Round-Efficient Adaptively Secure Threshold Signatures with Rewinding

Yanbo Chen[*]

April 8, 2025

Abstract

A threshold signature scheme allows distributing a signing key to n users, such that any t of them can jointly sign, but any $t-1$ cannot. It is desirable to prove *adaptive security* of threshold signature schemes, which considers adversaries that can adaptively corrupt honest users even after interacting with them. For a class of signatures that relies on security proofs with rewinding, such as Schnorr signatures, proving adaptive security entails significant challenges.

This work proposes two threshold signature schemes that are provably adaptively secure with rewinding proofs. Our proofs are solely in the random oracle model (ROM), without relying on the algebraic group model (AGM).

- We give a 3-round scheme based on the algebraic one-more discrete logarithm (AOMDL) assumption. The scheme outputs a standard Schnorr signature.

- We give a 2-round scheme based on the DL assumption. Signatures output by the scheme contain one more scalar than a Schnorr signature.

We follow the recent work by Katsumata, Reichle, and Takemure (Crypto 2024) that proposed the first threshold signature scheme with a rewinding proof of full adaptive security. Their scheme is a 5-round threshold Schnorr scheme based on the DL assumption. Our results significantly improve the round complexity.

Katsumata et al.'s protocol can be viewed as applying a masking technique to Sparkle, a threshold Schnorr signature scheme by Crites, Komlo, and Maller (Crypto 2023). This work shows wider applications of the masking technique. Our first scheme is obtained by masking FROST, a threshold Schnorr protocol by Komlo and Goldberg (SAC 2020). The second scheme is obtained by masking a threshold version of HBMS, a multi-signature scheme by Bellare and Dai (Asiacrypt 2021).

Katsumata et al. masked Sparkle at the cost of 2 additional rounds. Our main insight is that this cost varies across schemes, especially depending on how to simulate signing in the security proofs. The cost is 1 extra round for our first scheme, and is 0 for our second scheme.

Keywords. Threshold Signature, Adaptive Security, Schnorr Signature

1 Introduction

A threshold signature scheme is a protocol for distributing a main secret key to a group of n users, such that any subgroup of t users can jointly sign under the common public key using their own secret key shares, while any set of $t-1$ users cannot. In recent years, threshold signatures have attracted significant interest from both academia and industry, mainly for their wide applications in distributed systems.

A basic security notion for threshold signatures is the *static security*, where the adversary decides on which $t-1$ users to control before the protocol starts. A more advanced notion is *adaptive*

[*]University of Ottawa. Email: ychen918@uottawa.ca.

security, which allows the adversary to adaptively corrupt at most $t-1$ users during the execution of the protocol. In particular, the adversary can decide which parties to corrupt depending on its interaction with the honest users.

Many important signature schemes are proved to be secure based on *rewinding* in the random oracle model (ROM). The Schnorr signature scheme is a representative example. Rewinding-based security reductions face difficulties when considering adaptive adversaries: rewinding doubles the number of corruptions that the reduction needs to handle. Intuitively, the reduction thus needs to know up to $2(t-1) \geq t$ secret key shares. In many cases, these shares allow reconstructing the main secret key, so the reduction cannot embed an instance of a hard problem in the main public key, as for Schnorr signatures.

Adaptive security for such schemes was first considered by Crites, Komlo, and Maller [CKM23] who studied the adaptive security of their threshold Schnorr signature scheme Sparkle. Owing to the above difficulty, their proofs either achieve partially adaptive security, only allowing the adversary to adaptively corrupt $(t-1)/2$ users, or rely on the algebraic group model (AGM) [FKL18] to bypass rewinding. Bacho et al. [BLT$^+$24] proposed a scheme with fully adaptive security without the AGM, but the signatures are modified to support a DDH-based rewinding-free proof. The resulting signatures are relatively large and not compatible with systems that employ Schnorr signatures.

Recently, Katsumata, Reichle, and Takemure [KRT24a] proposed an ingenious solution to prove full adaptive security with rewinding. Their result is a threshold Schnorr signature scheme with a rewinding-based proof of adaptive security without the AGM. One main drawback is a 4-round and stateful signing protocol. Specifically, before the protocol runs, the signing users have to agree on a session identifier that is distinct for each signing, which requires them to store states that provide information about used identifiers. The authors also gave a general transformation to convert the scheme into a 5-round and stateless scheme. More recently, Bacho et al. [BDLR24] proposed a threshold Schnorr scheme that improves upon [KRT24a] in several aspects but is still 5-round. In comparison, most schemes proposed in recent years, which may not be proved to achieve adaptively secure without the AGM, are 2 or 3-round. Our main objective is to construct provably adaptively secure schemes of low round complexity.

1.1 Our Contribution

We propose two new schemes in this work. Both of them are proved to be adaptively secure with a rewinding proof in the ROM without the AGM.

- In Section 3, we present FROST-Mask, a stateful 2-round scheme based on the algebraic one-more discrete logarithm (AOMDL) assumption. The resulting signature is a standard Schnorr signature. The general transformation in [KRT24a] can compile FROST-Mask into a stateless 3-round scheme.
- In Section 4, we present HBTS-Mask, a stateless 2-round scheme based on the DL assumption. The resulting signature contains one more scalar than a Schnorr signature.

We review the masking technique used in Katsumata et al.'s construction [KRT24a] in depth and find wider applications of this technique. Katsumata et al.'s construction can be viewed as an application of the masking technique to Sparkle [CKM23]. To get our first scheme, we mask FROST [KG20], a well-known 2-round threshold Schnorr scheme. For our second scheme, we mask HBTS, a threshold version of 2-round multi-signature scheme HBMS [BD21].

Our main insight is that the cost of masking varies for different underlying schemes, particularly depending on how signing is simulated in the security proofs. The cost of masking Sparkle is being stateful and an extra round. For FROST, we only introduce the state. To mask HBTS, neither of these is required!

Table 1 summarizes our schemes and compares them with existing schemes in the pairing-free discrete logarithm setting. Previously, only [BLT$^+$24] (Twinkle), [KRT24a], [BDLR24] (Glacius) , and [Che25] (Dazzle) were proved to be adaptively secure without the AGM. Only [KRT24a] and

Scheme	Adaptive	Assumption	AGM	Rounds	State	Sig. Size
[KG20] (FROST)	✗	AOMDL	✗	2	✗	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
[TZ23]	✗	DL	✗	2	✗	$1\langle\mathbb{G}\rangle + 2\langle\mathbb{Z}_p\rangle$
[CKM23] (Sparkle)	✗	DL	✗	3	✗	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
[CKM23] (Sparkle)	half	AOMDL	✗	3	✗	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
[CKM23] (Sparkle)	✓	AOMDL	✓	3	✗	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
[BLT+24] (Twinkle)	✓	DDH	✗	3	✗	$2\langle\mathbb{G}\rangle + 3\langle\mathbb{Z}_p\rangle$
[Che25] (Dazzle)	✓	DDH	✗	2	✗	$1\langle\mathbb{G}\rangle + 3\langle\mathbb{Z}_p\rangle$
[KRT24a]	✓	DL	✗	4	✓	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
[BDLR24] (Glacius)	✓	DDH	✗	4	✓	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
FROST-Mask	✓	AOMDL	✗	2	✓	$1\langle\mathbb{G}\rangle + 1\langle\mathbb{Z}_p\rangle$
HBTS-Mask	✓	DL	✗	2	✗	$1\langle\mathbb{G}\rangle + 2\langle\mathbb{Z}_p\rangle$

Table 1: Comparison between our schemes and existing schemes. We compare whether they are proved to be secure in the adaptive model, the algebraic assumptions they rely on, whether the proofs are in the AGM, their round complexity, whether their signing algorithms are stateful, and the size of their signatures. In column "Adaptive", "half" means that the adversary is allowed to adaptively corrupt $(t-1)/2$ users (and statically corrupt the others). We remark that stateful schemes can be turned stateless at the cost of one extra round. We let $\langle\mathbb{G}\rangle$ and $\langle\mathbb{Z}_p\rangle$ denote the size of a group element and a scalar, respectively.

[BDLR24] produce standard Schnorr signatures, ensuring compatibility with systems that currently employ Schnorr signatures, a highly desirable property. Our FROST-Mask also outputs a standard Schnorr signature. While relying on a one-more assumption, it reduces the number of rounds by 2. Our HBTS-Mask does not output a standard Schnorr signature, but it only contains one more scalar. Its main advantage is being 2-round and stateless. Previously, only [Che25] achieved this, while our HBTS-Mask is based on DL rather than DDH and outputs signatures containing one fewer scalar.

We also mention two downsides of our schemes. First, as pointed out in [KRT24a], the masking technique makes signers' signature shares unverifiable and thus prevents non-interactive identifiable abort, a functionality that non-interactively traces misbehaving users that prevent the generation of valid signatures. Except [KRT24a] and our schemes, all other schemes in Table 1 support non-interactive identifiable abort. Second, the masking technique also causes FROST to lose the online-offline property. The first round of FROST can be preprocessed before the signing group and the message are decided. In contrast, FROST-Mask needs the signing group to be known in the first round to generate masks.

Future Work. Katsumata et al. [KRT24a] also gave a lattice-based version of their threshold Schnorr scheme. We note that the schemes we mask, FROST and HBMS, both have lattice variants [EKT24, Che23]. We consider finding lattice versions of FROST-Mask and HBTS-Mask as an interesting direction of future work.

1.2 Technical Overview

The Difficulty of Adaptive Threshold Schnorr with Rewinding.

The Schnorr signature works over a cyclic group $\mathbb{G}$ of order p with generator g. The secret key is a scalar $x \in \mathbb{Z}_p$, and the public key is its exponentiation $X = g^x$. To sign message μ, the signer commits randomness r in $R := g^r$, obtains a challenge by hashing $c := \mathsf{H}(\mu, R)$, responds with $z := r + cx$, and finally outputs (R, z) as the signature.

The Schnorr scheme is proved to be secure from the DL assumption in the ROM by rewinding. After receiving a forgery (R, z), the security proof runs the adversary for a second time. By reprogramming the random oracle, it expects to receive another forgery (R, z'), with the same commitment R but responding to a different challenge. These two related forgeries are used to extract the discrete logarithm of X.

A homomorphic signature scheme can be compiled into a threshold scheme as follows. The users secret-share the main secret key using Shamir's t-out-of-n secret sharing, each holding a share x_i. For a group $\mathcal{S}$ of t users to jointly sign, each user signs with $\lambda_{\mathcal{S},i} \cdot x_i$ as the secret key to provide a signature share σ_i, where $\lambda_{\mathcal{S},i}$ is the reconstruction coefficient. Then using the homomorphism, the signature shares can be combined into a valid signature $\sigma = \sum_{i \in \mathcal{S}} \sigma_i$ under the main key. Although the Schnorr signature scheme is not strictly homomorphic, the idea of the above compiler still works.

However, when aiming at adaptive security, the above framework faces a significant challenge when thresholdizing the Schnorr signature scheme. In this framework, each user issues Schnorr signatures using $\lambda_{\mathcal{S},i} \cdot x_i$. Even if the "public key share" $X_i = g^{x_i}$ is not made public at the beginning, the signature shares will totally reveal X_i. The public shares provide binding commitments to the secret key shares: whenever user i gets corrupted, the reduction must output $x_i = \mathsf{dlog}_g X_i$.

The problem arises when the rewinding happens before the adversary A corrupts parties, but after the public shares are revealed. The former means that A may corrupt different sets of users in two runs, forcing the reduction to output up to $2(t-1) \geq t$ secret key shares. The secret key shares have been committed before the rewinding point, so they must be related in the two runs, and any t of them reconstruct the main secret key x. Therefore, the reduction has to be able to figure out $x = \mathsf{dlog}_g X$ on its own, so it cannot embed a DL instance into the public key X.

One-Time Masks to The Rescue.

The above problem occurs because the public key shares are revealed as commitments to the secret key shares. What if they are never made public, and each user's signature share is independent of its secret key share? Then each secret key share x_i is independent of A's view until user i gets corrupted. Consequently, the security game is statistically unchanged if we defer the sampling of x_i to the corruption of user i. Since at most $t-1$ users can be corrupted in the game, we only need to sample $t-1$ shares. By the security of Shamir's secret sharing, these $t-1$ shares are independently, uniformly distributed and independent of the main secret key x.

Now there is no problem for rewinding! The answer to each corruption query is just a newly sampled secret key share. For a query made after the rewinding point, the answers in two runs are sampled independently. The secret key shares output in the second run do not reconstruct x with those output in the first round, so the reduction does not have to know x by itself.

Katsumata et al. [KRT24a] found that the masking technique, first used in lattice-based threshold signatures [DKM+24, EKT24] for a different purpose than adaptive security, can make signature shares independent of their secret key shares. The t signers generate one-time random masks $\{\Delta_i\}_{i \in \mathcal{S}}$ that sum to 0. Each of them masks its signature share using the mask and output $\sigma_i + \Delta_i = \tilde{\sigma}_i$. Since the masks sum to 0, the signature shares still combine to $\sum_{i \in \mathcal{S}} \tilde{\sigma} = \sum_{i \in \mathcal{S}} \sigma_i = \sigma$, but every $(t-1)$-subset of these signatures is uniformly random and leaks no information. The adversary can get information from honest users' signature shares only by combining them, and the information it obtains is no more than a signature σ issued with the main secret key x. Nothing about secret key shares is leaked.

Moreover, Katsumata et al. showed how to non-interactively generate the masks. For this purpose, we require every pair of users i and j to share seeds $\mathsf{seed}_{i,j}$ and $\mathsf{seed}_{j,i}$. We also need some distinct information sinf related to each signing session. User $k \in \mathcal{S}$ generates the mask as

$$\Delta_k := \sum_{i \in \mathcal{S} \setminus \{k\}} (\mathsf{H}(\mathsf{seed}_{k,i}, \mathsf{sinf}) - \mathsf{H}(\mathsf{seed}_{i,k}, \mathsf{sinf})). \tag{1}$$

It can be verified that $\sum_{i \in \mathcal{S}} \Delta_i = 0$, and in the ROM $\{\Delta_i\}_{i \in \mathcal{S}}$ are $(T-1)$-wise independently, uniformly random.

Masking Sparkle, and The Cost.

An immediate challenge in masking threshold Schnorr is the interactive signing protocol. The users do not output their signature shares in one shot. Instead, two components of Schnorr signatures are generated separately in a multi-round protocol. The signing protocol of Sparkle is as follows:

- In the first round, each signer k samples $r_k \leftarrow \mathbb{Z}_p$ and computes $R_k := g^{r_k}$. It outputs a hash commitment $d_i := \mathsf{H}_{\text{com}}(k, R_k)$.[1]
- In the second round, user k receives other users' hash commitments and outputs R_k.
- In the third round, user k receives $\{R_i\}_{i \in \mathcal{S}}$ and verifies that they are consistent with the hash commitments. It computes $R := \prod_{i \in \mathcal{S}} R_i$ and obtains the challenge $c := \mathsf{H}_{\text{chal}}(\mu, R)$. It outputs the response $z_k := r_k + c \cdot \lambda_{\mathcal{S},k} \cdot x_k$.
- Finally, a combiner receives $\{z_i\}_{i \in \mathcal{S}}$ and aggregates them into $z := \sum_{i \in \mathcal{S}} z_i$. The threshold signature is $\sigma := (R, z)$.

The homomorphism needed to combine signature shares holds only when every signer responds to a common challenge c. Therefore, the protocol lets the users first produce their commitments and exchange them. They can obtain the common challenge c by hashing the aggregated commitment R.

To make signature share (R_k, z_k) independent of x_k, it suffices to mask the response z_k. The masked $\tilde{z}_k$ then becomes independent of R_k and leaks no information about x_k. We generate the mask Δ_k by *Eq.* (1) using H_{mask}, where the session information sinf contains the message μ and the set of commitments $\{R_i\}_{i \in \mathcal{S}}$. We have successfully hidden secret key shares. Unfortunately, this is insufficient for adaptive security, as two subtle problems arise.

So far we argued that secret key shares are statistically hidden. What remains is to efficiently simulate the security game in the reduction. The problems are about how to simulate signing without the secret key. Thanks to the masks, we only need to simulate signatures under the main public key X. Importantly, the signature is issued *interactively*, which means that two components of the simulated signature (R_*, z_*) should be produced. More precisely, the simulator controls honest users in a signing session as follows. It decides honest users' commitments with R_* embedded in. The aggregated commitment R is determined after the corrupted users also output their commitments. Then the simulator decides honest users' responses based on z_*, where (R_*, z_*) is required to be a signature under X in response to $c = \mathsf{H}_{\text{chal}}(\mu, R)$. Essentially, this procedure is the same as simulating a single honest user in the unmasked scheme, so the simulation method of the masked scheme is inherited from the unmasked one.

The simulation method of Sparkle works as follows. The simulator samples a challenge c and calls the honest-verifier zero-knowledge (HVZK) simulator of Schnorr protocol to obtain a signature (R_*, z_*), a signature under X responding to c. The simulator has to make sure $\mathsf{H}_{\text{chal}}(\mu, R) = c$ even though R is partially decided by the corrupted users. In the ROM, the hash commitment becomes extractable for the reduction while still hiding in A's view. Such properties allow the simulator to know the corrupted users' commitments before revealing honest users' commitments, and therefore know R before the adversary. Hence, the simulator can program $\mathsf{H}_{\text{chal}}(\mu, R) := c$ before A queries $\mathsf{H}_{\text{chal}}(\mu, R)$.

Now we are ready to discuss the two remaining problems. First, corruption not only leaks the secret key shares, but also secret states in signing sessions, specifically r_k. Suppose that rewinding happens after $\{R_i\}_{i \in \mathcal{S}}$ are all output in a signing session. During the two runs, A may corrupt more than t different users, so the reduction may need to reveal r_i for all $i \in \mathcal{S}$. This means that there is no place to embed R_* whose discrete logarithm is unknown.

[1] We stress the difference between "hash commitment" d and "commitment" R (to randomness r).

The solution is to also mask the commitments with another set of one-time masks that sum to 0. To understand the effect, one can analogize r_k to x_k, R_k to X_k, and $\prod_{i\in\mathcal{S}} R_i$ to X. Now R_k as the commitment to r_k is never made public before rewinding, and the masked commitment $\tilde{R}_k$ is independent of r_k. This allows the reduction to defer sampling r_k until user k is corrupted. As a result, randomness r_i's output in the second run do not reconstruct the discrete logarithm of $\prod_{i\in\mathcal{S}} R_i$ with the randomness r_i's output in the first run. The reduction does not have to know the discrete logarithm of $\prod_{i\in\mathcal{S}} R_i$, which provides a place to embed R_*.

