\mathsf{M}\}$ and **req** $[\![\mathsf{M}^* \notin \mathsf{Q_M}]\!]$, respectively. Note that $\mathsf{Q_M}$ is an initially empty set. For adaptive security, replace line 3 in $\mathcal{O}_{\mathsf{Sign}_r}$ and line 7 in Fig. 2 by $\mathsf{Q_M} \leftarrow \mathsf{Q_M} \cup \{\mathsf{M}\}$ and **req** $[\![\mathsf{M}^* \notin \mathsf{Q_M}]\!]$, respectively. We omit details.

A.2 Rounding and Norms Modulo q

This section is taken almost verbatim from [dPKM+24]. In all of the following we fix positive integers q and n. We aim at giving a systematic treatment of the adaptation of the notions of norms and rounding maps to the ring of integers modulo q, $\mathbb{Z}_q$ and more generally in the free module $\mathbb{Z}_q^n$ of vectors mod q.

A.2.1 Length over Modular Integers.

In this work we use the so-called *canonical* unsigned representation of integers modulo q. Given an integer $x \in \mathbb{Z}$, this representation is the unique non-negative element $0 \leqslant t \leqslant q-1$ such that $x = t \mod q$. We will generically note this element $(x \mod q)$. Conversely, given a class $x + q\mathbb{Z} \in \mathbb{Z}_q$, we define the corresponding lift $\bar{x}$ to the unique integer in $x + q\mathbb{Z} \cap [0, \ldots q-1]$.

For any norm $\|\cdot\|$ over $\mathbb{Q}^n$, we define the *length* of a (vector) class $\mathbf{x} + q\mathbb{Z}^n$ to be $\min_{\mathbf{z} \in \mathbf{x} + q\mathbb{Z}^n} \|\mathbf{z}\|$, and overload the notation as $\|\mathbf{x} + q\mathbb{Z}^n\|$, $\|\mathbf{x} \mod q\|$ or even $\|\mathbf{x}\|$ if the context is clear enough to avoid any ambiguity. As for the integers, we prefer to write simply $|x|$ when $n = 1$ to refer to the absolute value. [dPKM+24] show that with the choices in this definition, $\|\cdot\|$ is indeed a *F-norm* over free modules over $\mathbb{Z}_q$. only non trivial point to show is the triangular inequality.

Lemma A.2. *For any $q, n \in \mathbb{N}\backslash\{0\}$, and $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^n$, we have*

$$|\|\mathbf{x}\| - \|\mathbf{y}\|| \leqslant \|\mathbf{x} + \mathbf{y}\| \leqslant \|\mathbf{x}\| + \|\mathbf{y}\|.$$

A.2.2 Modular Most-Significant Bit Decomposition.

Let $\nu \in \mathbb{N}\backslash\{0\}$. Any integer $x \in \mathbb{Z}$ can be *uniquely* decomposed as:

$$x = 2^\nu \cdot x_\top + x_\perp, \quad (x_\top, x_\perp) \in \mathbb{Z} \times [-2^{\nu-1}, 2^{\nu-1} - 1], \tag{18}$$

which consists essentially in separating the lower-order bits from the higher-order ones. We define the function

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

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

$$\lfloor\cdot\rceil_\nu : \mathbb{Z}_q \mapsto \mathbb{Z}_{q_\nu} = \mathbb{Z}_{\lfloor q/2^\nu\rceil} \quad \text{s.t.} \quad \lfloor x\rceil_\nu = \lfloor \bar{x}/2^\nu\rceil + q_\nu\mathbb{Z} = (\bar{x})_\top + q_\nu\mathbb{Z},$$

The function $\lfloor\cdot\rceil_\nu$ naturally extends to vectors coefficient-wise. The following is a special case of [dPKM+24]. This bound on modular rounding operations are useful when arguing the small offset caused by performing modular rounding for efficiency.

Lemma A.3. *Let ν, q be positive integers such that $q > 2^\nu$, $\nu \geqslant 4$, and set $q_\nu = \lfloor q/2^\nu\rceil$. Moreover, assume q and ν satisfy $q_\nu = \lfloor q/2^\nu\rfloor$, that is, q can be decomposed as $q = 2^\nu \cdot q_\nu + q_\perp$ for $q_\perp \in [0, 2^{\nu-1} - 1]$. Then, for any $x \in \mathbb{Z}_q$, we have*

$$\left|x - 2^\nu \cdot \overline{\lfloor x\rceil_\nu}\right| \leqslant 2^\nu - 1. \tag{19}$$

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

$$\left\|2^\nu \left(\overline{\lfloor \mathbf{x} + \boldsymbol{\delta}\rceil_\nu - \lfloor \mathbf{x}\rceil_\nu}\right)\right\| \leqslant \left\|2^\nu \overline{\lfloor \boldsymbol{\delta}\rceil_\nu}\right\| + \|\mathbf{1}\| \cdot \mathbf{2}^\nu.$$

Throughout this work, we will not be as precise as above for better readability. For instance, we might informally use x instead of the lift $\bar{x}$ or write $|2^\nu \cdot x|$ instead of $|2^\nu \cdot \bar{x} \mod q|$ when the context is clear and the distinction is unimportant.

A.3 Hardness of Lattice-Related Problems

Here we provide all the omitted details on the lattice-related hardness problems. The following is the standard notions of the MLWE and MSIS problem.

Definition A.4 (MLWE)**.** *Let ℓ, k, q be integers and $\mathcal{D}$ be a probability distribution over $\mathcal{R}_q$. The advantage of an adversary $\mathcal{A}$ against the* Module Learning with Errors $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ *problem is defined as:*

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

where $(\mathbf{A}, \mathbf{b}, \mathbf{s}, \mathbf{e}) \leftarrow \mathcal{R}_q^{k\times\ell} \times \mathcal{R}_q^k \times \mathcal{D}^\ell \times \mathcal{D}^k$. The $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage. We may write $\mathsf{MLWE}_{q,\ell,k,\sigma}$ as a shorthand for $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ when $\mathcal{D}$ is the Gaussian distribution of standard deviation σ. Lastly, we also define a variant called uniform MLWE (UMLWE) *where the secret key is sampled from the uniform distribution $\mathcal{R}_q^\ell$.*

Definition A.5 (MSIS)**.** *Let ℓ, k, q be integers and $\beta > 0$ a real number. The advantage of an adversary $\mathcal{A}$ against the* Module Short Integer Solution $\mathsf{MSIS}_{q,\ell,k,\beta}$ *problem, is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{MSIS}}(1^\lambda) = \Pr\left[\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k\times\ell}, \mathbf{s} \xleftarrow{\$} \mathcal{A}(\mathbf{A}) : (0 < \|\mathbf{s}\|_2 \leqslant \beta) \ \wedge\ \left[\mathbf{A} \mid \mathbf{I}\right]\mathbf{s} = \mathbf{0} \mod q\right].$$

The $\mathsf{MSIS}_{q,\ell,k,\beta}$ assumption states that any efficient adversary $\mathcal{A}$ has negligible advantage.

Lastly, we define a useful distribution associated to the rounded MLWE instance.

Definition A.6. *For any $\mathbf{A} \in \mathcal{R}_q^{k\times\ell}$, positive integers rep and ν, let $\mathcal{D}_{q,\ell,k,\sigma,\mathsf{rep},\nu}^{\mathsf{bd\text{-}MLWE}}(\mathbf{A})$ be the distribution defined as $\left\{ \lfloor \mathbf{A}\cdot\mathbf{s} + \mathbf{e} \rceil_\nu \mid (\mathbf{s}, \mathbf{e}) = (\sum_{i\in[\mathsf{rep}]} \mathbf{s}_i, \sum_{i\in[\mathsf{rep}]} \mathbf{e}_i), \forall i \in [\mathsf{rep}], (\mathbf{s}_i, \mathbf{e}_i) \xleftarrow{\$} \mathcal{D}_\sigma^\ell \times \mathcal{D}_\sigma^k \right\}$. That is, it samples rep $\mathsf{MLWE}_{q,\ell,k,\sigma}$ instances, aggregates them, and drops ν trailing bits.*

The following results establish the worst-case to average-case reductions for the MLWE and MSIS problems.

Lemma A.7 (Hardness of MLWE [LS15]). *Let $k(\lambda)$, $\ell(\lambda)$, $q(\lambda)$, $n(\lambda)$, $\sigma(\lambda)$ such that $q \leqslant \mathsf{poly}(n\ell)$, $k \leqslant \mathsf{poly}(\ell)$, and $\sigma \geqslant \sqrt{\ell}\cdot\omega(\sqrt{\log n})$. If $\mathcal{D}$ is a discrete Gaussian distribution with standard deviation σ, then the $\mathsf{MLWE}_{q,\ell,k,\mathcal{D}}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (*GIVP*) in dimension $N = n\ell$ with approximation factor $\sqrt{8\cdot N\ell}\cdot\omega(\sqrt{\log \ell})\cdot q/\sigma$.*

Lemma A.8 (Hardness of MSIS [LS15]). *For any $k(\lambda), \ell(\lambda), q(\lambda), n(\lambda), \beta(\lambda)$ such that $q > \beta\sqrt{n\ell}\cdot\omega(\log(n\ell))$, $k \leqslant \mathsf{poly}(\ell)$, and $\log q \leqslant \mathsf{poly}(n\ell)$. The $\mathsf{MSIS}_{q,\ell,k,\beta}$ problem is as hard as the worst-case lattice Generalized-Independent-Vector-Problem (*GIVP*) in dimension $N = n\ell$ with approximation factor $\beta\sqrt{N}\cdot\omega(\sqrt{\log N})$.*

We also recall the following MLWE to Hint-MLWE reduction. Below, $s_1(c)$ denotes the spectral norm of $c \in \mathcal{R}_q$ and c^* denotes the Hermitian adjoint of c. Kim et al. [KLSS23] showed that the reduction is tight.

Lemma A.9 (Hardness of Hint-MLWE [KLSS23]). *For any integers ℓ, k, q, n, Q, set $\mathcal{C} \subset \mathcal{R}_q$, and positive reals $B_{\mathsf{hint}}, \sigma, \sigma_\mathcal{D}, \sigma_\mathcal{G}$ such that $\Pr[s_1(\sum_{i\in[Q]} c_i\cdot(c_i)^*) < B_{\mathsf{hint}} : c_i \leftarrow \mathcal{C}] \geqslant 1 - \mathsf{negl}(\lambda)$, $\sigma = \omega(\sqrt{\log n})$, and $\frac{1}{\sigma^2} = 2\cdot\left(\frac{1}{\sigma_\mathcal{D}^2} + \frac{B_{\mathsf{hint}}}{\sigma_\mathcal{G}^2}\right)$, the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q,\sigma_\mathcal{D},\sigma_\mathcal{G},\mathcal{C}}$ problem is as hard as the $\mathsf{MLWE}_{q,\ell,k,\sigma}$ problem.*

Lastly, we recall the following MSIS to SelfTargetMSIS reduction. The reduction is a simple invocation of the forking lemma [PS00, BN06], running the adversary against the SelfTargetMSIS problem twice via rewinding. del Pino et al. [dPKM+24] provides a concrete bound on the reduction.

Lemma A.10 (Hardness of SelfTargetMSIS [dPKM+24]). *Let ℓ, k, q be integers and $B_{\mathsf{stmsis}} > 0$ be a real number. Let $\mathcal{C}$ be a subset of $\mathcal{R}_q$ and let $\mathsf{H} : \mathcal{R}_q^k \times \{0,1\}^{2\lambda} \to \mathcal{C}$ be a cryptographic hash function modeled as a random oracle. For any adversary $\mathcal{A}$ against the $\mathsf{SelfTargetMSIS}_{q,\ell,k,\mathsf{H},\mathcal{C},B_{\mathsf{stmsis}}}$ problem making at most Q_H queries to H, there exists an adversary $\mathcal{B}$ against the $\mathsf{MSIS}_{q,\ell,k,B_{\mathsf{msis}}}$ problem with $B_{\mathsf{msis}} = 2B_{\mathsf{stmsis}}$ such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetMSIS}}(\lambda) \leqslant \sqrt{Q_\mathsf{H}\cdot\mathsf{Adv}_{\mathcal{B}}^{\mathsf{MSIS}}(\lambda)} + \frac{Q_\mathsf{H}}{|\mathcal{C}|},$$

where $\mathsf{Time}(\mathcal{B}) \approx 2\cdot\mathsf{Time}(\mathcal{A})$.

A.4 Hardness of DL-Related Problems

Here we provide all the omitted details on the DL-related hardness problems. Let GenG be an algorithm that on input 1^λ, outputs a tuple $(\mathbb{G}, p, G)$, where G is a generator of cyclic group $\mathbb{G}$ of prime order p.

First, we recall the standard notions of the DL problem.

Definition A.11. *Let $(\mathbb{G}, p, G) \leftarrow \mathsf{GenG}(1^\lambda)$. The advantage of an adversary $\mathcal{A}$ against the DL problem, is defined as:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{DL}}(\lambda) = \Pr\left[x \xleftarrow{\$} \mathbb{Z}_p, X := x \cdot G, x' \xleftarrow{\$} \mathcal{A}^{\mathsf{H}}(X) : X = x' \cdot G\right].$$

The DL assumption states that any efficient adversary $\mathcal{A}$ has no more than negligible advantage.

Now, we recall the following two DL to $\mathsf{SelfTargetDL}$ reductions. In [BD21], Bellare and Dai provided the non-tight reduction and the tight reduction in the algebraic group model (AGM). Note that $\mathsf{SelfTargetDL}$ is called IDL. In particular, the non-tight reduction rewinds the adversary against $\mathsf{SelfTargetDL}$ problem once, and the success probability of the reduction is derived by using the forking lemma.

Lemma A.12 (Hardness of $\mathsf{SelfTargetDL}$ [BD21]). *Let $(\mathbb{G}, p, G) \leftarrow \mathsf{GenG}(1^\lambda)$. Let $\mathsf{H} : \mathbb{G}^2 \times \{0,1\}^{2\lambda} \to \mathbb{Z}_p$ be a cryptographic function modeled as a random oracle. For any adversary $\mathcal{A}$ against the $\mathsf{SelfTargetDL}$ problem making at most Q_{H} queries to H, there exists an adversary $\mathcal{B}$ against the DL problem such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetDL}}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}} \cdot \mathsf{Adv}_{\mathcal{B}}^{\mathsf{DL}}(\lambda)} + \frac{Q_{\mathsf{H}}}{p}$$

where $\mathsf{Time}(\mathcal{B}) \approx 2 \cdot \mathsf{Time}(\mathcal{A})$.

Moreover, for any algebraic *adversary $\mathcal{A}$ against the $\mathsf{SelfTargetDL}$ problem making at most Q_{H} queries to H, there exists an adversary $\mathcal{B}$ against the DL problem such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{SelfTargetDL}}(\lambda) \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{DL}}(\lambda) + \frac{Q_{\mathsf{H}}}{p}$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

A.5 Forking Lemmas

The forking lemma was originally introduced by Pointcheval and Stern [PS00] in the context of signature schemes. The lemma was later reformulated by Bellare and Neven [BN06] which extracts the purely probabilistic nature of the forking lemma. Below, we review the Bellare-Neven general forking lemma.

Lemma A.13 (General Forking Lemma). *Fix an integer $q \geqslant 1$ and a set $\mathcal{H}$ of size $h \geqslant 2$. Let $\mathcal{A}$ be a randomized algorithm on input $\mathsf{par}, \vec{h} :=:= (h_1, \cdots, h_q)$ that returns $J \in [0, \cdots, q]$ 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:*

$$\mathsf{acc} = \Pr\left[\mathsf{par} \xleftarrow{\$} \mathsf{IG},\ \vec{h} \xleftarrow{\$} \mathcal{H}^q,\ (J, \sigma) \xleftarrow{\$} \mathcal{A}(\mathsf{par}, \vec{h}) : J \geqslant 1\right].$$

The forking algorithm $\mathsf{Fork}_{\mathcal{A}}$ *associated to $\mathcal{A}$ is a randomized algorithm that takes input par and proceeds as in Fig. 9. Let*

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

Then,

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

Algorithm $\mathsf{Fork}_{\mathcal{A}}(\mathsf{par})$

$\mathsf{coin} \xleftarrow{\$} \{0,1\}^{\ell_{\mathcal{A}}}$ // $\ell_{\mathcal{A}}$-bit randomness used by $\mathcal{A}$

$\vec{h} := (h_1, \cdots, h_q) \xleftarrow{\$} \mathcal{H}^q$

$(J, \sigma) := \mathcal{A}(\mathsf{par}, \vec{h}; \rho)$

if $I = 0$ **then**

  return $(0, (\perp, \perp))$

$(h'_I, \cdots, h'_q) \xleftarrow{\$} \mathcal{H}^{q-I+1}$

$\vec{h}' := (h_1, \cdots, h_{I-1}, h'_I, \cdots, h'_q)$

$(J', \sigma') := \mathcal{A}(\mathsf{par}, \vec{h}'; \rho)$

if $J = J' \ \wedge \ h_J \neq h'_J$ **then**

  return $(1, (\sigma, \sigma'))$

else

  return $(0, (\perp, \perp))$

Figure 9: Description of the forking algorithm $\mathsf{Fork}_{\mathcal{A}}$.

B Details of Our 4-Round Threshold Raccoon

B.1 Construction

Here, we provide the construction of our 4-round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ in Fig. 10. We only show the procedure of the singing protocol since the setup, key generation, and verification algorithms are the same as those of 5 round scheme $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ in Fig. 4. Parameters, the helper algorithm $\mathsf{ZeroShare}$, the signature scheme, and hash functions are identical to those in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. For the detail of them, see Section 5.1.

B.2 Security

Below, we provide the main theorem establishing adaptive security of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$.

Theorem B.1. *The 4-round threshold signature* $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ *in Figs. 4 and 10 is adaptive secure under the* $\mathsf{Hint\text{-}MLWE}$ *and* MSIS *assumptions.*

Formally, for any N *and* T *with* $T \leqslant N$ *and an adversary* $\mathcal{A}$ *against the adaptive security game making at most* Q_{H_c}, $Q_{\mathsf{H}_{\mathsf{com}}}$, $Q_{\mathsf{H}_{\mathsf{mask}}}$, *and* Q_{S} *queries to the random oracles* H_c, $\mathsf{H}_{\mathsf{com}}$, *and* $\mathsf{H}_{\mathsf{mask}}$ *and the signing oracle, respectively, there exists adversaries* $\mathcal{B}$, $\mathcal{B}'$, *and* $\mathcal{B}_{\mathsf{S}}$ *against the* $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$, $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B_{\mathsf{stmsis}}}$ *problems, and the unforgeability of signatures, respectively, such that*

$$\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T, 1) \leqslant Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(1^\lambda) + \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(1^\lambda) + N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S},\mathcal{B}_{\mathsf{S}}}(\lambda)$$
$$+ \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{2^{n-1}} + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda)$$

where $\mathsf{Time}(\mathcal{B}), \mathsf{Time}(\mathcal{B}'), \mathsf{Time}(\mathcal{B}_{\mathsf{S}}) \approx \mathsf{Time}(\mathcal{A})$. *From Lemma A.10, we can replace* $\mathcal{B}'$ *by an adversary* $\mathcal{B}''$ *against the* $\mathsf{MSIS}_{q,\ell+1,k,2B}$ *problem with* $\mathsf{Time}(\mathcal{B}'') \approx 2 \cdot \mathsf{Time}(\mathcal{B}')$ *such that*

$$\mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{MSIS}}_{\mathcal{B}''}(\lambda)} + \frac{Q_{\mathsf{H}_c}}{|\mathcal{C}|}.$$

$\mathsf{Sign}_1(\mathsf{vk}, \boxed{\mathsf{sid}}, \mathsf{SS}, i, \mathsf{sk}_i, \mathsf{st}_i)$

- $\textbf{req}\ [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
- $\textbf{parse}\ (\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},i})_{i\in[N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$
- $\boxed{\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|\mathsf{sid}}$
- $\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$
- $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^\ell \times \mathcal{D}_{\mathbf{w}}^k$
- $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$
- $\widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$
- $\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \widetilde{\mathbf{w}}_i)$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\textbf{return}\ (\mathsf{pm}_{1,i} := \mathsf{cmt}_i, \mathsf{st}_i)$

$\mathsf{Sign}_2(\mathsf{vk}, \boxed{\mathsf{sid}}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

- $\textbf{req}\ [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{pm}_{1,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
- $\textbf{pick}\ (\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\ \textbf{from}\ \mathsf{st}_i$
- $\textbf{parse}\ (\mathsf{cmt}_j)_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}\setminus\{i\}}$ with $\mathsf{pm}_{1,i} = \mathsf{cmt}_i$
- $\boxed{\mathsf{M}_\mathsf{S} := \mathsf{SS}\|\mathsf{M}\|\mathsf{sid}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}}$
- $\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_\mathsf{S}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_\mathsf{S})$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\textbf{return}\ (\mathsf{pm}_{2,i} := \sigma_{\mathsf{S},i}, \mathsf{st}_i)$

$\mathsf{Sign}_3(\mathsf{vk}, \boxed{\mathsf{sid}}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

- $\textbf{req}\ [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{2,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
- $\textbf{pick}\ (\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\ \textbf{from}\ \mathsf{st}_i$ with $\mathsf{pm}_{2,i} = \sigma_{\mathsf{S},i}$
- $\textbf{parse}\ (\sigma_{\mathsf{S},j})_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}\setminus\{i\}}$
- $\boxed{\mathsf{M}_\mathsf{S} := \mathsf{SS}\|\mathsf{M}\|\mathsf{sid}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}}$
- $\textbf{req}\ [\![\forall j \in \mathsf{SS}\setminus\{i\}, \mathsf{Verify}_\mathsf{S}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_\mathsf{S}) = \top]\!]$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\textbf{return}\ (\mathsf{pm}_{3,i} := \widetilde{\mathbf{w}}_i, \mathsf{st}_i)$

$\mathsf{Sign}_4(\mathsf{vk}, \boxed{\mathsf{sid}}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

- $\textbf{req}\ [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}, \cdot) \in \mathsf{st}_i]\!]$
- $\textbf{parse}\ (\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}\setminus\{i\}}$
- $\textbf{pick}\ (\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\ \textbf{from}\ \mathsf{st}_i$ with $\mathsf{pm}_{3,i} = \widetilde{\mathbf{w}}_i$
- $\textbf{req}\ [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)]\!]$
- $\boxed{\mathsf{ctnt}_{\mathbf{z}} := 1\|\mathsf{SS}\|\mathsf{M}\|\mathsf{sid}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}\|(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}}$
- $\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
- $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ // $c \in \mathcal{C}$
- $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^\ell$
- $\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^\ell$
- $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
- $\textbf{return}\ (\mathsf{pm}_{4,i} := \widetilde{\mathbf{z}}_i, \mathsf{st}_i)$

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{b,j})_{(b,j)\in[4]\times\mathsf{SS}})$

- $\textbf{parse}\ (\widetilde{\mathbf{w}}_j, \widetilde{\mathbf{z}}_j)_{j\in\mathsf{SS}} \leftarrow \boxed{(\mathsf{pm}_{3,j}, \mathsf{pm}_{4,j})_{j\in\mathsf{SS}}}$
- $\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}}$
- $\mathbf{z} := \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{z}}_j \in \mathcal{R}_q^\ell$
- $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$
- $\mathbf{y} := \lfloor \mathbf{A}\mathbf{z} - 2^{\nu_{\mathbf{t}}} \cdot c \cdot \mathbf{t} \rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
- $\mathbf{h} := \mathbf{w} - \mathbf{y} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$
- $\textbf{return}\ \mathsf{sig} := (c, \mathbf{z}, \mathbf{h})$

Figure 10: The signing protocol of our four round threshold signature $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$. In the above, $L_{\mathsf{SS},i}$ denotes the Lagrange coefficient of user i in the set $\mathsf{SS} \subseteq [N]$ (see Section 2.3 for the definition), and sid is a session identifier that is never been reused. **pick** X **from** Y denotes the process of picking an element X from the set Y. The setup Setup, key generation KeyGen, verification $TSVf$ algorithm are identical to those of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ in Fig. 4.

We omit the formal proof and only provide the rough proof sketch since the proof of this theorem is almost identical to the proof of Theorem 6.1. The main difference is that the modification in Game_2 in the proof of Theorem 6.1 is not required. Recall that this modification is to ensure that the same $\mathsf{ctnt}_{\mathbf{w}}$ is never reused in the singing oracle by showing that the same string str_i is never generated twice. On the other hand, in $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$, this statement is immediately guaranteed by non-reuseability of the session identifier sid. In the remaining parts of proof, we can argue similarly to the proof of Theorem 6.1, using the fact that sid is never reused instead of the fact that $(\mathsf{str}_j)_{j\in\mathsf{SS}}$ is not reused. Eventually, we can bound the advantage of the adversary $\mathcal{A}$ as in the above theorem. Notice that the loss $Q_{\mathsf{S}}^2/2^{2\lambda}$, that arises from the modification in Game_2, is disappeared compared to Theorem 6.1.

C Details of Our 4-Round Threshold Schnorr

C.1 Construction

Here, we provide the construction of our 4-round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ in Fig. 11. We only show the procedure of the singing protocol since the setup, key generation, and verification algorithms are the same as those of 5 round scheme $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ in Fig. 6. Parameters, the helper algorithm $\mathsf{ZeroShare}$, the signature scheme, and hash functions are identical to those in $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. For the detail of them, see Section 7.

C.2 Security

Below, we provide the main theorem establishing adaptive security of $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$.

Theorem C.1. *The 5-round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$ in Figs. 6 and 11 is adaptive secure under the $\mathsf{SelfTargetDL}$ assumption.*

Formally, for any N and T with $T \leqslant N$ and an adversary $\mathcal{A}$ against the adaptive security game making at most Q_{H_c}, $Q_{\mathsf{H}_{\mathsf{com}}}$, $Q_{\mathsf{H}_{\mathsf{mask}}}$, and Q_{S} queries to the random oracles H_c, $\mathsf{H}_{\mathsf{com}}$, and $\mathsf{H}_{\mathsf{mask}}$ and the signing oracle, respectively, there exists adversaries $\mathcal{B}$ and $\mathcal{B}_{\mathsf{S}}$ against the DL problem and the unforgeability of signatures, respectively, such that

$$\mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T, 1) \leqslant Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetDL}}_{\mathcal{B}}(1^\lambda) + N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S},\mathcal{B}_{\mathsf{S}}}(\lambda) + \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{p} + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}},$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ and $\mathsf{Time}(\mathcal{B}_{\mathsf{S}}) \approx \mathsf{Time}(\mathcal{A})$. From Lemma A.12, we can replace $\mathcal{B}$ by an adversary $\mathcal{B}'$ against the DL problem with $\mathsf{Time}(\mathcal{B}') \approx 2 \cdot \mathsf{Time}(\mathcal{B})$ such that

$$\mathsf{Adv}^{\mathsf{SelfTargetDL}}_{\mathcal{B}}(\lambda) \leqslant \sqrt{Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathcal{B}'}(\lambda)} + \frac{Q_{\mathsf{H}_c}}{p}.$$

This theorem also immediately is obtained from Theorem 7.1. The idea of the proof is identical to that of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$. For the details of the idea, see Appendix B.

D Candidate Parameters for Our Threshold Raccoon

We propose three threshold signatures from Raccoon: $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$, $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$, and $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$. These schemes can all be proven correct and secure under the same set of parameters. More importantly, these are exactly the same as those used by the statically secure 3 round threshold Raccoon by del Pino et al. [dPKM+24].

For completeness, we provide a set of candidate asymptotic parameters for which all three of our schemes are secure under. This is taken from [dPKM+24, Section 7.1].

$\mathsf{Sign}_1(\mathsf{vk}, \mathsf{sid}, \mathsf{SS}, i, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
$\textbf{parse } \mathsf{sk}_i \leftarrow (x_i, (\mathsf{vk}_{\mathsf{S},i})_{i\in[N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i)$
$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| \mathsf{sid}$
$\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathbb{G}$
$r_i \xleftarrow{\$} \mathbb{Z}_p; R_i := r_i \cdot G$
$\widetilde{R}_i := R_i + \widetilde{\mathbf{\Delta}}_i \in \mathbb{G}$
$\mathsf{cmt}_i := \mathsf{H}_{\mathsf{com}}(i, \widetilde{R}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{\mathsf{str}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{1,i} := \mathsf{cmt}_i, \mathsf{st}_i)$

$\mathsf{Sign}_2(\mathsf{vk}, \mathsf{sid}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{pm}_{1,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\textbf{parse } (\mathsf{cmt}_j)_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}\backslash\{i\}}$
$\quad \text{with } \mathsf{pm}_{2,i} = \mathsf{cmt}_i$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| \mathsf{sid} \| (\mathsf{cmt}_j)_{j\in\mathsf{SS}}$
$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_{\mathsf{S}}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_{\mathsf{S}})$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{sid}, \mathsf{SS}, \mathsf{cmt}_i, \widetilde{R}_i, r_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{2,i} := \sigma_{\mathsf{S},i}, \mathsf{st}_i)$

$\mathsf{Sign}_3(\mathsf{vk}, \mathsf{sid}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{2,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\quad \text{with } \mathsf{pm}_{2,i} = \sigma_{\mathsf{S},i}$
$\textbf{parse } (\sigma_{\mathsf{S},j})_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}\backslash\{i\}}$
$\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| \mathsf{sid} \| (\mathsf{cmt}_j)_{j\in\mathsf{SS}}$
$\textbf{req } [\![\forall j \in \mathsf{SS}\backslash\{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{R}_i, r_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{3,i} := \widetilde{R}_i, \mathsf{st}_i)$

$\mathsf{Sign}_4(\mathsf{vk}, \mathsf{sid}, \mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}}, \mathsf{sk}_i, \mathsf{st}_i)$

$\textbf{req } [\![(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{parse } (\widetilde{R}_j)_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}\backslash\{i\}}$
$\textbf{pick } (\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{R}_i, r_i) \textbf{ from } \mathsf{st}_i$
$\quad \text{with } \mathsf{pm}_{3,i} = \widetilde{R}_i$
$\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{R}_j)]\!]$
$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} \| \mathsf{M} \| \mathsf{sid} \| (\mathsf{cmt}_j)_{j\in\mathsf{SS}} \| (\widetilde{R}_j)_{j\in\mathsf{SS}}$
$R := \sum_{j\in\mathsf{SS}} \widetilde{R}_j \in \mathbb{G}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, R) \quad /\!/ \ c \in \mathbb{Z}_p$
$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathbb{Z}_p$
$\widetilde{z}_i := c \cdot L_{\mathsf{SS},i} \cdot x_i + r_i + \mathbf{\Delta}_i \in \mathbb{Z}_p$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{sid}, \mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{R}_i, r_i)\}$
$\textbf{return } (\mathsf{pm}_{5,i} := \widetilde{z}_i, \mathsf{st}_i)$

$\mathsf{Agg}(\mathsf{vk}, \mathsf{SS}, \mathsf{M}, (\mathsf{pm}_{b,j})_{(b,j)\in[4]\times\mathsf{SS}})$

$\textbf{parse } (\widetilde{R}_j, \widetilde{z}_j)_{j\in\mathsf{SS}} \leftarrow (\mathsf{pm}_{3,j}, \mathsf{pm}_{4,j})_{j\in\mathsf{SS}}$
$R := \sum_{j\in\mathsf{SS}} \widetilde{R}_j$
$z := \sum_{j\in\mathsf{SS}} \mathbf{z}_j$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, R)$
$\textbf{return } \mathsf{sig} := (c, z)$

Figure 11: The Signing protocol of our four round threshold signature $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}}$. In the above, $L_{\mathsf{SS},i}$ denotes the Lagrange coefficient of user i in the set $\mathsf{SS} \subseteq [N]$ (see Section 2.3 for the definition), and sid is a session identifier that is never been reused. **pick** X **from** Y denotes the process of picking an element X from the set Y. The setup Setup, key generation KeyGen, verification Verify algorithm are identical to those of $\mathsf{TSchnorr}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$ in Fig. 6.

- $n, \ell, k = \mathsf{poly}(\lambda)$ such that $n \geqslant \lambda$.
- $W = \omega(1)$ for $|\mathcal{C}| \geqslant 2^\lambda$ for Lemma A.10 (hardness of MSIS).
- $B_{\mathsf{hint}} = Q_{\mathsf{S}} \cdot W \cdot \left(1 + n\frac{1}{\sqrt{Q_{\mathsf{S}}}}(\lambda + 1 + 2\log(n))\right)$, $\frac{1}{\sigma^2} = 2 \cdot \left(\frac{1}{\sigma_{\mathbf{t}}^2} + \frac{B_{\mathsf{hint}}}{\sigma_{\mathbf{w}}^2}\right)$, and $\sigma \geqslant \sqrt{\ell} \cdot \omega(\sqrt{\log n})$ for Lemmata A.7 and A.9 and [dPKM+24, Lemma B.2] (reduction from Hint-MLWE to MLWE and hardness of MLWE).
- $(\sigma_{\mathbf{t}}, \sigma_{\mathbf{w}}) = \left(2\sqrt{\ell} \cdot \log n, 2\sqrt{B_{\mathsf{hint}} \cdot \ell} \cdot \log n\right)$.
- $\nu_{\mathbf{t}}, \nu_{\mathbf{w}} = O(\log \lambda)$, where $\nu_{\mathbf{w}} \geqslant 4$ for correctness (see Lemmata 3.1 and 5.1).
- $B = e^{1/4} \cdot (W\,\sigma_{\mathbf{t}} + \sqrt{T}\,\sigma_{\mathbf{w}})\sqrt{n(k+\ell)} + (W \cdot 2^{\nu_{\mathbf{t}}} + 2^{\nu_{\mathbf{w}}+1}) \cdot \sqrt{nk}$ for correctness (see Lemmata 3.1 and 5.1).
- $B_{\mathsf{stmsis}} = B + \sqrt{W} + (W \cdot 2^{\nu_{\mathbf{t}}} + 2^{\nu_{\mathbf{w}}+1}) \cdot \sqrt{nk}$ for Lemmata E.7 and E.18.
- q is the smallest prime larger than $2B_{\mathsf{stmsis}} \cdot \sqrt{nk} \cdot \log(nk)^2$ such that $(q, \nu_{\mathbf{t}}, \nu_{\mathbf{w}})$ satisfy the condition in Table 3 (hardness of MSIS).

For concreter parameter sets aiming $\{128, 192, 256\}$-bits security, we refer the readers to [dPKM+24, Section 8].

E Formal Security Proofs

In this section, we provide the full proofs that were deferred from the main body.

E.1 Formal Security Proof of $\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}}$

We now provide a formal proof of Theorem 4.1.

Proof. Let $\mathcal{A}$ be an adversary against the selective security game. We consider a sequence of games where the first hybrid is the original game and the last is a game that can be reduced to the MSIS problem. Throughout the game, we divide the signer set SS into honest users $\mathsf{sHS} := \mathsf{SS} \cap \mathsf{HS}$ and corrupt users $\mathsf{sCS} := \mathsf{SS} \cap \mathsf{CS}$. We relate the advantage of $\mathcal{A}$ for each adjacent games, where ϵ_i denotes the advantage of $\mathcal{A}$ in Game_i.

Game_1: This is the real unforgeability game. Formally, this is depicted in Fig. 12. By definition, we have

$$\epsilon_1 := \mathsf{Adv}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T, 1).$$

Game_2: In this game, the challenger postpones generating $\mathbf{w}_i$ until $\mathcal{O}_{\mathsf{Sign}_2}$. This is depicted in Fig. 13. Specifically, the challenger outputs a random hash commitment $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_1}$ and removes the commitment-related values $(\widetilde{\mathbf{w}}_i, \mathbf{r}_i)$ from the state st_i. In $\mathcal{O}_{\mathsf{Sign}_2}$, it computes $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ as in $\mathcal{O}_{\mathsf{Sign}_1}$ of Game_1 and programs $\mathsf{H}_{\mathsf{com}}$ via $\mathsf{ProgramHashCom}$ such that we have $\mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i) = \mathsf{cmt}_i$. Also, it reintroduces $(\mathbf{w}_i, \mathbf{r}_i)$ into the state st_i at the end of $\mathcal{O}_{\mathsf{Sign}_2}$. Note that it aborts if $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}_i] \neq \bot$ holds in $\mathsf{ProgramHashCom}$.

Conditioned on the game does not abort, the view of two games are identically distributed to $\mathcal{A}$, since cmt_i and $\mathbf{w}_i$ are generated in the same manner and cmt_i is opened in $\mathcal{O}_{\mathsf{Sign}_2}$ in both games. Since the commitment $\mathbf{w}_i$ is honestly generated, the probability that $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \mathbf{w}_i] \neq \bot$ is at most $(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})/2^{n-1}$ with overwhelming probability due to Lemma 2.8. Since $\mathcal{A}$ makes at most Q_{S} signing queries, we have

$$|\epsilon_2 - \epsilon_1| \leqslant \frac{Q_{\mathsf{S}}(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})}{2^{n-1}} + \mathsf{negl}(\lambda).$$

$\mathsf{Game}_1 := \mathsf{Game}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T)$

$\mathsf{Q_M}[\cdot] := \varnothing, \mathsf{Q}_{\mathsf{H}_c}[\cdot] := \bot, \mathsf{Q}_{\mathsf{H_{com}}}[\cdot] := \bot, \mathsf{Q}_{\mathsf{H_{mask}}}[\cdot] := \bot$
$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times \ell}$
$(\mathsf{CS}, \mathsf{st}_\mathcal{A}) \xleftarrow{\$} \mathcal{A}^{\mathsf{H}_c, \mathsf{H}_\beta, \mathsf{H_{mask}}}(\mathbf{A}, N, T)$
req $[\![\mathsf{CS} \subseteq [N]]\!] \wedge [\![|\mathsf{CS}| \leqslant T-1]\!]$
$\mathsf{HS} := [N] \backslash \mathsf{CS}$
for $i \in \mathsf{HS}$ **do** $\mathsf{st}_i := \varnothing$
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_\mathbf{t}^\ell \times \mathcal{D}_\mathbf{t}^k$
$\mathbf{t} := \lfloor \mathbf{A}\mathbf{s} + \mathbf{e} \rceil_{\nu_\mathbf{t}} \in \mathcal{R}^k_{q_{\nu_\mathbf{t}}}$
for $i \in [N]$ **do**
  for $j \in [N]$ **do**
    $\mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$
    $\mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$
$(\vec{\mathsf{seed}}_i)_{i \in [N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in [N]}\right)_{i \in [N]}$
$\vec{P} \xleftarrow{\$} \mathcal{R}_q^\ell[X]$ with $\deg(\vec{P}) = T-1, \vec{P}(0) = \mathbf{s}$
$(\mathbf{s}_i)_{i \in [N]} := (\vec{P}(i))_{i \in [N]}$
$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$
$(\mathsf{sk}_i)_{i \in [N]} := \left(\mathbf{s}_i, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$
$\mathsf{oracles} := (\mathcal{O}_{\mathsf{Sign}_1}, \mathcal{O}_{\mathsf{Sign}_2}, \mathcal{O}_{\mathsf{Sign}_3}, \mathsf{H}_c, \mathsf{H_{com}}, \mathsf{H_{mask}})$
$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{CS}}, \mathsf{st}_\mathcal{A})$
req $[\![\left|\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}\right| \leqslant T-1]\!]$
return $\mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathcal{O}_{\mathsf{Sign}_1}(i)$

req $[\![i \in \mathsf{HS}]\!]$
$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_\mathbf{w}^\ell \times \mathcal{D}_\mathbf{w}^k$
$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$
$\mathsf{cmt}_i := \mathsf{H_{com}}(i, \mathbf{w}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)\}$
return $\mathsf{pm}_{1,i} := \mathsf{cmt}_i$

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}})$

req $[\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
req $[\![(\mathsf{pm}_{1,i}, \cdot, \cdot) \in \mathsf{st}_i]\!] \wedge [\![i \in \mathsf{SS}]\!]$
pick $(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{1,i} = \mathsf{cmt}_i$
parse $(\mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{cmt}_i, \mathbf{w}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
return $\mathsf{pm}_{2,i} := \mathbf{w}_i$

$\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$

if $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] = \bot]\!]$ **then**
  $c \xleftarrow{\$} \mathcal{C}$
  $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$
return $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}]$

$\mathsf{H_{com}}(i, \mathbf{w}_i)$

if $[\![\mathsf{Q}_{\mathsf{H_{com}}}[i, \mathbf{w}_i] = \bot]\!]$ **then**
  $\mathsf{cmt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
  $\mathsf{Q}_{\mathsf{H_{com}}}[i, \mathbf{w}_i] \leftarrow \mathsf{cmt}$
return $\mathsf{Q}_{\mathsf{H_{com}}}[i, \mathbf{w}_i]$

$\mathsf{H_{mask}}(\mathsf{seed}, \mathsf{ctnt}_\mathbf{z})$

if $[\![\mathsf{Q}_{\mathsf{H_{mask}}}[\mathsf{seed}, \mathsf{ctnt}_\mathbf{z}] = \bot]\!]$ **then**
  $\mathbf{m} \xleftarrow{\$} \mathcal{R}_q^\ell$
  $\mathsf{Q}_{\mathsf{H_{mask}}}[\mathsf{seed}, \mathsf{ctnt}_\mathbf{z}] \leftarrow \mathbf{m}$
return $\mathsf{Q}_{\mathsf{H_{mask}}}[\mathsf{seed}, \mathsf{ctnt}_\mathbf{z}]$

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

req $[\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{2,i}, \cdot) \in \mathsf{st}_i]\!]$
parse $(\mathbf{w}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}$
pick $(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)$ **from** st_i
  with $\mathsf{pm}_{2,i} = \mathbf{w}_i$
req $[\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H_{com}}(\mathsf{SS}, \mathsf{M}, j, \mathbf{w}_j)]\!]$
$\mathsf{ctnt}_\mathbf{z} := \mathsf{SS} || \mathsf{M} || (\mathsf{cmt}_j, \mathbf{w}_j)_{j \in \mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_\mathbf{w}}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$
$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_\mathbf{z}) \in \mathcal{R}_q^\ell$
$\mathbf{z}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
$\mathsf{Q_M}[\mathsf{M}] \leftarrow \mathsf{Q_M}[\mathsf{M}] \cup \{i\}$
return $\mathsf{pm}_{3,i} := \mathbf{z}_i$

Figure 12: The first game, identical to the real selective security game.

Game_2:

$\mathcal{O}_{\mathsf{Sign}_1}(\mathsf{SS}, \mathsf{M}, i)$

$\textbf{req}\ [\![i \in \mathsf{HS}]\!]$
$\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{cmt}_i)\}$
$\textbf{return}\ \mathsf{pm}_{1,i} := \mathsf{cmt}_i$

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, \mathsf{M}, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}})$

$\textbf{req}\ [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$
$\textbf{req}\ [\![(\mathsf{pm}_{1,i}) \in \mathsf{st}_i]\!] \wedge [\![i \in \mathsf{SS}]\!]$
$\textbf{parse}\ (\mathsf{cmt}_j)_{j\in\mathsf{SS}} \leftarrow (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}}$
$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$
$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$
$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \mathbf{w}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{cmt}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
$\textbf{return}\ \mathsf{pm}_{2,i} := \mathbf{w}_i$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \mathbf{w}_i)$:

$\textbf{abort if}\ [\![\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(i, \mathbf{w}_i) \neq \perp]\!]$
$\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(i, \mathbf{w}_i) \leftarrow \mathsf{cmt}_i$

Game_3:

$\mathcal{O}_{\mathsf{Sign}_1}(\mathsf{SS}, \mathsf{M}, i)$

$\textbf{req}\ [\![i \in \mathsf{HS}]\!]$
$\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\textbf{abort if}\ [\![\mathsf{cmt}_i \in \mathsf{Cmt}]\!]$
$\mathsf{Cmt} := \mathsf{Cmt} \cup \{\mathsf{cmt}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{cmt}_i)\}$
$\textbf{return}\ \mathsf{pm}_{1,i} := \mathsf{cmt}_i$

$\mathsf{H}_{\mathsf{com}}(i, \mathbf{w}_i)$

$\textbf{if}\ [\![\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \mathbf{w}_i] = \perp]\!]\ \textbf{then}$
$\quad \mathsf{cmt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\quad \textbf{abort if}\ [\![\mathsf{cmt} \in \mathsf{Cmt}]\!]$
$\quad \mathsf{Cmt} := \mathsf{Cmt} \cup \{\mathsf{cmt}\}$
$\quad \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \mathbf{w}_i] \leftarrow \mathsf{cmt}$
$\textbf{return}\ \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \mathbf{w}_i]$

Figure 13: The second and third games. The differences are highlighted in blue. We assume Game_3 initializes an empty set $\mathsf{Cmt} := \varnothing$ at the beginning of the game. Algorithm $\mathsf{ProgramHashCom}$ is a helper algorithm for programming the random oracle $\mathsf{H}_{\mathsf{com}}$ to open the hash commitments cmt_i consistently. This is assumed to have a joint state with the challenger and random oracle $\mathsf{H}_{\mathsf{com}}$ used by the unforgeability game.

Game_3**:** In this game, the challenger aborts if there is a collision in $\mathsf{H}_{\mathsf{com}}$. This is depicted in Fig. 13. Specifically, the challenger initially prepares an empty set $\mathsf{Cmt} := \varnothing$. In $\mathcal{O}_{\mathsf{Sign}_1}$ and $\mathsf{H}_{\mathsf{com}}$, it checks whether the sampled commitment cmt is already in Cmt, *i.e.*, was sampled at an earlier point in the game. If so, it aborts the game. Otherwise, it adds cmt to Cmt and continues as before. Since cmt is sampled uniformly at random from $\{0,1\}^{2\lambda}$, we have

$$|\epsilon_3 - \epsilon_2| \leqslant \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2}{2^{2\lambda}}.$$

Before we proceed, let us show a useful lemma.

Lemma E.1. *All invocations of* $\mathcal{O}_{\mathsf{Sign}_3}$ *with* $\mathsf{ctnt}_{\mathbf{w}}$ *share the identical value* $\mathsf{ctnt}_{\mathbf{z}}$.

Proof. Let us inspect the first call to $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$. Here, the challenger sets $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{SS}||\mathsf{M}||(\mathsf{cmt}_j, \mathbf{w}_j)_{j\in\mathsf{SS}}$. Due to the modification made in Game_3, there is no collision in $\mathsf{H}_{\mathsf{com}}$. Also, $\widetilde{\mathbf{w}}_j$ is uniquely determined by $\mathsf{ctnt}_{\mathbf{w}}$ in $\mathcal{O}_{\mathsf{Sign}_3}$ since the challenger checks in $\mathcal{O}_{\mathsf{Sign}_3}$ that there is a partial commitment $\mathbf{w}_j$ such that $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(j, \mathbf{w}_j) = \mathsf{cmt}_j$. Thus, $\mathsf{ctnt}_{\mathbf{z}}$ is uniquely determined by $\mathsf{ctnt}_{\mathbf{w}}$. This completes the proof. ☐

Game_4**:** In this game, the challenger introduces several additional tables: $\mathsf{UnOpenedHS}$ and $\mathsf{SumComRnd}$. These tables are indexed by $\mathsf{ctnt}_{\mathbf{w}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$ and indicate the following.

- $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the set of honest uses that have not executed the second round with $\mathsf{ctnt}_{\mathbf{w}}$ yet.
- $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the set of honest uses that have not executed the third round with $\mathsf{ctnt}_{\mathbf{w}}$ yet.

This is depicted in Fig. 14.

Specifically, in $\mathcal{O}_{\mathsf{Sign}_2}$, the challenger checks if $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ is bot. If so, it sets $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$. At the end of this oracle, it updates $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\backslash\{i\}$. Also, in $\mathcal{O}_{\mathsf{Sign}_2}$, it also sets $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$ after generating $\mathbf{w}_i$.

Since these are conceptual modification, we have

$$\epsilon_4 = \epsilon_3.$$

Game_5 In this game, the challenger introduces several additional tables: $\mathsf{UnSignedHS}$, $\mathsf{Mask}_{\mathbf{z}}$, and $\mathsf{MaskedResp}$. These tables are indexed by $\mathsf{ctnt}_{\mathbf{w}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$ and and indicate the following.

- $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the sum of the commitment randomness $\mathbf{r}_j$ for honest users that have already executed the second round with $\mathsf{ctnt}_{\mathbf{w}}$.
- $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the mask $\mathbf{\Delta}_i$.
- $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the masked response $\widetilde{\mathbf{z}}_i$.

This is depicted in Fig. 14.

Specifically, in $\mathcal{O}_{\mathsf{Sign}_3}$, the challenger checks if $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ is bot. If so, it sets $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$. At the end of this oracle, it updates $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]\backslash\{i\}$. Also, in $\mathcal{O}_{\mathsf{Sign}_3}$, it stores $\mathbf{\Delta}_i$ and $\widetilde{\mathbf{z}}_i$ in $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}]$, respectively.

Since these are conceptual modification, we have

$$\epsilon_5 = \epsilon_4.$$

Game_4:

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, \mathsf{M}, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}})$

$\textbf{req}\ [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{SS}]\!]$

$\textbf{req}\ [\![(\mathsf{pm}_{1,i}) \in \mathsf{st}_i]\!] \wedge [\![i \in \mathsf{SS}]\!]$

$\textbf{parse}\ (\mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}}$

$\mathsf{ctnt}_{\mathbf{w}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j \in \mathsf{SS}}$

$\textbf{if}\ [\![\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]\ \textbf{then}$

$\quad \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$

$\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \mathbf{w}_i)$

$\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{cmt}_i)\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

$\textbf{return}\ \mathsf{pm}_{2,i} := \mathbf{w}_i$

Game_5:

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

$\textbf{req}\ [\![(\cdot, \cdot, \cdot, \mathsf{pm}_{2,i}, \cdot) \in \mathsf{st}_i]\!]$

$\textbf{parse}\ (\mathbf{w}_j)_{j \in \mathsf{SS}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}}$

$\textbf{pick}\ (\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\ \textbf{from}\ \mathsf{st}_i$

$\quad$ with $\mathsf{pm}_{2,i} = \mathbf{w}_i$

$\textbf{req}\ [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(\mathsf{SS}, \mathsf{M}, j, \mathbf{w}_j)]\!]$

$\mathsf{ctnt}_{\mathbf{z}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j, \mathbf{w}_j)_{j \in \mathsf{SS}}$

$$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$$

$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$

$\mathsf{ctnt}_{\mathbf{w}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j \in \mathsf{SS}}$

$\textbf{if}\ [\![\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]\ \textbf{then}$

$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$

$\mathbf{z}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$

$\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$

$\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

$\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

$\textbf{return}\ \mathsf{pm}_{3,i} := \mathbf{z}_i$

Figure 14: The fourth game and fifth games. The differences are highlighted in blue. We assume that, at the beginning of the game, both games initialize two empty lists $\mathsf{UnOpenedHS}[\cdot], \mathsf{SumComRnd}[\cdot] := \perp$, and Game_5 additionally initialize three empty lists $\mathsf{UnSignedHS}[\cdot], \mathsf{Mask}_{\mathbf{z}}[\cdot], \mathsf{MaskedResp}[\cdot] := \perp$.

Game_6:

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 10 in Game_3

$\quad$ **for** $j \in \mathsf{sCS}$ **do**
$\quad\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad$ **for** $j \in \mathsf{sHS} \backslash \{i\}$ **do**
$\quad\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad \mathbf{\Delta}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j}) \in \mathcal{R}_q^{\ell}$
$\quad \mathbf{z}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$
$\quad \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$
$\quad \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$
$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
$\quad \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$
$\quad$ **return** $\mathsf{pm}_{3,i} := \mathbf{z}_i$

Game_7:

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 10 in Game_3

$\quad$ **for** $j \in \mathsf{sCS}$ **do**
$\quad\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$
$\quad \widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$
$\quad$ **for** $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}}$ **do**
$\quad\quad \mathbf{m}_{i,j} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$
$\quad\quad \mathbf{m}_{j,i} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$
$\quad$ **for** $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}} \backslash \{i\}$ **do**
$\quad\quad \mathbf{m}_{i,j} \xleftarrow{\$} \mathcal{R}_q^{\ell}, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{i,j}$
$\quad\quad \mathbf{m}_{j,i} \xleftarrow{\$} \mathcal{R}_q^{\ell}, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{j,i}$
$\quad \mathbf{z}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$
$\quad \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$
$\quad \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$
$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$
$\quad \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$
$\quad$ **return** $\mathsf{pm}_{3,i} := \mathbf{z}_i$

$\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}})$

$\quad$ **if** $[\![i \| j \| \mathsf{rand} \leftarrow \mathsf{seed} \text{ correctly parses}]\!]$ **then**
$\quad\quad$ **abort if** $[\![(i,j) \in \mathsf{HS}^2]\!] \wedge [\![\mathsf{rand} = \mathsf{rand}_{i,j}]\!]$
$\quad$ **if** $[\![\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}] = \perp]\!]$ **then**
$\quad\quad \mathbf{m} \xleftarrow{\$} \mathcal{R}_q^{\ell}$
$\quad\quad \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}$
$\quad$ **return** $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}]$

Figure 15: The sixth game and seventh games. The differences are highlighted in blue.

Game_6**:** In this game, we expand the definition of $\mathsf{ZeroShare}$ for every invocation of $\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$. This is depicted in Fig. 15. Both games are identical and we have

$$\epsilon_6 = \epsilon_5.$$

Game_7**:** In this game, an abort condition is added in the random oracle $\mathsf{H}_{\mathsf{mask}}$ and the challenger modifies how it generates masks $\mathbf{\Delta}_i$ in $\mathcal{O}_{\mathsf{Sign}_3}$. This is depicted in Fig. 15. Specifically, at the beginning of $\mathsf{H}_{\mathsf{mask}}$, the challenger aborts the game if $i\|j\|\mathsf{rand} \leftarrow \mathsf{seed}$ correctly parses and $i, j \in \mathsf{HS}$ and $\mathsf{seed} = \mathsf{seed}_{i,j}$ holds. In $\mathcal{O}_{\mathsf{Sign}_3}$, it first computes $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ for $j \in \mathsf{sCS}$ as before. It sets $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ which represents the honest signers, which have not executed round 3 with $\mathsf{ctnt}_{\mathbf{w}}$. Then, for $j \in \mathsf{sHS}\backslash\widetilde{\mathsf{sHS}}_{\mathbf{z}}$ (*i.e.*, honest users after round 3), it retrieves $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ from $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$, respectively. For $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$, it picks $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ uniformly at random from $\mathcal{R}_q^{\ell}$ and stores them in $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$, respectively. Finally, it sets $\mathbf{\Delta}_i := \sum_{j \in \mathsf{SS}\backslash\{i\}}(\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$ as before.

Let us analyze the advantage of $\mathcal{A}$ in this game. First, we upper bound the probability that the challenger aborts in $\mathsf{H}_{\mathsf{mask}}$. Let $Q_{i,j}$ be the number of the random oracle queries with $i, j \in \mathsf{HS}$. Note that $\sum_{i,j \in \mathsf{HS}} Q_{i,j} \leqslant Q_{\mathsf{H}_{\mathsf{mask}}}$. In each such random oracle query to $\mathsf{H}_{\mathsf{mask}}$, the probability that $\mathsf{rand} = \mathsf{rand}_{i,j}$ is $1/2^{\lambda}$ since $\mathsf{rand}_{i,j}$ is chosen uniformly at random from $\{0,1\}^{\lambda}$ and $\mathsf{rand}_{i,j}$ is information-theoretically hidden from $\mathcal{A}$ until either user i or j is corrupted. Thus, the abort probability for fixed pairs (i, j) is at most $Q_{i,j}/2^{\lambda}$. A union bound across all honest user pairs $(i, j) \in \mathsf{HS}^2$ allows us to upper bound the abort probability with $\frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^{\lambda}}$.

Further, we have to show that if $j \in \mathsf{sHS}\backslash\widetilde{\mathsf{sHS}}_{\mathbf{z}}$, then $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$ are already initialized with the $\mathsf{H}_{\mathsf{mask}}$ outputs. Since all signers in $\mathsf{sHS}\backslash\widetilde{\mathsf{sHS}}_{\mathbf{z}}$ have already executed $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$, these values were initialized in the corresponding $\mathcal{O}_{\mathsf{Sign}_3}$ invocation with $\mathsf{ctnt}_{\mathbf{w}}$ due to Lemma E.1. Also, we have to show that if $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$, then $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}] = \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}] = \perp$ (*i.e.*, the outputs are not yet defined and are thus distributed uniformly at this point). From Lemma E.1, these are not defined in $\mathcal{O}_{\mathsf{Sign}_3}$ invocation with other $\mathsf{ctnt}'_{\mathbf{w}} (\neq \mathsf{ctnt}_{\mathbf{w}})$. Thus, $\mathcal{O}_{\mathsf{Sign}_3}$ for j with $\mathsf{ctnt}_{\mathbf{w}}$ is not invoked when $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$. Also, due to the abort condition, the adversary $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{mask}}$ on honest seeds directly. Combining these facts concludes the proof. Therefore, we have,

$$|\epsilon_7 - \epsilon_6| \leqslant \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^{\lambda}}.$$

Game_8**:** In this game, the challenger samples the masks $\mathbf{\Delta}_i$ without $\mathsf{H}_{\mathsf{mask}}$. The last mask is set consistently and the others are sampled at random. This is depicted in Fig. 16. In more detail, when the challenger computes $\mathbf{\Delta}_i$ in $\mathcal{O}_{\mathsf{Sign}_3}$, then it checks if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$, where $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ is the set of honest users that are not executed the third round with $\mathsf{ctnt}_{\mathbf{w}}$. If so, it samples $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$ at random. Otherwise, user i is the last user, so the challenger computes $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$ for $j \in \mathsf{sCS}$ and sets $\mathbf{\Delta}_i := -\sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$. As before, all masks $\mathbf{\Delta}_i$ are stored in the table $\mathsf{Mask}_{\mathbf{z}}$. Note that now, the challenger no longer programs $\mathsf{H}_{\mathsf{mask}}$ related to the correct seed $\mathsf{seed}_{i,j}$ for $i, j \in \mathsf{HS}$.

We show that Game_7 and Game_8 are identically distributed. As in Game_8, the (potential) observable differences between both games are how the challenger programs $\mathsf{H}_{\mathsf{mask}}$ for $i, j \in \mathsf{HS}$ in $\mathcal{O}_{\mathsf{Sign}_3}$ and the distribution of the masks $\mathbf{\Delta}_i$. Since $\mathsf{seed}_{i,j}$ for $i, j \in \mathsf{HS}$ is never queried to $\mathsf{H}_{\mathsf{mask}}$ due to the abort condition added in Game_7, the distribution of the output of $\mathsf{H}_{\mathsf{mask}}$ in Game_8 remains identical even though $\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \cdot)$ for $i, j \in \mathsf{sHS}$ is no longer programmed. Then, we only need to show that the masks $\mathbf{\Delta}_i$ are identically distributed in both games. We initially fix some arbitrary $\mathsf{ctnt}_{\mathbf{w}}$ and later apply a hybrid argument to conclude.

Game_8:

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}})$

// Identical to Lines 1 to 10 in Game_3

11 : $\textbf{for } j \in \mathsf{sCS} \textbf{ do}$

12 : $\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

13 : $\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

14 : $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

15 : $\textbf{if } [\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!] \textbf{ then}$

16 : $\quad \mathbf{\Delta}_i \stackrel{\$}{\leftarrow} \mathcal{R}_q^{\ell}$

17 : $\textbf{else}$

18 : $\quad \textbf{for } j \in \mathsf{sCS}$

19 : $\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

20 : $\quad \mathbf{\Delta}_i := -\sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$

21 : $\mathbf{z}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$

22 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$

23 : $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

24 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

25 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

26 : $\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

27 : $\textbf{return } \mathsf{pm}_{3,i} := \mathbf{z}_i$

Game_9:

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}})$

1 : $\textbf{req } [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{2,i}, \cdot) \in \mathsf{st}_i]\!]$

2 : $\textbf{parse } (\mathbf{w}_j)_{j\in\mathsf{SS}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}}$

3 : $\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i$

$\quad$ with $\mathsf{pm}_{2,i} = \mathbf{w}_i$

4 : $\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(\mathsf{SS}, \mathsf{M}, j, \mathbf{w}_j)]\!]$

5 : $\mathsf{ctnt}_{\mathbf{z}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j, \mathbf{w}_j)_{j\in\mathsf{SS}}$

6 : $\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$

7 : $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$

8 : $\mathsf{ctnt}_{\mathbf{w}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j\in\mathsf{SS}}$

9 : $\textbf{if } [\![\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!] \textbf{ then}$

10 : $\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $\quad \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

12 : $\textbf{req } [\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c]\!]$

// Identical to Lines 11 to 27 in Game_8

Figure 16: The eighth and ninth games. The differences are highlighted in blue. We assume Game_9 initializes a empty list $\mathsf{Chall}[\cdot] := \bot$ at the beginning of the game.

Lemma E.2. *Let* $\mathsf{ctnt}_{\mathbf{w}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j)_{j\in\mathsf{SS}}$ *be fixed. If* $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$*, in both games, we have for* $i \in \mathsf{sHS}$ *that*

1. $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$ *if user* i *has not executed the third round with* $\mathsf{ctnt}_{\mathbf{w}}$*, else*
2. $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \sim \mathcal{U}_{\mathcal{R}_q^\ell}$ *is distributed at random, if there remains another honest signer* $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$ *before third round with* $\mathsf{ctnt}_{\mathbf{w}}$*, and*
3. $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = -\sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$*, if* i *was the last user between second and third with* $\mathsf{ctnt}_{\mathbf{w}}$ *(i.e., if* $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$*).*

Proof. The first statement holds in both games by construction. The second and third statement hold for Game_8 by construction. Let us inspect the distribution of $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ in Game_7. Observe that all values stored in $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ are computed as depicted in Fig. 15. Recall that, due to Lemma E.1, $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ for $\mathsf{ctnt}_{\mathbf{z}}$ is defined only when $\mathcal{O}_{\mathsf{Sign}_3}$ for user i or j is invoked with $\mathsf{ctnt}_{\mathbf{w}}$ that uniquely determines $\mathsf{ctnt}_{\mathbf{z}}$. If there exists some $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}}\backslash\{i\}$, then $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ are sampled at random over $\mathcal{R}_q^\ell$. Thus, $\mathbf{\Delta}_i = \sum_{j\in\mathsf{SS}\backslash\{i\}}(\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$ is distributed at random over $\mathcal{R}_q^k$. If on the other hand $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$, then all individual masks $(\mathbf{m}_{i,j}, \mathbf{m}_{j,i})_{j\in\mathsf{sHS}}$ are retrieved from $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}$ and thus, $\mathbf{\Delta}_i$ is fully determined. Because we have that $\sum_{j\in\mathsf{SS}} \mathbf{\Delta}_j = \mathbf{0}$, where $\mathbf{\Delta}_j = \sum_{\kappa\in\mathsf{SS}\backslash\{j\}}(\mathbf{m}_{\kappa,j} - \mathbf{m}_{j,\kappa})$, we have that

$$\mathbf{\Delta}_i = -\sum_{j\in\mathsf{SS}\backslash\{i\}} \mathbf{\Delta}_j$$

where $\mathbf{\Delta}_j = \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$ for $j \in \mathsf{sCS}$. Finally, observe that every time a user $j \in \mathsf{sHS}$ is removed from $\widetilde{\mathsf{sHS}}_{\mathbf{z}}$, the value $\mathbf{\Delta}_j$ is stored in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j]$. Recall that there is only one $\mathsf{ctnt}_{\mathbf{z}}$ corresponding to $\mathsf{ctnt}_{\mathbf{w}}$. Thus, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$, we have that

$$\sum_{j\in\mathsf{SS}\backslash\{i\}} \mathbf{\Delta}_j = \sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j.$$

Combining the both equations concludes. □

When we apply a hybrid argument over all $\mathsf{ctnt}_{\mathbf{w}}$ in order of occurrence to the above lemma, we have

$$\epsilon_8 = \epsilon_7.$$

Game_9: In this game, the challenger checks whether all honest users uses the same challenge c in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$. This is depicted in Fig. 16. Specifically, in $\mathcal{O}_{\mathsf{Sign}_3}$, the challenger additionally stores $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ in $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$, i.e., user i is the first user in the third round. Also, it checks if $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c$. If so, it continues the game as before. Otherwise, it aborts the game.