Masking commitments has a cost of turning the scheme stateful. Recall that one-time masks are generated by hashing some distinct session-specific information sinf. The distinctness is crucial for ensuring the masks are indeed "one-time". When masking z_k, the distinctness of sinf is guaranteed by the high entropy of $\{R_i\}_{i\in\mathcal{S}}$ contained in sinf. However, now we want to mask R_k in the first round, before any protocol message is output. We require users to agree on a distinct session id sid and use it to generate the mask. To ensure the distinctness of sid, users have to maintain long-term states across different challenges.

The second problem is more subtle. Recall that the HVZK simulator prepares a signature (R_*, z_*) for a *predetermined* challenge c. However, we will show that adaptive corruptions make the challenge c not determined before the simulator needs R_* even with the help of hash commitments in the first round. We remark a signature under X will be revealed to A at the moment when *a signing session has been consistently completed by all honest users.* This may happen not only following a signing query, but also a corruption query. Consider the following behavior of A. It lets t users *inconsistently* finish a signing session by sending different protocol messages to different honest signers. As a result, the honest signers saw different aggregated commitments and responded to different challenges $\{c_i\}_{i\in\mathcal{S}}$. So far A gets no information, because the honest users' one-time masks were generated by hashing different sinf and do not sum to 0. However, A then corrupts all users that participated in the session except one user k. As the only remaining honest user in the session, it definitely has "consistently completed" the session, and a signature responding to c_k should be revealed. In the above process, multiple candidate challenges remain until A corrupts $t-1$ users, and more importantly remain after the simulator used R_* to decide honest users' commitments. The HVZK simulator fails in such a situation.

The solution is to detect inconsistency in the signing protocol. The users run an additional echo round after they exchange the hash commitments. In the echo round, the users authentically exchange what they received from the first round (in fact, it suffices that each user broadcasts a signature). As a result, A can no longer send inconsistent protocol messages to different honest users, as the inconsistent messages cannot be both authenticated all honest users in the echo round.

In summary, two problems arise when masking $\mathsf{Sparkle}$. The problems are caused by the following features of the signature simulator of $\mathsf{Sparkle}$, respectively:

- The simulator embeds R_* with unknown discrete logarithm into honest users' commitments.
- The simulator can only respond to a predetermined challenge.

To resolve them, Katsumata et al. make the users stateful and add an echo round, respectively, resulting in a 4-round stateful protocol.

Masking FROST.

This work explores wider applications of the masking technique. Since we want to improve on round complexity, masking FROST [KG20, BCK+22], a well-known 2-round threshold Schnorr scheme, becomes a natural choice. For a set $\mathcal{S}$ of users to sign μ, the signing protocol of FROST works as follows:

- In the first round, each user k samples $r_{1,k}$, $r_{2,k} \leftarrow \mathbb{Z}_p$ and outputs commitments $R_{1,k} := g^{r_{1,k}}$, $R_{2,k} := g^{r_{2,k}}$.
- In the second round, user k receives protocol messages $\mathcal{M} = \{(R_{1,i}, R_{2,i})\}_{i\in\mathcal{S}}$. It hashes μ and $\mathcal{M}$ to obtain a combination coefficient $b := \mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mu, \mathcal{M})$. Then it calculates the aggregated

commitment $R := \prod_{i\in\mathcal{S}} R_{1,i}R_{2,i}^b$. It obtains the challenge $c := \mathsf{H}_{\text{chal}}(\mu, R)$ and outputs a response $z_k := r_{1,k} + br_{2,k} + c \cdot \lambda_{\mathcal{S},k} \cdot x_k$.
- Finally, a combiner aggregates the individual responses into $z := \sum_{i\in\mathcal{S}} z_i$ and outputs the threshold signature as $\sigma := (R, z)$.

The roadmap toward adaptive security is the same as Sparkle. First, we mask the responses with one-time masks. As a result, we can reach a statistically indistinguishable security game where each secret key share x_k is sampled only when user k gets corrupted. Next, the reduction needs to efficiently simulate this game. We need a simulator that interactively issues signatures under X. The simulation method is inherited from FROST. Let us review the simulation technique of FROST.

Instead of plain DL, FROST relies on the algebraic one-more discrete-logarithm (AOMDL) assumption. The AOMDL problem asks to solve $q + 1$ discrete logarithms with q calls to oracle DLog that solves the discrete logarithm. By "algebraic", every DLog query is required to be represented as a linear combination of the $q + 1$ input elements. The simulation of FROST crucially exploits the DLog oracle.

The reduction takes an input element as the main public key X. Let $\mathcal{C}$ denote the set of corrupted users. The simulator decides honest users' commitments $\{R_{1,i}\}_{i\in\mathcal{S}\setminus\mathcal{C}}$ with $R_{1,*}$ embedded in and $\{R_{2,i}\}_{i\in\mathcal{S}\setminus\mathcal{C}}$ with $R_{2,*}$ embedded in, where $R_{1,*}$ and $R_{2,*}$ are two input elements. When the corrupted users also output their commitments, the combining coefficient b and the challenge c are determined. After that, the simulator decides honest users' responses based on z_*, where $(R_{1,*}, R_{2,*}, z_*)$ is required to satisfy $g^{z_*} = R_{1,*}R_{2,*}^b X^c$, like a signature share in FROST. The simulator makes a query $\text{DLog}(R_{1,*}R_{2,*}^b X^c)$ to obtain such a response z_*. We remark that in the rewinding-based reduction, the simulator may have to respond to a different challenge in the second run. The key is that the reduction takes $2q_\text{s} + 1$ input elements, where q_s is the maximum number of signing queries, Each signing session uses two input elements as $R_{1,*}$, $R_{2,*}$, and the reduction has two chances to call dlog, one for each run.

Now let us examine the cost of masking FROST by checking if it also has the features of Sparkle that causes problems. First, FROST simulator also embeds $R_{1,*}$, $R_{2,*}$ with unknown discrete logarithm into honest users' commitments. The reduction faces the problem that during the two runs, all honest users in a signing session may be corrupted and leak their signing randomness $r_{1,i}$, $r_{2,i}$. To resolve this problem, we mask the commitments. Generating masks in the first round relies on distinct session id and requires users to be stateful.

In contrast, the second feature of FROST simulator is different from Sparkle. FROST generates the response z_* using dlog oracle instead of the HVZK simulator. The oracle can handle any challenge c, not just a pre-determined one. Therefore, the additional echo round is not necessary to mask FROST. In conclusion, the only cost is to be stateful!

Masking HBTS.

Can we do even better? We introduce a new DL-based 2-round threshold signature scheme. The 2-round multi-signature scheme HBMS (which stands for "Hash-Based Multi-Signature") [BD21] can be adapted to the following threshold signature scheme, which we call HBTS:

- In the first round, each user $k \in \mathcal{S}$ obtains a second generator $h := \mathsf{H}_{\text{gen}}(\mu)$, samples r_k, $y_k \leftarrow \mathbb{Z}_p$, and outputs commitment $R_k := g^{r_k}h^{y_k}$.
- In the second round, user k receives $\{R_i\}_{i\in\mathcal{S}}$ and computes the aggregated commitment $R := \prod_{i\in\mathcal{S}} R_i$. It obtains challenge $c := \mathsf{H}_{\text{chal}}(\mu, R)$ and outputs a response (z_k, y_k), where $z_k := r_k + c \cdot \lambda_{\mathcal{S},k} \cdot x_k$.
- Finally, a combiner aggregates the individual responses into $z := \sum_{i\in\mathcal{S}} z_i$ and $y := \sum_{i\in\mathcal{S}} y_i$ and outputs $\sigma := (R, z, y)$.

The resulting signature contains one more scalar than a Schnorr signature. It can be verified by the equation $g^z h^y = RX^c$.

Let us apply the roadmap toward adaptive security for the third time. We mask the responses

(z_k, y_k) with one-time masks, which statistically hide x_k until user k is corrupted. We then use the simulation method from HBTS to simulate signatures.

To understand the simulation of HBTS, we point out a connection to the Okamoto signature scheme [Oka93]. The Okamoto scheme is similar to the Schnorr scheme but works with two generators g and h. The secret key is $(x, w) \in \mathbb{Z}_p^2$, and the public key is $X := g^x h^w$. The signer samples r, $s \leftarrow \mathbb{Z}_p$ and computes commitment $R := g^r h^s$. It responds to the challenge $c := \mathsf{H}_{\mathrm{chal}}(\mu, R)$ with (z, y), where $z = r + cx$ and $y = s + cw$. The verification is to check whether $g^z h^y = RX^c$. The important fact for us is that the Okamoto scheme is *witness indistinguishable*. Specifically, a public key X corresponds to many possible secret keys (x, w), while signatures issued with different secret keys are identically distributed.

In HBTS, each user actually produces an Okamoto signature, with h being *message-specific* and $(x, 0)$ as the secret key. The reduction partitions the message space, such that for some messages, $\mathsf{H}_{\mathrm{gen}}(\mu) = h = X^{1/w}$ with some known w, while for the remaining messages, $\mathsf{H}_{\mathrm{gen}}(\mu) = h = g^a$ with known a. The first partition of messages are signable for the reduction. It uses $(0, w)$ as the secret key to issue an Okamoto signature, which by the witness indistinguishability, is identical to what an honest user produces using $(x, 0)$. Forgery on the second partition of messages can be used to find the discrete logarithm of X.

Signature simulation just does the same thing as normally signing. The simulator does not have to carefully embed some R_* into honest users' commitment. Instead, it normally generates honest users' signing randomnesses and commitments. They decide the randomness and commitment R_* for the simulator to sign under X. The simulator decides honest users' responses based on (z_*, y_*), which it obtains by responding to the challenge with Okamoto secret key $(0, w)$.

Let us examine the cost. The simulator has exactly the same features as a normal signer: the committed randomness is known, and any challenge can be handled. None of the two problems come up, and we already reach adaptive security, statelessly and without extra rounds!

1.3 Related Work

The reader is referred to [KRT24a] and [BDLR24] for comprehensive reviews of related work. Here we mention two important topics that we do not focus on in this work.

Robustness and Identifiable Abort. This work does not consider robustness. A robust threshold signature scheme outputs a valid signature even with malicious users participating. This property is desirable in some distributed systems but usually requires higher complexity. See recently proposed robust threshold Schnorr schemes in [RRJ+22, Sho23, BHK+24, GS24, BLSW24].

Identifiable abort provides a similar functionality: tracing the malicious users that prevent the protocol from issuing a valid signature. Prior works that we mentioned in Table 1, except [KRT24a], all support non-interactive identifiable abort, since users output verifiable signature shares in the protocol. In contrast, signature shares become unverifiable after masking, so [KRT24a] and this work lose identifiable abort.

Distributed Key Generation. This work assumes a trusted dealer for key distribution, like prior works including [TZ23, CKM23, BLT+24, KRT24a]. In general, the trusted dealer can be replaced with multi-party computation. Specially designed distributed key generation for DL-based schemes was studied in [Ped92, CGJ+99, JL00, GJKR07, KMS20, DYX+22, KGS23].

2 Preliminaries

Notations. For an integer n, $[n]$ denotes $\{1, 2, \ldots, n\}$. For vectors of group elements $\left[\begin{smallmatrix} X_1 \\ X_2 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} Y_1 \\ Y_2 \end{smallmatrix}\right]$, the multiplication is element-wise: $\left[\begin{smallmatrix} X_1 \\ X_2 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} Y_1 \\ Y_2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} X_1 Y_1 \\ X_2 Y_2 \end{smallmatrix}\right]$. For scalar a, the exponentiation is also

```
$q$-AOMDL$^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$
1: $\mathsf{ctr} := 0$
2: $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
3: for $i \in [q+1]$ do
4:     $x_i \leftarrow \mathbb{Z}_p$
5:     $X_i := g^{x_i}$
6: return $[\![\mathsf{A}(\mathbb{G}, p, g, X_1, \ldots, X_{q+1})^{\textsc{DLog}} = (x_1, \ldots, x_{q+1})]\!]$

$\textsc{DLog}(X, \alpha_1, \ldots, \alpha_{q+1})$
7: if $\mathsf{ctr} = q \vee X \neq \prod_{i=1}^{q+1} X_i^{\alpha_i}$ then
8:     return $\perp$
9: $\mathsf{ctr} := \mathsf{ctr} + 1$
10: return $\sum_{i=1}^{q+1} \alpha_i x_i$
```

Figure 1: Game q-$\mathsf{AOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$.

element-wise: $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}^a = \begin{bmatrix} X_1^a \\ X_2^a \end{bmatrix}$.

Algebraic Assumptions. A group generation algorithm GrGen takes a security parameter 1^λ as input and outputs a set of group parameters $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of order p, and g is a generator.

Definition 1 (DL)**.** The discrete logarithm (DL) problem is (τ, ε)-hard for group generation algorithm GrGen if for all algorithm A that runs in time at most $\tau(\lambda)$,

$$\Pr[\mathsf{A}(\mathbb{G}, p, g, g^x) = x : (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda); x \leftarrow \mathbb{Z}_p] \leq \varepsilon(\lambda).$$

The algebraic one-more discrete logarithm (AOMDL) problem asks to solve $q+1$ discrete logarithms with q calls to the DLog oracle. "Algebraic" means that queries to DLog oracle are required to be represented as a linear combination of the input elements. This allows a challenger to efficiently implement the DLog oracle, so the AOMDL assumption is falsifiable.

Definition 2 (AOMDL)**.** The q-algebraic one-more discrete logarithm (q-AOMDL) problem is (τ, ε)-hard for group generation algorithm GrGen if for all algorithm A that runs in time at most $\tau(\lambda)$, $\Pr[q\text{-}\mathsf{AOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda) = 1] \leq \varepsilon(\lambda)$, where the game q-$\mathsf{AOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$ is defined in Fig. 1.

Shamir's Secret Sharing. Shamir's secret sharing scheme [Sha79] allows sharing secrets over $\mathbb{Z}_p$. On input the number of users $n < p$, the threshold $t \leq n$, and the secret $x \in \mathbb{Z}_p$, the sharing algorithm $\mathsf{Share}(n, t, x)$ samples t coefficients $\alpha_0, \ldots, \alpha_{t-1} \leftarrow \mathbb{Z}_p$ and returns the secret shares $\{x_i\}_{i \in [n]}$ where $x_i = \sum_{j=0}^{T-1} \alpha_j i^j$. On input $\{x_i\}_{i \in \mathcal{S}}$ where $|\mathcal{S}| = T$, the reconstruction algorithm $\mathsf{Rec}(\{x_i\}_{i \in \mathcal{S}})$ recovers x by polynomial interpolation. In particular, $x = \sum_{i \in \mathcal{S}} \lambda_{\mathcal{S},i} x_i$, where the Lagrange coefficient for $\mathcal{S}$ is defined as $\lambda_{\mathcal{S},i} = \prod_{j \in \mathcal{S}, j \neq i} j/(i-j)$. Shamir's secret sharing scheme is perfectly secure.

General Forking Lemma. We state the general forking lemma by Bellare and Neven [BN06].

Lemma 1 (General forking lemma [BN06])**.** *Fix an interger $q \geq 1$, a set $\mathcal{H}$ of size $|\mathcal{H}| \geq 2$. Let A be a randomized algorithm that takes as inputs X, $h_1, \ldots, h_q$, takes as random coin tosses from set $\mathcal{R}$, and outputs a tuple (I, Y) where $I \in \{0, \ldots, q\}$, and Y is what we call "side output". Let $\mathcal{D}_X$ be an unspecified distribution. Let*

$$\varepsilon = \Pr[I \geq 1 : X \leftarrow \mathcal{D}_X; h_1, \ldots, h_q \leftarrow \mathcal{H}; (I, Y) \leftarrow \mathsf{A}(X, h_1, \ldots, h_q)].$$

```
$\underline{\mathsf{fork}_\mathsf{A}(X)}$
$\rho \leftarrow \mathcal{R}$
$h_1 \ldots, h_q \leftarrow \mathcal{H}$
$(I, Y) \leftarrow \mathsf{A}(X, h_1, \ldots, h_q; \rho)$
**if** $I = 0$ **then**
    **return** $\perp$
$h'_I, \ldots, h'_q \leftarrow \mathcal{H}$
$(I', Y') \leftarrow \mathsf{A}(X, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \rho)$
**if** $I \neq I' \vee h_I = h'_I$ **then**
    **return** $\perp$
**return** $(I, Y, Y')$
```

Figure 2: Algorithm $\mathsf{fork}_\mathsf{A}(X)$ in Lemma 1.

```
$\underline{\mathsf{Exec}(\mathcal{S}, \{\mathsf{sk}_i\}_{i\in\mathcal{S}}, \mu, \mathsf{sid})}$
**for** $j \in \mathcal{S}$ **do**
    $(\mathsf{st}_j, \mathsf{msg}_j) \leftarrow \mathsf{Sign}(\mathcal{S}, j, \mathsf{sk}_j, \mu, \mathsf{sid})$
**for** $j \in \mathcal{S}$ **do**
    $\mathsf{msg}'_j \leftarrow \mathsf{Sign}'(\mathcal{S}, j, \mathsf{sk}_j, \mu, \mathsf{sid}, \mathsf{st}_j, \{\mathsf{msg}_i\}_{i\in\mathcal{S}})$
**return** $\sigma \leftarrow \mathsf{Combine}(\mathcal{S}, \mu, \mathsf{sid}, \{\mathsf{msg}_i\}_{i\in\mathcal{S}}, \{\mathsf{msg}'_i\}_{i\in\mathcal{S}})$
```

Figure 3: Procedure Exec for defining the completeness of threshold signature schemes.

Define the forking algorithm fork_A *with respect to* A *as in Fig. 2. Let*

$$\varepsilon' = \Pr[\mathsf{fork}_\mathsf{A}(X) \neq \perp : X \leftarrow \mathcal{D}_X].$$

Then

$$\varepsilon' \geq \varepsilon\left(\frac{\varepsilon}{q} - \frac{1}{|\mathcal{H}|}\right).$$

2.1 Threshold Signatures

We define stateful and stateless two-round threshold signature schemes. Text in orange is only for stateful schemes; text in purple is only for stateless schemes. The key generation in our syntax is assumed to be trusted but can be implemented by a distributed protocol in practice. The signing protocol is described by three algorithms that each signer runs locally. The first stage of signing returns a secret state and a protocol message. After exchanging protocol messages with each other, each signer runs the second stage of signing to obtain the second protocol message. The last stage is a combining algorithm that takes all the previous protocol messages as inputs and outputs the final threshold signature. The combining algorithm does not take any secret state as input and can be executed by any designated party. An honest execution of the signing protocol is described as the procedure Exec in Fig. 3, which we use to define completeness.