Let us show that Game_8 and Game_9 are identically distributed. To show this, we show the following lemma.

Lemma E.3. *Let* $\mathsf{ctnt}_{\mathbf{w}}$ *be arbitrary. In* $\mathcal{O}_{\mathsf{Sign}_3}$ *with* $\mathsf{ctnt}_{\mathbf{w}}$*, all honest users in* sHS *use the same* $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$*.*

Proof. Since M is included in $\mathsf{ctnt}_{\mathbf{w}}$ and vk is fixed, we need to show that $\mathbf{w}$ computed in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$ is identical for all honest users in sHS. Recall that $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{cmt}_j, \mathbf{w}_j)_{j\in\mathsf{SS}}$ and $\mathbf{w}$ is computed by $\left\lfloor \sum_{j\in\mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$. Thus, $\mathbf{w}$ is uniquely determined by $\mathsf{ctnt}_{\mathbf{z}}$. Due to Lemma E.1, the same $\mathsf{ctnt}_{\mathbf{z}}$ is used in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for all uses in sHS. Therefore, all users in sHS compute the same $\mathbf{w}$ in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$. This computes the proof. □

$\underline{\mathsf{Game}_{10}:}$

$\underline{\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})}$

// Identical to Lines 1 to 12 in Game_9

13 : **for** $j \in \mathsf{sCS}$ **do**

14 : $\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

15 : $\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

16 : $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

17 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

18 : $\quad \tilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$

19 : **else**

20 : $\quad$ **for** $j \in \mathsf{sCS}$

21 : $\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

22 : $\quad \tilde{\mathbf{z}}_i := c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$

23 : $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \tilde{\mathbf{z}}_i$

24 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

25 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

26 : $\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

27 : **return** $\mathsf{pm}_{3,i} := \mathbf{z}_i$

$\underline{\mathsf{Game}_{11}:}$

$\underline{\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, \mathsf{M}, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}})}$

// Identical to Lines 1 to 6 in Game_4

7 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

8 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

9 : $\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

10 : $\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$

11 : **else**

12 : $\quad c \xleftarrow{\$} \mathcal{C}$

13 : $\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

14 : $\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$

15 : $\quad \mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \mathbf{w}_i)$

16 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$

17 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \mathbf{w}_i)$

18 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

19 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{cmt}_i)\}$

20 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

21 : **return** $\mathsf{pm}_{2,i} := \mathbf{w}_i$

$\underline{\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})}$

// Identical to Lines 1 to 8 in Game_9

9 : **if** $[\![\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$ **then**

10 : $\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $\quad \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

12 : **req** $[\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \neq c]\!]$

13 : **abort if** $[\![\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top]\!]$

// Identical to Lines 13 to 27 in Game_{10}

Figure 17: The tenth and eleventh games. The differences are highlighted in blue. We assume Game_{11} initializes a empty list $\mathsf{BadCtnt}[\cdot] := \perp$ at the beginning of the game. The algorithm $\mathsf{ProgramHashChall}$ is defined in Fig. 18.

$\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \mathbf{w}_i)$:

$\quad$ **parse** $\mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$

$\quad$ **if** $[\![\forall j \in \mathsf{SS} \backslash \{i\}, \exists! \mathbf{w}_j, \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(j, \mathbf{w}_j) = \mathsf{cmt}_j]\!]$

$$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$$

$\quad\quad$ **abort if** $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \neq \perp]\!]$

$\quad\quad$ $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$

$\quad$ **else**

$\quad\quad$ $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] := \top$

Figure 18: A helper algorithm $\mathsf{ProgramHashChall}$ for programming the random oracle H_c for input $\mathbf{w}$ derived from $\mathsf{ctnt}_{\mathbf{w}}$ (and optionally $\mathbf{w}_i$) to a given output c. The algorithm $\mathsf{ProgramHashChall}$ is assumed to have a joint state with the challenger and random oracle H_c used by the unforgeability game.

By the above lemma, the game never aborts due to the added abort conditions. Thus, we have

$$\epsilon_9 = \epsilon_8.$$

Game_{10} In this game, the challenger samples $\widetilde{\mathbf{z}}_i$ directly either at random or consistently for the last user in $\mathcal{O}_{\mathsf{Sign}_3}$. This is depicted in Fig. 17. We describe the changes in more detail. In $\mathcal{O}_{\mathsf{Sign}_3}$, instead of sampling $\mathbf{\Delta}_i$, it samples $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ at random if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$ and otherwise, it sets $\widetilde{\mathbf{z}}_i = c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$. The value $\widetilde{\mathbf{z}}_i$ is stored in $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ but $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ remains $\perp$. Note that $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is no longer used through the game.

Let us show that Game_9 and Game_{10} are identically distributed. To show this, we only need to show that $\widetilde{\mathbf{z}}_i$ in $\mathcal{O}_{\mathsf{Sign}_3}$ in both games are identically distributed. Let us first show a useful lemma.

Lemma E.4. *Let $\mathsf{ctnt}_{\mathbf{w}}$ be arbitrary. If $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ holds in $\mathcal{O}_{\mathsf{Sign}_3}$, then $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for all honest users in sHS are completed. Moreover, in Game_{10}, we have in line 22 in $\mathcal{O}_{\mathsf{Sign}_3}$ that $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}} \mathbf{r}_j$ where $\mathbf{r}_j$ is a commitment randomness generated in $\mathcal{O}_{\mathsf{Sign}_2}$ for user j with $\mathsf{ctnt}_{\mathbf{w}}$.*

Proof. We first show the first statement. When $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ holds in $\mathcal{O}_{\mathsf{Sign}_3}$ for user i where $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{z}}$, $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for all honest users in $\mathsf{sHS} \backslash \{i\}$ are finished. Also, for user $j \in \mathsf{sHS}$, $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$ is executed only when $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ is completed. Thus, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ holds in $\mathcal{O}_{\mathsf{Sign}_3}$, then all honest users in sHS finished $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$.

We can obtain the second statement from the first statement and the fact that a commitment randomness $\mathbf{r}_j$ for user j is added to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ at the end of $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for user j. This completes the proof. $\square$

Next, let us consider an intermediate game of $\mathsf{Game}_{9,*}$, where instead of sampling $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ at random in $\mathcal{O}_{\mathsf{Sign}_3}$, we sample $\mathbf{\Delta}_i^* \xleftarrow{\$} \mathcal{R}_q^\ell$ and set $\mathbf{\Delta}_i := \mathbf{\Delta}_i^* - (c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i)$. This game is identically distributed to Game_9. Then, we have in $\mathcal{O}_{\mathsf{Sign}_3}$ that

$$\widetilde{\mathbf{z}}_i = c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i$$

$$= \mathbf{\Delta}_i^* \sim \mathcal{U}_{\mathcal{R}_q^\ell}$$

which is distributed as in Game_{10}. Similarly, if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ we have that

$$\begin{aligned}
\widetilde{\mathbf{z}}_i &= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \widetilde{\mathbf{\Delta}}_i \\
&= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\
&= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i - \sum_{j \in \mathsf{sHS} \backslash \{i\}} (\mathbf{\Delta}_j^* - (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j)) - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\
&= \sum_{j \in \mathsf{sHS}} (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j) - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \widetilde{\mathbf{z}}_i - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\
&= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{CS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \sum_{j \in \mathsf{sHS}} \mathbf{r}_j - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j
\end{aligned}$$

where the third equation follows from Lemma E.3, and the last equation follows from the correctness of the Shamir secret sharing. Due to Lemma E.4, we have that $\widetilde{\mathbf{z}}_i$ is identically distributed in $\mathsf{Game}_{8,*}$ and Game_9.

Combining all arguments, we conclude that

$$\epsilon_{10} = \epsilon_9.$$

Game_{11}: In this game, the challenger precomputes the challenge c for $\mathcal{O}_{\mathsf{Sign}_3}$ when the last signer passes round 2. This is depicted in Fig. 17. In more detail, in $\mathcal{O}_{\mathsf{Sign}_2}$, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, then it samples a challenge $c \xleftarrow{\$} \mathcal{C}$ and programs H_c via a helper function $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ (cf. Fig. 18). Note that how it generates $\mathbf{w}_i$ is not changed. In $\mathsf{ProgramHashChall}$, the challenger parses $\mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$, and checks if for each cmt_j for $j \in \mathsf{SS} \backslash \{i\}$, there is a (unique) value $\mathbf{w}_j$ such that $\mathsf{H}_{\mathsf{com}}(j, \mathbf{w}_j) = \mathsf{cmt}_j$. If so, it sets $\mathbf{w} = \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}}$ and sets $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$ (but aborts if this value was previously set), and finally sets $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow (\mathsf{M}, c, \mathbf{w})$. Otherwise, it sets $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \top$. In $\mathcal{O}_{\mathsf{Sign}_3}$, it aborts the game if $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$.

Since how to generate $\mathbf{w}$ and sample c is not changed, both games are identically distributed conditioned on the game not aborting. Below, we bound the abort probability in Game_{10}. The challenger aborts the if (1) $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}]$ is already defined in $\mathsf{ProgramHashChall}$ or (2) $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ in $\mathcal{O}_{\mathsf{Sign}_3}$.

We first bound the probability of event (1). Observe that $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ is invoked with $\mathbf{w} = \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$ after $\mathbf{w}_i$ for the last user in $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ is sampled and before returning it to $\mathcal{A}$. Since $\mathbf{w}_i$ is generated honestly, $\mathbf{w}$ has min-entropy $n-1$ with overwhelming probability due to Lemma 2.8. Thus, the probability that $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \neq \perp$ is at most $(Q_{\mathsf{H}_c} + Q_{\mathsf{S}})/2^{n-1}$ with overwhelming probability. Since $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \mathbf{w}_i)$ is invoked when the last user with $\mathsf{ctnt}_{\mathbf{w}}$ passes round 2, the probability of event (1) is at most $Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})/2^{n-1}$.

It remains to bound the probability of event (2). Recall that $\mathsf{ProgramHashChall}$ is invoked when $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for the last user is invoked. If $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ holds in $\mathcal{O}_{\mathsf{Sign}_3}$, then $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for all honest users in sHS are completed. Then, cmt_j for all honest users in sHS is correctly defined via $\mathsf{ProgramHashCom}$ in $\mathcal{O}_{\mathsf{Sign}_2}$. Thus, we have $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ in $\mathcal{O}_{\mathsf{Sign}_3}$ only if there is at least one cmt_j for $j \in \mathsf{sCS}$ does not have a $\mathsf{H}_{\mathsf{com}}$ preimage of the form $(j, \mathbf{w}_j)$ when $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ is invoked (where cmt_j is determined by $\mathsf{ctnt}_{\mathbf{w}}$), but the $\mathcal{A}$ provides a valid preimage of cmt_j in $\mathcal{O}_{\mathsf{Sign}_3}$. Since the image cmt of $\mathsf{H}_{\mathsf{com}}$ is sampled uniformly at random from $\{0,1\}^{2\lambda}$ each $\mathsf{H}_{\mathsf{com}}$ query, the probability that $\mathcal{A}$ finds a valid preimage for cmt_j is at most $1/2^{2\lambda}$ per query. Thus, the probability of event (2) is at most $Q_{\mathsf{H}_{\mathsf{com}}}/2^{2\lambda}$. In conclusion, we have

$$|\epsilon_{11} - \epsilon_{10}| \leqslant \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_c} + Q_{\mathsf{S}})}{2^{n-1}} + \frac{Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda).$$

Game_{12}:

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, \mathsf{M}, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}})$

// Identical to Lines 1 to 6 in Game_4

$\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

if $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

  $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

  $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i$

  $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$

else

  $c \xleftarrow{\$} \mathcal{C}$

  $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

  $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r};\ \mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$

  $\mathbf{w}_i = \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} + \mathbf{z}'$

  $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{z}$

  $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \mathbf{w}_i)$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \mathbf{w}_i)$

$\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{cmt}_i)\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

return $\mathsf{pm}_{2,i} := \mathbf{w}_i$

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}})$

// Identical to Lines 1 to 13 in Game_{11}

for $j \in \mathsf{sCS}$ **do**

  $\mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

  $\mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

$\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

if $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

  $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}^{\ell}_q$

else

  abort if $[\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \neq (\mathsf{M}, c, \mathbf{w})]\!]$

  for $j \in \mathsf{sCS}$

    $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

  $\widetilde{\mathbf{z}}_i := \mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] - c \sum_{j\in\mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j$
  $+ \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$
  $- \sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$

$\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j\in\mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

$\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

return $\mathsf{pm}_{3,i} := \mathbf{z}_i$

Figure 19: The twelfth game. The differences are highlighted in blue. We assume that this game initializes a empty list $\mathsf{SimResp}[\cdot] := \perp$ at the beginning of the game.

Game_{12}: In this game, the challenger simulates the commitment for the last user in the second round. This is depicted in Fig. 19. Specifically, in $\mathcal{O}_{\mathsf{Sign}_2}$, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}$, where $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{w}}$, the challenger generates $\mathbf{w}_i$ honestly and add the commitment randomness $\mathbf{r}_i$ to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$. Otherwise, it samples c as before, and then samples $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$, sets $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}, \mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$ and simulates $\mathbf{w} = \mathbf{A} \cdot \mathbf{z} - c \cdot \hat{\mathbf{t}} + \mathbf{z}'$. The response $\mathbf{z}$ is stored in $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{z}$. In $\mathcal{O}_{\mathsf{Sign}_3}$, it generates $\widetilde{\mathbf{z}}_i$ using $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}]$ instead of $\mathbf{s}$. Note that in Game_{11}, $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ contains the sum of the commitment randomness $\mathbf{r}_j$ of all honest user in sHS, i.e., $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}} \mathbf{r}_j$. On the other hands, in Game_{12}, $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ contains the sum pf $\mathbf{r}_j$ for all honest users except for the last user in the second round, i.e., $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathbf{r}_j$.

Below, we show the distribution of the view of $\mathcal{A}$ remains identical. In Game_{11}, we have

$$\begin{aligned}
\widetilde{\mathbf{z}}_i &= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j \\
&= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \sum_{j \in \mathsf{sHS}} \mathbf{r}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j \\
&= c \cdot \mathbf{s} + \mathbf{r}_i - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathbf{r}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j
\end{aligned}$$

Due to the abort conditions in Game_{11} and the first statement in Lemma E.4, c in the computation of $\mathbf{z}_i$ for the last user in $\mathcal{O}_{\mathsf{Sign}_3}$ with $\mathsf{ctnt}_{\mathbf{w}}$ of Game_{11} is the same to c that is defined via $\mathsf{ProgramHashChall}$ when $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for the last user in the second round is invoked. Thus, c in $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] = c \cdot \mathbf{s} + \mathbf{r}_i$ used to compute $\mathbf{z}_i$ for the last user in Game_{12} is identical to that in the computation of $\mathbf{z}_i$ in Game_{11}. Also, in Game_{12}, $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Sign}_3}$ for the last user in the third round is equal to $\sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathbf{r}_j$ due to the first statement in Lemma E.4. Combining the above facts, we conclude that $\mathbf{z}_i$ is identically distributed in both games. It remains to argue that $\mathbf{w}_i$ generated in $\mathcal{O}_{\mathsf{Sign}_2}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for the last user in the second round is identically distributed. This follows since in Game_{12}, we have

$$\begin{aligned}
\mathbf{w}_i &= \mathbf{A} \cdot \mathbf{z} - c \cdot \hat{\mathbf{t}} + \mathbf{z}' \\
&= \mathbf{A} \cdot (c \cdot \mathbf{s} + \mathbf{r}_i) - c \cdot \hat{\mathbf{t}} + (c \cdot \mathbf{e} + \mathbf{e}'_i) \\
&= c(\mathbf{A} \cdot \mathbf{s} + \mathbf{e}) + \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i - c \cdot \hat{\mathbf{t}} \\
&= \mathbf{A} \cdot \mathbf{r}_i + \mathbf{e}'_i
\end{aligned}$$

where $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$. Hence, we have

$$\epsilon_{12} = \epsilon_{11}.$$

Game_{13}: In this game, the challenger samples $\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}_q^k$ at random. Also, it samples $\mathbf{s}_i$ only for corrupted users. This is depicted in Fig. 20. Concretely, the challenger samples $\hat{\mathbf{t}}$ uniformly at random over $\mathcal{R}_q^k$ instead of via $(\mathbf{s}, \mathbf{e})$. Also, it samples $\mathbf{s}_i$ uniformly at random from $\mathcal{R}_q^{\ell}$ for $i \in \mathsf{CS}$.

Due to Lemma E.6, which will be proven below, we can construct an $\mathsf{Hint\text{-}MLWE}$ adversary $\mathcal{B}$ solving the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$ problem such that

$$|\epsilon_{13} - \epsilon_{12}| \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^\lambda)$$

with $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Remark E.5. If we consider the weaker notion of security where the forgery's message M^* cannot be queried to any signing oracle as in [dPKM+24], then we can show that there exists a $\mathsf{SelfTargetMSIS}$ adversary $\mathcal{B}''$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ problem that internally runs an adversary $\mathcal{A}$ against Game_{13} such that $\epsilon_{13} \leqslant \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}(1^\lambda)$, where $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$.

Game_{13}:

$\mathsf{Q_M}[\cdot] := \varnothing, \mathsf{Com} := \varnothing$

$\mathsf{Q}_{\mathsf{H}_c}[\cdot], \mathsf{Q}_{\mathsf{H_{com}}}[\cdot], \mathsf{Q}_{\mathsf{H_{mask}}}[\cdot], \mathsf{UnOpenedHS}[\cdot], \mathsf{SumComRnd}[\cdot] := \perp$

$\mathsf{UnSignedHS}[\cdot], \mathsf{Mask}_{\mathbf{z}}[\cdot], \mathsf{MaskedResp}[\cdot], \mathsf{Chall}[\cdot], \mathsf{BadCtnt}[\cdot], \mathsf{SimResp}[\cdot] := \perp$

$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times \ell}$

$(\mathsf{CS}, \mathsf{st}_{\mathcal{A}}) \xleftarrow{\$} \mathcal{A}^{\mathsf{H}_c, \mathsf{H}_\beta, \mathsf{H_{mask}}}(\mathbf{A}, N, T)$

$\textbf{req } [\![\mathsf{CS} \subseteq [N]]\!] \wedge [\![|\mathsf{CS}| \leqslant T - 1]\!]$

$\mathsf{HS} := [N] \backslash \mathsf{CS}$

$\textbf{for } i \in \mathsf{HS} \textbf{ do } \mathsf{st}_i := \varnothing$

$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$

$\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}_q^k$

$\mathbf{t} := \left\lfloor \hat{\mathbf{t}} \right\rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}_{q_{\nu_{\mathbf{t}}}}^k$

$\textbf{for } i \in [N] \textbf{ do}$

$\quad \textbf{for } j \in [N] \textbf{ do}$

$\quad\quad \mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$

$\quad\quad \mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$

$(\vec{\mathsf{seed}}_i)_{i \in [N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in [N]} \right)_{i \in [N]}$

$\textbf{for } i \in \mathsf{CS} \textbf{ do}$

$\quad \mathbf{s}_i \xleftarrow{\$} \mathcal{R}_q^\ell$

$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$

$(\mathsf{sk}_i)_{i \in \mathsf{CS}} := \left(\mathbf{s}_i, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$

$(\mathsf{sk}_i)_{i \in \mathsf{HS}} := \left(\perp, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$

$\mathsf{oracles} := (\mathcal{O}_{\mathsf{Sign}_1}, \mathcal{O}_{\mathsf{Sign}_2}, \mathcal{O}_{\mathsf{Sign}_3}, \mathsf{H}_c, \mathsf{H_{com}}, \mathsf{H_{mask}})$

$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{CS}}, \mathsf{st}_{\mathcal{A}})$

$\textbf{req } [\![\left|\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}\right| \leqslant T - 1]\!]$

$\textbf{return } \mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

Figure 20: The thirteenth game. The differences are highlighted in blue.

Proof. This proof is identical to the proof by del Pino et al. [dPKM+24, Lemma 7.4] as our scheme has the same verification algorithm as theirs and the final step merely consists of extracting a solution from the forgery. In their proof, it is crucial that M^* is never queried for every honest user in $\mathcal{O}_{\mathsf{Sign}_2}$ because in that case, the output of H_c is programmed with a random challenge. Then, even if M^* is never queried in $\mathcal{O}_{\mathsf{Sign}_3}$, this H_c query does not help the adversary $\mathcal{B}'$ in finding a $\mathsf{SelfTargetMSIS}$ solution. $\square$

Game_{14}:

// Identical to Lines 1 to 22 in Game_{13}

23 : $(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk}, (\mathsf{sk}_i)_{i \in \mathsf{CS}}, \mathsf{st}_{\mathcal{A}})$

24 : **parse** $(c^*, \mathbf{z}^*, \mathbf{h}^*) \leftarrow \mathsf{sig}^*$

25 : let $q^*_{\mathsf{H}_c}$ be the value of $\mathsf{ctr}_{\mathsf{H}_c}$ when $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \lfloor \mathbf{Az} - 2^{\nu_\mathbf{t}} \cdot c \cdot \mathbf{t} \rceil_{\nu_\mathbf{w}} + \mathbf{h}]$ was set

26 : **abort if** $[\![q^*_{\mathsf{H}_c} \neq q_{\mathsf{H}_c}]\!]$

27 : **req** $[\![|\mathsf{Q}_\mathsf{M}[\mathsf{M}^*] \cup \mathsf{CS}| \leqslant T - 1]\!]$

28 : **return** $\mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathsf{ProgramHashChall}(\mathsf{ctnt}_\mathbf{w}, c, \mathbf{w}_i)$:

1 : **parse** $\mathsf{SS} \| \mathsf{M} \| (\mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{ctnt}_\mathbf{w}$

2 : **if** $[\![\forall j \in \mathsf{SS} \backslash \{i\}, \exists! \mathbf{w}_j, \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(j, \mathbf{w}_j) = \mathsf{cmt}_j]\!]$

3 : $\quad \mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \mathbf{w}_j \right\rceil_{\nu_\mathbf{w}} \in \mathcal{R}^k_{q_{\nu_\mathbf{w}}}$

4 : $\quad$ **abort if** $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \neq \bot]\!]$

5 : $\quad \mathsf{ctr}_{\mathsf{H}_c} \leftarrow \mathsf{ctr}_{\mathsf{H}_c} + 1$

6 : $\quad$ **if** $[\![\mathsf{ctr}_{\mathsf{H}_c} = q_{\mathsf{H}_c}]\!]$

$\quad\quad$ // Sample c' after $\mathbf{w}$ is defined

7 : $\quad\quad c' \xleftarrow{\$} \mathcal{C}$

8 : $\quad\quad \mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c'$

9 : $\quad\quad \mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] \leftarrow \top$

10 : $\quad$ **else**

11 : $\quad\quad \mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$

12 : **else**

13 : $\quad \mathsf{BadCtnt}[\mathsf{ctnt}_\mathbf{w}] := \top$

$\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$

1 : **if** $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] = \bot]\!]$ **then**

2 : $\quad c \xleftarrow{\$} \mathcal{C}$

3 : $\quad \mathsf{ctr}_{\mathsf{H}_c} \leftarrow \mathsf{ctr}_{\mathsf{H}_c} + 1$

4 : $\quad \mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$

5 : **return** $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}]$

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 13 in Game_{11}

14 : **for** $j \in \mathsf{sCS}$ **do**

15 : $\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_\mathbf{z})$

16 : $\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_\mathbf{z})$

17 : $\widetilde{\mathsf{sHS}}_\mathbf{z} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}]$

18 : **if** $[\![\widetilde{\mathsf{sHS}}_\mathbf{z} \neq \{i\}]\!]$ **then**

19 : $\quad \tilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}^\ell_q$

20 : **else**

21 : $\quad$ **abort if** $[\![\mathsf{Chall}[\mathsf{ctnt}_\mathbf{w}] \neq (\mathsf{M}, c, \mathbf{w})]\!]$

22 : $\quad$ **abort if** $[\![\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top]\!]$

23 : $\quad$ **for** $j \in \mathsf{sCS}$

24 : $\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_\mathbf{z})$

25 : $\quad \tilde{\mathbf{z}}_i := \mathsf{SimResp}[\mathsf{ctnt}_\mathbf{w}] - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j$
$\quad\quad + \mathsf{SumComRnd}[\mathsf{ctnt}_\mathbf{w}]$
$\quad\quad - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_\mathbf{w}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$

26 : $\mathsf{MaskedResp}[\mathsf{ctnt}_\mathbf{w}, i] \leftarrow \tilde{\mathbf{z}}_i$

27 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}] \backslash \{i\}$

28 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{cmt}_j)_{j \in \mathsf{SS}}, \mathbf{w}_i, \mathbf{r}_i)\}$

29 : $\mathsf{Q}_\mathsf{M}[\mathsf{M}] \leftarrow \mathsf{Q}_\mathsf{M}[\mathsf{M}] \cup \{i\}$

30 : **return** $\mathsf{pm}_{3,i} := \mathbf{z}_i$

Figure 21: The fourteenth game. The differences are highlighted in blue. We assume that this game initializes an empty list $\mathsf{BadGuess}[\cdot] := \bot$, a counter $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow 0$, and samples a guess $q_{\mathsf{H}_c} \xleftarrow{\$} [Q_{\mathsf{H}_c}]$ at the beginning of the game.

Game_{14}**:** In this game, the challenger guesses the H_c query associated to the adversary's forgery. For this query, the challenger never programs H_c via $\mathsf{ProgramHashChall}$ [12]. It also aborts if $\mathcal{O}_{\mathsf{Sign}_3}$ is invoked for every honest user but it did not program H_c in $\mathcal{O}_{\mathsf{Sign}_2}$ due to the aforementioned change. This is depicted in Fig. 21. In more detail, the challenger initially sets up a counter $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow 0$ and samples $q_{\mathsf{H}_c} \xleftarrow{\$} [Q_{\mathsf{H}_c}]$. Each time a table entry in $\mathsf{Q}_{\mathsf{H}_c}$ is changed, the challenger increases the counter $\mathsf{ctr}_{\mathsf{H}_c}$. This happens either in a fresh H_c query or when $\mathsf{ProgramHashChall}$ is invoked in $\mathcal{O}_{\mathsf{Sign}_2}$ and H_c is programmed. In the latter case, the challenger checks if $\mathsf{ctr}_{\mathsf{H}_c} = q_{\mathsf{H}_c}$ and sets $\mathsf{BadGuess}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ if so. It also aborts in $\mathcal{O}_{\mathsf{Sign}_3}$ if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ and $\mathsf{BadGuess}[\mathsf{ctnt}_{\mathbf{w}}] = \top$. After $\mathcal{A}$'s forgery $(\mathsf{sig}^*, \mathsf{M}^*)$ is output, the challenger retrieves the value $q^*_{\mathsf{H}_c}$ of $\mathsf{ctr}_{\mathsf{H}_c}$ when the query H_c associated to the forgery was made [13]. This happens either in $\mathsf{ProgramHashChall}$ or H_c.

Let us analyze the advantage of $\mathcal{A}$ in Game_{14}. Observe that the value $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}]$ is only used if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$ in $\mathcal{O}_{\mathsf{Sign}_3}$. If $\mathsf{BadGuess}[\mathsf{ctnt}_{\mathbf{w}}] = \bot$, then H_c was programmed via $\mathsf{ProgramHashChall}$ in $\mathcal{O}_{\mathsf{Sign}_2}$, so the challenge in $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] = c \cdot \mathbf{s} + \mathbf{r}$ is identical to the challenge in $\mathcal{O}_{\mathsf{Sign}_3}$. Thus, the view of $\mathcal{A}$ is identically distributed conditioned on no abort in Game_{13} and Game_{14}. If $\mathcal{A}$ is successful, then the challenger does not abort in $\mathcal{O}_{\mathsf{Sign}_3}$ if $q_{\mathsf{H}_c} = q^*_{\mathsf{H}_c}$ because the message M^* is not queried to $\mathcal{O}_{\mathsf{Sign}_3}$ for the last honest user. Note that the value q_{H_c} is hidden from $\mathcal{A}$. Thus, we have that

$$\begin{aligned}\epsilon_{14} &\geqslant \Pr[q_{\mathsf{H}_c} = q^*_{\mathsf{H}_c}] \cdot \epsilon_{13} \\ &\geqslant 1/Q_{\mathsf{H}_c} \cdot \epsilon_{13}.\end{aligned}$$

Due to Lemma E.7, which will be proven below, there exists an $\mathsf{SelfTargetMSIS}$ adversary $\mathcal{B}'$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ problem that internally runs an adversary $\mathcal{A}$ against Game_{14} such that

$$\epsilon_{14} \leqslant \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(1^\lambda).$$

Moreover, we have $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$. Collecting all bounds, we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ts\text{-}sel\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{sel}}_{\mathsf{3\text{-}rnd}},\mathcal{A}} \leqslant Q_{\mathsf{H}_c} \cdot \mathsf{Adv}^{\mathsf{SelfTargetMSIS}}_{\mathcal{B}'}(1^\lambda) + \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(1^\lambda) + \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{H}_c} + 2Q_{\mathsf{S}})}{2^{n-1}} \\ + \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda} + \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_{\mathsf{S}})^2 + Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda),\end{aligned}$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ and $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$.

To complete the proof, it remains to show Lemmata E.6 and E.7.

Lemma E.6. *There exists an adversary $\mathcal{B}$ against the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$ problem such that*

$$|\epsilon_{13} - \epsilon_{12}| \leqslant \mathsf{Adv}^{\mathsf{Hint\text{-}MLWE}}_{\mathcal{B}}(1^\lambda)$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Proof. Let $\mathcal{A}$ be an adversary that distinguishes Game_{13} and Game_{12}. To show this lemma, we construct an adversary $\mathcal{B}$ against the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_{\mathsf{S}},\sigma_{\mathbf{t}},\sigma_{\mathbf{w}},\mathcal{C}}$ problem that internally runs $\mathcal{A}$. $\mathcal{B}$ is given the $\mathsf{Hint\text{-}MLWE}$ problem instance $(\mathbf{A}, \mathbf{b}, (c_i, \mathbf{z}_i, \mathbf{z}'_i)_{i \in [Q_{\mathsf{S}}]})$ as input.

$\mathcal{B}$ behaves as the challenger in Game_{13} except for the initial phase and $\mathcal{O}_{\mathsf{Sign}_2}$. In the initial phase, it uses $\mathbf{A}$ given as input, instead of choosing a fresh $\mathbf{A}$ sampled from $\mathcal{R}_q^k$, and embeds $\lfloor \mathbf{b} \rceil_{\nu_{\mathbf{t}}}$ into $\mathbf{t}$. Also, when it generates the jth commitment for the last user in the second round in $\mathcal{O}_{\mathsf{Sign}_2}$, it uses $(c_j, \mathbf{z}_j, \mathbf{z}'_j)$, instead of sampling $c \xleftarrow{\$} \mathcal{C}$, $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^\ell_{\mathbf{w}} \times \mathcal{D}^k_{\mathbf{w}}$, and setting $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}_i$ and $\mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'_i$. Otherwise, it behaves as the challenger in Game_{13}.

We show that $\mathcal{B}$ perfectly simulates the challenger in Game_{12} (resp. Game_{13}) when $\mathbf{b}$ is a valid MLWE sample (resp. $\mathbf{b}$ is uniformly sampled from $\mathcal{R}_q^k$). When $\mathbf{b}$ is a valid MLWE sample, $\mathbf{t}$ is identically distributed

[12] More precisely, the challenger *always* samples the output of H_c *after* the input is defined for the guessed query.