Definition 3 (Two-Round Threshold Signature Schemes)**.** A two-round threshold signature scheme TS consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{par}$: On input the security parameter 1^λ, the setup algorithm Setup outputs global parameters par. All algorithms related to TS implicitly take par as input.

- $\mathsf{KGen}(n,t) \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$: On inputs an allowable number of user n and a threshold t, the key generation algorithm KGen outputs a public key pk and n secret key shares $\{\mathsf{sk}_i\}_{i\in[n]}$. Each secret key share implicitly indicates n and t.
- The signing protocol consists of three algorithms:
  - $\mathsf{Sign}(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid}) \to (\mathsf{st}, \mathsf{msg})$: On inputs a set of signer indices $\mathcal{S}$, a signer index k, a secret key share sk_k, a session identifier sid, and a message μ, the first-round signing algorithm Sign outputs a state st and a protocol message msg.
  - $\mathsf{Sign}'(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid}, \mathsf{st}, \mathcal{M}) \to \mathsf{msg}'$: On inputs a set of signer indices $\mathcal{S}$, a signer index k, a secret key share sk_k, a session identifier sid, a message μ, a state st and a set of protocol messages $\mathcal{M}$, the second-round signing algorithm Sign' outputs a protocol message msg'.
  - $\mathsf{Combine}(\mathcal{S}, \mu, \mathsf{sid}, \mathcal{M}, \mathcal{M}') \to \sigma$: On inputs a set of signer indices $\mathcal{S}$, a message μ, a session identifier sid, and two sets of protocol messages $\mathcal{M}$, $\mathcal{M}'$, the combining algorithm outputs a signature σ.
- $\mathsf{Vf}(\mathsf{pk}, \mu, \sigma) \to 0/1$: On inputs a public key pk, a message μ, and a signature σ, the verification algorithm Vf outputs 0 or 1.

For the completeness, we require that for all n and t that the scheme supports, all $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \in \mathsf{KGen}(n,t)$, all subset of signers $\mathcal{S} \subseteq [n]$ with $|\mathcal{S}| = t$, all messages μ and all identifiers sid, we have

$$\Pr[\mathsf{Vf}(\mathsf{pk}, \mu, \sigma) = 1 \mid \sigma \leftarrow \mathsf{Exec}(\mathcal{S}, \{\mathsf{sk}_i\}_{i\in\mathcal{S}}, \mu, \mathsf{sid})] = 1,$$

where the procedure Exec, modeling an honest execution of TS, is defined in Fig. 3.

Security Model.

We define the adaptive security of two-round threshold signature schemes. The adversary is given the public parameter, a public key, and oracles that allow corrupting and querying signatures from the signers. It can concurrently open many signing sessions with each signer. When the adversary corrupts a signer, it learns not only the secret key share but also all secret states st the signer output in the first round of each signing session, including completed sessions. The target of the adversary is to forge a signature σ^* on a message μ^* that it has never made signing queries on.

For stateless schemes, we maintain a counter per signer to set up session identifiers to keep track of signing sessions with the signer. We remark that this identifier appears in the security game but is not used in the scheme, unlike stateful schemes. For stateful schemes, a signing query is valid only if adversary provides a distinct session identifier. We store all previous identifiers for each user to check distinctness.

Our security definition models an adversary that fully controls the channels. In particular, we do *not* assume *authenticated channels*. As the inputs to the second-stage signing oracle, the adversary decides every protocol message, even on behalf of every honest party. In fact, honest signers will not realize the signing sessions of other signers. The adversary need not care about the consistency of protocol messages.

We also do *not* assume *secure erasure*. When the adversary corrupts a signer, it learns all randomness used in the signer's previous signing sessions. This implicitly requires the secret state st, output by the first-round signing algorithm Sign, to contain all randomness used during that session. While this not guaranteed by all schemes, it is true for our schemes, so we omit the corresponding definitional complication.

Definition 4 (Adaptive Security of Threshold Signature Schemes.)**.** A threshold signature scheme TS is (q_s, τ, ε)-adaptively secure if for all security parameter λ, all adversary A that makes at most $q_s = q_s(\lambda)$ signing queries and is of running time at most $\tau = \tau(\lambda)$, for all allowable n and t, $\Pr[\mathsf{adp\text{-}UF}^{\mathsf{A}}_{\mathsf{TS}}(\lambda, n, t) = 1] \le \varepsilon = \varepsilon(\lambda)$, where the security game $\mathsf{adp\text{-}UF}$ is defined in Fig. 4.

```
$\underline{\mathsf{adp\text{-}UF}^{\mathsf{A}}_{\mathsf{TS}}(\lambda, n, t)}$
$\mathcal{Q} := \emptyset, \mathcal{C} := \emptyset$
for $i \in [n]$ do
    $\mathsf{ctr}_i := 0$
    $\mathcal{I}_i := \emptyset$
$\mathsf{par} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i \in [n]}) \leftarrow \mathsf{KGen}(n, t)$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\textsc{Sign},\textsc{Sign}',\textsc{Corr}}(\mathsf{pk})$
return $[\![\mu^* \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, \mu^*, \sigma^*) = 1]\!]$

$\underline{\textsc{Corr}(k)}$
if $|\mathcal{C}| \geq t - 1$ then
    return $\perp$
$\mathcal{C} := \mathcal{C} \cup \{k\}$
return $(\mathsf{sk}_k, \{\mathsf{st}_{k,i}\}_{i \in [\mathsf{ctr}_k]}, \{\mathsf{st}_{k,i}\}_{i \in \mathcal{I}_k})$

$\underline{\textsc{Sign}(\mathcal{S}, k, \mu, \mathsf{sid})}$
if $k \in \mathcal{C} \vee \mathsf{sid} \in \mathcal{I}_k$ then
    return $\perp$
$\mathcal{Q} := \mathcal{Q} \cup \{\mu\}, \mathcal{I}_k := \mathcal{I}_k \cup \{\mathsf{sid}\}$
$\mathsf{ctr}_k := \mathsf{ctr}_k + 1$
$\mathsf{sid} := \mathsf{ctr}_k$
$(\mathsf{st}, \mathsf{msg}) \leftarrow \mathsf{Sign}(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid})$
$\mathsf{st}_{k,\mathsf{sid}} := \mathsf{st}$
$(\mathcal{S}_{k,\mathsf{sid}}, \mu_{k,\mathsf{sid}}) := (\mathcal{S}, \mu)$
$\mathsf{round}_{k,\mathsf{sid}} := 1$
return $\mathsf{msg}$

$\underline{\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})}$
if $k \in \mathcal{C} \vee \mathsf{round}_{k,\mathsf{sid}} \neq 1$ then
    return $\perp$
$(\mathcal{S}, \mu) := (\mathcal{S}_{k,\mathsf{sid}}, \mu_{k,\mathsf{sid}})$
$\mathsf{msg}' \leftarrow \mathsf{Sign}'(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid}, \mathsf{st}_{k,\mathsf{sid}}, \mathcal{M})$
$\mathsf{round}_{k,\mathsf{sid}} := 2$
return $\mathsf{msg}'$
```

Figure 4: The adp-UF security for defining the adaptive security of threshold signature schemes.

3 FROST-Mask

3.1 Scheme

In this section, we present our stateful two-round threshold Schnorr signature scheme FROST-Mask. The scheme is described in Fig. 5. With group parameters $(\mathbb{G}, p, g)$ chosen in the setup algorithm, the scheme uses hash functions $\mathsf{H}_{\text{mask}} : \{0,1\}^* \to \mathbb{G}^2$, $\mathsf{H}'_{\text{mask}} : \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_{\text{comb}} : \{0,1\}^* \to \mathbb{Z}_p$, $\mathsf{H}_{\text{chal}} : \{0,1\}^* \to \mathbb{Z}_p$. Based Shamir's secret sharing scheme, FROST-Mask supports all practical choices of (n, t). It is rather direct to verify the completeness of FROST-Mask. Note that in an honestly executed signing session, $\prod_{i \in \mathcal{S}} \Phi_i = [\,^{1}_{1}\,]$, and $\sum_{i \in \mathcal{S}} \Delta_i = 0$, and therefore $\prod_{i \in \mathcal{S}} \tilde{R}_{1,i} \tilde{R}^b_{2,i} = \prod_{i \in \mathcal{S}} R_{1,i} R^b_{2,i}$, $\sum_{i \in \mathcal{S}} \tilde{z}_i = \sum_{i \in \mathcal{S}} z_i$.

As shown in [KRT24b], such a stateful scheme can be generically transformed into a stateless one with one more round. In the additional round at the beginning of the signing protocol, each signer samples a random string. The concatenation of those strings plays the role of the session identifier.

3.2 Security

Theorem 2. *If* $2q_{\text{s}}$-AOMDL *is* $(\tau_{\text{aomdl}}, \varepsilon_{\text{aomdl}})$*-hard for* GrGen*, then* FROST-Mask *is* $(q_{\text{s}}, \tau_{\text{uf}}, \varepsilon_{\text{uf}})$*-adaptively secure in the random oracle model against any adversary that makes at most* q_{h} *hash queries in total, where essentially* $\tau_{\text{aomdl}} \approx 2\tau_{\text{uf}}$*, and*

$$\varepsilon_{\text{aomdl}} \geq \frac{1}{q}\left(\varepsilon_{\text{uf}}^2 - \frac{n^4 + q_{\text{h}}}{2^{\lambda - 1}}\right) - \frac{4q^2 + 1}{p},$$

where $q = q_{\text{s}} + q_{\text{h}} + 1$.

Proof (Part I). Assume that an adversary A $(q_{\text{s}}, \tau_{\text{uf}}, \varepsilon_{\text{uf}})$-breaks the adp-UF of FROST-Mask making at most q_{h} hash queries in total. The first part of this proof contains a series of hybrid games. Let

```
$\underline{\mathsf{Setup}(1^\lambda)}$
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\textbf{return } (\mathbb{G}, p, g)$

$\underline{\mathsf{KGen}(n, t)}$
$x \leftarrow \mathbb{Z}_p$
$X := g^x$
$\{x_i\}_{i \in [n]} \leftarrow \mathsf{Share}(n, t, x)$
$\textbf{for } i \neq j \in [n] \textbf{ do}$
    $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^\lambda$
$\mathsf{pk} := X$
$\textbf{for } i \in [n] \textbf{ do}$
    $\mathsf{sk}_i := (x_i, \{\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i}\}_{j \in [n] \setminus \{i\}})$
$\textbf{return } (\mathsf{pk}, \{\mathsf{sk}_i\}_{i \in [n]})$

$\underline{\mathsf{Sign}(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid})}$
$(x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i \in [n] \setminus \{k\}}) := \mathsf{sk}_k$
$r_{1,k},\, r_{2,k} \leftarrow \mathbb{Z}_p$
$\begin{bmatrix} R_{1,k} \\ R_{2,k} \end{bmatrix} := \begin{bmatrix} g^{r_{1,k}} \\ g^{r_{2,k}} \end{bmatrix}$
$\Phi_k := \prod_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{seed}_{k,i}, \mathsf{sid}) \cdot$
    $\prod_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{seed}_{i,k}, \mathsf{sid})^{-1}$
$\begin{bmatrix} \tilde{R}_{1,k} \\ \tilde{R}_{2,k} \end{bmatrix} := \begin{bmatrix} R_{1,k} \\ R_{2,k} \end{bmatrix} \cdot \Phi_k$
$\mathsf{st} := (r_{1,k}, r_{2,k})$
$\mathsf{msg} := (\tilde{R}_{1,k}, \tilde{R}_{2,k})$
$\textbf{return } (\mathsf{st}, \mathsf{msg})$

$\underline{\mathsf{Sign}'(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{sid}, \mathsf{st}, \mathcal{M})}$
$(x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i \in [n] \setminus \{k\}}) := \mathsf{sk}_k$
$(r_{1,k}, r_{2,k}) := \mathsf{st}$
$\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
$\textbf{for } i \in \mathcal{S} \textbf{ do}$
    $(\tilde{R}_{1,i}, \tilde{R}_{2,i}) := \mathsf{msg}_i$
$\mathsf{sinf} := (\mathsf{sid}, \mu, \mathcal{M})$
$b := \mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mathsf{sinf})$
$R := \prod_{i \in \mathcal{S}} \tilde{R}_{1,i} \tilde{R}_{2,i}^b$
$c := \mathsf{H}_{\mathrm{chal}}(\mu, R)$
$z_k := r_{1,k} + b r_{2,k} + c \cdot \lambda_{\mathcal{S},k} \cdot x_k$
$\Delta_k := \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{seed}_{k,i}, \mathsf{sinf}) -$
    $\sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{seed}_{i,k}, \mathsf{sinf})$
$\tilde{z}_k := z_k + \Delta_k$
$\textbf{return } \tilde{z}_k$

$\underline{\mathsf{Combine}(\mathcal{S}, \mathsf{sid}, \mu, \mathcal{M}, \mathcal{M}')}$
$\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
$\{\mathsf{msg}'_i\}_{i \in \mathcal{S}} := \mathcal{M}'$
$\textbf{for } i \in \mathcal{S} \textbf{ do}$
    $(\tilde{R}_{1,i}, \tilde{R}_{2,i}) := \mathsf{msg}_i$
    $\tilde{z}_i := \mathsf{msg}'_i$
$\mathsf{sinf} := (\mathsf{sid}, \mu, \mathcal{M})$
$b := \mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mathsf{sinf})$
$R := \prod_{i \in \mathcal{S}} \tilde{R}_{1,i} \tilde{R}_{2,i}^b$
$c := \mathsf{H}_{\mathrm{chal}}(\mu, R)$
$z := \sum_{i \in \mathcal{S}} \tilde{z}_i$
$\textbf{return } (R, z)$

$\underline{\mathsf{Vf}(\mathsf{pk}, \mu, \sigma)}$
$X := \mathsf{pk}$
$(R, z) := \sigma$
$c := \mathsf{H}_{\mathrm{chal}}(\mu, R)$
$\textbf{return } [\![ g^z = R X^c ]\!]$
```

Figure 5: FROST-Mask.

G_0 be the security game $\mathsf{adp\text{-}UF}^{\mathsf{A}}_{\mathsf{FROST\text{-}Mask}}(\lambda, n, t)$. Let ε_i denote the winning probability of A in G_i. We will end up with a game that can be simulated by a reduction that solves AOMDL. We provide the pseudocode description and a quick checklist of these games in Appendix A.

We make the variables used in the signing oracles global, so they can be referenced in other oracle calls. We rename them to avoid ambiguity. Specifically:

- We rename the variables $\begin{bmatrix} r_{1,k} \\ r_{2,k} \end{bmatrix}$, $\begin{bmatrix} R_{1,k} \\ R_{2,k} \end{bmatrix}$, $\begin{bmatrix} \tilde{R}_{1,k} \\ \tilde{R}_{2,k} \end{bmatrix}$, and Φ_k referenced in $\textsc{Sign}(\mathcal{S}, k, \mu, \mathsf{sid})$ to $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,k} \\ r_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$, $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$, $\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$, and $\Phi_{\mathcal{S},\mathsf{sid},k}$, respectively. Since for every user, each session has a distinct sid, the added index $(\mathcal{S}, \mathsf{sid})$ eliminates ambiguity.
- We rename the variables b, c, z_k, $\tilde{z}_k$, and Δ_k referenced in $\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})$ to $b_{\mathcal{S},\mathsf{sinf}}$, $c_{\mathcal{S},\mathsf{sinf}}$, $z_{\mathcal{S},\mathsf{sinf},k}$, $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$, and $\Delta_{\mathcal{S},\mathsf{sinf},k}$, where sinf is the one referenced in the same oracle call, i.e. $\mathsf{sinf} = (\mathsf{sid}, \mu, \mathcal{M})$ where $\mu = \mu_{\mathcal{S},\mathsf{sid}}$. Similarly, the added index $(\mathcal{S}, \mathsf{sinf})$ is distinct for every user.

  We need to be careful with b and c. They are renamed to $b_{\mathcal{S},\mathsf{sinf}}$ and $c_{\mathcal{S},\mathsf{sinf}}$, where the index does not contain k. Hence, those referenced in $\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})$ for different k may become the same global variables. We need to ensure that they do not conflict. This is true, because b and c is determined by $(\mathcal{S}, \mathsf{sinf})$: b is the hash of $(\mathcal{S}, \mathsf{sinf})$, and then c can be computed from sinf and b. In $\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})$ with different k, as long as $(\mathcal{S}, \mathsf{sinf})$ is the same, then b and c have the same value, so renaming them to $b_{\mathcal{S},\mathsf{sinf}}$ and $c_{\mathcal{S},\mathsf{sinf}}$ does not cause conflict.

We will use the terms "define" and "initialize" interchangeably for variables. At some point in the proof, we will define a variable but only assign it a pending value ?. A variable with a pending value should be distinguished from an undefined one.

Defer Sampling the Seeds.

The purpose of the first several games is to defer sampling the seeds.

G_1. In this game, we ensure the distinctness of seeds. After generating the seeds $\{\mathsf{seed}_{i,j}\}_{i \neq j \in [n]}$, G_1 aborts if the seeds are not distinct. The probability that G_1 aborts is at most $n^4/2^\lambda$, so

$$\varepsilon_1 \geq \varepsilon_0 - \frac{n^4}{2^\lambda}.$$

G_2. In this game, we modify how values are assigned to fresh hash queries to H_{mask} and $\mathsf{H}'_{\text{mask}}$ and introduce two families of indexed global variables $e_{\mathcal{S},\mathsf{sid},i,j}$ and $d_{\mathcal{S},\mathsf{sinf},i,j}$. Initially $e_{\mathcal{S},\mathsf{sid},i,j}$ and $d_{\mathcal{S},\mathsf{sinf},i,j}$ are undefined. When $e_{\mathcal{S},\mathsf{sid},i,j}$ or $d_{\mathcal{S},\mathsf{sinf},i,j}$ is referenced for the first time, the game initializes it to a uniformly random value. They are only referenced for the assignment to fresh hash queries, which we change to the following:

- When a fresh query in the form of $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,j})$ is made (internally by the game or externally by A), program $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,j}) := e_{\mathcal{S},\mathsf{sid},i,j}$.
- When a fresh query in the form of $\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$ is made, program $\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j}) := d_{\mathcal{S},\mathsf{sinf},i,j}$.

Recall that it has been ensured by G_1 that the seeds are distinct, so from $\mathsf{seed}_{i,j}$ being queried, G_2 can indeed determine the corresponding (i, j).

In this game, $e_{\mathcal{S},\mathsf{sid},i,j}$ and $d_{\mathcal{S},\mathsf{sinf},i,j}$ are just intermediate variables for programming the random oracle. The change in this game is invisible to A, so

$$\varepsilon_2 = \varepsilon_1.$$

G_3. This game changes the way that the signing oracles calculate the mask $\Phi_{\mathcal{S},\mathsf{sid},k}$ and $\Delta_{\mathcal{S},\mathsf{sinf},k}$. In response to query $\textsc{Sign}(\mathcal{S}, k, \mu, \mathsf{sid})$, we replace

$$\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S} \setminus \{k\}} (\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{k,i}) / \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,k}))$$

with

$$\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S} \setminus \{k\}} (e_{\mathcal{S},\mathsf{sid},k,i} / e_{\mathcal{S},\mathsf{sid},i,k}).$$

In response to query $\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})$ with signer group $\mathcal{S}$, we replace

$$\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \{k\}} (\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{k,i}) - \mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,k}))$$

with

$$\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \{k\}} (d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k}).$$

The signing oracles do not make internal hash queries to calculate the masks. We remark that if some $e_{\mathcal{S},\mathsf{sid},i,j}$ (resp. $d_{\mathcal{S},\mathsf{sinf},i,j}$) is referenced for the first time here, it will be uniformly sampled, while the corresponding $\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,j})$ (resp. $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$) will not be programmed at this time. Still it will be programmed to $\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,j}) := e_{\mathcal{S},\mathsf{sid},i,j}$ (resp. $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j}) := d_{\mathcal{S},\mathsf{sinf},i,j}$) when it is queried. The change in this game is invisible to A, so

$$\varepsilon_3 = \varepsilon_2.$$

G_4. In this game, seeds are not sampled in the setup phase. Instead, G_4 only samples $\mathsf{seed}_{i,j}$ when it has to be output for the corruption of user i or j. The game still aborts when some seeds collide.

G_3 has made the signing oracles not make $\mathsf{H}_{\mathrm{mask}}$ and $\mathsf{H}'_{\mathrm{mask}}$ queries, and therefore not use any seed. G_3 differs from G_4 only if A queries some $\mathsf{seed}_{i,j}$ to $\mathsf{H}_{\mathrm{mask}}$ or $\mathsf{H}'_{\mathrm{mask}}$ before user i or j is corrupted. In this case G_3 programs some $e_{\mathcal{S},\mathsf{sid},i,j}$ or $d_{\mathcal{S},\mathsf{sinf},i,j}$ to the random oracle, while G_4 programs a uniform value since $\mathsf{seed}_{i,j}$ has not been generated. The probability that A queries a seed that has not been output is at most $q_{\mathrm{h}}/2^{\lambda}$, so

$$\varepsilon_4 \geq \varepsilon_3 - \frac{q_{\mathrm{h}}}{2^{\lambda}}.$$

Defer Sampling Secret Key Shares.

The purpose of the following games is to defer the sampling of secret key shares to the corresponding corruption query. This is made possible by masking responses.

G_5. So far, $\Delta_{\mathcal{S},\mathsf{sinf},k}$ is only defined in $\textsc{Sign}'$ for uncorrupted user k. In this game, we additionally initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$ when answering $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{k,i})$ or $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,k})$ if user k is *corrupted*. Same as in $\textsc{Sign}'$, it is initialized to $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{\mathcal{S} \setminus \{k\}} (d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$, and also recall that $d_{\mathcal{S},\mathsf{sinf},i,j}$ will be initialized to a uniformly random value when it is referenced for the first time here.

We note that this change is invisible to A: $\Delta_{\mathcal{S},\mathsf{sinf},k}$ initialized in $\mathsf{H}'_{\mathrm{mask}}$ will never be referenced. Its initialization just causes some related $d_{\mathcal{S},\mathsf{sinf},i,j}$ being initialized, from the same distribution as in G_4.

$$\varepsilon_5 = \varepsilon_4.$$

G_6. In this game, we reverse the order of generating $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $d_{\mathcal{S},\mathsf{sinf},i,j}$, both in $\textsc{Sign}'$ and $\mathsf{H}'_{\mathrm{mask}}$. Instead of setting $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$, we directly sample $\Delta_{\mathcal{S},\mathsf{sinf},k}$ from its distribution. Then we initialize those $d_{\mathcal{S},\mathsf{sinf},i,j}$ that are related to $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and undefined.[2] The initialization of $d_{\mathcal{S},\mathsf{sinf},i,j}$ is modified accordingly: they are uniformly sampled under the constraint that $\Delta_{\mathcal{S},\mathsf{sinf},k} = \sum_{i\in\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$. Equivalently, all but one undefined $d_{\mathcal{S},\mathsf{sinf},i,j}$ are uniformly sampled, and then the last one is determined by the constraint.

Now we specify how to initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$:

- If $\Delta_{\mathcal{S},\mathsf{sinf},i}$ has been initialized for all $i \in \mathcal{S}\setminus\{k\}$, set $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i\in\mathcal{S}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}$.
- Else, uniformly sample $\Delta_{\mathcal{S},\mathsf{sinf},k}$.

To show that G_6 is identical to G_5, it suffices to show that the above initialization of $\Delta_{\mathcal{S},\mathsf{sinf},k}$ indeed follows its distribution in G_5. Below we analyze this distribution in G_5.

If $\Delta_{\mathcal{S},\mathsf{sinf},i}$ has been initialized for all $i \in \mathcal{S}\setminus\{k\}$, then $d_{\mathcal{S},\mathsf{sinf},i,j}$ for all $i \neq j \in \mathcal{S}$ has been initialized. One can verify that $\Delta_{\mathcal{S},\mathsf{sinf},k}$ is determined by $\Delta_{\mathcal{S},\mathsf{sinf},k} = -\sum_{i\in\mathcal{S}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}$, since

$$\sum_{i\in\mathcal{S}}\Delta_{\mathcal{S},\mathsf{sinf},i} = \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}\setminus\{i\}}(d_{\mathcal{S},\mathsf{sinf},i,j} - d_{\mathcal{S},\mathsf{sinf},j,i}) = \sum_{i\neq j\in\mathcal{S}} d_{\mathcal{S},\mathsf{sinf},i,j} - \sum_{i\neq j\in\mathcal{S}} d_{\mathcal{S},\mathsf{sinf},j,i} = 0.$$

In the other case, there exists some j such that $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $\Delta_{\mathcal{S},\mathsf{sinf},j}$ are both undefined. We claim that this implies $d_{\mathcal{S},\mathsf{sinf},k,j}$ and $d_{\mathcal{S},\mathsf{sinf},j,k}$ are both undefined. Therefore, $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i\in\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$ is uniformly random, since $d_{\mathcal{S},\mathsf{sinf},k,j}$ and $d_{\mathcal{S},\mathsf{sinf},j,k}$ will be referenced for the first time and uniformly sampled.

To see the claim, note that G_5 only references $d_{\mathcal{S},\mathsf{sinf},i,j}$ when

- initializing $\Delta_{\mathcal{S},\mathsf{sinf},i}$ or $\Delta_{\mathcal{S},\mathsf{sinf},j}$;
- or programming $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$.

For the second case, since $\mathsf{seed}_{i,j}$ has been generated, user i or j must be corrupted. Hence, G_5 must assure at least one of $\Delta_{\mathcal{S},\mathsf{sinf},i}$ or $\Delta_{\mathcal{S},\mathsf{sinf},j}$ has been initialized when A first queries $\mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$, before it is programmed. Therefore, $d_{\mathcal{S},\mathsf{sinf},i,j}$ only get referenced for or after the initialization of $\Delta_{\mathcal{S},\mathsf{sinf},i}$ or $\Delta_{\mathcal{S},\mathsf{sinf},j}$. Now in our claim, we have $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $\Delta_{\mathcal{S},\mathsf{sinf},j}$ both undefined, so $d_{\mathcal{S},\mathsf{sinf},k,j}$ and $d_{\mathcal{S},\mathsf{sinf},j,k}$ have never been referenced and initialized.

In conclusion, we have

$$\varepsilon_6 = \varepsilon_5.$$

G_7. In this game, we change the way to initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$ in the case that $\Delta_{\mathcal{S},\mathsf{sinf},i}$ has been initialized for all $i \in \mathcal{S}\setminus\{k\}$. In G_6 it is calculated as $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i\in\mathcal{S}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}$. Our goal is to avoid referencing $\Delta_{\mathcal{S},\mathsf{sinf},i}$ if user i is uncorrupted. It will reference the secret key shares, while we also eusure that it only references x_i for corrupted user i.

Here user k can be corrupted or uncorrupted. We first discuss the case that k is corrupted. Note that if i is uncorrupted, $\Delta_{\mathcal{S},\mathsf{sinf},i}$ could only be initialized in the signing oracle, and it holds that

$$\Delta_{\mathcal{S},\mathsf{sinf},i} = \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - z_{\mathcal{S},\mathsf{sinf},i} = \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}}\cdot r_{\mathcal{S},\mathsf{sid},2,i} + c_{\mathcal{S},\mathsf{sinf}}\cdot\lambda_{\mathcal{S},i}\cdot x_i),$$

for the sid contained in sinf. By substituting into $\Delta_{\mathcal{S},\mathsf{sinf},k} = -\sum_{i\in\mathcal{S}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}$ the above equation, we can instead calculate $\Delta_{\mathcal{S},\mathsf{sinf},k}$ by

$$\begin{aligned}\Delta_{\mathcal{S},\mathsf{sinf},k} &= -\sum_{i\in\mathcal{S}\setminus\mathcal{C}}(\tilde{z}_{\mathcal{S},\mathsf{sinf},i} - (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}}\cdot r_{\mathcal{S},\mathsf{sid},2,i} + c_{\mathcal{S},\mathsf{sinf}}\cdot\lambda_{\mathcal{S},i}\cdot x_i)) - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}\\ &= \sum_{i\in\mathcal{S}\setminus\mathcal{C}}(r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}}\cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}}(x - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\lambda_{\mathcal{S},i}\cdot x_i)\\ &\quad - \sum_{i\in\mathcal{S}\setminus\mathcal{C}}\tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{2}$$

[2]We say $d_{\mathcal{S},\mathsf{sinf},i,j}$ is related to $\Delta_{\mathcal{S},\mathsf{sinf},k}$ if $d_{\mathcal{S},\mathsf{sinf},i,j}$ appears in the equation $\Delta_{\mathcal{S},\mathsf{sinf},k} = \sum_{\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$, i.e., $(i,j) \in \{(k,i)\}_{i\in\mathcal{S}\setminus\{k\}} \cup \{(i,k)\}_{i\in\mathcal{S}\setminus\{k\}}$.

For the second equation, we used the reconstruction of Shamir's secret sharing $x = \sum_{i \in \mathcal{S}} \lambda_{\mathcal{S},i} x_i$.

For uncorrupted user k, we do a similar substitution and get

$$\begin{aligned}\Delta_{\mathcal{S},\mathsf{sinf},k} &= -\sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} (\tilde{z}_{\mathcal{S},\mathsf{sinf},i} - (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} r_{\mathcal{S},\mathsf{sid},2,i} + c_{\mathcal{S},\mathsf{sinf}} \cdot \lambda_{\mathcal{S},i} \cdot x_i)) - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i} \\ &= \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} (r_{\mathcal{S},\mathsf{sid},1,i} + b r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}} (x - \sum_{i \in \mathcal{S} \cap \mathcal{C} \cup \{k\}} \lambda_{\mathcal{S},i} \cdot x_i) \\ &\quad - \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{3}$$

This game just uses alternative ways to calculate $\Delta_{\mathcal{S},\mathsf{sinf},k} = -\sum_{i \in \mathcal{S} \setminus \{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$. The change is invisible, so

$$\varepsilon_7 = \varepsilon_6.$$

G_8. For any honest user k, $\Delta_{\mathcal{S},\mathsf{sinf},k}$ can only be initialized in the signing oracle. Since sinf contains sid, and each signing session has a distinct $(\mathcal{S}, \mathsf{sid})$, we know that $(\mathcal{S}, \mathsf{sinf})$ are distinct across different signing sessions for a single user. Therefore, each honest user k must initialize a $\Delta_{\mathcal{S},\mathsf{sinf},k}$ (instead of referencing an already defined one) in each signing session.

In this game, we swap the order of initializing $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and generating masked response $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$ in the signing oracle. In G_7, we first initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and then compute $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} := z_{\mathcal{S},\mathsf{sinf},k} + \Delta_{\mathcal{S},\mathsf{sinf},k}$. Now in G_8, we first sample $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$ from its distribution and then compute $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$.

Let us specify how to sample $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$. There are two cases. If there is some $i \in \mathcal{S} \setminus \{k\}$ such that $\Delta_{\mathcal{S},\mathsf{sinf},i}$ is undefined, then $\Delta_{\mathcal{S},\mathsf{sinf},k}$ here is uniformly random. Hence, we uniformly sample $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$.

Else, for all $i \in \mathcal{S} \setminus \{k\}$, $\Delta_{\mathcal{S},\mathsf{sinf},i}$ has been initialized. In this case, $\Delta_{\mathcal{S},\mathsf{sinf},k}$ is determined by Eq. (3). We have

$$\begin{aligned}\tilde{z}_{\mathcal{S},\mathsf{sinf},k} &= z_{\mathcal{S},\mathsf{sinf},k} + \Delta_{\mathcal{S},\mathsf{sinf},k} \\ &= r_{\mathcal{S},\mathsf{sid},1,k} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,k} + c_{\mathcal{S},\mathsf{sinf}} \cdot \lambda_{\mathcal{S},k} \cdot x_k \\ &\quad + \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}} (x - \sum_{i \in \mathcal{S} \cap \mathcal{C} \cup \{k\}} \lambda_{\mathcal{S},i} \cdot x_i) \\ &\quad - \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i} \\ &= \sum_{i \in \mathcal{S} \setminus \mathcal{C}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}} (x - \sum_{\mathcal{S} \cap \mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i) \\ &\quad - \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i},\end{aligned} \tag{4}$$

where sid in contained in sinf.

This game just changes the order of generating variables while keeps the distributions, which is invisible to A. Hence,

$$\varepsilon_8 = \varepsilon_7.$$

G_9. Recall that G_7 and G_8 has ensured that the generation of $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$ does not reference any $\Delta_{\mathcal{S},\mathsf{sinf},i}$ for uncorrupted user i. Now $\Delta_{\mathcal{S},\mathsf{sinf},k}$ only affects the view of A by constraining the initialization of related $d_{\mathcal{S},\mathsf{sinf},k,i}$ and $d_{\mathcal{S},\mathsf{sinf},i,k}$, which occurs right after the initialization of $\Delta_{\mathcal{S},\mathsf{sinf},k}$. Moreover, since G_6, $d_{\mathcal{S},\mathsf{sinf},i,j}$ is invisible to A until programmed into $\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$, which can happen only after the generation of $\mathsf{seed}_{i,j}$ when user i or j is corrupted.

Therefore, we can defer the *calculation* of $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$ and the initialization of related $d_{\mathcal{S},\mathsf{sinf},k,i}$ and $d_{\mathcal{S},\mathsf{sinf},i,k}$ from the signing oracle to the corruption of k. In particular, $\Delta_{\mathcal{S},\mathsf{sinf},k}$ is still initialized in the signing oracle, but to a pending value $\Delta_{\mathcal{S},\mathsf{sinf},k} :=?$. Only when user k gets corrupted, it is set to $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$, and we initialize related $d_{\mathcal{S},\mathsf{sinf},k,i}$ and $d_{\mathcal{S},\mathsf{sinf},i,k}$ that have not been initialized. We note that $d_{\mathcal{S},\mathsf{sinf},k,i}$ and $d_{\mathcal{S},\mathsf{sinf},i,k}$ may be initialized for the corruption of i rather than k if i is corrupted earlier than k. Changing the time of initializing variables $d_{\mathcal{S},\mathsf{sinf},i,j}$ does not change their distributions: they are still uniformly distributed under the constraint of related $\Delta_{\mathcal{S},\mathsf{sinf},i}$, whose distributions are the same as in G_8.

The modification in this game is invisible for A, so

$$\varepsilon_9 = \varepsilon_8.$$

G_{10}. Note that response $z_{\mathcal{S},\mathsf{sinf},k}$ is only referenced when calculating $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$, which was just deferred to the corruption of user k. It suffices to also defer the calculation of $z_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sid},1,k} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,k} + c_{\mathcal{S},\mathsf{sinf}} \cdot \lambda_{\mathcal{S},k} \cdot x_k$ until the corruption of user k, just before calculating $\Delta_{\mathcal{S},\mathsf{sinf},k}$. It holds that

$$\varepsilon_{10} = \varepsilon_9.$$

G_{11}. Note that secret key share x_k is only referenced when being output for the corruption of k, for calculating $z_{\mathcal{S},\mathsf{sinf},k}$, and in Eqs. (4) and (5). The calculation of $z_{\mathcal{S},\mathsf{sinf},k}$ was just deferred to the corruption of user k, and Eqs. (4) and (5) only reference x_i for corrupted user $i \in \mathcal{S} \cap \mathcal{C}$. Thus, it suffices to defer the sampling of x_k until user k gets corrupted, just before the calculation of $z_{\mathcal{S},\mathsf{sinf},k}$. Since at most $t-1$ users can be corrupted, by the security of Shamir's secret sharing, it suffices to sample x_k uniformly. It holds that

$$\varepsilon_{11} = \varepsilon_{10}.$$

Defer Sampling Committed Randomness.

The purpose of the following games is to defer the sampling of randomness $r_{\mathcal{S},\mathsf{sid},1,k}$, $r_{\mathcal{S},\mathsf{sid},2,k}$ until the corruption of user k. This is made possible by masking commitments.