[13] The adversary's forgery $(\mathsf{sig}^*, \mathsf{M}^*)$ is associated to some H_c query since we assume that Q_{H_c} also counts the challengers H_c queries without loss of generality.

to $\mathbf{t}$ in Game_{12}. Also, the secret shares $\mathbf{s}_i$ of each corrupted user $i \in \mathsf{CS}$ is uniformly distributed over $\mathcal{R}_q^{\ell}$. Moreover, since the leakage $(\mathbf{z}_i, \mathbf{z}'_i)$ satisfies

$$\mathbf{z}_i = c \cdot \mathbf{s} + \mathbf{r}_i, \text{ and } \mathbf{z}'_i = c \cdot \mathbf{e} + \mathbf{e}'_i \tag{20}$$

where $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$ and $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$, $\mathcal{B}$ perfectly simulates the singing oracle in Game_{12}.

When $\mathbf{b}$ is uniformly sampled from $\mathcal{R}_q^k$, the distribution of $\mathbf{t}$ is identical to that in Game_{13}. Moreover, $\mathcal{B}$ perfectly simulates the singing oracle in Game_{13} due to Eq. (20), where $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$. Note that the leakage no longer depends on $\mathbf{t}$. Combining all arguments, $\mathcal{B}$ perfectly simulates Game_{12} and Game_{13} when $\mathbf{b}$ is a valid MLWE sample and generated by $\mathbf{b} \xleftarrow{\$} \mathcal{R}_q^k$, respectively. Therefore, we have

$$|\epsilon_{13} - \epsilon_{12}| \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^{\lambda}).$$

Finally, it is clear $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ from the construction of $\mathcal{B}$. This completes the proof. ☐

Lemma E.7. *There exists a* $\mathsf{SelfTargetMSIS}$ *adversary* $\mathcal{B}'$ *solving the* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ *problem that internally runs an adversary* $\mathcal{A}$ *against* Game_{14} *such that*

$$\epsilon_{14} \leqslant \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}(1^{\lambda})$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Proof. Due to the added abort condition in Game_{14}, the challenge c associated to the adversary's forgery is sampled at random after $\mathbf{w}$ is fixed (either in H_c or in $\mathsf{ProgramHashChall}$). We can thus construct an adversary $\mathcal{B}'$ that answers $\mathcal{A}$'s queries to H_c via the oracle H provided by the $\mathsf{SelfTargetMSIS}$ reduction. Using the above observation, we can show the statement as in del Pino et al. [dPKM+24, Lemma 7.4]. We detail the modifications below.

First, $\mathcal{B}'$ is given $\mathbf{M} \in \mathcal{R}_q^{k \times (\ell+1)}$ by the $\mathsf{SelfTargetMSIS}$ challenger. Also, it is provided an oracle H. Then, $\mathcal{B}'$ sets $-\hat{\mathbf{t}}$ as the first column and $\mathbf{A}$ to be the remaining ℓ columns of $\mathbf{M}$. These values define the parameters $\mathsf{tspar} := (\mathbf{A}, N, T)$ which $\mathcal{B}'$ forwards to $\mathcal{A}$. After $\mathcal{A}$ outputs CS, adversary $\mathcal{B}'$ simulates the challenger in Game_{14} as before using verification key $\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$, where $\mathbf{t} := \lfloor \hat{\mathbf{t}} \rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}_{q_{\nu_{\mathbf{t}}}}^k$ with two modifications:

1. For every H_c query $(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ made by $\mathcal{A}$, $\mathcal{B}'$ outputs $c := \mathsf{H}(\mathsf{vk}, \mathsf{M}, 2^{\nu_{\mathbf{w}}} \cdot \overline{\mathbf{w}} \mod q)$, where recall that $\overline{\mathbf{m}}$ is the unique lift of values $\mathbf{m}$ in $\mathbb{Z}_{q_{\mathbf{w}}}^k$ to $\{0, 1, \cdots, q_{\mathbf{w}} - 1\}^k$.
2. If $\mathsf{ctr}_{\mathsf{H}_c} = q_{\mathsf{H}_c}$ in $\mathsf{ProgramHashChall}$, then $\mathcal{B}'$ uses $c' := \mathsf{H}(\mathsf{vk}, \mathsf{M}, 2^{\nu_{\mathbf{t}}} \cdot \overline{\mathbf{w}} \mod q)$ instead of a randomly sampled challenge.

We can argue as in [dPKM+24] that the view of $\mathcal{A}$ remains identical[14]. At the end of the game, $\mathcal{A}$ outputs a forgery $(c^*, \mathbf{z}^*, \mathbf{h}^*)$ such that $q_{\mathsf{H}_c} = q^*_{\mathsf{H}_c}$, where $q^*_{\mathsf{H}_c}$ is the value of $\mathsf{ctr}_{\mathsf{H}_c}$ when the H_c query corresponding to c^* was made. Due to the way $\mathcal{B}'$ simulates the oracle, we have that

$$c^* = \mathsf{H}\left(\mathsf{vk}, \mathsf{M}, 2^{\nu_{\mathbf{w}}} \cdot \left(\overline{\lfloor \mathbf{A} \cdot \mathbf{z}^* - 2^{\nu_{\mathbf{t}}} \cdot c^* \cdot \overline{\mathbf{t}} \rceil_{\nu_{\mathbf{w}}} + \mathbf{h}} \mod q\right)\right)$$

Here, it is important to note that this holds because $\mathcal{B}'$ simulates the forgery's H_c query via H by design. The rest of the proof is identical to the proof by del Pino et al. [dPKM+24, Lemma 7.4] as our scheme has the same verification algorithm as theirs and the final step merely consists of extracting a solution from the forgery. ☐

This completes the proof. ☐

[14]Roughly, this is because the mapping $\mathbf{w}$ to $2^{\nu_{\mathbf{t}}} \cdot \overline{\mathbf{w}} \mod q$ is injective.

E.2 Formal Security Proof of $\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}}$

We provide the full proof of Theorem 6.1.

Proof. Let $\mathcal{A}$ be an adversary against the adaptive security game. We consider a sequence of games where the first hybrid is the original game and the last is a game that can be reduced to the MSIS problem. We relate the advantage of $\mathcal{A}$ for each adjacent games, where ϵ_i denotes the advantage of $\mathcal{A}$ in Game_i.

Game_1**:** This is the real unforgeability game. Formally, this is depicted in Fig. 22. By definition, we have

$$\epsilon_1 = \mathsf{Adv}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T).$$

Game_2**:** In this game, the challenger adds an abort condition in $\mathcal{O}_{\mathsf{Sign}_1}$. This is depicted in Fig. 23. Specifically, the challenger initializes a set $\mathsf{Strings}$ to $\varnothing$ at the beginning of the game. In $\mathcal{O}_{\mathsf{Sign}_1}$, it aborts if str_i is included in $\mathsf{Strings}$. Otherwise, it adds str_i to $\mathsf{Strings}$ and continues.

Now we bound the probability that the challenger aborts the game. This is equalt to the probability that the same string str is generated twice. Since str_i is chosen uniformly at random from $\{0,1\}^{2\lambda}$ and at most Q_S strings are generated in total, a standard birthday bound argument yields

$$|\epsilon_2 - \epsilon_1| \leqslant \frac{Q_\mathsf{S}^2}{2^{2\lambda}}.$$

Game_3**:** This game is given in Figure 24. Roughly, the challenger postpones generating $\widetilde{\mathbf{w}}_i$ until $\mathcal{O}_{\mathsf{Sign}_4}$. For this, it outputs a random hash commitment $\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$ in $\mathcal{O}_{\mathsf{Sign}_2}$ and removes the commitment-related values $(\widetilde{\mathbf{w}}_i, \mathbf{r}_i)$ from the state st_i. The challenger behaves as in the previous game in $\mathcal{O}_{\mathsf{Sign}_3}$ except that the state st_i is parsed without $(\widetilde{\mathbf{w}}_i, \mathbf{r}_i)$. In $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger computes $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ and $\widetilde{\mathbf{\Delta}}_i$ as previously in $\mathcal{O}_{\mathsf{Sign}_2}$, and programs $\mathsf{H}_{\mathsf{com}}$ via $\mathsf{ProgramHashCom}$ (cf. Figure 25) such that we have $\mathsf{H}_{\mathsf{com}}(i, \widetilde{\mathbf{w}}_i) = \mathsf{cmt}_i$, where $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$. Also, the values $(\widetilde{\mathbf{w}}_i, \mathbf{r}_i)$ are reintroduced into the state st_i at the end of $\mathcal{O}_{\mathsf{Sign}_4}$. Note that the challenger aborts in case $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}_i] \neq \perp$ in $\mathsf{ProgramHashCom}$. Since the states st_i between rounds 2 and 4 are now inconsistent with the real game, the challenger also sanitizes the states in $\mathcal{O}_{\mathsf{Corrupt}}$. In more detail, it iterates over all states of signer i between round 2 and round 4, computes $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ and $(\widetilde{\mathbf{\Delta}}_i, \widetilde{\mathbf{w}}_i)$ as above, then programs $\mathsf{H}_{\mathsf{com}}$ via $\mathsf{ProgramHashCom}$, and finally reintroduces $(\widetilde{\mathbf{w}}_i, \mathbf{r}_i)$ into the state st_i.

Because cmt_i is first opened in $\mathcal{O}_{\mathsf{Sign}_4}$ in both Game_3 and Game_4, the view of the adversary $\mathcal{A}$ remains identical except if the challenger aborts, *i.e.*, if $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}_i] \neq \perp$, where $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$. Since the commitment $\mathbf{w}_i$ is generated independently of $\widetilde{\mathbf{\Delta}}_i$, the masked commitment $\widetilde{\mathbf{w}}_i$ has at least the same min-entropy as $\mathbf{w}_i$. Thus, the probability that $\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}_i] \neq \perp$ in either $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$ is at most $(Q_{\mathsf{H}_{\mathsf{com}}} + Q_\mathsf{S})/2^{n-1}$ with overwhelming probability due to Lemma 2.8. Note for each invocation of $\mathsf{ProgramHashCom}$ in $\mathcal{O}_{\mathsf{Corrupt}}$, there is a corresponding signing query with a state between round 2 and round 4. Since $\mathcal{A}$ makes at most Q_S signing queries, we have

$$|\epsilon_3 - \epsilon_2| \leqslant \frac{Q_\mathsf{S}(Q_{\mathsf{H}_{\mathsf{com}}} + Q_\mathsf{S})}{2^{n-1}} + \mathsf{negl}(\lambda).$$

Game_4**:** In this game, the challenger aborts in case there is a collision in $\mathsf{H}_{\mathsf{com}}$. This is depicted in Fig. 26. Specifically, the challenger initially prepares an empty set $\mathsf{Cmt} := \varnothing$. In $\mathcal{O}_{\mathsf{Sign}_2}$ and $\mathsf{H}_{\mathsf{com}}$, it checks whether the sampled commitment cmt is already in Cmt, *i.e.*, was sampled at an earlier point in the game. If so, it aborts the game. Otherwise, it adds cmt to Cmt and continues as before. Since cmt is sampled uniformly at random from $\{0,1\}^{2\lambda}$, we have

$$|\epsilon_4 - \epsilon_3| \leqslant \frac{(Q_{\mathsf{H}_{\mathsf{com}}} + Q_\mathsf{S})^2}{2^{2\lambda}}.$$

$\mathsf{Game}_1 := \mathsf{Game}^{\mathsf{ts\text{-}adp\text{-}uf}}_{\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{5\text{-}rnd}},\mathcal{A}}(1^\lambda, N, T)$

$\mathsf{Q_M}[\cdot] := \varnothing, \mathsf{Q_{H_c}}[\cdot] := \perp, \mathsf{Q_{H_{com}}}[\cdot] := \perp$
$\mathsf{Q_{H_{mask}}}[\cdot] := \perp, \mathsf{Q_{H_{\widehat{mask}}}}[\cdot] := \perp$
$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times \ell}$
$\mathsf{HS} := [N]$
$\textbf{for } i \in \mathsf{HS} \textbf{ do } \mathsf{st}_i := \varnothing$
$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^\ell \times \mathcal{D}_{\mathbf{t}}^k$
$\hat{\mathbf{t}} := \mathbf{A}\mathbf{s} + \mathbf{e}; \mathbf{t} := \lfloor \hat{\mathbf{t}} \rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{t}}}}$
$\textbf{for } i \in [N] \textbf{ do}$
$\quad (\mathsf{vk}_{\mathsf{S},i}, \mathsf{vk}_{\mathsf{S},i}) \xleftarrow{\$} \mathsf{KeyGen_S}(1^\lambda)$
$\quad \textbf{for } j \in [N] \textbf{ do}$
$\quad\quad \mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^\lambda$
$\quad\quad \mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$
$(\vec{\mathsf{seed}}_i)_{i \in [N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in [N]}\right)_{i \in [N]}$
$\vec{P} \xleftarrow{\$} \mathcal{R}_q^\ell[X] \text{ with } \deg(\vec{P}) = T-1, \vec{P}(0) = \mathbf{s}$
$(\mathbf{s}_i)_{i \in [N]} := (\vec{P}(i))_{i \in [N]}$
$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$
$(\mathsf{sk}_i)_{i \in [N]} := \left(\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$
$\mathsf{oracles} := ((\mathcal{O}_{\mathsf{Sign}_i})_{i \in [5]}, \mathcal{O}_{\mathsf{Corrupt}}, \mathsf{H}_c, \mathsf{H_{com}}, \mathsf{H_{mask}})$
$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk})$
$\textbf{req } [\![|\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}| \leqslant T-1]\!]$
$\textbf{return } \mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathcal{O}_{\mathsf{Sign}_1}(i)$

$\textbf{req } [\![i \in \mathsf{HS}]\!]$
$\mathsf{str}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{\mathsf{str}_i\}$
$\textbf{return } \mathsf{pm}_{1,i} := \mathsf{str}_i$

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}})$

$\textbf{req } [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{HS}]\!]$
$\textbf{pick } \mathsf{str}_i \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{1,i} = \mathsf{str}_i$
$\textbf{parse } (\mathsf{str}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$
$\tilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$
$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^\ell \times \mathcal{D}_{\mathbf{w}}^k$
$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$
$\tilde{\mathbf{w}}_i := \mathbf{w}_i + \tilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$
$\mathsf{cmt}_i := \mathsf{H_{com}}(i, \tilde{\mathbf{w}}_i)$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{\mathsf{str}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } \mathsf{pm}_{2,i} := \mathsf{cmt}_i$

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \cdot, \mathsf{pm}_{2,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{2,i} = \mathsf{cmt}_i$
$\textbf{parse } (\mathsf{cmt}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\mathsf{M_S} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign_S}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M_S})$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } \mathsf{pm}_{3,i} := \sigma_{\mathsf{S},i}$

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}, \cdot, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$
$\textbf{parse } (\sigma_{\mathsf{S},j})_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\mathsf{M_S} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$
$\textbf{req } [\![\forall j \in \mathsf{SS}\setminus\{i\}, \mathsf{Verify_S}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M_S}) = \top]\!]$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{return } \mathsf{pm}_{4,i} := \tilde{\mathbf{w}}_i$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}, \cdot) \in \mathsf{st}_i]\!]$
$\textbf{parse } (\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$
$\textbf{parse } (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}\setminus\{i\}}$
$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{4,i} = \tilde{\mathbf{w}}_i$
$\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H_{com}}(j, \tilde{\mathbf{w}}_j)]\!]$
$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$
$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \tilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$
$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) \quad /\!/ \ c \in \mathcal{C}$
$\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^\ell$
$\tilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^\ell$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\mathsf{Q_M}[\mathsf{M}] \leftarrow \mathsf{Q_M}[\mathsf{M}] \cup \{i\}$
$\textbf{return } \mathsf{pm}_{5,i} := \mathbf{z}_i$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![|\mathsf{CS}| \leqslant T-1]\!]$
$\mathsf{HS} \leftarrow \mathsf{HS} \setminus \{i\}$
$\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$
$\textbf{return } (\mathsf{sk}_i, \mathsf{st}_i)$

Figure 22: The first game, identical to the real adaptive security game.

Game_2:

$\mathcal{O}_{\mathsf{Sign}_1}(i)$

$\textbf{req}\ [\![i \in \mathsf{HS}]\!]$

$\mathsf{str}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$

$\textbf{abort if}\ [\![\mathsf{str}_i \in \mathsf{Strings}]\!]$

$\mathsf{Strings} \leftarrow \mathsf{Strings} \cup \{\mathsf{str}_i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{\mathsf{str}_i\}$

$\textbf{return}\ \mathsf{pm}_{1,i} := \mathsf{str}_i$

Figure 23: The second game. The differences are highlighted in blue. We assume that this game initializes a empty set $\mathsf{Strings} := \varnothing$ at the beginning of the game.

Game_5**:** In this game, the challenger adds an additional abort condition in $\mathcal{O}_{\mathsf{Sign}_4}$. This is depicted in Figure 27. In more detail, the challenger initially sets up an empty table $\mathsf{Signed}_\Sigma[\cdot] = \varnothing$. In $\mathcal{O}_{\mathsf{Sign}_3}$, it adds M_S to $\mathsf{Signed}_\Sigma[i]$ after generating the signature $\sigma_{\mathsf{S},i}$ on M_S. In $\mathcal{O}_{\mathsf{Sign}_4}$, it aborts if there is a honest signer $j \in \mathsf{sHS}\backslash\{i\}$ such that M_S was not signed by j previously in $\mathcal{O}_{\mathsf{Sign}_3}$, *i.e.*, if $\mathsf{M}_\mathsf{S} \notin \mathsf{Signed}_\Sigma[j]$. Otherwise, it continues as before.

Since the challenger checks in $\mathcal{O}_{\mathsf{Sign}_4}$ whether the signature $\sigma_{\mathsf{S},j}$ verifies with respect to M_S for $j \in \mathsf{SS}\backslash\{i\}$, the challenger aborts in Game_5 (but not in Game_4) iff the adversary $\mathcal{A}$ invokes $\mathcal{O}_{\mathsf{Sign}_4}$ with a valid signature $\sigma_{\mathsf{S},j}$ on M_S, where M_S was not signed by signer j in $\mathcal{O}_{\mathsf{Sign}_3}$ yet. It is straigthforward to construct an adversary $\mathcal{B}_\mathsf{S}$ such that $|\epsilon_5 - \epsilon_4| \leqslant N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S},\mathcal{B}_\mathsf{S}}(\lambda)$. Adversary $\mathcal{B}_\mathsf{S}$ simulates the view of $\mathcal{A}$ in Game_4 as follows. First, it guesses a signer $j_* \in [N]$ for which the adversary forges a signature. It obtains $\mathsf{vk}_{\mathsf{S},*}$ from the EUF-CMA game and proceeds as in Game_4, except that it sets $\mathsf{vk}_{\mathsf{S},j_*} := \mathsf{vk}_{\mathsf{S},*}$. In $\mathcal{O}_{\mathsf{Sign}_3}$, signatures for user i are generated via the signature oracle provided by the EUF-CMA game. If $\mathcal{A}$ invokes $\mathcal{O}_{\mathsf{Sign}_4}$ with a valid signature σ_{S,j_*} on $\mathsf{M}_\mathsf{S} \notin \mathsf{Signed}_\Sigma[j_*]$ and j_* is not yet corrupted, then $\mathcal{B}_\mathsf{S}$ forwards σ_{S,j_*} and M_S to the EUF-CMA game. Note that if $\mathsf{M}_\mathsf{S} \notin \mathsf{Signed}_\Sigma[j_*]$, then the message M_S was never queried in the EUF-CMA game by design. In case user j_* becomes corrupted, $\mathcal{B}_\mathsf{S}$ aborts.

Note that the value of j_* is never revealed to $\mathcal{A}$ unless $\mathcal{B}_\mathsf{S}$ aborts. Thus, conditioned on the challenger aborting in Game_5 but not in Game_4 (*i.e.*, $\mathcal{O}_{\mathsf{Sign}_4}$ is invoked for $j \in \mathsf{sHS}$ with a valid signature $\sigma_{\mathsf{S},j}$ on an unsigned message M_S) and $j = j_*$, the above adversary $\mathcal{B}_\mathsf{S}$ succeeds in the EUF-CMA game. Thus, we have

$$|\epsilon_5 - \epsilon_4| \leqslant N \cdot \mathsf{Adv}^{\mathsf{euf\text{-}cma}}_{\mathsf{S},\mathcal{B}_\mathsf{S}}(\lambda)$$

where $\mathsf{Time}(\mathcal{B}_\mathsf{S}) \approx \mathsf{Time}(\mathcal{A})$.

Before describing the next hybrid, we introduce some helpful lemmatas.

Lemma E.8. *We have the following:*

1. *The oracles* $\mathcal{O}_{\mathsf{Sign}_3}, \mathcal{O}_{\mathsf{Sign}_4}$ *and* $\mathcal{O}_{\mathsf{Sign}_5}$ *are invoked with* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*) at most once for each honest signer* $i \in \mathsf{HS}$*.*
2. *If* $\mathcal{O}_{\mathsf{Sign}_4}$ *is invoked with* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*), then* $\mathcal{O}_{\mathsf{Sign}_3}$ *was invoked with* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*) for all signers* $j \in \mathsf{sHS}$*.*
3. *If* $\mathcal{O}_{\mathsf{Sign}_5}$ *is invoked with* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*) on user* i*, then* $\mathcal{O}_{\mathsf{Sign}_4}$ *was invoked with* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*) for all users* $j \in \mathsf{sHS}$*.*

Here, we mean by invoked with $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*) that the values* SS *and* $(\mathsf{str}_j)_{j \in \mathsf{SS}}$ *(resp.* SS, M *and* $(\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$*) defined within the call to the oracle are identical to the values in* $\mathsf{ctnt}_\mathbf{w}$ *(resp.* M_S*).*

Game_3:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

req $[\![i \in \mathsf{HS}]\!] \wedge [\![|\mathsf{CS}| \leqslant T-1]\!]$

// state of user i between round 2 and 3

for $(\mathsf{SS}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i) \in \mathsf{st}_i$ **do**

$\quad \mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$

$\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

$\quad \tilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\overrightarrow{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

$\quad \tilde{\mathbf{w}}_i := \mathbf{w}_i + \tilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$

$\quad \mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i)$

$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i)\}$

$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

// state of user i between round 3 and 4

for $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i$ **do**

$\quad \mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$

$\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

$\quad \tilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\overrightarrow{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

$\quad \tilde{\mathbf{w}}_i := \mathbf{w}_i + \tilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$

$\quad \mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i)$

$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

$\mathsf{HS} \leftarrow \mathsf{HS} \backslash \{i\}$

$\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

return $(\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}})$

req $[\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{HS}]\!]$

pick str_i **from** st_i with $\mathsf{pm}_{1,i} = \mathsf{str}_i$

parse $(\mathsf{str}_j)_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j\in\mathsf{SS}\backslash\{i\}}$

$\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{\mathsf{str}_i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i)\}$

return $\mathsf{pm}_{2,i} := \mathsf{cmt}_i$

$\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}})$

req $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \cdot, \mathsf{pm}_{2,i}) \in \mathsf{st}_i]\!]$

pick $(\mathsf{SS}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i)$ **from** st_i with $\mathsf{pm}_{2,i} = \mathsf{cmt}_i$

parse $(\mathsf{cmt}_j)_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j\in\mathsf{SS}\backslash\{i\}}$

$\mathsf{M}_{\mathsf{S}} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$

$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_{\mathsf{S}}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_{\mathsf{S}})$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, (\mathsf{str}_j)_{j\in\mathsf{SS}}, \mathsf{cmt}_i)\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

return $\mathsf{pm}_{3,i} := \sigma_{\mathsf{S},i}$

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}})$

req $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}) \in \mathsf{st}_i]\!]$

pick $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})$ **from** st_i with $\mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$

parse $(\sigma_{\mathsf{S},j})_{j\in\mathsf{SS}\backslash\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}\backslash\{i\}}$

$\mathsf{M}_{\mathsf{S}} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$

req $[\![\forall j \in \mathsf{SS}\backslash\{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$

$\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$

$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

$\tilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\overrightarrow{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

$\tilde{\mathbf{w}}_i = \mathbf{w}_i + \tilde{\mathbf{\Delta}}_i$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i)$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

return $\mathsf{pm}_{4,i} := \tilde{\mathbf{w}}_i$

Figure 24: The third game. The differences are highlighted in blue. The algorithm ProgramHashCom is defined in Fig. 25

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$:

$\textbf{abort if } [\![\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(i, \widetilde{\mathbf{w}}_i) \neq \perp]\!]$
$\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(i, \widetilde{\mathbf{w}}_i) \leftarrow \mathsf{cmt}_i$

Figure 25: A helper algorithm for programming the random oracle $\mathsf{H}_{\mathsf{com}}$ to open the hash commitments cmt_i consistently. Algorithm $\mathsf{ProgramHashCom}$ is assumed to have a joint state with the challenger and random oracle $\mathsf{H}_{\mathsf{com}}$ used by the unforgeability game.

Game_4:

$\mathsf{H}_{\mathsf{com}}(i, \widetilde{\mathbf{w}})$

$\textbf{if } [\![\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}] = \perp]\!] \textbf{ then}$
$\quad \mathsf{cmt} \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\quad \textbf{abort if } [\![\mathsf{cmt} \in \mathsf{Cmt}]\!]$
$\quad \mathsf{Cmt} \leftarrow \mathsf{Cmt} \cup \{\mathsf{cmt}\}$
$\mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}] \leftarrow \mathsf{cmt}$
$\textbf{return } \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}[i, \widetilde{\mathbf{w}}]$

$\mathcal{O}_{\mathsf{Sign}_2}(\mathsf{SS}, i, (\mathsf{pm}_{1,j})_{j \in \mathsf{SS}})$

$\textbf{req } [\![\mathsf{SS} \subseteq [N]]\!] \wedge [\![i \in \mathsf{HS}]\!]$
$\textbf{pick } \mathsf{str}_i \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{1,i} = \mathsf{str}_i$
$\textbf{parse } (\mathsf{str}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{1,j})_{j \in \mathsf{SS} \setminus \{i\}}$
$\mathsf{cmt}_i \xleftarrow{\$} \{0,1\}^{2\lambda}$
$\textbf{abort if } [\![\mathsf{cmt}_i \in \mathsf{Cmt}]\!]$
$\mathsf{Cmt} \leftarrow \mathsf{Cmt} \cup \{\mathsf{cmt}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{\mathsf{str}_i\}$
$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i)\}$
$\textbf{return } \mathsf{pm}_{2,i} := \mathsf{cmt}_i$

Figure 26: The fourth game. The differences are highlighted in blue. We assume this game initializes a empty set $\mathsf{Cmt} := \varnothing$ at the beginning of the game.

$\underline{\mathsf{Game}_5\text{:}}$

$\underline{\mathcal{O}_{\mathsf{Sign}_3}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{2,j})_{j \in \mathsf{SS}})}$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \cdot, \mathsf{pm}_{2,i}) \in \mathsf{st}_i]\!]$

$\textbf{pick } (\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i) \textbf{ from } \mathsf{st}_i$

$\quad \text{with } \mathsf{pm}_{2,i} = \mathsf{cmt}_i$

$\textbf{parse } (\mathsf{cmt}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{2,j})_{j \in \mathsf{SS} \setminus \{i\}}$

$\mathsf{M}_\mathsf{S} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$

$\sigma_{\mathsf{S},i} \xleftarrow{\$} \mathsf{Sign}_\mathsf{S}(\mathsf{sk}_{\mathsf{S},i}, \mathsf{M}_\mathsf{S})$

$\mathsf{Signed}_\Sigma[i] \leftarrow \mathsf{Signed}_\Sigma[i] \cup \{\mathsf{M}_\mathsf{S}\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, (\mathsf{str}_j)_{j \in \mathsf{SS}}, \mathsf{cmt}_i)\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

$\textbf{return } \mathsf{pm}_{3,i} := \sigma_{\mathsf{S},i}$

$\underline{\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})}$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}) \in \mathsf{st}_i]\!]$

$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}) \textbf{ from } \mathsf{st}_i$

$\quad \text{with } \mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$

$\textbf{parse } (\sigma_{\mathsf{S},j})_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS} \setminus \{i\}}$

$\mathsf{M}_\mathsf{S} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$

$\textbf{req } [\![\forall j \in \mathsf{SS} \setminus \{i\}, \mathsf{Verify}_\mathsf{S}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_\mathsf{S}) = \top]\!]$

$\textbf{abort if } [\![\exists j \in \mathsf{sHS} \setminus \{i\}, \mathsf{M}_\mathsf{S} \notin \mathsf{Signed}_\Sigma[j]]\!]$

$\mathsf{ctnt}_\mathbf{w} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

$(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^\ell_\mathbf{w} \times \mathcal{D}^k_\mathbf{w}$

$\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}^k_q$

$\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_\mathbf{w}) \in \mathcal{R}^k_q$

$\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

$\textbf{return } \mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

Figure 27: The fifth game. The differences are highlighted in blue. We assume this game initializes a empty list $\mathsf{Signed}_\Sigma[\cdot] := \perp$ at the beginning of the game.

Proof. Let $\mathsf{ctnt}_{\mathbf{w}} = 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$ and $\mathsf{M}_\mathsf{S} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$.

Let us show the first statement. Note that each call to the signing oracle of round r for signer i with $\mathsf{ctnt}_{\mathbf{w}}$ (resp. M_S) consumes one $(r-1)$ round state in st_i with str_i. Since str_i is generated at most once in $\mathcal{O}_{\mathsf{Sign}_1}$ due to the abort condition introduced in Game_2, there can be at most one signing oracle call with $\mathsf{ctnt}_{\mathbf{w}}$ (resp. M_S) per round.

Let us show the second statement. If $\mathcal{O}_{\mathsf{Sign}_4}$ is invoked on signer i with M_S, then $\mathcal{O}_{\mathsf{Sign}_3}$ was invoked with M_S for signer $i \in \mathsf{sHS}$ (else the challenger cannot retrieve a matching state from st_i). Also, we know that $\mathsf{M}_\mathsf{S} \in \mathsf{Signed}_\Sigma[j]$ for $j \in \mathsf{sHS}\backslash\{i\}$. Since M_S is added to $\mathsf{Signed}_\Sigma[j]$ in $\mathcal{O}_{\mathsf{Sign}_3}$, it must hold that $\mathcal{O}_{\mathsf{Sign}_3}$ was invoked with M_S for signer $j \in \mathsf{sHS}\backslash\{i\}$. Since $\mathsf{ctnt}_{\mathbf{w}}$ is fully determined by M_S, we can conclude that $\mathsf{ctnt}_{\mathbf{w}}$ must be identical in $\mathcal{O}_{\mathsf{Sign}_4}$ and $\mathcal{O}_{\mathsf{Sign}_3}$, too.

Let us show the last statemet. If $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked with M_S, then we know that $\mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)$ for $j \in \mathsf{SS}$. Also, since M_S is defined by the values retrieved from st_i, we know that $\mathcal{O}_{\mathsf{Sign}_3}$ and $\mathcal{O}_{\mathsf{Sign}_4}$ was called with M_S for user i. Thus, we have that $\mathsf{M}_\mathsf{S} \in \mathsf{Signed}_\Sigma[j]$ for $j \in \mathsf{sHS}$ and consequently, cmt_j was sampled by the challenger in $\mathcal{O}_{\mathsf{Sign}_2}$ without preimage. At that point, cmt_j is also added to Cmt. If $\mathcal{O}_{\mathsf{Sign}_4}$ was called for user j with M_S, the preimage $(j, \widetilde{\mathbf{w}}_j)$ for cmt_j is initialized. Else, the adversary $\mathcal{A}$ must have found a preimage for cmt_j without invoking $\mathcal{O}_{\mathsf{Sign}_4}$ for user j with M_S. But because $\mathsf{cmt}_j \in \mathsf{Cmt}$, the challenger aborts due to the condition added in $\mathsf{H}_{\mathsf{com}}$ in Game_4. ☐

Game_6**:** In this game, the challenger introduces several additional tables: $\mathsf{InitializeOpen}$, $\mathsf{UnOpenedHS}$, $\mathsf{Mask}_{\mathbf{w}}$ and $\mathsf{MaskedCom}$. These tables are indexed by $\mathsf{ctnt}_{\mathbf{w}}$ and indicate the following.