G_{12}. Like G_5, we additionally initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ when answering $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{k,i})$ and $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,k})$ for corrupted user k. It is initialized to $\prod_{i \in \mathcal{S}\setminus\{k\}}(e_{\mathcal{S},\mathsf{sid},k,i}/e_{\mathcal{S},\mathsf{sid},i,k})$, where $e_{\mathcal{S},\mathsf{sid},k,i}$ and $e_{\mathcal{S},\mathsf{sid},i,k}$ are uniformly sampled if they are referenced for the first time. $\Phi_{\mathcal{S},\mathsf{sid},k}$ initialized in H_{mask} will never be referenced and does not change the distributions of $e_{\mathcal{S},\mathsf{sid},i,j}$. Hence, this change is invisible to A.

$$\varepsilon_{12} = \varepsilon_{11}.$$

G_{13}. Like G_6, we initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ by sampling directly from its distribution, then we initialize those related $e_{\mathcal{S},\mathsf{sid},k,i}$ and $e_{\mathcal{S},\mathsf{sid},i,k}$ under the constraint that $\Phi_{\mathcal{S},\mathsf{sid},k} = \prod_{i \in \mathcal{S}\setminus\{k\}}(e_{\mathcal{S},\mathsf{sid},k,i}/e_{\mathcal{S},\mathsf{sid},i,k})$. Specifically, we initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ as follows:

- If $\Phi_{\mathcal{S},\mathsf{sid},i}$ has been initialized for all $i \in \mathcal{S} \setminus \{k\}$, set $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$.
- Else, uniformly sample $\Phi_{\mathcal{S},\mathsf{sid},k}$.

The same argument as in G_6 can show that G_{13} is identical to G_{12}, so

$$\varepsilon_{13} = \varepsilon_{12}.$$

G_{14}. Like G_7, we change the way to initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ when $\Phi_{\mathcal{S},\mathsf{sid},i}$ has been defined for all $i \in \mathcal{S}\setminus\{k\}$. The goal is to avoid referencing $\Phi_{\mathcal{S},\mathsf{sid},i}$ if user i is uncorrupted.

User k can be corrupted or uncorrupted. We first discuss the case that k is corrupted. For any uncorrupted user i, $\Phi_{\mathcal{S},\mathsf{sid},i}$ could only be initialized in the signing oracle and used to mask the commitment, which satisfies $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix} = \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,i}\\ R_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix} \cdot \Phi_{\mathcal{S},\mathsf{sid},i}$. Substitute this equation into $\Phi_{\mathcal{S},\mathsf{sid},k} = \prod_{i\in\mathcal{S}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$, and we get

$$\Phi_{\mathcal{S},\mathsf{sid},k} = \prod_{i\in\mathcal{S}\setminus\mathcal{C}} (\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix} / \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,i}\\ R_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix})^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}. \tag{5}$$

For uncorrupted user k, the substitution gives

$$\Phi_{\mathcal{S},\mathsf{sid},k} = \prod_{i\in\mathcal{S}\setminus\mathcal{C}\setminus\{k\}} (\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix} / \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,i}\\ R_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix})^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}. \tag{6}$$

We solely change the way to calculate $\Phi_{\mathcal{S},\mathsf{sid},k}$ when it is already determined. The change is invisible to A, so

$$\varepsilon_{14} = \varepsilon_{13}.$$

G_{15}**.** The security game ensures that each session has a distinct $(\mathcal{S}, \mathsf{sid})$, so an honest user k must initialize a $\Phi_{\mathcal{S},\mathsf{sid},k}$ (instead of reference an already defined one) in each signing session. Like G_8, in this game we swap the order of initializing $\Phi_{\mathcal{S},\mathsf{sid},i}$ and generating masked commitment $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$. In G_{14}, we initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ and then compute $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} := \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,k}\\ R_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} \cdot \Phi_{\mathcal{S},\mathsf{sid},k}$. Now in G_{15}, we first sample $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$ from its distribution and then compute $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} / \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,k}\\ R_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$.

There are two cases. If there is some $i \in \mathcal{S} \setminus \{k\}$ such that $\Phi_{\mathcal{S},\mathsf{sid},i}$ is undefined, then $\Phi_{\mathcal{S},\mathsf{sid},k}$ here is uniformly random. Hence, we uniformly sample $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$.

Else, for all $i \in \mathcal{S}\setminus\{k\}$, $\Phi_{\mathcal{S},\mathsf{sid},i}$ has been initialized. In this case, $\Phi_{\mathcal{S},\mathsf{sid},k}$ is determined by Eq. (3). Hence, we have

$$\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} = \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,k}\\ R_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} \cdot \Phi_{\mathcal{S},\mathsf{sid},k} = \prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,i}\\ R_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix} \cdot \prod_{i\in\mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix}^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}. \tag{7}$$

We just swapped the order of generating variables while keeping the distributions, which is invisible. Thus,

$$\varepsilon_{16} = \varepsilon_{15}.$$

G_{16}**.** Recall that G_{14} made the initialization of $\Phi_{\mathcal{S},\mathsf{sid},k}$ not reference any $\Phi_{\mathcal{S},\mathsf{sid},i}$ for uncorrupted user i. G_{15} made the generation of $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$ not reference $\Phi_{\mathcal{S},\mathsf{sid},k}$. Now $\Phi_{\mathcal{S},\mathsf{sid},k}$ only affects the view of A by constraining the initialization of related $e_{\mathcal{S},\mathsf{sid},k,i}$ and $e_{\mathcal{S},\mathsf{sid},i,k}$, which occurs right after the initialization of $\Phi_{\mathcal{S},\mathsf{sid},k}$. Moreover, $e_{\mathcal{S},\mathsf{sid},i,j}$ is invisible to A until programmed into $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,j})$, which can happen only if user i or j is corrupted. Like G_9, it suffices to defer the calculation of $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} / \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,k}\\ R_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$ and the initialization of related $e_{\mathcal{S},\mathsf{sid},k,i}$ and $e_{\mathcal{S},\mathsf{sid},i,k}$ from the signing oracle to the corruption of user k or i. Specifically, $\Phi_{\mathcal{S},\mathsf{sid},k}$ is still initialized in the signing oracle, but to a pending value $\Phi_{\mathcal{S},\mathsf{sid},k} :=?$. Only when user k gets corrupted, it is set to $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix} / \begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,k}\\ R_{\mathcal{S},\mathsf{sid},2,k}\end{bmatrix}$, and then the related $e_{\mathcal{S},\mathsf{sid},k,i}$ and $e_{\mathcal{S},\mathsf{sid},i,k}$ are initialized. This is an invisible change, and we have

$$\varepsilon_{16} = \varepsilon_{15}.$$

G_{17}. Commitment $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$ is only referenced when calculating $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} / \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$ and in Eqs. (5) and (7). The calculation of $\Phi_{\mathcal{S},\mathsf{sid},k}$ was just deferred until user k gets corrupted. Eqs. (5) and (7) only need the product $\prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$. Therefore, it suffices to defer the calculation of single $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,k}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,k}} \end{bmatrix}$ until user k is corrupted. In Eqs. (5) and (7) we directly calculate the product by

$$\prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} := \begin{bmatrix} g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix}.$$

Specifically, we replace Eqs. (5) and (7) with

$$\Phi_{\mathcal{S},\mathsf{sid},k} = \begin{bmatrix} g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix} \cdot \prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Phi_{\mathcal{S},\mathsf{sid},i}^{-1}, \tag{8}$$

$$\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} = \begin{bmatrix} g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix} \cdot \prod_{i\in\mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}} \Phi_{\mathcal{S},\mathsf{sid},i}^{-1}, \tag{9}$$

respectively. This change is invisible, and

$$\varepsilon_{17} = \varepsilon_{16}.$$

G_{18}. In G_{17}, the committed randomness $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,k} \\ r_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$ is only referenced

- to calculate individual commitment $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$;
- to calculate individual response $z_{\mathcal{S},\mathsf{sinf},k}$;
- to calculate $\prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ in Eqs. (8) and (9);
- in Eqs. (2) and (4).

Recall that the calculation of individual commitment and response have been deferred until user k gets corrupted in G_{10} and G_{17}, respectively. In Eqs. (2), (4), (8) and (9), we only need the sum $\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$.

In this game, we make the following change. When $(\mathcal{S},\mathsf{sid})$ is queried to SIGN for the first time, we uniformly sample $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$. This is the sum $\sum_{i\in\mathcal{S}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ that we pre-sample here, where $\mathcal{C}_{\mathcal{S},\mathsf{sid}}$ is the current set of corrupted users.

When user $i \in \mathcal{C}_{\mathcal{S},\mathsf{sid}}$ is corrupted, we sample $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$. We will specify how to sample them later. When we need $\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ in Eqs. (2), (4), (8) and (9), since these can only occur after $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$ are sampled, we have $\mathcal{C}_{\mathcal{S},\mathsf{sid}} \subseteq \mathcal{C}$, and thus we can calculate it as

$$\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} = \sum_{i\in\mathcal{S}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} = \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}.$$

Specifically, Eqs. (2), (4), (8) and (9) are replaced with the following Eqs. (10) to (13), respectively:

$$\begin{aligned} \Delta_{\mathcal{S},\mathsf{sinf},k} = {} & r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x \\ & - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) - c_{\mathcal{S},\mathsf{sinf}} \cdot \sum_{i\in\mathcal{S}\cap\mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i \\ & - \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i} \end{aligned} \tag{10}$$

$$\begin{aligned}\tilde{z}_{\mathcal{S},\mathsf{sinf},k} = {} & r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x \\ & - \sum_{i \in \mathcal{S} \cap \mathcal{C} \setminus \mathcal{C}_{\mathcal{S},\mathsf{sid}}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) - c_{\mathcal{S},\mathsf{sinf}} \sum_{i \in \mathcal{S} \cap \mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i \\ & - \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}\end{aligned} \tag{11}$$

$$\Phi_{\mathcal{S},\mathsf{sid},k} = \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,*}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,*}} \end{bmatrix} \cdot \prod_{i \in \mathcal{S} \cap \mathcal{C} \setminus \mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S} \setminus \mathcal{C}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S} \cap \mathcal{C} \setminus \{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i} \tag{12}$$

$$\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} = \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,*}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,*}} \end{bmatrix} \cdot \prod_{i \in \mathcal{S} \cap \mathcal{C} \setminus \mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S} \cap \mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}. \tag{13}$$

It remains to specify how to sample $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ when user $i \in \mathcal{C}_{\mathcal{S},\mathsf{sid}}$ gets corrupted. It should be uniformly random under the constraint of the pre-sampled sum, i.e. $\sum_{\mathcal{S} \setminus \mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} = \begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$. Equivalently, for all but the last $i \in \mathcal{C}_{\mathcal{S},\mathsf{sid}}$, $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ is uniformly sampled. The last one is determined by the constraint. Moreover, since A can corrupt at most $t-1$ users, it is impossible that all users in $\mathcal{S}$ are corrupted, so the last $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ is never referenced. Hence, we always uniformly sample $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ when user $i \in \mathcal{C}_{\mathcal{S},\mathsf{sid}}$ gets corrupted.

The change is invisible to A, so

$$\varepsilon_{18} = \varepsilon_{17}.$$

Collect everything together, and we have

$$\varepsilon_{18} \geq \varepsilon_0 - \frac{N^4 + q_{\mathrm{h}}}{2^\lambda}.$$

□

The remaining part of this proof is a reduction that solves the $2q_{\mathrm{s}}$-AOMDL problem by simulating G_{18} for the adversary and applying the general forking lemma. The reduction is similar to the one for FROST [BCK+22].

Proof (Part II). The reduction B takes $2q_{\mathrm{s}} + 1$ group elements as inputs. We first show how a wrapper W takes these group elements and simulate G_{18} for A. The main structure of the reduction will then be applying the general forking lemma [BN06] to W.

W takes group elements U, $V_{1,1}$, $V_{2,1}$, …, $V_{1,q_{\mathrm{s}}}$, $V_{2,q_{\mathrm{s}}}$ as inputs. It simulates G_{18} for A using these group elements as follows:

- It sets the target public key X as the first input element. Hence, the secret key x is the unknown discrete logarithm of this element.
- Whenever a fresh $(\mathcal{S}, \mathsf{sid})$ is queried to Sign, $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$ should be sampled. Here, W takes two unused input elements as $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,*} \\ R_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix} = \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,*}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,*}} \end{bmatrix}$. Hence, $r_{\mathcal{S},\mathsf{sid},1,*}$ and $r_{\mathcal{S},\mathsf{sid},2,*}$ are the unknown discrete logarithms of these elements.

Now we specify how W simulates G_{18} when those unknown values x, $r_{\mathcal{S},\mathsf{sid},1,*}$, and $r_{\mathcal{S},\mathsf{sid},2,*}$ are referenced, which only occurs in Eqs. (10) to (13).

- In Eqs. (12) and (13), $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$ is referenced in the form of $\begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,*}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,*}} \end{bmatrix}$, i.e. $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,*} \\ R_{\mathcal{S},\mathsf{sid},2,*} \end{bmatrix}$, which is known by W.
- In Eqs. (10) and (11), the unknown part is $r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x$. Other terms in the equations are (publicly) known. W queries $R_{\mathcal{S},\mathsf{sid},1,*} \cdot R^{b_{\mathcal{S},\mathsf{sinf}}}_{\mathcal{S},\mathsf{sid},2,*} \cdot X^{c_{\mathcal{S},\mathsf{sinf}}}$ to DLog to obtain the unknown part.

For each $\mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mathsf{sinf})$ query with returned value b, we say $\mathsf{H}_{\mathrm{chal}}(\mu, R)$ is its resulting query if sinf parses into $(\mathsf{sid}, \mu, \{(\tilde{R}_{1,i}, \tilde{R}_{2,i})\}_{i \in \mathcal{S}})$, and $R = \prod_{i \in \mathcal{S}} \tilde{R}_{1,i} \tilde{R}_{2,i}^{b}$. The $\mathsf{H}_{\mathrm{chal}}$ query made in $\textsc{Sign}'$ is always the resulting query to the $\mathsf{H}_{\mathrm{comb}}$ query just made. W does the following in addition:

- When A makes a query $\mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mathsf{sinf})$, before returning b, W internally makes the resulting $\mathsf{H}_{\mathrm{chal}}$ query.

To fit in the syntax specified in Lemma 1, W takes $h_1, \ldots, h_q$ as inputs and assigns them to $\mathsf{H}_{\mathrm{chal}}$ in order. If the simulated G_{18} outputs 1, i.e., A outputs a valid forgery on a message that has never been queried to $\textsc{Sign}$, then W returns I such that h_I was assigned to $\mathsf{H}_{\mathrm{chal}}(\mu_*, R_*)$ that corresponds to the forgery. If the game aborts, or A does not output a valid forgery, then W returns $I = 0$. For simplicity, we do not specify the side output Y. We just note that Y can contain all internal variables of W that B needs.

Reduction B runs $\mathsf{fork}_{\mathsf{W}}(\mathsf{inp})$ as specified in Lemma 1, with inp being its input group elements.[3] If $\mathsf{fork}_{\mathsf{W}}$ returns non-$\perp$, then B gets two valid forgeries (R_*, z_*) and (R_*, z'_*) satisfying $g^{z_*} = R_* X^{c_*}$ and $g^{z'_*} = R_* X^{c'_*}$, with the same commitment R_* but different challenges $c_* \neq c'_*$. It then extracts the discrete logarithm of the first input element, i.e. the secret key, as $x := (z_* - z'_*)/(c_* - c'_*)$.

What remains is to extract the discrete logarithms of other input elements, i.e. those $\left[\begin{smallmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},2,*} \end{smallmatrix}\right]$. For each such a pair, W makes at most one $\textsc{DLog}$ query to obtain

$$r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x, \quad r_{\mathcal{S}',\mathsf{sid}',1,*} + b'_{\mathcal{S}',\mathsf{sinf}'} \cdot r_{\mathcal{S}',\mathsf{sid}',2,*} + c'_{\mathcal{S}',\mathsf{sinf}'} \cdot x$$

in each of the two executions, respectively. We note that here $\left[\begin{smallmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},1,*} \end{smallmatrix}\right] = \left[\begin{smallmatrix} r_{\mathcal{S}',\mathsf{sid}',1,*} \\ r_{\mathcal{S}',\mathsf{sid}',1,*} \end{smallmatrix}\right]$ are the discrete logarithms of the same pair of input elements — the same pair may correspond to different $(\mathcal{S}, \mathsf{sid})$ in two executions. Here, B already knows x, so the answers to such queries constitute two equations with two unknown $r_{\mathcal{S},\mathsf{sid},1,*}$ and $r_{\mathcal{S},\mathsf{sid},2,*}$. B can solve them to obtain $\left[\begin{smallmatrix} r_{\mathcal{S},\mathsf{sid},1,*} \\ r_{\mathcal{S},\mathsf{sid},1,*} \end{smallmatrix}\right]$ if $b_{\mathcal{S},\mathsf{sinf}} \neq b'_{\mathcal{S}',\mathsf{sinf}'}$. If W did not make such a query in one or both executions, then B can make the remaining queries to get enough equations. If W made such a query in both executions, and $(b_{\mathcal{S},\mathsf{sinf}}, c_{\mathcal{S},\mathsf{sinf}}) = (b'_{\mathcal{S},\mathsf{sinf}}, c'_{\mathcal{S}',\mathsf{sinf}'})$, then these two queries are repeated and only count as one, so B can also make one more query to get enough equations. The only bad case is $b_{\mathcal{S},\mathsf{sinf}} = b_{\mathcal{S}',\mathsf{sinf}'}$ and $c_{\mathcal{S},\mathsf{sinf}} \neq c_{\mathcal{S}',\mathsf{sinf}'}$, where two distinct queries have been made but only provide one equation.

We claim that if the returned values of $\mathsf{H}_{\mathrm{comb}}$ in the two executions are all distinct (except that those whose assignment is before the forking point must be identical), then this bad case never happens. These values are not distinct only with probability at most $4q^2/p$. To prove this claim, $b_{\mathcal{S},\mathsf{sinf}}$ and $c_{\mathcal{S},\mathsf{sinf}}$ are the answers to a $\mathsf{H}_{\mathrm{comb}}$ query and its resulting $\mathsf{H}_{\mathrm{chal}}$ query. Recall that W makes the resulting query internally. Therefore, the value $c_{\mathcal{S},\mathsf{sinf}}$ is assigned at the latest right after the assignment to $b_{\mathcal{S},\mathsf{sinf}}$ in an atomic operation (so the forking point cannot be in between). If $b_{\mathcal{S},\mathsf{sinf}} = b'_{\mathcal{S}',\mathsf{sinf}'}$, then they are the answers to the same query made before the forking point. Hence, their resulting queries must also be the same and made before the forking point, so $c_{\mathcal{S},\mathsf{sinf}} \neq c_{\mathcal{S}',\mathsf{sinf}'}$ is impossible.

By Lemma 1, and taking account of the above bad event, B solves $(2q_{\mathrm{s}} + 1)$-AOMDL with the stated probability. $\square$

4 HBTS-Mask

4.1 Scheme

In this section, we present our stateless two-round threshold signature scheme $\mathsf{HBTS\text{-}Mask}$. The scheme is described in Fig. 6. With group parameters $(\mathbb{G}, p, g)$ chosen in the setup algorithm, the scheme uses hash functions $\mathsf{H}_{\mathrm{mask}} : \{0,1\}^* \to \mathbb{Z}_p^2$, $\mathsf{H}_{\mathrm{gen}} : \{0,1\}^* \to \mathbb{G}$, $\mathsf{H}_{\mathrm{chal}} : \{0,1\}^* \to \mathbb{Z}_p$. It

[3]More formally, B also runs a subroutine that handles the $\textsc{DLog}$ queries of W by forwarding them to the $\textsc{DLog}$ oracle of B, and the forking is applied to W plus this subroutine with $\textsc{DLog}$ considered inside.

```
$\underline{\mathsf{Setup}(1^\lambda)}$
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\textbf{return } (\mathbb{G}, p, g)$

$\underline{\mathsf{KGen}(n, t)}$
$x \leftarrow \mathbb{Z}_p$
$X := g^x$
$\{x_i\}_{i \in [n]} \leftarrow \mathsf{Share}(n, t, x)$
$\textbf{for } i \neq j \in [n] \textbf{ do}$
    $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^\lambda$
$\mathsf{pk} := X$
$\textbf{for } i \in [n] \textbf{ do}$
    $\mathsf{sk}_i := (x_i, \{\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i}\}_{j \in [n] \setminus \{i\}})$
$\textbf{return } (\mathsf{pk}, \{\mathsf{sk}_i\}_{i \in [n]})$

$\underline{\mathsf{Sign}(\mathcal{S}, k, \mathsf{sk}_k, \mu)}$
$(x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i \in [n] \setminus \{k\}}) := \mathsf{sk}_k$
$h := \mathsf{H}_{\text{gen}}(\mu)$
$r_k,\ y_k \leftarrow \mathbb{Z}_p$
$R_k := g^{r_k} h^{y_k}$
$\mathsf{st} := (r_k, y_k)$
$\mathsf{msg} := R_k$
$\textbf{return } (\mathsf{st}, \mathsf{msg})$

$\underline{\mathsf{Sign}'(\mathcal{S}, k, \mathsf{sk}_k, \mu, \mathsf{st}, \mathcal{M})}$
$(x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i \in [n] \setminus \{k\}}) := \mathsf{sk}_k$
$(r_k, y_k) := \mathsf{st}$
$\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
$\textbf{for } i \in \mathcal{S} \textbf{ do}$
    $R_i := \mathsf{msg}_i$
$R := \prod_{i \in \mathcal{S}} R_i$
$c := \mathsf{H}_{\text{chal}}(\mu, R)$
$z_k := r_k + c \cdot \lambda_{\mathcal{S},k} \cdot x_k$
$\mathsf{sinf} := (\mu, \mathcal{M})$
$\Delta_k := \sum_{i \in \mathcal{S} \setminus \{k\}} (\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{seed}_{k,i}, \mathsf{sinf}) -$
$\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{seed}_{i,k}, \mathsf{sinf}))$
$\begin{bmatrix} \tilde{z}_k \\ \tilde{y}_k \end{bmatrix} := \begin{bmatrix} z_k \\ y_k \end{bmatrix} + \Delta_k$
$\textbf{return } (\tilde{z}_k, \tilde{y}_k)$

$\underline{\mathsf{Combine}(\mathcal{S}, \mu, \mathcal{M}, \mathcal{M}')}$
$\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
$\{\mathsf{msg}'_i\}_{i \in \mathcal{S}} := \mathcal{M}'$
$\textbf{for } i \in \mathcal{S} \textbf{ do}$
    $R_i := \mathsf{msg}_i$
    $(\tilde{z}_i, \tilde{y}_i) := \mathsf{msg}'_i$
$R := \prod_{i \in \mathcal{S}} R_i$
$c := \mathsf{H}_{\text{chal}}(\mu, R)$
$\begin{bmatrix} z \\ y \end{bmatrix} := \sum_{i \in \mathcal{S}} \begin{bmatrix} \tilde{z}_i \\ \tilde{y}_i \end{bmatrix}$
$\textbf{return } (R, z, y)$

$\underline{\mathsf{Vf}(\mathsf{pk}, \mu, \sigma)}$
$X := \mathsf{pk}$
$(R, z, y) := \sigma$
$h := \mathsf{H}_{\text{gen}}(\mu)$
$c := \mathsf{H}_{\text{chal}}(\mu, R)$
$\textbf{return } [\![ g^z h^y = R X^c ]\!]$
```

Figure 6: HBTS-Mask.

supports all practical choices of (n,t). To see the completeness of $\mathsf{HBTS\text{-}Mask}$, note that in an honestly executed signing session, $\sum_{i\in\mathcal{S}} \Delta_i = \left[\begin{smallmatrix}0\\0\end{smallmatrix}\right]$, and therefore $\sum_{i\in\mathcal{S}}\left[\begin{smallmatrix}\tilde{z}_i\\\tilde{y}_i\end{smallmatrix}\right] = \sum_{i\in\mathcal{S}}\left[\begin{smallmatrix}z_i\\y_i\end{smallmatrix}\right]$.

4.2 Security

Theorem 3. *If* DL *is* $(\tau_{\mathrm{dl}}, \varepsilon_{\mathrm{dl}})$*-hard for* GrGen*, then* $\mathsf{HBTS\text{-}Mask}$ *is* $(q_{\mathrm{s}}, \tau_{\mathrm{uf}}, \varepsilon_{\mathrm{uf}})$*-adaptively secure in the random oracle model against any adversary that makes at most* q_{h} *hash queries, where essentially* $\tau_{\mathrm{dl}} \approx 2\tau_{\mathrm{uf}}$*, and*

$$\varepsilon_{\mathrm{dl}} \geq \frac{1}{16{q_{\mathrm{s}}}^2 q}\left(\varepsilon_{\mathrm{uf}}^2 - \frac{N^4+q_{\mathrm{h}}}{2^{\lambda-1}} - \frac{2q_{\mathrm{s}}}{p}\right) - \frac{3}{4q_{\mathrm{s}}p}\varepsilon_{\mathrm{uf}},$$

where $q = q_{\mathrm{s}} + q_{\mathrm{h}} + 1$*.*

Proof (Part I). The first part of the proof is virtually the same as the proof of Theorem 2 until G_{11}. We do not need G_{12} to G_{18} here since the commitments are not masked. While we largely omit this part, the pseudocode description and a quick checklist of the games is provided in Appendix B. We highlight two main differences from the proof of Theorem 2.

First, recall that in the proof of Theorem 2, we globalize the variables in SIGN and SIGN′. In this proof, we only globalize the variables in SIGN′. Specifically, we rename the variables r_k, y_k, c, z_k, $\tilde{z}_k$, $\tilde{y}_k$, and Δ_k referenced in $\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})$ to $r_{\mathcal{S},\mathsf{sinf},k}$, $y_{\mathcal{S},\mathsf{sinf},k}$, $c_{\mathcal{S},\mathsf{sinf}}$, $z_{\mathcal{S},\mathsf{sinf},k}$, $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$, $\tilde{y}_{\mathcal{S},\mathsf{sinf},k}$, and $\Delta_{\mathcal{S},\mathsf{sinf},k}$, respectively, where $\mathcal{S} = \mathcal{S}_{k,\mathsf{sid}}$ and $\mathsf{sinf} = (\mu_{k,\mathsf{sid}}, \mathcal{M})$. The indexing ensuring unambiguous reference to these variables in other oracle calls. Similarly to the proof of Theorem 2, note that the index of $c_{\mathcal{S},\mathsf{sinf}}$ does not include k. This does not cause ambiguity, since sinf uniquely determines c referenced in SIGN′. Note that in this proof, SIGN is never modified and hence remains the same as in the original $\mathsf{adp\text{-}UF}$ game.

Second, a minor difference is that, since sinf does not contain sid, the distinctness of sinf relies on the distinctness of $\mathcal{M}$, which comes from the entropy of R_k rather than from the distinctness of sid. Thus, an additive loss of ${q_{\mathrm{s}}}^2/p$ is introduced in G_8. We end up with

$$\varepsilon_{11} \geq \varepsilon_0 - \frac{N^4+q_{\mathrm{h}}}{2^\lambda} - \frac{{q_{\mathrm{s}}}^2}{p}.$$

□

Similar to Eqs. (2) and (4), we have the following two crucial equations in G_{11} which correspond to the initialization of $\Delta_{\mathcal{S},\mathsf{sinf},k}$ for the last $k \in \mathcal{S}$ in $\mathsf{H}_{\mathrm{mask}}$ and SIGN′, respectively. These are the only places where x is referenced.

$$\begin{aligned}\Delta_{\mathcal{S},\mathsf{sinf},k} = &\sum_{i\in\mathcal{S}\setminus\mathcal{C}}\begin{bmatrix}r_{\mathcal{S},\mathsf{sinf},i}\\y_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}}\cdot\left(\begin{bmatrix}x\\0\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\begin{bmatrix}\lambda_{\mathcal{S},i}\cdot x_i\\0\end{bmatrix}\right)\\ &-\sum_{i\in\mathcal{S}\setminus\mathcal{C}}\begin{bmatrix}\tilde{z}_{\mathcal{S},\mathsf{sinf},i}\\\tilde{y}_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}}\Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{14}$$

$$\begin{aligned}\begin{bmatrix}\tilde{z}_{\mathcal{S},\mathsf{sinf},k}\\\tilde{y}_{\mathcal{S},\mathsf{sinf},k}\end{bmatrix} = &\sum_{i\in\mathcal{S}\setminus\mathcal{C}}\begin{bmatrix}r_{\mathcal{S},\mathsf{sinf},i}\\y_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}}\cdot\left(\begin{bmatrix}x\\0\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\begin{bmatrix}\lambda_{\mathcal{S},i}\cdot x_i\\0\end{bmatrix}\right)\\ &-\sum_{i\in\mathcal{S}\setminus\mathcal{C}\setminus\{k\}}\begin{bmatrix}\tilde{z}_{\mathcal{S},\mathsf{sinf},i}\\\tilde{y}_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{15}$$

Proof (Part II). In the second part of the proof, we construct a reduction that uses the adversary to solve DL. We want to embed the DL input group element into the target public key, but the reduction cannot calculate Eqs. (14) and (15) without knowing the secret key x. Therefore, we need to define more hybrid games.

G_{12}**.** We first introduce some internal queries. When A makes a H_{chal} query on message μ, the game makes an internal query $\mathsf{H}_{\text{gen}}(\mu)$. The total number of fresh $\mathsf{H}_{\text{gen}}(\mu)$ queries is at most q.

In this game, when a message μ is queried to H_{gen} for the first time, we flip a coin $\mathsf{coin}(\mu)$ with probability $\rho = q_{\text{s}}/(q_{\text{s}}+1)$ to be 1 and $1-\rho = 1/(q_{\text{s}}+1)$ to be 0. This game aborts if A queries a message μ to SIGN with $\mathsf{coin}(\mu) = 0$ or outputs μ_* with $\mathsf{coin}(\mu_*) = 1$ at the end. Note that SIGN and the verification of the forgery both make internal H_{gen} queries, so the coin must be defined. Since these coins are independent of the view of A until the game aborts for this reason, we have

$$\varepsilon_{12} = \rho^{q_{\text{s}}}(1-\rho)\varepsilon_{11} \geq \frac{1}{4q_{\text{s}}} \cdot \varepsilon_{11}.$$

G_{13}**.** In this game, we answer each fresh query $\mathsf{H}_{\text{gen}}(\mu)$ based on $\mathsf{coin}(\mu)$. If $\mathsf{coin}(\mu) = 1$, we sample $w_\mu \leftarrow \mathbb{Z}_p^*$ and let the answer $h := X^{1/w_\mu}$. If $\mathsf{coin}(\mu) = 0$, we sample $a_\mu \leftarrow \mathbb{Z}_p$ and let the answer $h := g^{a_\mu}$. Each answer h in G_{13} is at most $1/p$ away from the answer in G_{12} in statistical distance. Thus, we have

$$\varepsilon_{13} \geq \varepsilon_{12} - \frac{q}{p}.$$

G_{14}**.** In this game, Eqs. (14) and (15) are replaced with the following equations, respectively.

$$\begin{aligned}\Delta_{\mathcal{S},\mathsf{sinf},k} = &\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} \cdot \left(\begin{bmatrix} 0 \\ w_\mu \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \begin{bmatrix} \lambda_{\mathcal{S},i}\cdot x_i \\ 0 \end{bmatrix} \right) \\ &- \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{16}$$

$$\begin{aligned}\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k}\end{bmatrix} = &\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} \cdot \left(\begin{bmatrix} 0 \\ w_\mu \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \begin{bmatrix} \lambda_{\mathcal{S},i}\cdot x_i \\ 0 \end{bmatrix} \right) \\ &- \sum_{i\in\mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},i}\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}.\end{aligned} \tag{17}$$

That is, $\left[\begin{smallmatrix} x \\ 0 \end{smallmatrix}\right]$ is simply replaced with $\left[\begin{smallmatrix} 0 \\ w_\mu \end{smallmatrix}\right]$ for the μ contained in sinf. We note that the game calculates these equations only if μ has been queried to SIGN, and the game aborts if $\mathsf{coin}(\mu) = 0$. Therefore, such w_μ must exist. Our reduction will execute G_{14} for A. We claim that

$$\varepsilon_{14} = \varepsilon_{13}.$$

We prove this claim through additional hybrid games. We will modify G_{13} and G_{14} in parallel while keeping the view of A unchanged. We will end up with $\mathsf{G}_{13,3}$ and $\mathsf{G}_{14,3}$, which turn out to be exactly the same game. We note that the following games are not efficiently executable. This is not a problem since no reduction needs to execute them.

$\mathsf{G}_{13,1}$ **(resp.** $\mathsf{G}_{14,1}$**).** In this game, SIGN directly generates $R_k \leftarrow \mathbb{Z}_p$ and set r_k and y_k to ?, which denotes being pending. They remain to be pending when they are globalized to $r_{\mathcal{S},\mathsf{sinf},k}$ and $y_{\mathcal{S},\mathsf{sinf},k}$ in SIGN$'$. They are uniformly sampled conditioned on $R_k = g^{r_k}h^{y_k}$ for the corresponding R_k and h when being referenced elsewhere, specifically:

- In Eqs. (14) and (15) (resp. Eqs. (16) and (17)); and
- When being output as the secret state of user k in CORR(k).

One can verify that r_k and y_k are independent of the view of A until being referenced in these places. Thus, $\mathsf{G}_{13,1}$ (resp. $\mathsf{G}_{14,1}$) is identical to G_{13} (resp. G_{14}).

$\mathsf{G}_{13,2}$ **(resp. $\mathsf{G}_{14,2}$).** In this game, we directly sample the sum $\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \left[\begin{smallmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{smallmatrix}\right]$ in Eqs. (14) and (15) (resp. Eqs. (16) and (17)) conditioned on $g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sinf},i}} h^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} y_{\mathcal{S},\mathsf{sinf},i}} = \prod_{i\in\mathcal{S}\setminus\mathcal{C}} R_i$ for the corresponding h and R_i. Since not all users in $\mathcal{S}\setminus\mathcal{C}$ will be corrupted later, the sum sampled here will not constraint the individual r_i, y_i sampled in $\textsc{Corr}(i)$. That is, they are still uniformly sampled condition on $g^{r_i}h^{y_i} = R_i$.

$\mathsf{G}_{13,3}$ **(resp. $\mathsf{G}_{14,3}$).** In this game, in Eqs. (14) and (15) (resp. Eqs. (16) and (17)), we replace

$$\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} \cdot \begin{bmatrix} x \\ 0 \end{bmatrix} \quad \left(\text{resp.} \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} \cdot \begin{bmatrix} 0 \\ w_\mu \end{bmatrix}\right)$$

with uniformly random $\left[\begin{smallmatrix} \alpha \\ \beta \end{smallmatrix}\right]$ conditioned on $g^\alpha h^\beta = \prod_{i\in\mathcal{S}\setminus\mathcal{C}} R_i X$ for corresponding h and R_i. Given the distribution of $\sum_{i\in\mathcal{S}\setminus\mathcal{C}} \left[\begin{smallmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{smallmatrix}\right]$ and the fact that $g^x = X$ (resp. $h^{w_\mu} = X$), this modification does not change the view of A.

Note that $\mathsf{G}_{13,3}$ has exactly the same description as $\mathsf{G}_{14,3}$. We therefore conclude that the views of A in G_{13} and G_{14} are identical, and $\varepsilon_{14} = \varepsilon_{13}$.

Now we define a wrapper W that follows the syntax defined in Lemma 1. W takes a group element X and $h_1, \ldots, h_q$ as inputs. It executes G_{14} for A with X being the target public key and assigning $h_1, \ldots, h_q$ to H_{chal}. If A wins in G_{14}, W returns I such that h_I was assigned to $\mathsf{H}_{\text{chal}}(\mu_*, R_*)$ that corresponds to the forgery. Otherwise it returns $I = 0$. We assume that the unspecified side output contains all the internal information that our reduction needs.

On input X, the reduction B runs $\mathsf{fork}_{\mathsf{W}}(X)$ as specified in Lemma 1. If $\mathsf{fork}_{\mathsf{W}}$ returns non-$\perp$, then B gets two valid forgeries (R_*, z_*, y_*) and (R_*, z'_*, y'_*) satisfying $g^{z_*}h^{y_*} = R_* X^{c_*}$ and $g^{z'_*}h^{y'_*} = R_* X^{c'_*}$, with the same h and R_* but different challenges $c_* \neq c'_*$. We have the same h because since the forgeries in two executions correspond to the same $\mathsf{H}_{\text{chal}}(\mu_*, R_*)$ query, then it also corresponds to the same $\mathsf{H}_{\text{gen}}(\mu_*)$ query. Moreover, the assignment to $\mathsf{H}_{\text{gen}}(\mu_*)$ is earlier than the forking point $\mathsf{H}_{\text{chal}}(\mu_*, R_*)$, which is ensured by the internal queries introduced in G_{12}. Also note that h has a known discrete logarithm a_{μ_*}. Therefore, B can compute the discrete logarithm of X as $x = (z_* + a_{\mu_*}y_* - z'_* - a_{\mu_*}y'_*)/(c_* - c'_*)$. □

Acknowledgment

We would like to thank Andrej Bogdanov and the anonymous reviewers for helpful comments and for carefully pointing out a number of typos in an earlier version.