- $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \mathsf{SS}$ indicates that some honest user executed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$. If on the other hand $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$, then no user started round 4 with $\mathsf{ctnt}_{\mathbf{w}}$. (We could also set $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ instead, but we use SS for convenience to check in $\mathcal{O}_{\mathsf{Corrupt}}$ if $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$.)
- $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ stores the set of honest users $\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ that have not executed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$ yet.
- $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \widetilde{\mathbf{\Delta}}_i$ stores the mask $\widetilde{\mathbf{\Delta}}_i = \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$.
- $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] = \widetilde{\mathbf{w}}_i$ stores the masked commitment $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$.

The game is depicted in Figure 28. Let us detail how the tables are managed concretely. In $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger checks if $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$ after $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$ is set. If so, it sets $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$ and $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$. Additionally, after setting up $\widetilde{\mathbf{\Delta}}_i$ and $\widetilde{\mathbf{w}}_i$, it stores the values in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \widetilde{\mathbf{\Delta}}_i$ and $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \widetilde{\mathbf{w}}_i$, and finally updates $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\backslash\{i\}$. In $\mathcal{O}_{\mathsf{Corrupt}}$, the challenger iterates over all $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$. Note that in that case, there is a honest signer that finished round 4 with $\mathsf{ctnt}_{\mathbf{w}}$ but user i is between round 3 and round 4 with $\mathsf{ctnt}_{\mathbf{w}}$ due to Lemma E.8. For each such $\mathsf{ctnt}_{\mathbf{w}}$, the challenger samples $\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ honestly, stores $\widetilde{\mathbf{\Delta}}_i$ in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and removes i from $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$. Later, when iterating over all states of users between round 3 and round 4, the user checks if $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$. If so, it sets $\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ and $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$. After this check, when setting up $\widetilde{\mathbf{w}}_i$ within the loop, the user sets $\widetilde{\mathbf{\Delta}}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and then $\widetilde{\mathbf{w}}_i \leftarrow \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$ (instead of using $\widetilde{\mathbf{\Delta}}_i$ sampled via $\mathsf{ZeroShare}$). Finally, $\widetilde{\mathbf{w}}_i$ is stored in $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$.

Note that the view of adversary $\mathcal{A}$ is identical in Game_5 and Game_6, except in $\mathcal{O}_{\mathsf{Corrupt}}$, the challenger samples $\widetilde{\mathbf{\Delta}}_i$ via $\mathsf{ZeroShare}$ in Game_5 but sets $\widetilde{\mathbf{\Delta}}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ in Game_6 when iterating over states between round 3 and round 4. Since $\widetilde{\mathbf{\Delta}}_i$ is used to compute $\widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$, we need to show that $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ for all states $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})$ (*i.e.*, states between round 3 and round 4) with $\mathsf{ctnt}_{\mathbf{w}} = 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$. But here, $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is set to the

Game_6:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

$\quad$ // Identical to Lines 1 to 10 in Game_3

11 : $\textbf{for } \mathsf{ctnt}_{\mathbf{w}} \text{ s.t. } [\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot]\!] \textbf{ do}$

$\quad$ // $\exists$honest signer finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$

$\quad$ // Sign_3 for user i is completed

12 : $\quad \textbf{parse } 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$ $\quad$ // $i \in \mathsf{SS}$

13 : $\quad \widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

14 : $\quad \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

15 : $\quad \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

$\quad$ // state of user i between round 3 and 4

16 : $\textbf{for } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i \textbf{ do}$

17 : $\quad \mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

18 : $\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

19 : $\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

20 : $\quad \textbf{if } [\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!]$

21 : $\qquad \widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

22 : $\qquad \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

23 : $\quad \widetilde{\mathbf{\Delta}}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$

24 : $\quad \widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$

25 : $\quad \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

26 : $\quad \mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

27 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

28 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

29 : $\mathsf{HS} \leftarrow \mathsf{HS} \backslash \{i\}$

30 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

31 : $\textbf{return } (\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

1 : $\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}) \in \mathsf{st}_i]\!]$

2 : $\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}) \textbf{ from } \mathsf{st}_i$ with $\mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$

3 : $\textbf{parse } (\sigma_{\mathsf{S},j})_{j \in \mathsf{SS} \backslash \{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS} \backslash \{i\}}$

4 : $\mathsf{M}_{\mathsf{S}} := \mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$

5 : $\textbf{req } [\![\forall j \in \mathsf{SS} \backslash \{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$

6 : $\textbf{abort if } [\![\exists j \in \mathsf{sHS} \backslash \{i\}, \mathsf{M}_{\mathsf{S}} \notin \mathsf{Signed}_{\Sigma}[j]]\!]$

7 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

8 : $\textbf{if } [\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!] \textbf{ then}$

9 : $\quad \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

10 : $\quad \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

12 : $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

13 : $\widetilde{\mathbf{\Delta}}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}}) \in \mathcal{R}_q^k$

14 : $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$

15 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

16 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

17 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

18 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

19 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

20 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

21 : $\textbf{return } \mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

Figure 28: The sixth game. The differences are highlighted in blue. We assume that this game initializes four empty lists $\mathsf{InitializeOpen}[\cdot], \mathsf{UnOpenedHS}[\cdot], \mathsf{Mask}_{\mathbf{w}}[\cdot], \mathsf{MaskedCom}[\cdot] := \bot$ at the beginning of the game.

correct value either in line 14 (if $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$) or line 22 (if $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$). Hence, we have

$$\epsilon_6 = \epsilon_5.$$

$\underline{\mathsf{Game}_7\text{:}}$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$ and $\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

```
    // Replace any invocation of ZeroShare(seed⃗_i[SS], ctnt_w) with the following:
    for j ∈ sCS do
        m̃_{i,j} := H_mask(seed_{i,j}, ctnt_w)
        m̃_{j,i} := H_mask(seed_{j,i}, ctnt_w)
    for j ∈ sHS\{i} do
        m̃_{i,j} := H_mask(seed_{i,j}, ctnt_w)
        m̃_{j,i} := H_mask(seed_{j,i}, ctnt_w)
    Δ̃_i := Σ_{j∈SS\{i}} (m̃_{j,i} − m̃_{i,j}) ∈ R_q^k
```

Figure 29: The seventh game. We simple explicitly write down the description of $\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ for convenience.

Game_7: In this game, we expand the definition of $\mathsf{ZeroShare}$ for every invocation of $\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$. This is depicted in Fig. 29. Since both games are identical, we have

$$\epsilon_7 = \epsilon_6.$$

Game_8: In this game, an abort condition is added in the random oracle $\mathsf{H}_{\mathsf{mask}}$ and the challenger modifies how it generates masks $\tilde{\mathbf{\Delta}}_i$ in $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Sign}_4}$. This is depicted in Fig. 30. Specifically, at the beginning of $\mathsf{H}_{\mathsf{mask}}$, the challenger aborts the game if $i \| j \| \mathsf{rand} \leftarrow \mathsf{seed}$ correctly parses and $i, j \in \mathsf{HS}$ and $\mathsf{seed} = \mathsf{seed}_{i,j}$ holds. Further, whenever a mask $\tilde{\mathbf{\Delta}}_i$ is computed in $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger first computes $\tilde{\mathbf{m}}_{i,j}$ and $\tilde{\mathbf{m}}_{j,i}$ for $j \in \mathsf{sCS}$ as before, and then checks if $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$. If so, it sets $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$. Else, it sets $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{sHS}$. Note that in both cases, $\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ represents the honest signers which have not executed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$. Then, for $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ (*i.e.*, honest users after round 4), it retrieves $\tilde{\mathbf{m}}_{i,j}$ and $\tilde{\mathbf{m}}_{j,i}$ from $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$, respectively. For $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}} \backslash \{i\}$, it picks $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ uniformly at random from $\mathcal{R}_q^\ell$ and stores them in $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$, respectively. Finally, it sets $\tilde{\mathbf{\Delta}}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\tilde{\mathbf{m}}_{j,i} - \tilde{\mathbf{m}}_{i,j})$ as before.

Let us analyze the advantage of $\mathcal{A}$ in Game_8. First, we upper bound the probability that the challenger aborts in $\mathsf{H}_{\mathsf{mask}}$. Let $Q_{i,j}$ be the number of the random oracle queries with $i, j \in \mathsf{HS}$. Note that $\sum_{i,j \in \mathsf{HS}} Q_{i,j} \leqslant Q_{\mathsf{H}_{\mathsf{mask}}}$. In each such random oracle query to $\mathsf{H}_{\mathsf{mask}}$, the probability that $\mathsf{rand} = \mathsf{rand}_{i,j}$ is $1/2^\lambda$ since $\mathsf{rand}_{i,j}$ is chosen uniformly at random from $\{0,1\}^\lambda$ and $\mathsf{rand}_{i,j}$ is information-theoretically hidden from $\mathcal{A}$ until either user i or j is corrupted. Thus, the abort probability for fixed pairs (i, j) is at most $Q_{i,j}/2^\lambda$. A union bound across all honest user pairs $(i, j) \in \mathsf{HS}^2$ allows us to upper bound the abort probability with $\frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda}$.

Further, we have to show that if $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{w}}$, then $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$ are already initialized with the $\mathsf{H}_{\mathsf{mask}}$ outputs. Since all signers in $\mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ have already executed $\mathcal{O}_{\mathsf{Sign}_4}$

Game_8:

$\mathsf{H}_{\mathsf{mask}}(\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}})$

$\quad$ **if** $[\![i \| j \| \mathsf{rand} \leftarrow \mathsf{seed}$ correctly parses$]\!]$ **then**
$\quad\quad$ **abort if** $[\![(i, j) \in \mathsf{HS}^2]\!] \wedge [\![\mathsf{rand} = \mathsf{rand}_{i,j}]\!]$
$\quad$ **if** $[\![\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}] = \perp]\!]$ **then**
$\quad\quad$ **if** $[\![$first entry of $\mathsf{ctnt}_{\mathbf{z}}$ is $0]\!]$ **then**
$\quad\quad\quad$ $\mathbf{m} \xleftarrow{\$} \mathcal{R}_q^k$
$\quad\quad$ **else**
$\quad\quad\quad$ $\mathbf{m} \xleftarrow{\$} \mathcal{R}_q^\ell$
$\quad\quad$ $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}$
$\quad$ **return** $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{z}}]$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$ and $\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

$\quad$ // Replace the modification made in Game_7 with the following:
$\quad$ **for** $j \in \mathsf{sCS}$ **do**
$\quad\quad$ $\widetilde{\mathbf{m}}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}})$
$\quad\quad$ $\widetilde{\mathbf{m}}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}})$
$\quad$ **if** $[\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!]$ **then**
$\quad\quad$ $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$
$\quad$ **else**
$\quad\quad$ // No honest user invoked $\mathsf{ZeroShare}$ for $\mathsf{ctnt}_{\mathbf{w}}$ yet
$\quad\quad$ $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{sHS}$
$\quad$ **for** $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{w}}$ **do**
$\quad\quad$ $\widetilde{\mathbf{m}}_{i,j} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}]$
$\quad\quad$ $\widetilde{\mathbf{m}}_{j,i} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$
$\quad$ **for** $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}} \backslash \{i\}$ **do**
$\quad\quad$ $\widetilde{\mathbf{m}}_{i,j} \xleftarrow{\$} \mathcal{R}_q^k, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}] \leftarrow \widetilde{\mathbf{m}}_{i,j}$
$\quad\quad$ $\widetilde{\mathbf{m}}_{j,i} \xleftarrow{\$} \mathcal{R}_q^k, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}] \leftarrow \widetilde{\mathbf{m}}_{j,i}$
$\quad$ $\widetilde{\mathbf{\Delta}}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j}) \in \mathcal{R}_q^k$

Figure 30: The eighth game. The differences are highlighted in blue.

with $\mathsf{ctnt}_{\mathbf{w}}$, these values were initialized in the corresponding $\mathcal{O}_{\mathsf{Sign}_4}$ invocation. Also, we have to show that if $j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}} \backslash \{i\}$, then $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}}] = \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}] = \perp$ (*i.e.*, the outputs are not yet defined and are thus distributed uniformly at this point). Note that due to the abort condition, the adversary $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{mask}}$ on honest seeds directly. Also, $\mathcal{O}_{\mathsf{Sign}_4}$ was never invoked for j with $\mathsf{ctnt}_{\mathbf{w}}$ (and either, this is the first invocation of $\mathcal{O}_{\mathsf{Sign}_4}$ for i with $\mathsf{ctnt}_{\mathbf{w}}$, or user i is corrupted and $\mathcal{O}_{\mathsf{Sign}_4}$ was never invoked for user i). Combining these facts concludes the proof. In total, we have

$$|\epsilon_8 - \epsilon_7| \leqslant \frac{Q_{\mathsf{H}_{\mathsf{mask}}}}{2^\lambda}.$$

Game_9**:** In this game, the challenger samples all but the last mask $\widetilde{\mathbf{\Delta}}_i$ at random (without computing the individual masks $(\widetilde{\mathbf{m}}_{i,j}, \widetilde{\mathbf{m}}_{j,i})_{j \in \mathsf{sHS}}$), and samples the last mask $\widetilde{\mathbf{\Delta}}_i$ *consistently* if all other masks $(\widetilde{\mathbf{\Delta}}_j)_{j \in \mathsf{sHS} \backslash \{i\}}$ are already defined. Also, when a user is corrupted, it programs the oracle $\mathsf{H}_{\mathsf{mask}}$ in accordance. The game is detailed in Fig. 31. In more detail, whenever the challenger computes $\widetilde{\mathbf{\Delta}}_i$ and $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Sign}_4}$ and $\mathcal{O}_{\mathsf{Corrupt}}$, then the user checks if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}$, where $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ is the set of honest users that are still before round 4 with $\mathsf{ctnt}_{\mathbf{w}}$. If so, it samples $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$ at random. Otherwise, it computes $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$ for $j \in \mathsf{sCS}$ and sets $\widetilde{\mathbf{\Delta}}_i := -\sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j$. In $\mathcal{O}_{\mathsf{Corrupt}}$, when passing over states between round 3 and round 4, the user also samples $\widetilde{\mathbf{\Delta}}_i$ at random (if $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$). As before, all masks $\widetilde{\mathbf{\Delta}}_i$ are stored in the table $\mathsf{Mask}_{\mathbf{w}}$. Note that now, the user never invokes $\mathsf{H}_{\mathsf{mask}}$ to compute $\widetilde{\mathbf{\Delta}}_j$ for $j \in \mathsf{HS}$. Instead, at the end of $\mathcal{O}_{\mathsf{Corrupt}}$, the user programs the oracle $\mathsf{H}_{\mathsf{mask}}$ consistently for corrupted user i via $\mathsf{ProgramZeroShare}$ given in Fig. 32. That is, for $\mathsf{ctnt}_{\mathbf{w}} = 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

Game_9:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 10 in Game_3

11 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge$ $[\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!]$ **do**

  // $\exists$honest signer finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$

  // Sign_3 for user i is completed

12 : **parse** $0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$ // $i \in \mathsf{SS}$

13 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$

14 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$

15 : $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$

16 : **elseif** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!]$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

17 :   **for** $j \in \mathsf{sCS}$

18 : $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

19 : $\widetilde{\mathbf{\Delta}}_i := -\sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j$

20 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

21 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\setminus\{i\}$

// state of user i between round 3 and 4

22 : **for** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i$ **do**

23 : $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$

24 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

25 : $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

26 : **if** $[\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$

27 : $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$

28 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

29 : $\mathbf{\Delta}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$

30 : $\widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$

31 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

32 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

33 : $\mathsf{st}_i \leftarrow \mathsf{st}_i\setminus\{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

34 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

35 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **do**

36 : $\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

37 : $\mathsf{HS} \leftarrow \mathsf{HS}\setminus\{i\}$

38 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

39 : **return** $(\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}})$

1 : **req** $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}) \in \mathsf{st}_i]\!]$

2 : **pick** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})$ **from** st_i with $\mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$

3 : **parse** $(\sigma_{\mathsf{S},j})_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j\in\mathsf{SS}\setminus\{i\}}$

4 : $\mathsf{M}_{\mathsf{S}} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$

5 : **req** $[\![\forall j \in \mathsf{SS}\setminus\{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$

6 : **abort if** $[\![\exists j \in \mathsf{sHS}\setminus\{i\}, \mathsf{M}_{\mathsf{S}} \notin \mathsf{Signed}_{\Sigma}[j]]\!]$

7 : $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$

8 : **if** $[\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$ **then**

9 : $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

10 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

12 : $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

13 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

14 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

15 : $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$

16 : **else** // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

17 : **for** $j \in \mathsf{sCS}$

18 : $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

19 : $\widetilde{\mathbf{\Delta}}_i := -\sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \in \mathcal{R}_q^k$

20 : $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$

21 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

22 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

23 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\setminus\{i\}$

24 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

25 : $\mathsf{st}_i \leftarrow \mathsf{st}_i\setminus\{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

26 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

27 : **return** $\mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

Figure 31: The ninth game. The differences are highlighted in blue. The algorithm ProgramZeroShare is defined in Fig. 32.

$\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{z}}, i, \mathbf{\Delta}^*, \mathsf{sCS}, \mathsf{sHS})$:

$\textbf{req } [\![\mathbf{\Delta}^* \neq \bot]\!] \wedge [\![|\mathsf{sHS}| > 1]\!]$

$\textbf{if } [\![\mathbf{\Delta}^* \in \mathcal{R}_q^k]\!] \textbf{ then}$

$\quad t := k \quad$ // Zero share $\mathbf{\Delta}^* = \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ for commitment $\mathbf{w}_i$

$\textbf{else}$

$\quad t := \ell \quad$ // Zero share $\mathbf{\Delta}^* = \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{z}}, i]$ for commitment $\mathbf{z}_i$

$\textbf{for } j \in \mathsf{sCS}$

$\quad \mathbf{m}_{i,j} \leftarrow \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

$\quad \mathbf{m}_{j,i} \leftarrow \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

// Choose an arbitrary honest user other than i

$\textbf{pick } a \textbf{ from } \mathsf{sHS} \backslash \{i\}$

$\textbf{for } j \in \mathsf{sHS} \backslash \{i, a\} \textbf{ do}$

$\quad \mathbf{m}_{i,j} \xleftarrow{\$} \mathcal{R}_q^t, \ \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{i,j}$

$\quad \mathbf{m}_{j,i} \xleftarrow{\$} \mathcal{R}_q^t, \ \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{j,i}$

$\mathbf{m}_{i,a} \xleftarrow{\$} \mathcal{R}_q^t, \ \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,a}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{i,a}$

// Set final *individual* mask to be consistent with zero share $\mathbf{\Delta}^*$

$\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{a,i}, \mathsf{ctnt}_{\mathbf{w}}]$

$\quad \leftarrow \mathbf{\Delta}^* - \sum_{j \in \mathsf{SS} \backslash \{i,a\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j}) + \widetilde{\mathbf{m}}_{i,a}$

Figure 32: A helper algorithm for programming the random oracle $\mathsf{H}_{\mathsf{mask}}$ to open the zero shares $\mathbf{\Delta}_i \in \mathcal{R}_q^\ell$ and $\widetilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$ consistently. Algorithm $\mathsf{ProgramZeroShare}$ is assumed to have a joint state with the challenger and random oracle $\mathsf{H}_{\mathsf{mask}}$ used by the unforgeability game.

such that $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] \neq \perp$, it invokes $\mathsf{ProgramZeroShare}$ with input $(\mathsf{ctnt_w}, i, \widetilde{\mathbf{\Delta}}^*, \mathsf{sCS}, \mathsf{sHS})$, where $\widetilde{\mathbf{\Delta}}^* := \mathsf{Mask_w}[\mathsf{ctnt_w}, i]$, which programs $\mathsf{H_{mask}}$ as follows. Initially, two sanity checks are performed: (1) except i, there is at least another uncorrupted user $a \in \mathsf{sHS} \backslash \{i\}$ and (2) $\widetilde{\mathbf{\Delta}}^* \neq \perp$. Then, for each $j \in \mathsf{sCS}$, the challenger sets $\mathbf{m}_{i,j} \leftarrow \mathsf{H_{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt_w})$ and $\mathbf{m}_{j,i} \leftarrow \mathsf{H_{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt_w})$. For each $j \in \mathsf{sHS} \backslash \{i, a\}$, the signer samples $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ at random and programs $\mathsf{H_{mask}}$ accordingly. For user a, it samples only $\mathsf{Q_{H_{mask}}}[\mathsf{seed}_{i,a}, \mathsf{ctnt_w}] \leftarrow \mathbf{m}_{i,a} \xleftarrow{\$} \mathcal{R}_q^k$ at random, and the final individual mask $\mathbf{m}_{a,i}$ is set consistently, *i.e.*, $\mathsf{Q_{H_{mask}}}[\mathsf{seed}_{a,i}, \mathsf{ctnt_w}] \leftarrow \mathbf{\Delta}^* - \sum_{j \in \mathsf{SS} \backslash \{i,a\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j}) + \widetilde{\mathbf{m}}_{i,a}$.

We show that Game_9 and Game_8 are identically distributed. Observe that the (potential) observable differences between both games are the distribution of the masks $\widetilde{\mathbf{\Delta}}_i$ and the output of $\mathsf{H_{mask}}$. We first show that the masks $\widetilde{\mathbf{\Delta}}_i$ are identically distributed in both games. Then, we show that $\mathsf{ProgramZeroShare}$ programs $\mathsf{H_{mask}}$ as desired. We initially fix some arbitrary $\mathsf{ctnt_w}$ and later apply a hybrid argument to conclude.

Lemma E.9. *Let* $\mathsf{ctnt_w}$ *be fixed. If* $\mathsf{InitializeOpen}[\mathsf{ctnt_w}] \neq \perp$*, then in both games, we have for* $i \in \mathsf{sHS}$ *that*

1. $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] = \perp$ *if signer* i *is not being corrupted and has not passed round* 4 *with* $\mathsf{ctnt_w}$*, else*
2. $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] \sim \mathcal{U}_{\mathcal{R}_q^k}$ *is distributed at random, if there remains another honest signer* $j \in \widetilde{\mathsf{sHS}}_\mathbf{w} \backslash \{i\}$ *before round* 4 *with* $\mathsf{ctnt_w}$ *and*
3. $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] = -\sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{Mask_w}[\mathsf{ctnt_w}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j$*, if* i *was the last user between round* 3 *and round* 4 *with* $\mathsf{ctnt_w}$ *(i.e., if* $\widetilde{\mathsf{sHS}}_\mathbf{w} = \{i\}$*).*

Proof. The first statement holds in both games by construction. The second and third statmeent hold for Game_9 by construction. Let us inspect the distribution of $\mathsf{Mask_w}[\mathsf{ctnt_w}, i]$ in Game_8 for $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] \neq \perp$. Observe that all values stored in $\mathsf{Mask_w}[\mathsf{ctnt_w}, i]$ are computed as depicted in Fig. 30. If there exists some $j \in \widetilde{\mathsf{sHS}}_\mathbf{w} \backslash \{i\}$, then $\widetilde{\mathbf{m}}_{i,j}$ and $\widetilde{\mathbf{m}}_{j,i}$ are sampled at random over $\mathcal{R}_q^k$. Thus, $\widetilde{\mathbf{\Delta}}_i = \sum_{j \in \mathsf{SS} \backslash \{i\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j})$ is distributed at random over $\mathcal{R}_q^k$. If on the other hand $\widetilde{\mathsf{sHS}}_\mathbf{w} = \{i\}$, then all individual masks $(\widetilde{\mathbf{m}}_{i,j}, \widetilde{\mathbf{m}}_{j,i})_{j \in \mathsf{sHS}}$ are retrieved from $\mathsf{Q_{H_{mask}}}$ and thus, $\widetilde{\mathbf{\Delta}}_i$ is fully determined. Because we have that $\sum_{j \in \mathsf{SS}} \widetilde{\mathbf{\Delta}}_j = \mathbf{0}$, where $\widetilde{\mathbf{\Delta}}_j = \sum_{\kappa \in \mathsf{SS} \backslash \{j\}} (\widetilde{\mathbf{m}}_{\kappa,j} - \widetilde{\mathbf{m}}_{j,\kappa})$, we have that

$$\widetilde{\mathbf{\Delta}}_i = -\sum_{j \in \mathsf{SS} \backslash \{i\}} \widetilde{\mathbf{\Delta}}_j.$$

Finally, observe that every time a signer $j \in \mathsf{sHS}$ is removed from $\widetilde{\mathsf{sHS}}_\mathbf{w}$, the value $\widetilde{\mathbf{\Delta}}_j$ is stored in $\mathsf{Mask_w}[\mathsf{ctnt_w}, j]$. Thus, if $\widetilde{\mathsf{sHS}}_\mathbf{w} = \{i\}$, we have that

$$\sum_{j \in \mathsf{SS} \backslash \{i\}} \widetilde{\mathbf{\Delta}}_j = \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{Mask_w}[\mathsf{ctnt_w}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j.$$

Combining the both equations concludes. ◻

Lemma E.10. *Let* $\mathsf{ctnt_w}$ *be fixed. If* $\mathsf{InitializeOpen}[\mathsf{ctnt_w}] = \perp$ *and* $\mathsf{Mask_w}[\mathsf{ctnt_w}, i] \neq \perp$*, then* $\mathsf{Mask_w}[\mathsf{ctnt_w}, i]$ *is distributed uniformly over* $\mathcal{R}_q^k$ *in both games.*

Proof. This is immediate for Game_9. In Game_8, the values $\widetilde{\mathbf{\Delta}}_i$ stored in $\mathsf{Mask_w}[\mathsf{ctnt_w}, i]$ are computed as depicted in Fig. 30. Here, the challenger sets $\widetilde{\mathsf{sHS}}_\mathbf{w} \leftarrow \mathsf{sHS}$ when computing $\widetilde{\mathbf{\Delta}}_i$. Thus, all values $(\widetilde{\mathbf{m}}_{i,j}, \widetilde{\mathbf{m}}_{j,i})_{j \in \mathsf{sHS}}$ are sampled at random. Since $\mathsf{sHS} \neq \varnothing$, the statement follows. ◻

Next, we show that $\mathsf{ProgramZeroShare}$ programs the oracle $\mathsf{H_{mask}}$ as desired in Game_9.

Lemma E.11. *Let* $\mathsf{ctnt}_{\mathbf{w}}$ *be fixed. For every query of the form* $(\mathsf{seed}, \mathsf{ctnt}_{\mathbf{w}})$ *to* $\mathsf{H}_{\mathsf{mask}}$ *of* $\mathcal{A}$*, the value* $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}, \mathsf{ctnt}_{\mathbf{w}}]$ *is identically distributed in both games.*

Proof. For $i \in \mathsf{CS}$, we need to show that the individual masks $\mathbf{m}_{j,i} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}})$ and $\mathbf{m}_{i,j} = \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{w}})$ output by $\mathsf{H}_{\mathsf{mask}}$ are distributed at random in Game_9 (as in Game_8), where $j \in \mathsf{SS}$. (If both j and i are corrupt, then let us assume without loss of generality that i was corrupted first.) Note that if $(i, j) \in \mathsf{HS}^2$, then the challenger aborts in $\mathsf{H}_{\mathsf{mask}}$ in both games (if the seed matches). Further, since $\widetilde{\boldsymbol{\Delta}}_i$ is computed via $\mathsf{H}_{\mathsf{mask}}$ in Game_8, we need to show that in Game_9, we also have $\widetilde{\boldsymbol{\Delta}}_i = \sum_{j \in \mathsf{SS} \setminus \{i\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j})$ in $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Sign}_4}$.

Note that when $\widetilde{\boldsymbol{\Delta}}_i$ is sampled in Game_9, then it is stored in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$. Thus, $\mathsf{ProgramZeroShare}$ is invoked with input $(\mathsf{ctnt}_{\mathbf{w}}, i, \widetilde{\boldsymbol{\Delta}}_i, \mathsf{sCS}, \mathsf{sHS})$ when i is corrupted. By construction, we have that $\widetilde{\mathbf{m}}_{a,i} = \widetilde{\boldsymbol{\Delta}}_i - \sum_{j \in \mathsf{SS} \setminus \{i,a\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j}) + \widetilde{\mathbf{m}}_{i,a}$. Reordering the equation confirms that $\widetilde{\boldsymbol{\Delta}}_i = \sum_{j \in \mathsf{SS} \setminus \{i\}} (\widetilde{\mathbf{m}}_{j,i} - \widetilde{\mathbf{m}}_{i,j})$. It remains to show that each individual mask is distributed uniformly at random. By construction, this is immediate for all masks except $\widetilde{\mathbf{m}}_{a,i}$. If i is not the last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$, then $\widetilde{\boldsymbol{\Delta}}_i$ was sampled at random and thus $\widetilde{\mathbf{m}}_{a,i}$ is also distributed at random. Otherwise, we have $\widetilde{\boldsymbol{\Delta}}_i = -\sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j$. Since there is an honest user $a_* \in \mathsf{sHS}$ that is never corrupted, we know that $\widetilde{\boldsymbol{\Delta}}_{a_*} = \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, a_*]$ is distributed uniformly and independently at random and thus, $\widetilde{\mathbf{m}}_{a,i}$ is uniform. □

When we combine Lemmata E.9 to E.11 with a hybrid argument over all $\mathsf{ctnt}_{\mathbf{w}}$ in order of occurence, we have

$$\epsilon_9 = \epsilon_8.$$

Game_{10}**:** In this game, the challenger manages additional tables $\mathsf{UsedCom}$, SumCom and $\mathsf{SumComRnd}$. Also, it samples $\widetilde{\mathbf{w}}_i$ in $\mathcal{O}_{\mathsf{Sign}_4}$ differenlty. These tables are indexed by $\mathsf{ctnt}_{\mathbf{w}}$ and indicate the following.

- $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] = (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ stores the commitment $\mathbf{w}_i$ with its randomness $(\mathbf{r}_i, \mathbf{e}'_i)$ that is used by honest signer i in $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$.
- $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \mathbf{w}$ stores the (partial) sum $\mathbf{w} = \sum_{j \in \mathsf{sHS} \setminus \widetilde{\mathsf{sHS}}_{\mathbf{w}}} \mathbf{w}_j$ of all commitments $\mathbf{w}_j$ that honest users used in $\mathcal{O}_{\mathsf{Sign}_4}$.
- $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \mathbf{r}$ stores the (partial) sum $\mathbf{r} = \sum_{j \in \mathsf{sHS} \setminus \widetilde{\mathsf{sHS}}_{\mathbf{w}}} \mathbf{r}_j$ of the commitment randomness $\mathbf{r}_j$ that honest users used in $\mathcal{O}_{\mathsf{Sign}_4}$.

This is depicted in Fig. 33. In $\mathcal{O}_{\mathsf{Sign}_4}$, after sampling the values $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$, the challenger stores $\mathbf{w}_i$ in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and updates $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}_i$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$. Also, instead of sampling $\widetilde{\boldsymbol{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$ at random if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}$, the challenger samples $\widetilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$ directly. Similarly, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, the challenger sets $\widetilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$. Note that in Game_{10}, $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ remains undefined at this point. This step is delayed until user i is corrupted. In $\mathcal{O}_{\mathsf{Corrupt}}$, when passing over $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$, the challenger checks whether $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$. In that case, it retrieves $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$, sets $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] - \mathbf{w}_i$, $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{w}_i$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{r}_i$. Also, the masks for the last user are sampled via $\widetilde{\boldsymbol{\Delta}}_i = \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$ in $\mathcal{O}_{\mathsf{Corrupt}}$.

Let us show that both games are identically distributed. First, we show that SumCom and $\mathsf{SumComRnd}$ indeed store the intended partial sums.