References

[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, Cham, August 2022. doi:10.1007/978-3-031-15985-5_18. 6, 21

[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, Cham, December 2021. doi:10.1007/978-3-030-92068-5_22. 2, 7

[BDLR24] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. Cryptology ePrint Archive, Report 2024/1628, 2024. URL: https://eprint.iacr.org/2024/1628. 2, 3, 8

[BHK$^+$24] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Yiping Ma, and Tal Rabin. SPRINT: High-throughput robust distributed Schnorr signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 62–91. Springer, Cham, May 2024. doi:10.1007/978-3-031-58740-5_3. 8

[BLSW24] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. HARTS: High-threshold, adaptively secure, and robust threshold schnorr signatures. Cryptology ePrint Archive, Report 2024/280, 2024. URL: https://eprint.iacr.org/2024/280. 8

[BLT$^+$24] Renas Bacho, Julian Loss, Stefano Tessaro, Benedikt Wagner, and Chenzhi Zhu. Twinkle: Threshold signatures from DDH with full adaptive security. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 429–459. Springer, Cham, May 2024. doi:10.1007/978-3-031-58716-0_15. 2, 3, 8

[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. doi:10.1145/1180405.1180453. 9, 21

[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, Berlin, Heidelberg, August 1999. doi:10.1007/3-540-48405-1_7. 8

[Che23] Yanbo Chen. DualMS: Efficient lattice-based two-round multi-signature with trapdoor-free simulation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 716–747. Springer, Cham, August 2023. doi:10.1007/978-3-031-38554-4_23. 3

[Che25] Yanbo Chen. Dazzle: Improved adaptive threshold signatures from DDH. Cryptology ePrint Archive, Report 2025/264, 2025. URL: https://eprint.iacr.org/2025/264. 2, 3

[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, Cham, August 2023. doi:10.1007/978-3-031-38557-5_22. 2, 3, 8

[DKM$^+$24] Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen. Threshold raccoon: Practical threshold signatures from standard lattice assumptions. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024. doi:10.1007/978-3-031-58723-8_8. 4

[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. doi:10.1109/SP46214.2022.9833584. 8

[EKT24] Thomas Espitau, Shuichi Katsumata, and Kaoru Takemure. Two-round threshold signature from algebraic one-more learning with errors. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 387–424. Springer, Cham, August 2024. doi:10.1007/978-3-031-68394-7_13. 3, 4

[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, Cham, August 2018. doi:10.1007/978-3-319-96881-0_2. 2

[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. doi:10.1007/s00145-006-0347-3. 8

[GS24] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 370–400. Springer, Cham, May 2024. doi:10.1007/978-3-031-58740-5_13. 8

[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, Berlin, Heidelberg, May 2000. doi:10.1007/3-540-45539-6_16. 8

[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, Cham, October 2020. doi:10.1007/978-3-030-81652-0_2. 2, 3, 6

[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. URL: https://eprint.iacr.org/2023/292. 8

[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. doi:10.1145/3372297.3423364. 8

[KRT24a] Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 459–491. Springer, Cham, August 2024. doi:10.1007/978-3-031-68394-7_15. 2, 3, 4, 8

[KRT24b] Shuichi Katsumata, Michael Reichle, and Kaoru Takemure. Adaptively secure 5 round threshold signatures from MLWE/MSIS and DL with rewinding. Cryptology ePrint Archive, Report 2024/1033, 2024. URL: https://eprint.iacr.org/2024/1033. 12

[Oka93] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Berlin, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_3. 8

[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, Berlin, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_9. 8

[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. doi:10.1145/3548606.3560583. 8

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. doi:10.1145/359168.359176. 9

[Sho23] Victor Shoup. The many faces of schnorr. Cryptology ePrint Archive, Report 2023/1019, 2023. URL: https://eprint.iacr.org/2023/1019. 8

[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, Cham, April 2023. doi:10.1007/978-3-031-30589-4_22. 3, 8

A Figures for Proof of Theorem 2

In Figs. 7 to 12, we provide the pseudocode description of the games in part I of the proof of Theorem 2. For readability, we avoid putting all hybrid games together. Instead, we organize Sign (Fig. 9) using conditional branches, each covering a subset of the games: G_0–G_{12}, G_{13}–G_{14}, and G_{15}–G_{18}. We use a similar strategy for Sign$'$ (Fig. 10). Oracles H_{mask} and $\mathsf{H}'_{\text{mask}}$ (Figs. 11 and 12) are rewritten multiple times to reflect the changes across the games. In some places, we use informal description like "Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$" to simplify the presentation. The precise formulation of such statements is provided in Fig. 8. We omit H_{comb} and H_{chal}, since they remain unchanged throughout the hybrid game sequence.

We provide a quick checklist that outlines the modifications made in each hybrid game.

- G_1: Check the distinctness of seeds.
- G_2: Introduce intermediate variables $d_{\mathcal{S},\mathsf{sinf},i,j}$ and $e_{\mathcal{S},\mathsf{sid},i,j}$ for $\mathsf{H}_{\text{mask}}(\mathcal{S},\mathsf{sid},\mathsf{seed}_{i,j})$ and $\mathsf{H}'_{\text{mask}}(\mathcal{S},\mathsf{sinf},\mathsf{seed}_{i,j})$.
- G_3: Reference the intermediate variables instead of calling H_{mask} and $\mathsf{H}'_{\text{mask}}$ in Sign and Sign$'$, respectively.
- G_4: Defer sampling the seeds until the corresponding user is corrupted.
- G_5: Initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$ when answering $\mathsf{H}'_{\text{mask}}(\mathcal{S},\mathsf{sinf},\mathsf{seed}_{k,i})$ or $\mathsf{H}'_{\text{mask}}(\mathcal{S},\mathsf{sinf},\mathsf{seed}_{i,k})$ for corrupted user k.
- G_6: Reverse the order of generating $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and those related $d_{\mathcal{S},\mathsf{sinf},i,j}$.
- G_7: Avoid referencing $\Delta_{\mathcal{S},\mathsf{sinf},i}$ and x_i for uncorrupted user i when initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$.
- G_8: Reverse the order of generating $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$ in Sign$'$.
- G_9: Defer the calculation of $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and the initialization of those related $d_{\mathcal{S},\mathsf{sinf},i,j}$ from Sign$'$ to Corr(k).
- G_{10}: Defer the calculation of $z_{\mathcal{S},\mathsf{sinf},k}$ from Sign$'$ to Corr(k).
- G_{11}: Defer the generation of secret key shares until the corresponding user is corrupted.
- G_{12}: Initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$ when answering $\mathsf{H}_{\text{mask}}(\mathcal{S},\mathsf{sid},\mathsf{seed}_{k,i})$ or $\mathsf{H}'_{\text{mask}}(\mathcal{S},\mathsf{sid},\mathsf{seed}_{i,k})$ for corrupted user k.
- G_{13}: Reverse the order of generating $\Phi_{\mathcal{S},\mathsf{sid},k}$ and those related $e_{\mathcal{S},\mathsf{sid},i,j}$.
- G_{14}: Avoid referencing $\Phi_{\mathcal{S},\mathsf{sid},i}$ for uncorrupted user i when initialize $\Phi_{\mathcal{S},\mathsf{sid},k}$.
- G_{15}: Reverse the order of generating $\Phi_{\mathcal{S},\mathsf{sid},k}$ and $\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix}$ in Sign.
- G_{16}: Defer the calculation of $\Phi_{\mathcal{S},\mathsf{sid},k}$ and the initialization of those related $e_{\mathcal{S},\mathsf{sid},i,j}$ from Sign to Corr(k).

```
$\underline{\text{adp-UF}^{\mathsf{A}}_{\text{FROST-Mask}}}$
$\mathcal{Q} := \emptyset, \mathcal{C} := \emptyset$
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
$x \leftarrow \mathbb{Z}_p$
$X := g^x$
$\{x_i\}_{i\in[n]} \leftarrow \mathsf{Share}(n, t, x)$                          $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_{10}$
for $i \neq j \in [n]$ do                                                   $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_3$
    $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^\lambda$                            $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_3$
if $\exists (i,j) \neq (i',j') : \mathsf{seed}_{i,j} = \mathsf{seed}_{i',j'}$ then   $\triangleright$ $\mathsf{G}_1$–$\mathsf{G}_3$
    Game Abort                                                              $\triangleright$ $\mathsf{G}_1$–$\mathsf{G}_3$
$\mathsf{pk} := X$
for $i \in [n]$ do
    $\mathsf{sk}_i := (x_i, \{\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i}\}_{j\in[n]\setminus\{i\}})$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\textsc{Sign},\textsc{Sign}',\textsc{Corr},\mathsf{H}_{\text{mask}},\mathsf{H}_{\text{comb}},\mathsf{H}_{\text{chal}}}(\mathsf{pk})$
return $[\![\mu^* \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, \mu^*, \sigma^*) = 1]\!]$
```

Figure 7: Setup in proof of Theorem 2.

- G_{17}: Directly calculate $\prod_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ without calculating each $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$.
- G_{18}: Pre-sample the sum of $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ over uncorrupted users when $(\mathcal{S}, \mathsf{sid})$ is queried to $\textsc{Sign}$ for the first time, and defer the generation of each $\begin{bmatrix} r_{\mathcal{S},\mathsf{sid},1,i} \\ r_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}$ until the corresponding user is corrupted.

B Figures for Proof of Theorem 3

In Figs. 13 to 16, we provide the pseudocode description of the games in part I of the proof of Theorem 3. We omit $\textsc{Sign}$, H_{gen}, and H_{chal}, as they remain unchanged throughout the hybrid game sequence. The pseudocode description of the games defined in part II of the proof are not included here.

We provide a quick checklist that outlines the modifications made in each hybrid game.

- G_1: Check the distinctness of seeds.
- G_2: Introduce intermediate variables $d_{\mathcal{S},\mathsf{sinf},i,j}$ for $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,j})$.
- G_3: Reference the intermediate variables instead of calling H_{mask} in $\textsc{Sign}'$.
- G_4: Defer sampling the seeds until the corresponding user is corrupted.
- G_5: Initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$ when answering $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{k,i})$ or $\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,k})$ for corrupted user k.
- G_6: Reverse the order of generating $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and those related $d_{\mathcal{S},\mathsf{sinf},i,j}$.
- G_7: Avoid referencing $\Delta_{\mathcal{S},\mathsf{sinf},i}$ and x_i for uncorrupted user i when initialize $\Delta_{\mathcal{S},\mathsf{sinf},k}$.
- G_8: Reverse the order of generating $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and $\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix}$ in $\textsc{Sign}'$.
- G_9: Defer the calculation of $\Delta_{\mathcal{S},\mathsf{sinf},k}$ and the initialization of those related $d_{\mathcal{S},\mathsf{sinf},i,j}$ from $\textsc{Sign}'$ to $\textsc{Corr}(k)$.
- G_{10}: Defer the calculation of $z_{\mathcal{S},\mathsf{sinf},k}$ from $\textsc{Sign}'$ to $\textsc{Corr}(k)$.
- G_{11}: Defer the generation of secret key shares until the corresponding user is corrupted.

```
$\underline{\textsc{Corr}(k)}$
if $|\mathcal{C}| \geq t-1$ then
    return $\perp$
$\mathcal{C} := \mathcal{C} \cup \{k\}$
for $i \in [n] \setminus \{k\}$ do                                            $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
    if $i \notin \mathcal{C}$ then                                              $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
        $\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k} \leftarrow \{0,1\}^\lambda$                 $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
if $\exists (i,j) \neq (i',j') : \mathsf{seed}_{i,j} = \mathsf{seed}_{i',j'} \neq \perp$ then       $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
    Game Abort                                                                  $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
$x_k \leftarrow \mathbb{Z}_p$                                                  $\triangleright$ $\mathsf{G}_{11}$–$\mathsf{G}_{18}$
for $\mathsf{sid} \in \mathcal{I}_k$ do                                          $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
    $\mathcal{S} := \mathcal{S}_{k,\mathsf{sid}}$                                 $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
    $r_{\mathcal{S},\mathsf{sid},1,k}, r_{\mathcal{S},\mathsf{sid},2,k} \leftarrow \mathbb{Z}_p$          $\triangleright$ $\mathsf{G}_{18}$
    $\mathsf{st}_{k,\mathsf{sid}} := (r_{\mathcal{S},\mathsf{sid},1,k}, r_{\mathcal{S},\mathsf{sid},2,k})$    $\triangleright$ $\mathsf{G}_{18}$
    $\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,k}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,k}} \end{bmatrix}$    $\triangleright$ $\mathsf{G}_{17}$–$\mathsf{G}_{18}$
    $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} / \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$    $\triangleright$ $\mathsf{G}_{16}$–$\mathsf{G}_{18}$
    Initialize related $e_{\mathcal{S},\mathsf{sid},i,j}$ conditioned on $\Phi_{\mathcal{S},\mathsf{sid},k}$     $\triangleright$ $\mathsf{G}_{16}$–$\mathsf{G}_{18}$
    if $\mathsf{round}_{k,\mathsf{sid}} = 2$ then                                 $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
        $\mathsf{sinf} := \mathsf{sinf}_{k,\mathsf{sid}}$                         $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
        $z_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sid},1,k} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,k} + c_{\mathcal{S},\mathsf{sinf}} \cdot \lambda_{\mathcal{S},k} \cdot x_k$    $\triangleright$ $\mathsf{G}_{10}$–$\mathsf{G}_{18}$
        $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$    $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
        Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$    $\triangleright$ $\mathsf{G}_9$–$\mathsf{G}_{18}$
$\mathsf{sk}_k := (x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i \in [n] \setminus \{k\}})$    $\triangleright$ $\mathsf{G}_4$–$\mathsf{G}_{18}$
return $(\mathsf{sk}_k, \{\mathsf{st}_{k,\mathsf{sid}}\}_{\mathsf{sid} \in \mathcal{I}_i})$

$\underline{\text{Initialize related } d_{\mathcal{S},\mathsf{sinf},i,j} \text{ conditioned on } \Delta_{\mathcal{S},\mathsf{sinf},k}}$
Let $i^*$ be the smallest number in $\mathcal{S} \setminus \{k\}$ such that $d_{k,i} = \perp$
for $(i',j') \in \{(k,i)\}_{i \in \mathcal{S} \setminus \{k\}} \cup \{(i,k)\}_{i \in \mathcal{S} \setminus \{k\}} \setminus \{(k,i^*)\}$ do    $\triangleright$ Sample all but one
    if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then
        $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p$
$d_{\mathcal{S},\mathsf{sinf},k,i^*} := \Delta_{\mathcal{S},\mathsf{sinf},k} - \sum_{i \in \mathcal{S} \setminus \{k\} \setminus \{i^*\}} d_{\mathcal{S},\mathsf{sinf},k,i} + \sum_{i \in \mathcal{S} \setminus \{k\}} d_{\mathcal{S},\mathsf{sinf},i,k}$    $\triangleright$ Calculate the last one

$\underline{\text{Initialize related } e_{\mathcal{S},\mathsf{sid},i,j} \text{ conditioned on } \Phi_{\mathcal{S},\mathsf{sid},k}}$
Let $i^*$ be the smallest number in $\mathcal{S} \setminus \{k\}$ such that $e_{\mathcal{S},\mathsf{sid},k,i} = \perp$
for $(i',j') \in \{(k,i)\}_{i \in \mathcal{S} \setminus \{k\}} \cup \{(i,k)\}_{i \in \mathcal{S} \setminus \{k\}} \setminus \{(k,i^*)\}$ do    $\triangleright$ Sample all but one
    if $e_{\mathcal{S},\mathsf{sid},i',j'} = \perp$ then
        $e_{\mathcal{S},\mathsf{sid},i',j'} \leftarrow \mathbb{G}^2$
$e_{\mathcal{S},\mathsf{sid},k,i^*} := \Phi_{\mathcal{S},\mathsf{sid},k} \cdot \prod_{i \in \mathcal{S} \setminus \{k\} \setminus \{i^*\}} e^{-1}_{\mathcal{S},\mathsf{sid},k,i} \cdot \prod_{i \in \mathcal{S} \setminus \{k\}} e_{\mathcal{S},\mathsf{sid},i,k}$    $\triangleright$ Calculate the last one
```