Game_{10}:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

$\quad$ // Identical to Lines 1 to 10 in Game_3

11 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot]\!]$ **do**

$\quad$ // $\exists$honest signer finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$

$\quad$ // Sign_3 for user i is completed

12 : $\quad$ **parse** $0\|\mathsf{SS}\|(\mathsf{str}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$ $\quad$ // $i \in \mathsf{SS}$

13 : $\quad$ **if** $[\![\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot]\!]$ **then**

$\quad\quad$ // user i finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$

14 : $\quad\quad$ $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

15 : $\quad\quad$ $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{w}_i$

16 : $\quad\quad$ $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{w}_i$

17 : $\quad\quad$ $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{r}_i$

18 : $\quad$ **else** $\quad$ // user i is between Round 3 and Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$

19 : $\quad\quad$ $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ $\quad$ // $i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$

20 : $\quad\quad$ **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$

21 : $\quad\quad\quad$ $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$

22 : $\quad\quad$ **elseif** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!]$ $\quad$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

23 : $\quad\quad\quad$ **for** $j \in \mathsf{sCS}$

24 : $\quad\quad\quad\quad$ $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

25 : $\quad\quad\quad$ $\widetilde{\mathbf{\Delta}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS}\setminus\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

26 : $\quad\quad$ $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$

27 : $\quad\quad$ $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

$\quad$ // Identical to Lines 21 to 38 in Game_9

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

1 : **req** $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{3,i}) \in \mathsf{st}_i]\!]$

2 : **pick** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})$ **from** st_i with $\mathsf{pm}_{3,i} = \sigma_{\mathsf{S},i}$

3 : **parse** $(\sigma_{\mathsf{S},j})_{j \in \mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}\setminus\{i\}}$

4 : $\mathsf{M}_{\mathsf{S}} := \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}$

5 : **req** $[\![\forall j \in \mathsf{SS}\setminus\{i\}, \mathsf{Verify}_{\mathsf{S}}(\mathsf{vk}_{\mathsf{S},j}, \sigma_{\mathsf{S},j}, \mathsf{M}_{\mathsf{S}}) = \top]\!]$

6 : **abort if** $[\![\exists j \in \mathsf{sHS}\setminus\{i\}, \mathsf{M}_{\mathsf{S}} \notin \mathsf{Signed}_{\Sigma}[j]]\!]$

7 : $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j \in \mathsf{SS}}$

8 : **if** $[\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!]$ **then**

9 : $\quad$ $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

10 : $\quad$ $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

12 : $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$

13 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

14 : $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}_i$

15 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}_i$

16 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

17 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

18 : $\quad$ $\widetilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$

19 : **else** $\quad$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

20 : $\quad$ **for** $j \in \mathsf{sCS}$

21 : $\quad\quad$ $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

22 : $\quad$ $\widetilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS}\setminus\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

23 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

24 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

25 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

26 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

27 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

28 : **return** $\mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

Figure 33: The tenth game. The differences are highlighted in blue. We assume that this game initializes three empty lists $\mathsf{UsedCom}[\cdot], \mathsf{SumCom}[\cdot], \mathsf{SumComRnd}[\cdot] := \bot$ at the beginning of the game.

Lemma E.12. *Let* $\mathsf{ctnt}_{\mathbf{w}}$ *be arbitrary. Let* $\mathsf{sHS}' = \mathsf{sHS}$ *(resp.* $\mathsf{sHS}' = \mathsf{sHS}\backslash\{i\}$*). In* Game_{10}*, we have in line* 22 *in* $\mathcal{O}_{\mathsf{Sign}_4}$ *(resp. line* 25 *in* $\mathcal{O}_{\mathsf{Corrupt}}$*) that* $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}'\backslash\widetilde{\mathsf{sHS}}_{\mathbf{w}}} \mathbf{w}_j$ *and* $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in \mathsf{sHS}'\backslash\widetilde{\mathsf{sHS}}_{\mathbf{w}}} \mathbf{r}_j$*, where* $(\mathbf{w}_j, \mathbf{r}_j, \mathbf{e}'_j) = \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$.

Proof. Note that $\mathsf{sHS}'\backslash\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ is the set of honest users that passed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$ (excluding user i if it is in the process of being corrupted). Recall that in $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger adds $\mathbf{w}_i$ and $\mathbf{r}_i$ to SumCom and $\mathsf{SumComRnd}$, respectively, and initializes $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] = \widetilde{\mathbf{w}}_i$. Thus, each $\mathcal{O}_{\mathsf{Sign}_4}$ call keeps the invariant. If user $i \in \mathsf{sHS}\backslash\widetilde{\mathsf{sHS}}_{\mathbf{w}}$ is being corrupted, then we have $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$ and $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$. Consequently, the values $\mathbf{w}_i$ and $\mathbf{r}_i$ are removed from SumCom and $\mathsf{SumComRnd}$ in line 16 and line 17, respectively, if previously added. The statement follows. $\square$

Next, let us consider an intermedite game of $\mathsf{Game}_{9,*}$, where instead of sampling $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$ at random in $\mathcal{O}_{\mathsf{Sign}_4}$, we sample $\widetilde{\mathbf{\Delta}}_i^* \xleftarrow{\$} \mathcal{R}_q^k$ and set $\widetilde{\mathbf{\Delta}}_i := \widetilde{\mathbf{\Delta}}_i^* - \mathbf{w}_i$. Clearly, this game is identically distributed to Game_9. Then, we have that

$$\begin{aligned} \widetilde{\mathbf{w}}_i &= \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \\ &= \widetilde{\mathbf{\Delta}}_i^* \sim \mathcal{U}_{\mathcal{R}_q^k} \end{aligned}$$

which is distributed as in Game_{10}. Consequently, we have if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ that

$$\begin{aligned} \widetilde{\mathbf{w}}_i &= \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \\ &= \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \\ &= \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} (\widetilde{\mathbf{\Delta}}_j^* - \mathbf{w}_j) - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \\ &= \sum_{j \in \mathsf{sHS}} \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} \widetilde{\mathbf{\Delta}}_j^* - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \\ &= \sum_{j \in \mathsf{sHS}} \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j. \end{aligned}$$

Similarly, we have in $\mathcal{O}_{\mathsf{Corrupt}}$ if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ that

$$\begin{aligned} \widetilde{\mathbf{\Delta}}_i &= -\sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \\ &= -\sum_{j \in \mathsf{sHS}\backslash\{i\}} \widetilde{\mathbf{\Delta}}_j^* - \mathbf{w}_j \\ &= \sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} \widetilde{\mathbf{\Delta}}_j^* - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j \\ &= \sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathbf{w}_i - \sum_{j \in \mathsf{sHS}\backslash\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j. \end{aligned}$$

Due to Lemma E.12, we have that $\widetilde{\mathbf{w}}_i$ and $\widetilde{\mathbf{\Delta}}_i$ are identically distributed in $\mathsf{Game}_{9,*}$ and Game_{10}. Finally, note that while $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is initialized in $\mathcal{O}_{\mathsf{Sign}_4}$ in $\mathsf{Game}_{9,*}$ but not in Game_{10}, whenever $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is used by the challenger, the value is identically distributed. We conclude that

$$\epsilon_{10} = \epsilon_9.$$

Game_{11} – part 1:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

$/\!/$ Identical to Lines 1 to 10 in Game_3
$\textbf{for } \mathsf{ctnt}_{\mathbf{w}} \text{ s.t. } [\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge$
$\quad [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!] \textbf{ do}$
$\quad /\!/$ $\exists$honest signer finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
$\quad /\!/$ Sign_3 for user i is completed
$\quad \textbf{parse } 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}} \quad /\!/\ i \in \mathsf{SS}$
$\quad \textbf{if } [\![\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!] \textbf{ then}$
$\quad\quad /\!/$ user i finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
$\quad\quad \textbf{if } [\![\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \varnothing]\!] \textbf{ then}$
$\quad\quad\quad \textbf{if } [\![\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!] \textbf{ then}$
$\quad\quad\quad\quad /\!/$ user i is between Round 4 and 5
$\quad\quad\quad\quad \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$
$\quad\quad\quad (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
$\quad\quad\quad \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{r}_i$
$\quad\quad \textbf{else} \quad /\!/$ There is still a honest user in Round 3 with $\mathsf{ctnt}_{\mathbf{w}}$
$\quad\quad\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$
$\quad\quad\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$
$\quad\quad\quad \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$
$\quad\quad \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] - \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
$\quad \textbf{else} \quad /\!/$ user i is between Round 3 and Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
$\quad\quad \widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \quad /\!/\ i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$
$\quad\quad \textbf{if } [\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$
$\quad\quad\quad \widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$
$\quad\quad \textbf{elseif } [\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!] \quad /\!/$ Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$
$\quad\quad\quad \textbf{for } j \in [|\mathsf{sHS}| - 1] \textbf{ do}$
$\quad\quad\quad\quad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$
$\quad\quad\quad\quad \mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}_q^k$
$\quad\quad\quad\quad \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$
$\quad\quad\quad\quad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$
$\quad\quad\quad\quad \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$
$\quad\quad\quad \textbf{for } j \in \mathsf{sCS}$
$\quad\quad\quad\quad \widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$
$\quad\quad\quad \widetilde{\mathbf{\Delta}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$
$\quad\quad \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$
$\quad\quad \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\backslash\{i\}$

$/\!/$ Continuation of $\mathcal{O}_{\mathsf{Corrupt}}$
$/\!/$ state of user i between round 3 and 4
$\textbf{for } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i \textbf{ do}$
$\quad \mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$
$\quad (\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$
$\quad \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}_q^k$
$\quad \textbf{if } [\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$
$\quad\quad \widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}_q^k$
$\quad\quad \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$
$\quad \mathbf{\Delta}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$
$\quad \widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i \in \mathcal{R}_q^k$
$\quad \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$
$\quad \mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i\backslash\{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \sigma_{\mathsf{S},i})\}$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$/\!/$ state of user i between round 4 and 5
$\textbf{for } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i) \in \mathsf{st}_i \textbf{ do}$
$\quad \mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$
$\quad (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i\backslash\{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i)\} \textbf{ do}$
$\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
$\textbf{for } \mathsf{ctnt}_{\mathbf{w}} \text{ s.t. } [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!] \textbf{ do}$
$\quad \mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$
$\mathsf{HS} \leftarrow \mathsf{HS}\backslash\{i\}$
$\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$
$\textbf{return } (\mathsf{sk}_i, \mathsf{st}_i)$

Figure 34: The first part of the eleventh game. The differences are highlighted in blue.

Game_{11} – part 2:

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 10 in Game_{10}

11 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

12 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

13 : $\quad \tilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$

14 : **else** // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

15 : $\quad$ **for** $j \in [|\mathsf{sHS}|]$ **do**

16 : $\quad\quad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

17 : $\quad\quad \mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}_q^k$

18 : $\quad\quad \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$

19 : $\quad\quad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$

20 : $\quad\quad \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$

21 : $\quad$ **for** $j \in \mathsf{sCS}$

22 : $\quad\quad \tilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

23 : $\quad \tilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \tilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

24 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \tilde{\mathbf{w}}_i$

25 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

26 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \tilde{\mathbf{w}}_i)$

27 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

28 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i)\}$

29 : **return** $\mathsf{pm}_{4,i} := \tilde{\mathbf{w}}_i$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

1 : **req** $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

2 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

3 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$

4 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

5 : **parse** $(\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$

6 : **parse** $(\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS} \setminus \{i\}}$

7 : **pick** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{4,i} = \tilde{\mathbf{w}}_i$

8 : **req** $[\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \tilde{\mathbf{w}}_j)]\!]$

9 : $\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$

10 : $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \tilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$

11 : $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ // $c \in \mathcal{C}$

12 : $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$

13 : $\tilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell}$

14 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i)\}$

15 : $\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

16 : **return** $\mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 35: The second part of the eleventh game. The differences are highlighted in blue.

Game_{11}**:** In this game, the challenger delays sampling the commitments until the last signer finishes $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$. These commitments are stored in a table $\mathsf{UnUsedCom}$ and assigned in $\mathcal{O}_{\mathsf{Sign}_5}$ (or $\mathcal{O}_{\mathsf{Corrupt}}$) to signers in $\mathsf{UsedCom}$. This is depicted in Figs. 34 and 35. Let us detail the changes. In $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger samples all commitments at once if user i is the last honest signer in round 4, *i.e.*, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$. That is, it generates $|\mathsf{sHS}|$ commitments $(\mathbf{w}_j)_{j\in|\mathsf{sHS}|}$ and stores them in the table $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$. At that point, it also computes and stores the sums in SumCom and $\mathsf{SumComRnd}$. Notably, the commitments are not attributed to specific users in $\mathcal{O}_{\mathsf{Sign}_4}$ and the randomness $\mathbf{r}_i$ is removed from the state until $\mathcal{O}_{\mathsf{Sign}_5}$. In $\mathcal{O}_{\mathsf{Sign}_5}$, the user first removes an unused commitment from $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$, stores $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and proceeds as before with this commitment. In $\mathcal{O}_{\mathsf{Corrupt}}$, if $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$ when passing over $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$, the challenger checks if $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \varnothing$. Note that this is the case if all honest users passed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$. In that case, the user retrieves $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$, else it chooses an unused commitment $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$. Then, it removes $\mathbf{w}_i$ and $\mathbf{r}_i$ from $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$, respectively. If otherwise $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \varnothing$, then a fresh commitment $\mathbf{w}_i$ is sampled and stored with its randomness in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$. Again, if i is the last user before round 4, it generates $|\mathsf{sHS}| - 1$ differents commitments and stores them in the table $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$. At that point, it also computes and stores the sums in SumCom and $\mathsf{SumComRnd}$. At the end of $\mathcal{O}_{\mathsf{Corrupt}}$, the challenger passes over states of users between round 4 and 5 and reintroduces $\mathbf{r}_i$ into the states via $\mathsf{UsedCom}$.

Let us show that both games are identically distributed. Let $\mathsf{ctnt}_{\mathbf{w}}$ be fixed. Observe that the output $\widetilde{\mathbf{w}}_i$ in $\mathcal{O}_{\mathsf{Sign}_4}$ is identically distributed in both games for all but the last honest signer. Let us inspect the distribution of $\widetilde{\mathbf{w}}_i$ when $\mathcal{O}_{\mathsf{Sign}_4}$ is called for the last user i with $\mathsf{ctnt}_{\mathbf{w}}$, *i.e.*, $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$. Since $\widetilde{\mathbf{w}}_i$ is computed via $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$, we need to show that in Game_{11}, the table $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ stores the sum of the commitments $\mathbf{w}_j$ that are used by honest users $j \in \mathsf{sHS}$ in $\mathcal{O}_{\mathsf{Sign}_4}$. Note that while in Game_{11}, these commitments $\mathbf{w}_j$ are not attributed internally to any user in $\mathcal{O}_{\mathsf{Sign}_4}$ yet, the sum stored in $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is identically distributed by construction. The commitments are stored in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$. Similarly, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ in $\mathcal{O}_{\mathsf{Corrupt}}$, then $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ are initialized before $\widetilde{\mathbf{\Delta}}_i$ is computed. We can reason as above that $\widetilde{\mathbf{\Delta}}_i$ is distributed identically in $\mathcal{O}_{\mathsf{Corrupt}}$ in Game_{11} in that case, and $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is initialized with the commitments that sum up to $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$, *i.e.*, $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{(\mathbf{w}_j, \mathbf{r}_j, \mathbf{e}'_j) \in \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]} \mathbf{w}_j$. The above allows is to conclude that

$$\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \varnothing \implies \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp. \tag{21}$$

Observe that the signer state st_i and the distribution of $\mathcal{O}_{\mathsf{Sign}_5}$ is identically distributed in both games, if $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is identically distributed when accessed by the simulator. Note that $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is handled identically in both games if user i never signed with $\mathsf{ctnt}_{\mathbf{w}}$ via $\mathcal{O}_{\mathsf{Sign}_4}$ yet: it is not set for $i \in \mathsf{sHS}$ and freshly sampled in $\mathcal{O}_{\mathsf{Corrupt}}$ when user i is corrupted. Let us inspect the remaining cases:

1. User i is corrupted between round 4 and round 5 with $\mathsf{ctnt}_{\mathbf{w}}$, and there remains another honest users before round 4 with $\mathsf{ctnt}_{\mathbf{w}}$, *i.e.*, $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \varnothing$. In both games, we have that $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$. In Game_{11}, $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is initialized in line 23 and in Game_{10}, it is initialized in $\mathcal{O}_{\mathsf{Sign}_4}$. The commitment $\mathbf{w}_i$ is sampled in the same manner in both games. In Game_{11}, its randomness does *not* influence the value of $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$. In Game_{10}, it is first added to $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Sign}_4}$, but removed in $\mathcal{O}_{\mathsf{Corrupt}}$ in line 17 before $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is accessed to compute $\widetilde{\mathbf{\Delta}}_i$ or $\widetilde{\mathbf{w}}_i$.
2. User i is corrupted between round 4 and round 5 with $\mathsf{ctnt}_{\mathbf{w}}$, and $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \varnothing$. In Game_{10}, the value $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ is set in $\mathcal{O}_{\mathsf{Sign}_4}$ and its randomness is included in the sum $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ when it is accessed to compute $\widetilde{\mathbf{\Delta}}_i$ or $\widetilde{\mathbf{w}}_i$. In Game_{11}, the value $\mathbf{w}_j$ is chosen from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ (cf. line 17), *i.e.*, its randomness is also included in the sum $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ when it is accessed to compute $\widetilde{\mathbf{\Delta}}_i$ or $\widetilde{\mathbf{w}}_i$.

3. User i is in (or after) round 5 with $\mathsf{ctnt}_{\mathbf{w}}$. Then, in Game_{10}, the value of $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is distributed as in the previous case (since it remains unchanged). In Game_{11}, the value is chosen from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, *i.e.*, is distributed as in the previous case, too.

Note that in Game_{11}, whenever a value is chosen from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, then it is removed from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and thus, its randomness is counted at most once in $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ as in Game_{11}. Also, observe that $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Sign}_5}$ removes at most one commitment from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$. This occurs in line 17 (resp. line 3) if user i has a state between round 4 and 5 with $\mathsf{ctnt}_{\mathbf{w}}$ in $\mathcal{O}_{\mathsf{Corrupt}}$ (resp. if $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked for user i with $\mathsf{ctnt}_{\mathbf{w}}$). Recall that due to Lemma E.8, $\mathcal{O}_{\mathsf{Sign}_5}$ is called at most once per $\mathsf{ctnt}_{\mathbf{w}}$ and $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked only if all honest users passed round 4 with $\mathsf{ctnt}_{\mathbf{w}}$, *i.e.*, $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \varnothing$, and thus, $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$ due to Eq. (21). Also, note that if $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is set in $\mathcal{O}_{\mathsf{Sign}_5}$, then no commitment is removed from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Corrupt}}$. The above observations allow us to conlude that each time a commitment is chosen from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, the set is non-empty. In summary, we have that $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is identically distributed in Game_{10} and Game_{11}.

Finally, a hybrid argument over all $\mathsf{ctnt}_{\mathbf{w}}$ shows that

$$\epsilon_{11} = \epsilon_{10}$$

Game_{12}: In this game, the challenger introduces several additional tables $\mathsf{InitializeSign}, \mathsf{UnSignedHS}, \mathsf{Mask}_{\mathbf{z}}$ and $\mathsf{MaskedResp}$. All tables are indexed by $\mathsf{ctnt}_{\mathbf{w}}$ and the first four tables are the functional equivalents to $\mathsf{InitializeOpen}$, $\mathsf{UnOpenedHS}$, $\mathsf{Mask}_{\mathbf{w}}$, $\mathsf{MaskedCom}$, respectively, but for the masks $\mathbf{\Delta}_i$ in $\mathcal{O}_{\mathsf{Sign}_5}$ instead of $\widetilde{\mathbf{\Delta}}_i$. Explicitly, each table represents the following.

- $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \mathsf{SS}$ indicates that some honest user executed round 5 with $\mathsf{ctnt}_{\mathbf{w}}$. If on the other hand $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$, then no user started round 5 with $\mathsf{ctnt}_{\mathbf{w}}$.
- $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \widetilde{\mathsf{sHS}}_{\mathbf{z}}$ stores the set of honest users $\widetilde{\mathsf{sHS}}_{\mathbf{z}}$ that have not executed round 5 with $\mathsf{ctnt}_{\mathbf{w}}$ yet.
- $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \mathbf{\Delta}_i$ stores the mask $\mathbf{\Delta}_i$.
- $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] = \widetilde{\mathbf{z}}_i$ stores the masked response $\widetilde{\mathbf{z}}_i$.

The game is depicted in Figure 36. Let us detail how the tables are managed concretely. In $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger sets $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$ and checks if $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \perp$. If so, it sets $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$ and $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$. Additionally, after setting up $\mathbf{\Delta}_i$ and $\widetilde{\mathbf{z}}_i$, it stores the values in $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$ and $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \widetilde{\mathbf{z}}_i$, and finally updates $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$. In $\mathcal{O}_{\mathsf{Corrupt}}$, the challenger iterates over all $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$. Note that in that case, there is a honest signer that finished round 5 with $\mathsf{ctnt}_{\mathbf{w}}$ but user i is between round 4 and round 5 with $\mathsf{ctnt}_{\mathbf{w}}$ due to Lemma E.8. For each such $\mathsf{ctnt}_{\mathbf{w}}$, the challenger samples $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$ honestly, stores $\mathbf{\Delta}_i$ in $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and removes i from $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$.

This change is purely conceputal to ease the introduction of future hybrids. Hence, we have

$$\epsilon_{12} = \epsilon_{11}.$$

Before we proceed, let us show a useful lemma.

Lemma E.13. *All invocations of* $\mathcal{O}_{\mathsf{Sign}_5}$ *with* $\mathsf{ctnt}_{\mathbf{w}}$ *share the identical value* $\mathsf{ctnt}_{\mathbf{z}}$.

Proof. Let us inspect the first call to $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}} = 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$. Here, the challenger sets $\mathsf{ctnt}_{\mathbf{z}} = 1\|\mathsf{SS}||\mathsf{M}||(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}\|(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$. Let us set $\mathsf{M}_{\mathsf{S}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$. Due to Lemma E.8, if $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked with M_{S}, we know that $\mathcal{O}_{\mathsf{Sign}_4}$ was invoked for all users $j \in \mathsf{sHS}$ with M_{S}. Notably, M_{S} determines $\mathsf{ctnt}_{\mathbf{w}}$ and thus, the values $\mathsf{ctnt}_{\mathbf{w}}$ and M_{S} are identical across each of the aforementioned

Game_{12}

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 53 in Game_{11}

54 : $\textbf{for } \mathsf{ctnt}_{\mathbf{w}} \textit{ s.t. } [\![i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!]$

// $\exists$honest user finished Round 5 with $\mathsf{ctnt}_{\mathbf{w}}$

// all honest users completed Sign_4

55 : $\quad \mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

56 : $\quad \mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$

57 : $\quad \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$

// state of user i between round 4 and 5

58 : $\textbf{for } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i) \in \mathsf{st}_i \textbf{ do}$

59 : $\quad \mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

60 : $\quad (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

61 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\} \textbf{ do}$

62 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

63 : $\textbf{for } \mathsf{ctnt}_{\mathbf{w}} \text{ s.t. } [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!] \textbf{ do}$

64 : $\quad \mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

65 : $\mathsf{HS} \leftarrow \mathsf{HS} \backslash \{i\}$

66 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

67 : $\textbf{return } (\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

1 : $\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

2 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

3 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$

4 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

5 : $\textbf{parse } (\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$

6 : $\textbf{parse } (\widetilde{\mathbf{w}}_j)_{j \in \mathsf{SS} \backslash \{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS} \backslash \{i\}}$

7 : $\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i \text{ with } \mathsf{pm}_{4,i} = \widetilde{\mathbf{w}}_i$

8 : $\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)]\!]$

9 : $\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\widetilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$

10 : $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$

11 : $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) \quad // \ c \in \mathcal{C}$

12 : $\textbf{if } [\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!] \textbf{ then}$

13 : $\quad \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

14 : $\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

15 : $\quad \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

16 : $\mathbf{\Delta}_i := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}}) \in \mathcal{R}_q^{\ell}$

17 : $\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}_q^{\ell}$

18 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$

19 : $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

20 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \backslash \{i\}$

21 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

22 : $\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

23 : $\textbf{return } \mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 36: The twelfth game. The differences are highlighted in blue. We assume that this game initializes four empty lists $\mathsf{InitializeSign}[\cdot], \mathsf{UnSignedHS}[\cdot], \mathsf{Mask}_{\mathbf{z}}[\cdot], \mathsf{MaskedResp}[\cdot] := \perp$ at the beginning of the game.

$\mathcal{O}_{\mathsf{Sign}_4}$ invocations. Also, due to Lemma E.8, there are no further calls to $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$. In summary, all states st_j of users $j \in \mathsf{sHS}$ between round 4 and 5 with $\mathsf{ctnt}_{\mathbf{w}}$ share the same value M_{S}. Thus, if $\mathcal{O}_{\mathsf{Sign}_5}$ is invoked with $\mathsf{ctnt}_{\mathbf{w}}$, then M_{S} is identical. Since for each cmt_j for $j \in \mathsf{SS}$, there is exactly one $\mathsf{H}_{\mathsf{com}}$ preimage $(j, \widetilde{\mathbf{w}}_j)$, the value $\mathsf{ctnt}_{\mathbf{z}}$ is determined by M_{S}. The statement follows. ◻

Game_{13}:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$ and $\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

// Replace any invocation of $\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$ with the following:

for $j \in \mathsf{sCS}$ **do**

  $\mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

  $\mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

for $j \in \mathsf{sHS} \backslash \{i\}$ **do**

  $\mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

  $\mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

$\mathbf{\Delta}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j}) \in \mathcal{R}_q^{\ell}$

Figure 37: The thirteenth game.

Game_{13}**:** In this game, we expand the definition of $\mathsf{ZeroShare}$ for every invocation of $\mathsf{ZeroShare}(\vec{\mathsf{seed}}_i[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$. This is depicted in Fig. 37. Both games are identical and we have

$$\epsilon_{13} = \epsilon_{12}.$$

Game_{14}**:** In this game, the challenger modifies how masks $\mathbf{\Delta}_i$ are sampled. This is depcicted in Fig. 38. That is, whenever a mask $\mathbf{\Delta}_i$ is computed in $\mathcal{O}_{\mathsf{Corrupt}}$ and $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger first computes $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ for $j \in \mathsf{sCS}$ as before. It sets $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ which represents the honest signers which have not executed round 5 with $\mathsf{ctnt}_{\mathbf{w}}$. Then, for $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}}$ (*i.e.*, honest users after round 5), it retrieves $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ from $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{w}}]$, respectively. For $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}} \backslash \{i\}$, it picks $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ uniformly at random from $\mathcal{R}_q^{\ell}$ and stores them in $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$, respectively. Finally, it sets $\mathbf{\Delta}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$ as before.

Let us show that both games are identically distributed. We have to show that if $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}}$, then $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$ are already initialized. We know that all signers $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}}$ have already executed $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$ due to Lemma E.13. Thus, the oracles were initialized correctly in the corresponding $\mathcal{O}_{\mathsf{Sign}_5}$ invocation for $j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}}$. Also, we have to show that if $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}} \backslash \{i\}$, then both $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}] = \perp$ and $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}] = \perp$ are not yet defined. This follow readily from the following two facts: (1) due to the abort condition in $\mathsf{H}_{\mathsf{mask}}$, the adversary $\mathcal{A}$ never queries $\mathsf{H}_{\mathsf{mask}}$ on honest seeds directly, and (2) $\mathcal{O}_{\mathsf{Sign}_5}$ was never invoked for $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}} \backslash \{i\}$ with $\mathsf{ctnt}_{\mathbf{w}}$. In total, we have

$$\epsilon_{14} = \epsilon_{13}.$$

Game_{14}:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$ and $\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

// Replace the modification made in Game_{13} with the following:

$\textbf{for } j \in \mathsf{sCS} \textbf{ do}$

$\quad \mathbf{m}_{i,j} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}})$

$\quad \mathbf{m}_{j,i} := \mathsf{H}_{\mathsf{mask}}(\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}})$

$\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

$\textbf{for } j \in \mathsf{sHS} \backslash \widetilde{\mathsf{sHS}}_{\mathbf{z}} \textbf{ do}$

$\quad \mathbf{m}_{i,j} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}]$

$\quad \mathbf{m}_{j,i} \leftarrow \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}]$

$\textbf{for } j \in \widetilde{\mathsf{sHS}}_{\mathbf{w}} \backslash \{i\} \textbf{ do}$

$\quad \mathbf{m}_{i,j} \xleftarrow{\$} \mathcal{R}_q^{\ell}, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{i,j}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{i,j}$

$\quad \mathbf{m}_{j,i} \xleftarrow{\$} \mathcal{R}_q^{\ell}, \mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}[\mathsf{seed}_{j,i}, \mathsf{ctnt}_{\mathbf{z}}] \leftarrow \mathbf{m}_{j,i}$

$\mathbf{\Delta}_i := \sum_{j \in \mathsf{SS} \backslash \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j}) \in \mathcal{R}_q^{\ell}$

Figure 38: The fourteenth game. The differences are highlighted in blue.

Game_{15} : In this game, the challenger samples the masks $\mathbf{\Delta}_i$ without $\mathsf{H}_{\mathsf{mask}}$. The last mask is set consistently and the others are sampled at random. Also, when a user is corrupted, it programs the oracle $\mathsf{H}_{\mathsf{mask}}$ in accordance. This is depicted in Fig. 39. In more detail, whenever the challenger computes $\mathbf{\Delta}_i$ and $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Sign}_5}$ and $\mathcal{O}_{\mathsf{Corrupt}}$, then the user checks if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$, where $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ is the set of honest users that are still before round 5 with $\mathsf{ctnt}_{\mathbf{w}}$. If so, it samples $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$ at random. Otherwise, i is the last user, so the challenger computes $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$ for $j \in \mathsf{sCS}$ and sets $\mathbf{\Delta}_i := -\sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$. As before, all masks $\mathbf{\Delta}_i$ are stored in the table $\mathsf{Mask}_{\mathbf{z}}$. Note that now, the user never invokes $\mathsf{H}_{\mathsf{mask}}$ to compute $\mathbf{\Delta}_j$ for $j \in \mathsf{HS}$. Instead, at the end of $\mathcal{O}_{\mathsf{Corrupt}}$, the user programs the oracle $\mathsf{H}_{\mathsf{mask}}$ consistently for corrupted user i via $\mathsf{ProgramZeroShare}$ given in Fig. 32 as in Game_9.

We show that Game_{15} and Game_{14} are identically distributed. As in Game_9, the (potential) observable differences between both games are the distribution of the masks $\mathbf{\Delta}_i$ and the output of $\mathsf{H}_{\mathsf{mask}}$. The statement follows almost in Lemmata E.9 and E.11.

First, observe that in both games, we have that $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot$ in Game_{16} iff $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot$ in Game_{15}. We can argue as in the proof of Lemma E.11 that thanks to $\mathsf{ProgramZeroShare}$, the distribuion of $\mathsf{H}_{\mathsf{mask}}$ is identical in the view of $\mathcal{A}$ if $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{z}}]$ is distributed identically in Game_{14} and Game_{15}. (Note that apart from the dimension of the mask, $\mathsf{ProgramZeroShare}$ behaves identically in both cases, and that Lemma E.13 ensures that $\mathsf{ctnt}_{\mathbf{z}}$ is correctly set.) So it remains to show that $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is distributed identically in both games if $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot$. Observe that all values stored in $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ are computed as depicted in Fig. 38 in Game_{14}. If there exists some $j \in \widetilde{\mathsf{sHS}}_{\mathbf{z}} \backslash \{i\}$, then $\mathbf{m}_{i,j}$ and $\mathbf{m}_{j,i}$ are sampled at random over $\mathcal{R}_q^{\ell}$. Thus, $\mathbf{\Delta}_i = \sum_{j \in \mathsf{SS} \backslash \{i\}} (\mathbf{m}_{j,i} - \mathbf{m}_{i,j})$ is distributed at random over $\mathcal{R}_q^{\ell}$. Otherwise, we have that $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$ and all individual masks $(\mathbf{m}_{i,j}, \mathbf{m}_{j,i})_{j \in \mathsf{sHS}}$ are retrieved from $\mathsf{Q}_{\mathsf{H}_{\mathsf{mask}}}$. In that case, $\mathbf{\Delta}_i$ is fully determined. A simple calculation (cf. Lemma E.9 for details) shows that indeed, $\mathbf{\Delta}_i$ is identically distributed in this case, too. In conclusion, we have that

Game_{15}

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 53 in Game_{11}

54 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ *s.t.* $[\![i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot]\!]$

// $\exists$honest user finished Round 5 with $\mathsf{ctnt}_{\mathbf{w}}$

// all honest users completed Sign_4

55 : $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \bot$

56 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

57 : $\boldsymbol{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$

58 : **else** // user i is the last user for $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

59 : $\mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

60 : **for** $j \in \mathsf{sCS}$

61 : $\boldsymbol{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

62 : $\boldsymbol{\Delta}_i := -\sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j$

63 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \boldsymbol{\Delta}_i$

// state of user i between round 4 and 5

64 : **for** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i) \in \mathsf{st}_i$ **do**

65 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

66 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

67 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i)\}$ **do**

68 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

69 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot]\!]$ **do**

70 : $\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

71 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot]\!]$ **do**

72 : $\mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

73 : $\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{z}}, i, \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

74 : $\mathsf{HS} \leftarrow \mathsf{HS} \setminus \{i\}$

75 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

76 : **return** $(\mathsf{sk}_i, \mathsf{st}_i)$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

1 : **req** $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

2 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

3 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$

4 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

5 : **parse** $(\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$

6 : **parse** $(\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS} \setminus \{i\}}$

7 : **pick** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{4,i} = \tilde{\mathbf{w}}_i$

8 : **req** $[\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \tilde{\mathbf{w}}_j)]\!]$

9 : $\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\tilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$

10 : $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \tilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$

11 : $c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ // $c \in \mathcal{C}$

12 : **if** $[\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!]$ **then**

13 : $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

14 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

15 : $\mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

16 : $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

17 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

18 : $\boldsymbol{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$

19 : **else**

20 : **for** $j \in \mathsf{sCS}$

21 : $\boldsymbol{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

22 : $\boldsymbol{\Delta}_i := -\sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j$

23 : $\tilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \boldsymbol{\Delta}_i \in \mathcal{R}_q^{\ell}$

24 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \boldsymbol{\Delta}_i$

25 : $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \tilde{\mathbf{z}}_i$

26 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

27 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \tilde{\mathbf{w}}_i)\}$

28 : $\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

29 : **return** $\mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 39: The fifteenth game. The differences are highlighted in blue.

$$\epsilon_{15} = \epsilon_{14}.$$

Game_{16} : In this game, the challenger checks whether all honest users uses the same challenge c in $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$. This is depicted in Fig. 40. Specifically, in $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger additionally stores $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ in $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \bot$, i.e., user i is the first user in the fifth round. Also, it checks if $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c$. If so, it continues the game as before. Otherwise, it aborts the game.

Let us show that Game_{15} and Game_{16} are identically distributed. To show this, we show the following lemma.

Lemma E.14. *Let* $\mathsf{ctnt}_{\mathbf{w}}$ *be arbitrary. In* $\mathcal{O}_{\mathsf{Sign}_5}$ *with* $\mathsf{ctnt}_{\mathbf{w}}$*, all honest users in* sHS *use the same* $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$.

Proof. From Lemma E.13, the same $\mathsf{ctnt}_{\mathbf{z}}$ is used in $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$ for all uses in sHS. Recall that $\mathsf{ctnt}_{\mathbf{z}} = 1\|\mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}\|(\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$ and $\mathbf{w}$ is computed by $\left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}}$. Thus, $\mathbf{w}$ is uniquely determined by $\mathsf{ctnt}_{\mathbf{z}}$. Also, since $\mathsf{ctnt}_{\mathbf{z}}$ contains M and vk is fixed through the game, M and vk are also uniquely determined by $\mathsf{ctnt}_{\mathbf{z}}$. Therefore, all users in sHS compute the same $c = \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ in $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$. □

By the above lemma, the game never aborts due to the added abort conditions. Thus, we have

$$\epsilon_{16} = \epsilon_{15}.$$

Game_{17} : In this game, the challenger samples $\widetilde{\mathbf{z}}_i$ directly either at random or consistently for the last user in $\mathcal{O}_{\mathsf{Sign}_5}$. Also, the challenger delays attributing commmitments from $\mathsf{UnUsedCom}$ until a user is corrupted via $\mathcal{O}_{\mathsf{Corrupt}}$. This is depicted in Figs. 41 and 42. We describe the changes in more detail. In $\mathcal{O}_{\mathsf{Sign}_5}$, the challenger does not setup $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ via $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ yet. Also, instead of sampling $\mathbf{\Delta}_i$, it samples $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ at random if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$ and otherwise, it sets $\widetilde{\mathbf{z}}_i = c \cdot \mathbf{s} - c\sum_{j\in\mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$. The value $\widetilde{\mathbf{z}}_i$ is stored in $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ but $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ remains $\bot$. In $\mathcal{O}_{\mathsf{Corrupt}}$, when passing over $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot$, the challenger checks if $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot$ and $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \varnothing$. If so, it chooses some commitment $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$, stores it in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$, and removes $\mathbf{w}_i$ and $\mathbf{r}_i$ from SumCom and $\mathsf{SumComRnd}$, respectively. Note that $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot$ at that point since it no longer is set in $\mathcal{O}_{\mathsf{Sign}_5}$. Further, when passing over $\mathsf{ctnt}_{\mathbf{w}}$ such that $i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot$, the challenger retrieves $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and $c \leftarrow \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$. Note that $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$ is already defined since there is at least one user who finished $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$. If $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \bot$, then it sets $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}] - c \cdot \mathbf{s}_i - \mathbf{r}_i$. If else $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] = \bot$, the challenger samples $\mathbf{\Delta}_i$ at random if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}$ as before, but if $\widetilde{\mathsf{sHS}}_{\mathbf{z}} = \{i\}$, it samples $\mathbf{\Delta}_i = c \cdot \mathbf{s} - c\sum_{j\in\mathsf{sCS}\cup\{i\}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{r}_i - \sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$.

Let us show that both games are identically distributed. Let us first show a useful lemma.

Lemma E.15. *Let* $\mathsf{ctnt}_{\mathbf{w}}$ *be arbitrary. In* Game_{17}*, we have in line 22 in* $\mathcal{O}_{\mathsf{Sign}_5}$ *(resp. line 65 in* $\mathcal{O}_{\mathsf{Corrupt}}$*) that* $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{(\mathbf{w},\mathbf{r},\mathbf{e}')\in\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]} \mathbf{r}$.

Proof. When $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is initialized, this holds by construction. Further, whenever a commitment is removed from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, the invariant is retained. □

$\underline{\mathsf{Game}_{16}}$

$\underline{\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j\in\mathsf{SS}})}$

$\textbf{req}\ [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j\in\mathsf{SS}}$

$(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$

$\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

$\textbf{parse}\ \left(\mathbf{s}_i, \vec{\mathsf{seed}}_i\right) \leftarrow \mathsf{sk}_i$

$\textbf{parse}\ (\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j\in\mathsf{SS}\setminus\{i\}}$

$\textbf{pick}\ (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\ \textbf{from}\ \mathsf{st}_i$ with $\mathsf{pm}_{4,i} = \widetilde{\mathbf{w}}_i$

$\textbf{req}\ [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)]\!]$

$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}} \| (\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$

$\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$

$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})\quad /\!/\ c \in \mathcal{C}$

$\textbf{if}\ [\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!]\ \textbf{then}$

$\quad \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

$\quad \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

$\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

$\quad \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

$\textbf{req}\ [\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c]\!]$

$\textbf{if}\ [\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]\ \textbf{then}$

$\quad \mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}^\ell_q$

$\textbf{else}$

$\quad \textbf{for}\ j \in \mathsf{sCS}$

$\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

$\quad \mathbf{\Delta}_i := -\sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$

$\widetilde{\mathbf{z}}_i := c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \in \mathcal{R}^\ell_q$

$\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$

$\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

$\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

$\textbf{return}\ \mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 40: The sixteenth game. The differences are highlighted in blue. We assume that this game initializes a empty lists $\mathsf{Chall}[\cdot] := \bot$ at the beginning of the game.

Game_{17} – part 1:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 10 in Game_3
11 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![i \in \mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!]$ **do**
// $\exists$honest signer finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
// Sign_3 for user i is completed
12 : **parse** $0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}} \leftarrow \mathsf{ctnt}_{\mathbf{w}}$ // $i \in \mathsf{SS}$
13 : **if** $[\![\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **then**
// user i finished Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
14 : **if** $[\![\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] = \varnothing]\!]$ **then**
15 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$
16 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
17 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \mathbf{r}_i$
18 : **else** // There is still a honest user in Round 3 with $\mathsf{ctnt}_{\mathbf{w}}$
19 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$
20 : $\mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}^k_q$
21 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$
22 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}] - \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
23 : **else** // user i is between Round 3 and Round 4 with $\mathsf{ctnt}_{\mathbf{w}}$
24 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$
25 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$
26 : $\widetilde{\mathbf{\Delta}}_i \xleftarrow{\$} \mathcal{R}^k_q$
27 : **elseif** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!]$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$
28 : **for** $j \in [|\mathsf{sHS}| - 1]$ **do**
29 : $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$
30 : $\mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}^k_q$
31 : $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$
32 : $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$
33 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$
34 : **for** $j \in \mathsf{sCS}$
35 : $\widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$
36 : $\widetilde{\mathbf{\Delta}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$
37 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{\Delta}}_i$
38 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]\setminus\{i\}$

// Continuation of $\mathcal{O}_{\mathsf{Corrupt}}$
// Identical to Lines 41 to 53 in Game_{11}
52 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ *s.t.* $[\![i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!]$
// $\exists$honest user finished Round 5 with $\mathsf{ctnt}_{\mathbf{w}}$
// all honest users completed Sign_4
// $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$ due to line 15 of $\mathcal{O}_{\mathsf{Corrupt}}$
53 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
54 : $c \leftarrow \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$
55 : **if** $[\![\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ // user i completed Sign_5
56 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] - c \cdot \mathbf{s}_i - \mathbf{r}_i$
57 : **else** // user i is between round 4 and round 5
58 : $\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$
59 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**
60 : $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}^{\ell}_q$
61 : **else** // user i is the last user for $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$
62 : $\mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$
63 : **for** $j \in \mathsf{sCS}$
64 : $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$
65 : $\mathbf{\Delta}_i := c \cdot \mathbf{s} - c \sum_{j\in\mathsf{sCS}\cup\{i\}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$
66 : $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathbf{\Delta}_i$
// state of user i between round 4 and 5
67 : **for** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i) \in \mathsf{st}_i$ **do**
68 : $\mathsf{ctnt}_{\mathbf{w}} := 0\|\mathsf{SS}\|(\mathsf{str}_j)_{j\in\mathsf{SS}}$
69 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$
70 : $\mathsf{st}_i \leftarrow \mathsf{st}_i\setminus\{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$ **do**
71 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$
72 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **do**
73 : $\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$
74 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **do**
75 : $\mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$
76 : $\mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{z}}, i, \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$
77 : $\mathsf{HS} \leftarrow \mathsf{HS}\setminus\{i\}$
78 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$
79 : **return** $(\mathsf{sk}_i, \mathsf{st}_i)$

Figure 41: The first part of the seventeenth. The differences are highlighted in blue.

Game_{17} – part 2:

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j\in\mathsf{SS}})$

$\textbf{req } [\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j\in\mathsf{SS}}$

$\textbf{parse } \left(\mathbf{s}_i, \vec{\mathsf{seed}}_i\right) \leftarrow \mathsf{sk}_i$

$\textbf{parse } (\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}\setminus\{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j\in\mathsf{SS}\setminus\{i\}}$

$\textbf{pick } (\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i) \textbf{ from } \mathsf{st}_i$

$\quad\quad\text{with } \mathsf{pm}_{4,i} = \widetilde{\mathbf{w}}_i$

$\textbf{req } [\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)]\!]$

$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}} \| (\widetilde{\mathbf{w}}_j)_{j\in\mathsf{SS}}$

$$\mathbf{w} := \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$$

$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w}) \quad /\!\!/ \ c \in \mathcal{C}$

$\textbf{if } [\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \bot]\!] \textbf{ then}$

$\quad \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

$\quad \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

$\quad \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

$\textbf{req } [\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c]\!]$

$\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

$\textbf{if } [\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!] \textbf{ then}$

$\quad \widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}^\ell_q$

$\textbf{else}$

$\quad \textbf{for } j \in \mathsf{sCS}$

$\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

$$\widetilde{\mathbf{z}}_i := c \cdot \mathbf{s} - c \sum_{j\in\mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sHS}\setminus\{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j\in\mathsf{sCS}} \mathbf{\Delta}_j$$

$\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

$\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

$\textbf{return } \mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 42: The second part of the seventeenth game. The differences are highlighted in blue.

Next, let us consider an intermedite game of $\mathsf{Game}_{16,*}$, where instead of sampling $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}_q^\ell$ at random in $\mathcal{O}_{\mathsf{Sign}_5}$, we sample $\mathbf{\Delta}_i^* \xleftarrow{\$} \mathcal{R}_q^\ell$ and set $\mathbf{\Delta}_i := \mathbf{\Delta}_i^* - (c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i)$. This game is identically distributed to Game_{16}. Then, we have in $\mathcal{O}_{\mathsf{Sign}_5}$ that

$$\begin{aligned}\widetilde{\mathbf{z}}_i &= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \mathbf{\Delta}_i \\ &= \mathbf{\Delta}_i^* \sim \mathcal{U}_{\mathcal{R}_q^\ell}\end{aligned}$$

which is distributed as in Game_{17}. Similarly, if $\widetilde{\mathsf{sHS}_{\mathbf{z}}} = \{i\}$ we have that

$$\begin{aligned}\widetilde{\mathbf{z}}_i &= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i + \widetilde{\mathbf{\Delta}}_i \\ &= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i - \sum_{j \in \mathsf{sHS} \setminus \{i\}} (\mathbf{\Delta}_j^* - (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j)) - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= \sum_{j \in \mathsf{sHS}} (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j) - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \widetilde{\mathbf{z}}_i - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{CS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \sum_{j \in \mathsf{sHS}} \mathbf{r}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j\end{aligned}$$

where the third equation follows from Lemma E.14, and the last equation follows from the correctness of the Shamir secret sharing. Similarly, we have in $\mathcal{O}_{\mathsf{Corrupt}}$ if $\widetilde{\mathsf{sHS}_{\mathbf{w}}} = \{i\}$ that

$$\begin{aligned}\mathbf{\Delta}_i &= - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= - \sum_{j \in \mathsf{sHS} \setminus \{i\}} (\mathbf{\Delta}_j^* - (c \cdot L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathbf{r}_j)) - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} (\widetilde{\mathbf{z}} + \mathbf{r}_j) - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j \\ &= c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathbf{r}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j\end{aligned}$$

Note that $\sum_{(\mathbf{w},\mathbf{r},\mathbf{e}') \in \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]} \mathbf{r}$ is equal to the sum of commitment randomness in the above equations by design. Due to Lemma E.15, we have that $\widetilde{\mathbf{z}}_i$ and $\mathbf{\Delta}_i$ are identically distributed in $\mathsf{Game}_{16,*}$ and Game_{17}. Since $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ are no longer initialized in $\mathcal{O}_{\mathsf{Sign}_5}$ in Game_{17}, it remains to show that their value is identically distributed when accessed in $\mathsf{Game}_{16,*}$ and Game_{17}. In Game_{17}, if a user after round 5 for $\mathsf{ctnt}_{\mathbf{w}}$ is corrupted, the challenger chooses a commitment from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$, stores it in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ and adapts $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ accordingy. In $\mathsf{Game}_{16,*}$, the commitment $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is also chosen from $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ in $\mathcal{O}_{\mathsf{Sign}_5}$ (and never accessed before $\mathcal{O}_{\mathsf{Corrupt}}$), whereas $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$ is adapted in $\mathcal{O}_{\mathsf{Corrupt}}$. Thus, $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is identically distributed when accessed. In $\mathsf{Game}_{16,*}$, the value $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \mathbf{\Delta}_i^* - (c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i + \mathbf{r}_i)$ is initialized in $\mathcal{O}_{\mathsf{Sign}_5}$. In Game_{17}, the challenger sets this value via $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ when passing over $\mathsf{ctnt}_{\mathbf{w}}$ with $i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]$ and $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp$. That is, if $\mathcal{O}_{\mathsf{Sign}_5}$ was executed for user i with $\mathsf{ctnt}_{\mathbf{w}}$, *i.e.*, if $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$, it sets $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] - c \cdot L_{\mathsf{SS},i} \cdot \mathbf{s}_i - \mathbf{r}_i$. Here, $\mathbf{r}_i$ is retrieved from $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ (which is identically distributed due to the argument above) and c is retrieved from $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$. Because $\mathsf{ctnt}_{\mathbf{z}}$ determines c uniquely and because there is a unique $\mathsf{ctnt}_{\mathbf{z}}$ for each $\mathsf{ctnt}_{\mathbf{w}}$ in $\mathcal{O}_{\mathsf{Sign}_5}$ (cf. Lemma E.13), we know that c is identical to the challenge of the

$\mathcal{O}_{\mathsf{Sign}_5}$ invocation when $\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i]$ was set. Thus, $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is identically distributed in both games at that point. Note that beforehand, the $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is not accessed in both games. In total, whenever $\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i]$ or $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ is accessed by the challenger, the values are identically distributed in Game_{17} and $\mathsf{Game}_{16,*}$. We conclude that

$$\epsilon_{17} = \epsilon_{16}.$$

Game_{18}**:** In this game, the challenger precomputes the challenge c for $\mathcal{O}_{\mathsf{Sign}_5}$ when the last signer passes round 4. This is depcicted in Fig. 43. In more detail, in $\mathcal{O}_{\mathsf{Sign}_4}$, the challenger stores $\mathsf{SimContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{M}_{\mathsf{S}}$ if $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \bot$. Further, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, then it samples a challenge $c \xleftarrow{\$} \mathcal{C}$ and programs H_c via a helper function $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ (cf. Fig. 44). In $\mathsf{ProgramHashChall}$, the challenger retrieves $\mathsf{M}_{\mathsf{S}} \leftarrow \mathsf{SimContent}[\mathsf{ctnt}_{\mathbf{w}}]$, where $\mathsf{M}_{\mathsf{S}} = \mathsf{SS}\|\mathsf{M}\|(\mathsf{str}_j, \mathsf{cmt}_j)_{j\in\mathsf{SS}}$, and checks if for each cmt_j for $j \in \mathsf{SS}\backslash\{i\}$, there is a (unique) value $\widetilde{\mathbf{w}}_j$ such that $\mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j) = \mathsf{cmt}_j$. If so, it sets $\mathbf{w} = \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}}$ and sets $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$ (but aborts if this value was previously set). Otherwise, it sets $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \top$. Similarly, in $\mathcal{O}_{\mathsf{Corrupt}}$, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, then the challenger samples a challenge $c \xleftarrow{\$} \mathcal{C}$ and programs H_c via $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$. Note that here, $\widetilde{\mathbf{w}}_i$ is setup at that point, instead of when passing over states between round 3 and round 4. Finally, in $\mathcal{O}_{\mathsf{Sign}_5}$, it aborts the game if $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$. Moreover, instead of setting $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \bot$, the challenger always checks if $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = (\mathsf{M}, c, \mathbf{w})$ and aborts if not.

Observe that both games are identically distributed conditioned on the game not aborting. In $\mathcal{O}_{\mathsf{Corrupt}}$ in Game_{17}, the table $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is initialized with a freshly sampled commitment $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$ when passing over the states of user i between round 3 and round 4. In $\mathcal{O}_{\mathsf{Corrupt}}$ in Game_{18}, the value is identically sampled and stored in $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$ in line 25. Later, when passing over states between round 3 and round 4, the value is retrieved from $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$. None of the other changes in Game_{18} impact the view of $\mathcal{A}$ (since $c \xleftarrow{\$} \mathcal{C}$ remains uniform in Game_{18}), and thus both games are identically distributed conditioned on the game not aborting. It remains to bound the abort probability in Game_{18}. The challenger aborts the if (1) $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}]$ is already defined in $\mathsf{ProgramHashChall}$ or (2) $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ in $\mathcal{O}_{\mathsf{Sign}_5}$.

We first bound the probability of event (1). Observe that $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ is invoked with $\mathbf{w} = \left\lfloor \sum_{j\in\mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$ after $\widetilde{\mathbf{w}}_i$ is sampled for the last user in $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$ and before returning it to $\mathcal{A}$. The commitment $\widetilde{\mathbf{w}}_i$ is either set in $\mathcal{O}_{\mathsf{Corrupt}}$ to $\widetilde{\mathbf{w}}_i = \mathbf{w}_i + \widetilde{\mathbf{\Delta}}_i$ or to $\widetilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j\in\mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j\in\mathsf{sHS}\backslash\{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$ in $\mathcal{O}_{\mathsf{Sign}_4}$. In both cases, $\widetilde{\mathbf{w}}_i$ is computed via at least one freshly sampled commitment (either $\mathbf{w}_i$ or within $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$). Thus, due to Lemma 2.8, $\mathbf{w}$ has min-entropy $n-1$ with overwhelming probability. Since $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ is invoked when the last user with $\mathsf{ctnt}_{\mathbf{w}}$ passes round 4 or is corrupted, the probability of event (1) is at most $Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_c} + Q_{\mathsf{S}})/2^{n-1}$.

Next, we bound the probability of event (2). Since cmt_j for each honest user has at most one preimage $(j, \widetilde{\mathbf{w}}_j)$ due to previous modifications, we have $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top$ only if $\mathcal{A}$ some cmt_j does not have a $\mathsf{H}_{\mathsf{com}}$ preimage of the form $(j, \widetilde{\mathbf{w}}_j)$ when $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$ is invoked (where cmt_j is determined by $\mathsf{ctnt}_{\mathbf{w}}$), but the $\mathcal{A}$ provides a valid preimage of cmt_j in $\mathcal{O}_{\mathsf{Sign}_5}$. Since the image cmt of $\mathsf{H}_{\mathsf{com}}$ is sampled uniformly at random from $\{0,1\}^{2\lambda}$ each $\mathsf{H}_{\mathsf{com}}$ query, the probability that $\mathcal{A}$ finds a valid preimage for cmt_j is at most $1/2^{2\lambda}$ per query. Thus, the probability of event (2) is at most $Q_{\mathsf{H}_{\mathsf{com}}}/2^{2\lambda}$. In conclusion, we have

$$|\epsilon_{18} - \epsilon_{17}| \leqslant \frac{Q_{\mathsf{S}} \cdot (Q_{\mathsf{H}_c} + Q_{\mathsf{S}})}{2^{n-1}} + \frac{Q_{\mathsf{H}_{\mathsf{com}}}}{2^{2\lambda}} + \mathsf{negl}(\lambda).$$

Game_{19}**:** In this game, the challenger simulates one of the sampled commitments. This is depicted in Figs. 45 and 46. In $\mathcal{O}_{\mathsf{Sign}_4}$, if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, it samples c as before, and then samples $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^k_{\mathbf{w}}$, sets

Game_{18}:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 23 in Game_{17}

// Recall that user i is between Round 3 and 4 with $\mathsf{ctnt}_{\mathbf{w}}$

24 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$

// Prepare commitment for line 46

25 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}};\ \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}^k_q$

26 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

27 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$

28 : $\widetilde{\boldsymbol{\Delta}}_i \xleftarrow{\$} \mathcal{R}^k_q$

29 : **elseif** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!]$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

30 : $c \xleftarrow{\$} \mathcal{C}$

31 : **for** $j \in [|\mathsf{sHS}| - 1]$ **do**

32 : $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

33 : $\mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}^k_q$

34 : $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$

35 : $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$

36 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$

37 : **for** $j \in \mathsf{sCS}$

38 : $\widetilde{\boldsymbol{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

39 : $\widetilde{\boldsymbol{\Delta}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

40 : $\widetilde{\mathbf{w}}_i \leftarrow \mathbf{w}_i + \widetilde{\boldsymbol{\Delta}}_i$

41 : $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$

42 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\boldsymbol{\Delta}}_i$

43 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

// state of user i between round 3 and 4

44 : **for** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i$ **do**

45 : $\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

46 : $(\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

47 : **if** $[\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$

48 : $\widetilde{\boldsymbol{\Delta}}_i \xleftarrow{\$} \mathcal{R}^k_q$

49 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\boldsymbol{\Delta}}_i$

50 : $\boldsymbol{\Delta}_i \leftarrow \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i]$

51 : $\widetilde{\mathbf{w}}_i := \mathbf{w}_i + \widetilde{\boldsymbol{\Delta}}_i \in \mathcal{R}^k_q$

52 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

53 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

54 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

55 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

// Identical to lines 52 to 79 in Game_{17}

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 7 in Game_{10}

8 : **if** $[\![\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$ **then**

9 : $\mathsf{InitializeOpen}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

10 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

11 : $\mathsf{SimContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{M}_{\mathsf{S}}$

12 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

13 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

14 : $\widetilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}^k_q$

15 : **else** // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

16 : $c \xleftarrow{\$} \mathcal{C}$

17 : **for** $j \in [|\mathsf{sHS}|]$ **do**

18 : $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

19 : $\mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}^k_q$

20 : $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$

21 : $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$

22 : $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$

23 : **for** $j \in \mathsf{sCS}$

24 : $\widetilde{\boldsymbol{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

25 : $\widetilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

26 : $\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$

27 : $\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

28 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

29 : $\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

30 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

31 : $\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

32 : **return** $\mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 9 in Game_{16}

10 : **if** $[\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$ **then**

11 : $\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

12 : $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

13 : $\mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

14 : $\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

15 : **req** $[\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c]\!]$

16 : **abort if** $[\![\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top]\!]$

17 : // Identical to Lines 16 to 27 in Game_{16}

Figure 43: The eighteenth game. The differences are highlighted in blue. We assume that this game initializes two empty lists $\mathsf{SimContent}[\cdot], \mathsf{BadCtnt}[\cdot] := \perp$ at the beginning of the game. The algorithm $\mathsf{ProgramHashChall}$ is defined in Fig. 44.

$\mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$:

$\mathsf{M}_\mathsf{S} \leftarrow \mathsf{SimContent}[\mathsf{ctnt}_{\mathbf{w}}]$
parse $\mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{M}_\mathsf{S}$
if $[\![\forall j \in \mathsf{SS} \backslash \{i\}, \exists! \widetilde{\mathbf{w}}_j, \mathsf{Q}_{\mathsf{H}_{\mathsf{com}}}(j, \widetilde{\mathbf{w}}_j) = \mathsf{cmt}_j]\!]$
  $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}^k_{q_{\nu_{\mathbf{w}}}}$
  abort if $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \neq \bot]\!]$
  $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$
else
  $\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] := \top$

Figure 44: A helper algorithm for programming the random oracle H_c for input $\mathbf{w}$ derived from $\mathsf{ctnt}_{\mathbf{w}}$ (and optionally $\widetilde{\mathbf{w}}_i$) to a given output c. Algorithm $\mathsf{ProgramHashChall}$ is assumed to have a joint state with the challenger and random oracle H_c used by the unforgeability game.

$\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}, \mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$ and simulates $\mathbf{w} = \mathbf{A} \cdot \mathbf{z} - c \cdot \hat{\mathbf{t}} + \mathbf{z}'$. The response $\mathbf{z}$ is stored in $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{z}$ and $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{w}$ is initialized with $\mathbf{w}$. Note that $\mathbf{r}$ is *not* added to $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}]$. Then, the challenger proceeds as before, except only $|\mathsf{sHS}| - 1$ commitments are generated instead of $|\mathsf{sHS}|$ many. In $\mathcal{O}_{\mathsf{Corrupt}}$, the first commitment is also simulated (as described above) if $\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}$, and only $|\mathsf{sHS}| - 2$ further commitments are generated (instead of $|\mathsf{sHS}| - 1$). Finally, the challenger generates $\mathbf{\Delta}_i$ (resp. $\widetilde{\mathbf{z}}_i$) in $\mathcal{O}_{\mathsf{Corrupt}}$ (resp. $\mathcal{O}_{\mathsf{Sign}_5}$) using $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}]$. Note that in Game_{18}, it $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in |\mathsf{sHS}|} \mathbf{r}_j$ contains the sum of the randomness $\mathbf{r}_j$ of all honest commitments (stored in $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$). In Game_{19}, the table $\mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] = \sum_{j \in |\mathsf{sHS}-1|} \mathbf{r}_j$ contains the sum the randomness $\mathbf{r}_j$ of the generated commitments except the randomness $\mathbf{r}$ of the simulated commitment. (Note that it is updated as in Game_{18}, so the invariant is kept.) In Game_{18}, we have

$$\widetilde{\mathbf{z}}_i = c \cdot \mathbf{s} - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j.$$

Due to the abort conditions in Game_{18} and Lemma E.8, c in the computation of $\mathbf{z}_i$ for the last user in $\mathcal{O}_{\mathsf{Sign}_5}$ with $\mathsf{ctnt}_{\mathbf{w}}$ of Game_{18} is the same to c that is defined via $\mathsf{ProgramHashChall}$ when $\mathcal{O}_{\mathsf{Sign}_4}$ with $\mathsf{ctnt}_{\mathbf{w}}$ or $\mathcal{O}_{\mathsf{Corrupt}}$. Thus, c in $\mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] = c \cdot \mathbf{s} + \mathbf{r}_i$ used to compute $\mathbf{z}_i$ for the last user in Game_{19} is identical to that in the computation of $\mathbf{z}_i$ in Game_{18}. Combining the above facts, we conclude that $\widetilde{\mathbf{z}}_i$ is identically distributed in both games. A similar argument yields that $\widetilde{\mathbf{\Delta}}_i$ is identically distributed in both games. We remark also that $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$—initialized via $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}].\mathsf{pop}(1)$—is identically distributed in both games because at least one user $j \in \mathsf{sHS}$ remains uncorrupted, so $\mathsf{pop}(1)$ is invoked at most $|\mathsf{sHS}| - 1$ (resp. $|\mathsf{sHS}| - 2$) times on $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ if $\mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is setup in $\mathcal{O}_{\mathsf{Sign}_4}$ (resp. $\mathcal{O}_{\mathsf{Corrupt}}$). It remains to argue that $\mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}]$ is identically distributed. This follows since in Game_{19}, the simulated commitment

$$\begin{aligned}
\mathbf{w} &= \mathbf{A} \cdot \mathbf{z} - c \cdot \hat{\mathbf{t}} + \mathbf{z}' \\
&= \mathbf{A} \cdot (c \cdot \mathbf{s} + \mathbf{r}) - c \cdot \hat{\mathbf{t}} + (c \cdot \mathbf{e} + \mathbf{e}') \\
&= c(\mathbf{A} \cdot \mathbf{s} + \mathbf{e}) + \mathbf{A} \cdot \mathbf{r} + \mathbf{e}' - c \cdot \hat{\mathbf{t}} \\
&= \mathbf{A} \cdot \mathbf{r} + \mathbf{e}'
\end{aligned}$$

Game_{19} – part 1:

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 23 in Game_{17}

// Recall that user i is between Round 3 and 4 with $\mathsf{ctnt}_{\mathbf{w}}$

24 : $\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $i \in \widetilde{\mathsf{sHS}}_{\mathbf{w}}$

// Prepare commitment for line 46

25 : $(\mathbf{r}_i, \mathbf{e}'_i) \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}};\ \mathbf{w}_i := \mathbf{A}\mathbf{r}_i + \mathbf{e}'_i \in \mathcal{R}^k_q$

26 : $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i)$

27 : **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$

28 : $\quad \widetilde{\boldsymbol{\Delta}}_i \xleftarrow{\$} \mathcal{R}^k_q$

29 : **elseif** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} = \{i\}]\!]$ // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

30 : $\quad c \xleftarrow{\$} \mathcal{C}$

31 : $\quad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

32 : $\quad \mathbf{z} := c \cdot \mathbf{s} + \mathbf{r};\ \mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$

33 : $\quad \mathbf{w} = \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} + \mathbf{z}'$

34 : $\quad \mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{z}$

35 : $\quad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{w}$

36 : $\quad$ **for** $j \in [|\mathsf{sHS}| - 2]$ **do**

37 : $\qquad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}^{\ell}_{\mathbf{w}} \times \mathcal{D}^{k}_{\mathbf{w}}$

38 : $\qquad \mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}^k_q$

39 : $\qquad \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$

40 : $\qquad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$

41 : $\qquad \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$

42 : $\quad$ **for** $j \in \mathsf{sCS}$

43 : $\qquad \widetilde{\boldsymbol{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

44 : $\qquad \widetilde{\boldsymbol{\Delta}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\boldsymbol{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$

45 : $\qquad \widetilde{\mathbf{w}}_i \leftarrow \mathbf{w}_i + \widetilde{\boldsymbol{\Delta}}_i$

46 : $\quad \mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$

47 : $\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\boldsymbol{\Delta}}_i$

48 : $\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

// Continuation of $\mathcal{O}_{\mathsf{Corrupt}}$

// Identical to Lines 44 to 55 in Game_{18}

61 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ *s.t.* $[\![i \in \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}]]\!] \wedge [\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] = \perp]\!]$

// $\exists$honest user finished Round 5 with $\mathsf{ctnt}_{\mathbf{w}}$

// all honest users completed Sign_4

// $\mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp$ due to Line 15 of $\mathcal{O}_{\mathsf{Corrupt}}$

62 : $\quad (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

63 : $\quad c \leftarrow \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}]$

64 : $\quad$ **if** $[\![\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ // user i completed Sign_5

65 : $\qquad \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] - c \cdot \mathbf{s}_i - \mathbf{r}_i$

66 : $\quad$ **else** // user i is between round 4 and round 5

67 : $\qquad \widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$ // $\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \neq \perp$

68 : $\qquad$ **if** $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

69 : $\qquad\quad \boldsymbol{\Delta}_i \xleftarrow{\$} \mathcal{R}^{\ell}_q$

70 : $\qquad$ **else** // user i is the last user for $\mathsf{ctnt}_{\mathbf{z}} = \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

71 : $\qquad\quad \mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

72 : $\qquad\quad$ **for** $j \in \mathsf{sCS}$

73 : $\qquad\qquad \boldsymbol{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

74 : $\qquad\quad \boldsymbol{\Delta}_i := \mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] - c \sum_{j \in \mathsf{sCS} \cup \{i\}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \boldsymbol{\Delta}_j$

75 : $\quad \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \boldsymbol{\Delta}_i$

// state of user i between round 4 and 5

76 : **for** $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i}) \in \mathsf{st}_i$ **do**

77 : $\quad \mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

78 : $\quad (\mathbf{w}_i, \mathbf{r}_i, \mathbf{e}'_i) \leftarrow \mathsf{UsedCom}[\mathsf{ctnt}_{\mathbf{w}}, i]$

79 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$ **do**

80 : $\quad \mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)\}$

81 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **do**

82 : $\quad \mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{w}}, i, \mathsf{Mask}_{\mathbf{w}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

83 : **for** $\mathsf{ctnt}_{\mathbf{w}}$ s.t. $[\![\mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i] \neq \perp]\!]$ **do**

84 : $\quad \mathsf{ctnt}_{\mathbf{z}} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}]$

85 : $\quad \mathsf{ProgramZeroShare}(\mathsf{ctnt}_{\mathbf{z}}, i, \mathsf{Mask}_{\mathbf{z}}[\mathsf{ctnt}_{\mathbf{w}}, i], \mathsf{sCS}, \mathsf{sHS})$

86 : $\mathsf{HS} \leftarrow \mathsf{HS} \setminus \{i\}$

87 : $\mathsf{CS} \leftarrow \mathsf{CS} \cup \{i\}$

88 : **return** $(\mathsf{sk}_i, \mathsf{st}_i)$

Figure 45: The first part of the nineteenth game. The differences are highlighted in blue. We assume that this game initializes a empty list $\mathsf{SimResp}[\cdot] := \perp$ at the beginning of the game.

Game_{19} – part 2:

$\mathcal{O}_{\mathsf{Sign}_4}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{3,j})_{j \in \mathsf{SS}})$

$\quad$ // Identical to Lines 1 to 11 in Game_{18}

$\widetilde{\mathsf{sHS}}_{\mathbf{w}} \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

if $[\![\widetilde{\mathsf{sHS}}_{\mathbf{w}} \neq \{i\}]\!]$ **then**

$\quad \widetilde{\mathbf{w}}_i \xleftarrow{\$} \mathcal{R}_q^k$

else // Last honest signer for $\mathsf{ctnt}_{\mathbf{w}}$

$\quad c \xleftarrow{\$} \mathcal{C}$

$\quad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\quad \mathbf{z} := c \cdot \mathbf{s} + \mathbf{r};\ \mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$

$\quad \mathbf{w} = \mathbf{A} \cdot \mathbf{z} - c \cdot \mathbf{t} + \mathbf{z}'$

$\quad \mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{z}$

$\quad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathbf{w}$

$\quad$ **for** $j \in [|\mathsf{sHS}| - 1]$ **do**

$\quad\quad (\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_{\mathbf{w}}^{\ell} \times \mathcal{D}_{\mathbf{w}}^{k}$

$\quad\quad \mathbf{w} := \mathbf{A}\mathbf{r} + \mathbf{e}' \in \mathcal{R}_q^k$

$\quad\quad \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnUsedCom}[\mathsf{ctnt}_{\mathbf{w}}] \cup (\mathbf{w}, \mathbf{r}, \mathbf{e}')$

$\quad\quad \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{w}$

$\quad\quad \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] + \mathbf{r}$

$\quad$ **for** $j \in \mathsf{sCS}$

$\quad\quad \widetilde{\mathbf{\Delta}}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{w}})$

$$\widetilde{\mathbf{w}}_i := \mathsf{SumCom}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sCS}} \widetilde{\mathbf{\Delta}}_j - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, j]$$

$\quad \mathsf{ProgramHashChall}(\mathsf{ctnt}_{\mathbf{w}}, c, \widetilde{\mathbf{w}}_i)$

$\mathsf{MaskedCom}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{w}}_i$

$\mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnOpenedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

$\mathsf{ProgramHashCom}(i, \mathsf{cmt}_i, \widetilde{\mathbf{w}}_i)$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \sigma_{\mathsf{S},i})\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \cup \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

return $\mathsf{pm}_{4,i} := \widetilde{\mathbf{w}}_i$

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

req $[\![i \in \mathsf{HS}]\!] \wedge [\![(\mathsf{SS}, \mathsf{M}, \cdot, \mathsf{pm}_{4,i}) \in \mathsf{st}_i]\!]$

$\mathsf{ctnt}_{\mathbf{w}} := 0 \| \mathsf{SS} \| (\mathsf{str}_j)_{j \in \mathsf{SS}}$

parse $(\mathbf{s}_i, \vec{\mathsf{seed}}_i) \leftarrow \mathsf{sk}_i$

parse $(\widetilde{\mathbf{w}}_j)_{j \in \mathsf{SS} \setminus \{i\}} \leftarrow (\mathsf{pm}_{4,j})_{j \in \mathsf{SS} \setminus \{i\}}$

pick $(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i, \mathbf{r}_i)$ **from** st_i with $\mathsf{pm}_{4,i} = \widetilde{\mathbf{w}}_i$

req $[\![\forall j \in \mathsf{SS},\ \mathsf{cmt}_j = \mathsf{H}_{\mathsf{com}}(j, \widetilde{\mathbf{w}}_j)]\!]$

$\mathsf{ctnt}_{\mathbf{z}} := 1 \| \mathsf{SS} || \mathsf{M} || (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \| (\widetilde{\mathbf{w}}_j)_{j \in \mathsf{SS}}$

$$\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_{\mathbf{w}}} \in \mathcal{R}_{q_{\nu_{\mathbf{w}}}}^k$$

$c := \mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$ // $c \in \mathcal{C}$

if $[\![\mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] = \perp]\!]$ **then**

$\quad \mathsf{InitializeSign}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{SS}$

$\quad \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{sHS}$

$\quad \mathsf{SignContent}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{ctnt}_{\mathbf{z}}$

$\quad \mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow c$

req $[\![\mathsf{Chall}[\mathsf{ctnt}_{\mathbf{w}}] = c]\!]$

abort if $[\![\mathsf{BadCtnt}[\mathsf{ctnt}_{\mathbf{w}}] = \top]\!]$

$\widetilde{\mathsf{sHS}}_{\mathbf{z}} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}]$

if $[\![\widetilde{\mathsf{sHS}}_{\mathbf{z}} \neq \{i\}]\!]$ **then**

$\quad \widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}_q^{\ell}$

else

$\quad$ **for** $j \in \mathsf{sCS}$

$\quad\quad \mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_{\mathbf{z}})$

$$\widetilde{\mathbf{z}}_i := \mathsf{SimResp}[\mathsf{ctnt}_{\mathbf{w}}] - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_{\mathbf{w}}] - \sum_{j \in \mathsf{sHS} \setminus \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$$

$\mathsf{MaskedResp}[\mathsf{ctnt}_{\mathbf{w}}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_{\mathbf{w}}] \setminus \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \setminus \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

$\mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \leftarrow \mathsf{Q}_{\mathsf{M}}[\mathsf{M}] \cup \{i\}$

return $\mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 46: The second part of the nineteenth game. The differences are highlighted in blue.

is distributed like a honest commitment. Hence, we have

$$\epsilon_{19} = \epsilon_{18}.$$

Game_{20}:

$\mathsf{Q_M}[\cdot] := \varnothing, \mathsf{Strings}, \mathsf{ComSet} := \varnothing, \mathsf{Q}_{\mathsf{H}_c}[\cdot], \mathsf{Q}_{\mathsf{H_{com}}}[\cdot], \mathsf{Q}_{\mathsf{H_{mask}}}[\cdot], \mathsf{Q}_{\mathsf{H}_{\widehat{\mathsf{mask}}}}[\cdot], \mathsf{ProgramHashCom}[\cdot], \mathsf{Signed}_{\Sigma}[\cdot] := \perp$

$\mathsf{InitializeOpen}[\cdot], \mathsf{UnOpenedHS}[\cdot], \mathsf{Mask}_{\mathbf{w}}[\cdot], \mathsf{MaskedCom}[\cdot], \mathsf{UsedCom}[\cdot], \mathsf{SumCom}[\cdot], \mathsf{SumComRnd}[\cdot] := \perp$

$\mathsf{InitializeSign}[\cdot], \mathsf{UnSignedHS}[\cdot], \mathsf{Mask}_{\mathbf{z}}[\cdot], \mathsf{MaskedResp}[\cdot], \mathsf{Chall}[\cdot], \mathsf{SimContent}[\cdot], \mathsf{BadCtnt}[\cdot], \mathsf{SimResp}[\cdot] := \perp$

$\mathbf{A} \xleftarrow{\$} \mathcal{R}_q^{k \times \ell}$

$\mathsf{HS} := [N]$

$\textbf{for } i \in \mathsf{HS} \textbf{ do } \mathsf{st}_i := \varnothing$

$(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_{\mathbf{t}}^{\ell} \times \mathcal{D}_{\mathbf{t}}^{k}$

$\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}_q^k$

$\mathbf{t} := \left\lfloor \hat{\mathbf{t}} \right\rceil_{\nu_{\mathbf{t}}} \in \mathcal{R}_{q_{\nu_{\mathbf{t}}}}^k$

$\textbf{for } i \in [N] \textbf{ do}$

$\quad (\mathsf{vk}_{\mathsf{S},i}, \mathsf{vk}_{\mathsf{S},i}) \xleftarrow{\$} \mathsf{KeyGen}_{\mathsf{S}}(1^{\lambda})$

$\quad \textbf{for } j \in [N] \textbf{ do}$

$\quad\quad \mathsf{rand}_{i,j} \xleftarrow{\$} \{0,1\}^{\lambda}$

$\quad\quad \mathsf{seed}_{i,j} := i \| j \| \mathsf{rand}_{i,j}$

$(\vec{\mathsf{seed}}_i)_{i \in [N]} := \left((\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i})_{j \in [N]}\right)_{i \in [N]}$

$\vec{P} \xleftarrow{\$} \mathcal{R}_q^{\ell}[X]$ with $\deg(\vec{P}) = T - 1, \vec{P}(0) = \mathbf{s}$

$\mathsf{vk} := (\mathsf{tspar}, \mathbf{t})$

$(\mathsf{sk}_i)_{i \in [N]} := \left(\perp, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$

$\mathsf{oracles} := ((\mathcal{O}_{\mathsf{Sign}_i})_{i \in [5]}, \mathcal{O}_{\mathsf{Corrupt}}, \mathsf{H}_c, \mathsf{H_{com}}, \mathsf{H_{mask}})$

$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk})$

$\textbf{req } \llbracket |\mathsf{Q_M}[\mathsf{M}^*] \cup \mathsf{CS}| \leqslant T - 1 \rrbracket$

$\textbf{return } \mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

$\textbf{req } \llbracket \mathsf{SS} \subseteq [N] \rrbracket \wedge \llbracket i \in \mathsf{HS} \rrbracket$

$\mathbf{s}_i \leftarrow \mathcal{R}_q^{\ell}$

$\mathsf{sk}_i \leftarrow \left(\mathbf{s}_i, (\mathsf{vk}_{\mathsf{S},i})_{i \in [N]}, \mathsf{sk}_{\mathsf{S},i}, \vec{\mathsf{seed}}_i\right)_{i \in [N]}$

// Identical to Lines 1 to 88 in Game_{19}

Figure 47: The twentieth game. The differences are highlighted in blue.

Game_{20}: In this game the challenger samples $\hat{\mathbf{t}} \xleftarrow{\$} \mathcal{R}_q^k$ at random. Also, it samples $\mathbf{s}_i$ only if user i becomes corrupted. This is depicted in Fig. 47. Concretely, the challegner samples $\hat{\mathbf{t}}$ uniformly at random over $\mathcal{R}_q^k$ instead of via $(\mathbf{s}, \mathbf{e})$. Also, it postpones generating the secret share $\mathbf{s}_i$ until user i is corrupted via $\mathcal{O}_{\mathsf{Corrupt}}$. In $\mathcal{O}_{\mathsf{Corrupt}}$, it first picks $\mathbf{s}_i$ uniformly at random from $\mathcal{R}_q^{\ell}$ and appends it to the secret key sk_i.

Due to Lemma E.17, which will be proven below, we can construct an $\mathsf{Hint\text{-}MLWE}$ adversary $\mathcal{B}$ solving the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_\mathsf{S},\sigma_\mathbf{t},\sigma_\mathbf{w},\mathcal{C}}$ problem such that

$$|\epsilon_{20} - \epsilon_{19}| \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^\lambda)$$

with $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Remark E.16. If we consider the weaker notion of security where the forgery's message M^* cannot be queried to any signing oracle as in [dPKM+24], then we can show that there exists a $\mathsf{SelfTargetMSIS}$ adversary $\mathcal{B}''$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ problem that internally runs an adversary $\mathcal{A}$ against Game_{20} such that $\epsilon_{20} \leqslant \mathsf{Adv}_{\mathcal{B}''}^{\mathsf{SelfTargetMSIS}}(1^\lambda)$, where $\mathsf{Time}(\mathcal{B}'') \approx \mathsf{Time}(\mathcal{A})$.

Proof. This follows as in Remark E.5. □

Game_{21}: In this game, the challenger guesses the H_c query associated to the adversary's forgery. For this query, the challenger never programs H_c via $\mathsf{ProgramHashChall}$ [15]. It also aborts if $\mathcal{O}_{\mathsf{Sign}_5}$ for the last user is invoked or the last user is corrupted but it did not program H_c in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$ due to the aforementioned change. This is depicted in Fig. 48. In more detail, the challenger initially sets up a counter $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow 0$ and samples $q_{\mathsf{H}_c} \xleftarrow{\$} [Q_{\mathsf{H}_c}]$. Each time a table entry in $\mathsf{Q}_{\mathsf{H}_c}$ is changed, the challenger increases the counter $\mathsf{ctr}_{\mathsf{H}_c}$. This happens either in a fresh H_c query, or when $\mathsf{ProgramHashChall}$ is invoked in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$ and H_c is programmed. In the latter case, the challenger checks if $\mathsf{ctr}_{\mathsf{H}_c} = q_{\mathsf{H}_c}$ and sets $\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top$ if so. It aborts in $\mathcal{O}_{\mathsf{Sign}_5}$ and $\mathcal{O}_{\mathsf{Corrupt}}$ if $\widetilde{\mathsf{sHS}}_\mathbf{z} = \{i\}$ and $\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top$. After $\mathcal{A}$'s forgery $(\mathsf{sig}^*, \mathsf{M}^*)$ is output, the challenger retrieves the value $q^*_{\mathsf{H}_c}$ of $\mathsf{ctr}_{\mathsf{H}_c}$ when the query H_c associated to the forgery was made [16]. This happens either in $\mathsf{ProgramHashChall}$ or H_c.

Let us analyze the advantage of $\mathcal{A}$ in Game_{21}. First, observe that the view of $\mathcal{A}$ is identically distributed conditioned on no abort since $\mathsf{SimResp}[\mathsf{ctnt}_\mathbf{w}]$ for $\mathsf{ctnt}_\mathbf{w}$ such that $\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top$, that is no longer consistent due to the modified $\mathsf{ProgramHashChall}$, is not used throughout the game. If $\mathcal{A}$ is successful and $q_{\mathsf{H}_c} = q^*_{\mathsf{H}_c}$, then the challenger does not abort in $\mathcal{O}_{\mathsf{Sign}_5}$ and $\mathcal{O}_{\mathsf{Corrupt}}$ because the last user involved in the singing queries on M^* is not corrupt and does not execute $\mathcal{O}_{\mathsf{Sign}_5}$. Note that the value q_{H_c} is hidden from $\mathcal{A}$. Thus, we have that

$$\begin{aligned}\epsilon_{21} &\geqslant \Pr[q_{\mathsf{H}_c} = q^*_{\mathsf{H}_c}] \cdot \epsilon_{20} \\ &\geqslant 1/Q_{\mathsf{H}_c} \cdot \epsilon_{20}.\end{aligned}$$

Due to Lemma E.18, which will be proven below, there exists an $\mathsf{SelfTargetMSIS}$ adversary $\mathcal{B}'$ solving the $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ problem that internally runs an adversary $\mathcal{A}$ against Game_{21} such that

$$\epsilon_{21} \leqslant \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}(1^\lambda).$$

Moreover, we have $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$. Collecting all bounds, we have

$$\begin{aligned}\mathsf{Adv}_{\mathsf{TRaccoon}^{\mathsf{adp}}_{\mathsf{4\text{-}rnd}},\mathcal{A}}^{\mathsf{ts\text{-}adp\text{-}uf}}(1^\lambda, N, T) \leqslant{}& Q_{\mathsf{H}_c} \cdot \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}(1^\lambda) + \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^\lambda) + N \cdot \mathsf{Adv}_{\mathsf{S},\mathcal{B}_\mathsf{S}}^{\mathsf{euf\text{-}cma}}(\lambda) \\ &+ \frac{Q_\mathsf{S} \cdot (Q_{\mathsf{H}_\mathsf{com}} + Q_{\mathsf{H}_c} + 2Q_\mathsf{S})}{2^{n-1}} + \frac{Q_{\mathsf{H}_\mathsf{mask}}}{2^\lambda} + \frac{Q_\mathsf{S}^2 + (Q_{\mathsf{H}_\mathsf{com}} + Q_\mathsf{S})^2 + Q_{\mathsf{H}_\mathsf{com}}}{2^{2\lambda}} + \mathsf{negl}(\lambda)\end{aligned}$$

where $\mathsf{Time}(\mathcal{B}), \mathsf{Time}(\mathcal{B}_\mathsf{S}) \approx \mathsf{Time}(\mathcal{A})$, $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$.

To complete the proof, it remains to show Lemmata E.17 and E.18.

[15] More precisely, the challenger *always* samples the output of H_c *after* the input is defined for the guessed query.

[16] The adversary's forgery $(\mathsf{sig}^*, \mathsf{M}^*)$ is associated to some H_c query since we assume that Q_{H_c} also counts the challengers H_c queries in verification without loss of generality.

Game_{21}:

// Identical to Lines 1 to 19 in Game_{20}

$(\mathsf{sig}^*, \mathsf{M}^*) \xleftarrow{\$} \mathcal{A}^{\mathsf{oracles}}(\mathsf{vk})$

parse $(c^*, \mathbf{z}^*, \mathbf{h}^*) \leftarrow \mathsf{sig}^*$

let $q^*_{\mathsf{H}_c}$ be the value of $\mathsf{ctr}_{\mathsf{H}_c}$ when $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \lfloor \mathbf{Az} - 2^{\nu_\mathbf{t}} \cdot c \cdot \mathbf{t} \rceil_{\nu_\mathbf{w}} + \mathbf{h}]$ was set

abort if $[\![q^*_{\mathsf{H}_c} \neq q_{\mathsf{H}_c}]\!]$

req $[\![|\mathsf{Q}_\mathsf{M}[\mathsf{M}^*] - \mathsf{CS}| \leqslant T - 1]\!]$

return $\mathsf{Verify}(\mathsf{tspar}, \mathsf{vk}, \mathsf{M}^*, \mathsf{sig}^*)$

$\mathsf{ProgramHashChall}(\mathsf{ctnt}_\mathbf{w}, c, \widetilde{\mathbf{w}}_i)$:

$\mathsf{M}_\mathsf{S} \leftarrow \mathsf{SimContent}[\mathsf{ctnt}_\mathbf{w}]$

parse $\mathsf{SS} \| \mathsf{M} \| (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}} \leftarrow \mathsf{M}_\mathsf{S}$

if $[\![\forall j \in \mathsf{SS} \backslash \{i\}, \exists! \widetilde{\mathbf{w}}_j, \mathsf{Q}_{\mathsf{H}_\mathsf{com}}(j, \widetilde{\mathbf{w}}_j) = \mathsf{cmt}_j]\!]$

  $\mathbf{w} := \left\lfloor \sum_{j \in \mathsf{SS}} \widetilde{\mathbf{w}}_j \right\rceil_{\nu_\mathbf{w}} \in \mathcal{R}^k_{q_{\nu_\mathbf{w}}}$

  abort if $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \neq \bot]\!]$

  $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow \mathsf{ctr}_{\mathsf{H}_c} + 1$

  if $[\![\mathsf{ctr}_{\mathsf{H}_c} = q_{\mathsf{H}_c}]\!]$

    // Sample c' after $\mathbf{w}$ is defined

    $c' \xleftarrow{\$} \mathcal{C}$

    $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c'$

    $\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] \leftarrow \top$

  else

    $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$

else

  $\mathsf{BadCtnt}[\mathsf{ctnt}_\mathbf{w}] := \top$

$\mathsf{H}_c(\mathsf{vk}, \mathsf{M}, \mathbf{w})$

if $[\![\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] = \bot]\!]$ **then**

  $c \xleftarrow{\$} \mathcal{C}$

  $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow \mathsf{ctr}_{\mathsf{H}_c} + 1$

  $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}] \leftarrow c$

return $\mathsf{Q}_{\mathsf{H}_c}[\mathsf{vk}, \mathsf{M}, \mathbf{w}]$

$\mathcal{O}_{\mathsf{Corrupt}}(i)$

// Identical to Lines 1 to 65 in Game_{19}

  else // user i is between round 4 and round 5

    $\widetilde{\mathsf{sHS}}_\mathbf{z} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}]$ // $\mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}] \neq \bot$

    if $[\![\widetilde{\mathsf{sHS}}_\mathbf{z} \neq \{i\}]\!]$ **then**

      $\mathbf{\Delta}_i \xleftarrow{\$} \mathcal{R}^\ell_q$

    else // user i is the last user for $\mathsf{ctnt}_\mathbf{z} = \mathsf{SignContent}[\mathsf{ctnt}_\mathbf{w}]$

      abort if $[\![\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top]\!]$

      $\mathsf{ctnt}_\mathbf{z} \leftarrow \mathsf{SignContent}[\mathsf{ctnt}_\mathbf{w}]$

      for $j \in \mathsf{sCS}$

        $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_\mathbf{z})$

      $\mathbf{\Delta}_i := \mathsf{SimResp}[\mathsf{ctnt}_\mathbf{w}] - c \sum_{j \in \mathsf{sCS} \cup \{i\}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_\mathbf{w}] - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_\mathbf{w}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$

    $\mathsf{Mask}_\mathbf{z}[\mathsf{ctnt}_\mathbf{w}, i] \leftarrow \mathbf{\Delta}_i$

// Identical to Lines 76 to 88 in Game_{19}

$\mathcal{O}_{\mathsf{Sign}_5}(\mathsf{SS}, \mathsf{M}, i, (\mathsf{pm}_{4,j})_{j \in \mathsf{SS}})$

// Identical to Lines 1 to 16 in Game_{19}

$\widetilde{\mathsf{sHS}}_\mathbf{z} \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}]$

if $[\![\widetilde{\mathsf{sHS}}_\mathbf{z} \neq \{i\}]\!]$ **then**

  $\widetilde{\mathbf{z}}_i \xleftarrow{\$} \mathcal{R}^\ell_q$

else

  abort if $[\![\mathsf{BadGuess}[\mathsf{ctnt}_\mathbf{w}] = \top]\!]$

  for $j \in \mathsf{sCS}$

    $\mathbf{\Delta}_j := \mathsf{ZeroShare}(\vec{\mathsf{seed}}_j[\mathsf{SS}], \mathsf{ctnt}_\mathbf{z})$

  $\widetilde{\mathbf{z}}_i := \mathsf{SimResp}[\mathsf{ctnt}_\mathbf{w}] - c \sum_{j \in \mathsf{sCS}} L_{\mathsf{SS},j} \cdot \mathbf{s}_j + \mathsf{SumComRnd}[\mathsf{ctnt}_\mathbf{w}] - \sum_{j \in \mathsf{sHS} \backslash \{i\}} \mathsf{MaskedResp}[\mathsf{ctnt}_\mathbf{w}, j] - \sum_{j \in \mathsf{sCS}} \mathbf{\Delta}_j$

$\mathsf{MaskedResp}[\mathsf{ctnt}_\mathbf{w}, i] \leftarrow \widetilde{\mathbf{z}}_i$

$\mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}] \leftarrow \mathsf{UnSignedHS}[\mathsf{ctnt}_\mathbf{w}] \backslash \{i\}$

$\mathsf{st}_i \leftarrow \mathsf{st}_i \backslash \{(\mathsf{SS}, \mathsf{M}, (\mathsf{str}_j, \mathsf{cmt}_j)_{j \in \mathsf{SS}}, \widetilde{\mathbf{w}}_i)\}$

$\mathsf{Q}_\mathsf{M}[\mathsf{M}] \leftarrow \mathsf{Q}_\mathsf{M}[\mathsf{M}] \cup \{i\}$

return $\mathsf{pm}_{5,i} := \mathbf{z}_i$

Figure 48: The twenty-first game. The differences are highlighted in blue. We assume that this game initializes an empty list $\mathsf{BadGuess}[\cdot] := \bot$, a counter $\mathsf{ctr}_{\mathsf{H}_c} \leftarrow 0$, and samples a guess $q_{\mathsf{H}_c} \xleftarrow{\$} [Q_{\mathsf{H}_c}]$ at the beginning of the game.

Lemma E.17. *There exists an adversary $\mathcal{B}$ against the* $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_\mathsf{S},\sigma_\mathbf{t},\sigma_\mathbf{w},\mathcal{C}}$ *problem such that*

$$|\epsilon_{20} - \epsilon_{19}| \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^\lambda)$$

where $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$.

Proof. Let $\mathcal{A}$ be an adversary that distinguishes Game_{19} and Game_{20}. To show this lemma, we construct an adversary $\mathcal{B}$ against the $\mathsf{Hint\text{-}MLWE}_{q,\ell,k,Q_\mathsf{S},\sigma_\mathbf{t},\sigma_\mathbf{w},\mathcal{C}}$ problem that internally runs $\mathcal{A}$. $\mathcal{B}$ is given the $\mathsf{Hint\text{-}MLWE}$ problem instance $(\mathbf{A}, \mathbf{b}, (c_i, \mathbf{z}_i, \mathbf{z}'_i)_{i\in[Q_\mathsf{S}]})$ as input.

$\mathcal{B}$ behaves as the challegner in Game_{20} except for the initial phase, $\mathcal{O}_{\mathsf{Sign}_4}$, and $\mathcal{O}_{\mathsf{Corrupt}}$. In the initial phase, it uses $\mathbf{A}$ given as input, instead of choosing a fresh $\mathbf{A}$ sampled from $\mathcal{R}_q^k$, and embeds $\lfloor \mathbf{b} \rceil_{\nu_\mathbf{t}}$ into $\mathbf{t}$. Note that it no longer generates secret shares $(\mathbf{s}_i)_{i\in[N]}$. Also, when it generates the ith simulated commitment in $\mathcal{O}_{\mathsf{Sign}_4}$ or $\mathcal{O}_{\mathsf{Corrupt}}$, it uses $(c_i, \mathbf{z}_i, \mathbf{z}'_i)$, instead of sampling $c \xleftarrow{\$} \mathcal{C}$, $(\mathbf{r}, \mathbf{e}') \xleftarrow{\$} \mathcal{D}_\mathbf{w}^\ell \times \mathcal{D}_\mathbf{w}^k$, and setting $\mathbf{z} := c \cdot \mathbf{s} + \mathbf{r}$ and $\mathbf{z}' := c \cdot \mathbf{e} + \mathbf{e}'$. Otherwise, it behaves as the challenger in Game_{20}.

We show that $\mathcal{B}$ perfectly simulates the challenger in Game_{19} (resp. Game_{20}) when $\mathbf{b}$ is a valid MLWE sample (resp. $\mathbf{b}$ is uniformly sampled from $\mathcal{R}_q^k$). When $\mathbf{b}$ is a valid MLWE sample, $\mathbf{t}$ is identically distributed to $\mathbf{t}$ in Game_{19}. Since $\mathcal{A}$ can corrupt at most $T-1$ honest users, the secret shares $\mathbf{s}_i$ of each corrupted user $i \in \mathsf{CS}$ is uniformly distributed over $\mathcal{R}_q^\ell$. Also, since the leakage $(\mathbf{z}_i, \mathbf{z}'_i)$ satisfies

$$\mathbf{z}_i = c \cdot \mathbf{s} + \mathbf{r}, \text{ and } \mathbf{z}'_i = c \cdot \mathbf{e} + \mathbf{e}' \tag{22}$$

where $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_\mathbf{t}^\ell \times \mathcal{D}_\mathbf{t}^k$ and $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$, $\mathcal{B}$ perfectly simulates the singing and corruption oracles in Game_{19}.

When $\mathbf{b}$ is uniformly sampled from $\mathcal{R}_q^k$, the distribution of $\mathbf{t}$ is identical to that in Game_{20}. Moreover, $\mathcal{B}$ perfectly simulates the singing and corruption oracles in Game_{20} due to Eq. (22), where $(\mathbf{s}, \mathbf{e}) \xleftarrow{\$} \mathcal{D}_\mathbf{t}^\ell \times \mathcal{D}_\mathbf{t}^k$. Note that the leakage no longer depends on $\mathbf{t}$. Combining all arguments, $\mathcal{B}$ perfectly simulates Game_{19} and Game_{20} when $\mathbf{b}$ is a valid MLWE sample and generated by $\mathbf{b} \xleftarrow{\$} \mathcal{R}_q^k$, respectively. Therefore, we have

$$|\epsilon_{20} - \epsilon_{19}| \leqslant \mathsf{Adv}_{\mathcal{B}}^{\mathsf{Hint\text{-}MLWE}}(1^\lambda).$$

Finally, it is clear $\mathsf{Time}(\mathcal{B}) \approx \mathsf{Time}(\mathcal{A})$ from the construction of $\mathcal{B}$. This completes the proof. ☐

Lemma E.18. *There exists a* $\mathsf{SelfTargetMSIS}$ *adversary* $\mathcal{B}'$ *solving the* $\mathsf{SelfTargetMSIS}_{q,\ell+1,k,\mathsf{H}_c,\mathcal{C},B}$ *problem that internally runs an adversary* $\mathcal{A}$ *against* Game_{21} *such that*

$$\epsilon_{21} \leqslant \mathsf{Adv}_{\mathcal{B}'}^{\mathsf{SelfTargetMSIS}}(1^\lambda)$$

where $\mathsf{Time}(\mathcal{B}') \approx \mathsf{Time}(\mathcal{A})$.

Proof. This follows as in Lemma E.7. ☐

This completes the proof. ☐