Figure 8: Corr in proof of Theorem 2.

```
$\textsc{Sign}(\mathcal{S}, k, \mu, \mathsf{sid})$
if $k \in \mathcal{C} \vee \mathsf{sid} \in \mathcal{I}_k$ then
    return $\perp$
$\mathcal{Q} := \mathcal{Q} \cup \{\mu\}$
$\mathcal{I}_k := \mathcal{I}_k \cup \{\mathsf{sid}\}$
$r_{\mathcal{S},\mathsf{sid},1,k},\ r_{\mathcal{S},\mathsf{sid},2,k} \leftarrow \mathbb{Z}_p$                                   $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_{17}$
$\begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,k}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,k}} \end{bmatrix}$                                   $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_{16}$
if $\mathcal{C}_{\mathcal{S},\mathsf{sid}} = \perp$ then          $\triangleright$ $(\mathcal{S}, \mathsf{sid})$ *is queried for the first time,* $\mathsf{G}_{18}$
    $r_{\mathcal{S},\mathsf{sid},1,*},\ r_{\mathcal{S},\mathsf{sid},2,*} \leftarrow \mathbb{Z}_p$                                   $\triangleright$ $\mathsf{G}_{18}$
    $\mathcal{C}_{\mathcal{S},\mathsf{sid}} := \mathcal{C}$                                   $\triangleright$ $\mathsf{G}_{18}$

$\mathsf{G}_0$–$\mathsf{G}_{12}$
$\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S}\setminus\{k\}} (\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{k,i}) / \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}_{i,k}))$          $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_2$
for $(i', j') \in \{(k,i)\}_{i \in \mathcal{S}\setminus\{k\}} \cup \{(i,k)\}_{i \in \mathcal{S}\setminus\{k\}}$ do          $\triangleright$ *Initialize for the first reference,* $\mathsf{G}_3$–$\mathsf{G}_{12}$
    if $e_{\mathcal{S},\mathsf{sid},i',j'} = \perp$ then                                   $\triangleright$ $\mathsf{G}_3$–$\mathsf{G}_{12}$
        $e_{\mathcal{S},\mathsf{sid},i',j'} \leftarrow \mathbb{G}^2$                                   $\triangleright$ $\mathsf{G}_3$–$\mathsf{G}_{12}$
$\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S}\setminus\{k\}} (e_{\mathcal{S},\mathsf{sid},k,i} / e_{\mathcal{S},\mathsf{sid},i,k})$                                   $\triangleright$ $\mathsf{G}_3$–$\mathsf{G}_{12}$
$\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} \cdot \Phi_{\mathcal{S},\mathsf{sid},k}$                                   $\triangleright$ $\mathsf{G}_0$–$\mathsf{G}_{14}$

$\mathsf{G}_{13}$–$\mathsf{G}_{14}$
if $\exists i \in \mathcal{S} \setminus \{k\} : \Phi_{\mathcal{S},\mathsf{sid},k} = \perp$ then                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{18}$
    $\Phi_{\mathcal{S},\mathsf{sid},k} \leftarrow \mathbb{G}^2$                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{14}$
else                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{18}$
    $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$                                   $\triangleright$ $\mathsf{G}_{13}$
    $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} ( \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} / \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} )^{-1} \cdot \prod_{i \in \mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$          $\triangleright$ *Eq.* (6), $\mathsf{G}_{14}$
$\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} \cdot \Phi_{\mathcal{S},\mathsf{sid},k}$                                   $\triangleright$ $\mathsf{G}_{12}$–$\mathsf{G}_{14}$
Initialize related $e_{\mathcal{S},\mathsf{sid},i,j}$ conditioned on $\Phi_{\mathcal{S},\mathsf{sid},k}$                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{15}$

$\mathsf{G}_{15}$–$\mathsf{G}_{18}$
if $\exists i \in \mathcal{S} \setminus \{k\} : \Phi_{\mathcal{S},\mathsf{sid},i} = \perp$ then                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{18}$
    $\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} \leftarrow \mathbb{G}^2$                                   $\triangleright$ $\mathsf{G}_{15}$–$\mathsf{G}_{18}$
else                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{18}$
    $\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \prod_{i \in \mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,i} \\ R_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix} \cdot \prod_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$          $\triangleright$ *Eq.* (7), $\mathsf{G}_{15}$–$\mathsf{G}_{16}$
    $\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} := \begin{bmatrix} g^{\sum_{i \in \mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{\sum_{i \in \mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix} \cdot \prod_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$          $\triangleright$ *Eq.* (9), $\mathsf{G}_{17}$
    $\begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} = \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,*}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,*}} \end{bmatrix} \cdot \prod_{i \in \mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}} \begin{bmatrix} g^{r_{\mathcal{S},\mathsf{sid},1,i}} \\ g^{r_{\mathcal{S},\mathsf{sid},2,i}} \end{bmatrix}^{-1}$
        $\cdot \prod_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,i} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i} \end{bmatrix}^{-1} \cdot \prod_{i \in \mathcal{S}\cap\mathcal{C}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$          $\triangleright$ *Eq.* (13), $\mathsf{G}_{18}$
$\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix} \tilde{R}_{\mathcal{S},\mathsf{sid},1,k} \\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix} / \begin{bmatrix} R_{\mathcal{S},\mathsf{sid},1,k} \\ R_{\mathcal{S},\mathsf{sid},2,k} \end{bmatrix}$                                   $\triangleright$ $\mathsf{G}_{15}$
Initialize related $e_{\mathcal{S},\mathsf{sid},i,j}$ conditioned on $\Phi_{\mathcal{S},\mathsf{sid},k}$                                   $\triangleright$ $\mathsf{G}_{13}$–$\mathsf{G}_{15}$
$\Phi_{\mathcal{S},\mathsf{sid},k} :=?$                                   $\triangleright$ $\mathsf{G}_{16}$–$\mathsf{G}_{18}$

$\mathsf{st}_{k,\mathsf{sid}} := (r_{\mathcal{S},\mathsf{sid},1,k}, r_{\mathcal{S},\mathsf{sid},2,k})$
$\mathsf{msg} := (\tilde{R}_{\mathcal{S},\mathsf{sid},1,k}, \tilde{R}_{\mathcal{S},\mathsf{sid},2,k})$
$(\mathcal{S}_{k,\mathsf{sid}}, \mu_{k,\mathsf{sid}}) := (\mathcal{S}, \mu)$
$\mathsf{round}_{k,\mathsf{sid}} := 1$
return $\mathsf{msg}$
```

Figure 9: Sign in the proof of Theorem 2.

```
$\underline{\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})}$
1:  if $k \in \mathcal{C} \vee \mathsf{round}_{k,\mathsf{sid}} \neq 1$ then
2:      return $\perp$
3:  $(\mathcal{S}, \mu) := (\mathcal{S}_{k,\mathsf{sid}}, \mu_{k,\mathsf{sid}})$
4:  $\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
5:  for $i \in \mathcal{S}$ do
6:      $(\tilde{R}_{1,i}, \tilde{R}_{2,i}) := \mathsf{msg}_i$
7:  $\mathsf{sinf} := (\mathsf{sid}, \mu, \mathcal{M})$
8:  $b_{\mathcal{S},\mathsf{sinf}} := \mathsf{H}_{\mathrm{comb}}(\mathcal{S}, \mathsf{sinf})$
9:  $R := \prod_{i \in \mathcal{S}} \tilde{R}_{1,i} \tilde{R}_{2,i}^{b}$
10: $c_{\mathcal{S},\mathsf{sinf}} := \mathsf{H}_{\mathrm{chal}}(\mu, \tilde{R})$
11: $z_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sid},1,k} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,k} + c_{\mathcal{S},\mathsf{sinf}} \cdot \lambda_{\mathcal{S},k} \cdot x_k$    ▷ $\mathsf{G}_0$–$\mathsf{G}_9$

    $\underline{\mathsf{G}_0\text{–}\mathsf{G}_5}$
12: $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{k,i}) - \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{H}'_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,k})$    ▷ $\mathsf{G}_0$–$\mathsf{G}_2$
13: for $(i', j') \in \{(k, i)\}_{i \in \mathcal{S} \setminus \{k\}} \cup \{(i, k)\}_{i \in \mathcal{S} \setminus \{k\}}$ do    ▷ *Initialize for the first reference,* $\mathsf{G}_3$–$\mathsf{G}_5$
14:     if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then    ▷ $\mathsf{G}_3$–$\mathsf{G}_5$
15:         $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p$    ▷ $\mathsf{G}_3$–$\mathsf{G}_5$
16: $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \{k\}} (d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$    ▷ $\mathsf{G}_3$–$\mathsf{G}_5$
17: $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} := z_{\mathcal{S},\mathsf{sinf},k} + \Delta_{\mathcal{S},\mathsf{sinf},k}$    ▷ $\mathsf{G}_0$–$\mathsf{G}_7$

    $\underline{\mathsf{G}_6\text{–}\mathsf{G}_7}$
18: if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},i} = \perp$ then    ▷ $\mathsf{G}_6$–$\mathsf{G}_{18}$
19:     $\Delta_{\mathcal{S},\mathsf{sinf},k} \leftarrow \mathbb{Z}_p$    ▷ $\mathsf{G}_6$–$\mathsf{G}_7$
20: else    ▷ $\mathsf{G}_6$–$\mathsf{G}_{18}$
21:     $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i \in \mathcal{S} \setminus \{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$    ▷ $\mathsf{G}_6$
22:     $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}}(x - \sum_{i \in \mathcal{S} \cap \mathcal{C} \cup \{k\}} \lambda_{\mathcal{S},i} \cdot x_i)$
        $- \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}$    ▷ *Eq.* (3), $\mathsf{G}_7$
23: $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} := z_{\mathcal{S},\mathsf{sinf},k} + \Delta_{\mathcal{S},\mathsf{sinf},k}$    ▷ $\mathsf{G}_5$–$\mathsf{G}_7$
24: Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$    ▷ $\mathsf{G}_6$–$\mathsf{G}_8$

    $\underline{\mathsf{G}_8\text{–}\mathsf{G}_{18}}$
25: if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},k} = \perp$ then    ▷ $\mathsf{G}_6$–$\mathsf{G}_{18}$
26:     $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} \leftarrow \mathbb{Z}_p$    ▷ $\mathsf{G}_8$–$\mathsf{G}_{18}$
27: else    ▷ $\mathsf{G}_6$–$\mathsf{G}_{18}$
28:     $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S} \setminus \mathcal{C}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}} \cdot (x - \sum_{\mathcal{S} \cap \mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i)$
        $- \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}$    ▷ *Eq.* (4), $\mathsf{G}_8$–$\mathsf{G}_{17}$
29:     $\tilde{z}_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x$
        $- \sum_{i \in \mathcal{S} \cap \mathcal{C} \setminus \mathcal{C}_{\mathcal{S},\mathsf{sid}}} (r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) - c_{\mathcal{S},\mathsf{sinf}} \sum_{i \in \mathcal{S} \cap \mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i$
        $- \sum_{i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i \in \mathcal{S} \cap \mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}$    ▷ *Eq.* (11), $\mathsf{G}_{18}$
30: $\Delta_{\mathcal{S},\mathsf{sinf},k} := \tilde{z}_{\mathcal{S},\mathsf{sinf},k} - z_{\mathcal{S},\mathsf{sinf},k}$    ▷ $\mathsf{G}_8$
31: $\Delta_{\mathcal{S},\mathsf{sinf},k} :=?$    ▷ $\mathsf{G}_9$–$\mathsf{G}_{18}$
32: Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$    ▷ $\mathsf{G}_6$–$\mathsf{G}_8$

33: $\mathsf{sinf}_{k,\mathsf{sid}} := \mathsf{sinf}$
34: $\mathsf{round}_{k,\mathsf{sid}} := 2$
35: return $\tilde{z}_{\mathcal{S},\mathsf{sinf},k}$
```

Figure 10: Sign′ in the proof of Theorem 2.

```
$\underline{\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})}$                                              ▷ $\mathsf{G}_0$–$\mathsf{G}_1$
if $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) = \perp$ then
    $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) \leftarrow \mathbb{G}^2$
return $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})$
```

```
$\underline{\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})}$                                              ▷ $\mathsf{G}_2$–$\mathsf{G}_{11}$
if $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        $e_{\mathcal{S},\mathsf{sid},i,j} \leftarrow \mathbb{G}^2$
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) := e_{\mathcal{S},\mathsf{sid},i,j}$
    else
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) \leftarrow \mathbb{G}^2$
return $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})$
```

```
$\underline{\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})}$                                              ▷ $\mathsf{G}_{12}$
if $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            for $(i', j') \in \{(k, i)\}_{\mathcal{S}\setminus\{k\}} \cup \{(i, k)\}_{i\in\mathcal{S}\setminus\{k\}}$ do        ▷ *Initialize for the first reference*
                if $e_{\mathcal{S},\mathsf{sid},i',j'} = \perp$ then
                    $e_{\mathcal{S},\mathsf{sid},i',j'} \leftarrow \mathbb{G}^2$
            $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i\in\mathcal{S}\setminus\{k\}}(e_{\mathcal{S},\mathsf{sid},k,i}/e_{\mathcal{S},\mathsf{sid},i,k})$
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) := e_{\mathcal{S},\mathsf{sid},i,j}$
    else
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) \leftarrow \mathbb{G}^2$
return $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})$
```

```
$\underline{\mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})}$                                              ▷ $\mathsf{G}_{13}$–$\mathsf{G}_{18}$
if $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            if $\Phi_{\mathcal{S},\mathsf{sid},k} = \perp$ then
                if $\exists i \in \mathcal{S} \setminus \{k\} : \Phi_{\mathcal{S},\mathsf{sid},i} = \perp$ then
                    $\Phi_{\mathcal{S},\mathsf{sid},k} \leftarrow \mathbb{G}^2$
                else
                    $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i\in\mathcal{S}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$        ▷ $\mathsf{G}_{13}$
                    $\Phi_{\mathcal{S},\mathsf{sid},k} := \prod_{i\in\mathcal{S}\setminus\mathcal{C}}(\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix}/\begin{bmatrix}R_{\mathcal{S},\mathsf{sid},1,i}\\ R_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix})^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$        ▷ *Eq.* (5), $\mathsf{G}_{14}$–$\mathsf{G}_{16}$
                    $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix}g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},1,i}}\\ g^{\sum_{i\in\mathcal{S}\setminus\mathcal{C}} r_{\mathcal{S},\mathsf{sid},2,i}}\end{bmatrix} \cdot \prod_{i\in\mathcal{S}\setminus\mathcal{C}}\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix}^{-1}$
                    $\cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$        ▷ *Eq.* (8), $\mathsf{G}_{17}$
                    $\Phi_{\mathcal{S},\mathsf{sid},k} := \begin{bmatrix}g^{r_{\mathcal{S},\mathsf{sid},1,*}}\\ g^{r_{\mathcal{S},\mathsf{sid},2,*}}\end{bmatrix} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}}\begin{bmatrix}g^{r_{\mathcal{S},\mathsf{sid},1,i}}\\ g^{r_{\mathcal{S},\mathsf{sid},2,i}}\end{bmatrix}^{-1} \cdot$
                    $\prod_{i\in\mathcal{S}\setminus\mathcal{C}}\begin{bmatrix}\tilde{R}_{\mathcal{S},\mathsf{sid},1,i}\\ \tilde{R}_{\mathcal{S},\mathsf{sid},2,i}\end{bmatrix}^{-1} \cdot \prod_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Phi^{-1}_{\mathcal{S},\mathsf{sid},i}$        ▷ *Eq.* (12), $\mathsf{G}_{18}$
                Initialize related $e_{\mathcal{S},\mathsf{sid},i,j}$ conditioned on $\Phi_{\mathcal{S},\mathsf{sid},k}$
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) := e_{\mathcal{S},\mathsf{sid},i,j}$
    else
        $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed}) \leftarrow \mathbb{G}^2$
return $\mathcal{T}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sid}, \mathsf{seed})$
```

Figure 11: $\mathsf{H}_{\mathrm{mask}}$ in the proof of Theorem 2.

```
$\underline{\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_0$–$\mathsf{G}_1$
if $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p$
return $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_2$–$\mathsf{G}_4$
if $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        $d_{\mathcal{S},\mathsf{sinf},i,j} \leftarrow \mathbb{Z}_p$
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p$
return $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_5$
if $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            for $(i', j') \in \{(k, i)\}_{\mathcal{S}\setminus\{k\}} \cup \{(i, k)\}_{i\in\mathcal{S}\setminus\{k\}}$ do      ▷ Initialize for the first reference
                if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then
                    $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p$
            $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i\in\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p$
return $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{18}$
if $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            if $\Delta_{\mathcal{S},\mathsf{sinf},k} = \perp$ then
                if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},i} = \perp$ then
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} \leftarrow \mathbb{Z}_p$
                else
                    $(\mathsf{sid}, \mu, \mathcal{M}) := \mathsf{sinf}$
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i\in\mathcal{S}\setminus k} \Delta_{\mathcal{S},\mathsf{sinf},i}$                    ▷ $\mathsf{G}_6$
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i\in\mathcal{S}\setminus\mathcal{C}}(r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) + c_{\mathcal{S},\mathsf{sinf}}(x - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i)$
                    $- \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$      ▷ Eq. (2), $\mathsf{G}_7$–$\mathsf{G}_{17}$
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sid},1,*} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,*} + c_{\mathcal{S},\mathsf{sinf}} \cdot x$
                    $- \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\mathcal{C}_{\mathcal{S},\mathsf{sid}}}(r_{\mathcal{S},\mathsf{sid},1,i} + b_{\mathcal{S},\mathsf{sinf}} \cdot r_{\mathcal{S},\mathsf{sid},2,i}) - c_{\mathcal{S},\mathsf{sinf}} \cdot \sum_{i\in\mathcal{S}\cap\mathcal{C}} \lambda_{\mathcal{S},i} \cdot x_i$
                    $- \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$      ▷ Eq. (10), $\mathsf{G}_{18}$
                Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p$
return $\mathcal{T}'_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$
```

Figure 12: $\mathsf{H}'_{\text{mask}}$ in the proof of Theorem 2.

```
$\underline{\mathsf{adp\text{-}UF}^{\mathsf{A}}_{\mathsf{HBTS\text{-}Mask}}}$
$\mathcal{Q} := \emptyset, \mathcal{C} := \emptyset$
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
$x \leftarrow \mathbb{Z}_p$
$X := g^x$
$\{x_i\}_{i\in[n]} \leftarrow \mathsf{Share}(n, t, x)$                                  ▷ $G_0$–$G_{10}$
for $i \neq j \in [n]$ do                                                                 ▷ $G_0$–$G_3$
    $\mathsf{seed}_{i,j} \leftarrow \{0,1\}^\lambda$                                      ▷ $G_0$–$G_3$
if $\exists (i,j) \neq (i',j') : \mathsf{seed}_{i,j} = \mathsf{seed}_{i',j'}$ then      ▷ $G_1$–$G_3$
    Game Abort                                                                            ▷ $G_1$–$G_3$
$\mathsf{pk} := X$
for $i \in [n]$ do
    $\mathsf{sk}_i := (x_i, \{\mathsf{seed}_{i,j}, \mathsf{seed}_{j,i}\}_{j\in[n]\setminus\{k\}})$
$(\mu^*, \sigma^*) \leftarrow \mathsf{A}^{\textsc{Sign},\textsc{Sign}',\textsc{Corr},\mathsf{H}_{\mathrm{mask}},\mathsf{H}_{\mathrm{gen}},\mathsf{H}_{\mathrm{chal}}}(\mathsf{pk})$
return $[\![\mu^* \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, \mu^*, \sigma^*) = 1]\!]$
```

Figure 13: Setup in proof of Theorem 3.

```
$\underline{\textsc{Corr}(k)}$
if $|\mathcal{C}| \geq t-1$ then
    return $\perp$
$\mathcal{C} := \mathcal{C} \cup \{k\}$
for $i \in [n] \setminus \{k\}$ do                                                        ▷ $G_4$–$G_{11}$
    if $i \notin \mathcal{C}$ then                                                        ▷ $G_4$–$G_{11}$
        $\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k} \leftarrow \{0,1\}^\lambda$             ▷ $G_4$–$G_{11}$
if $\exists (i,j) \neq (i',j') : \mathsf{seed}_{i,j} = \mathsf{seed}_{i',j'} \neq \perp$ then   ▷ $G_4$–$G_{11}$
    Game Abort                                                                            ▷ $G_4$–$G_{11}$
$x_k \leftarrow \mathbb{Z}_p$                                                             ▷ $G_{11}$
for $\mathsf{sid} \in [\mathsf{ctr}_k]$ do                                                ▷ $G_9$–$G_{11}$
    if $\mathsf{round}_{k,\mathsf{sid}} = 2$ then                                         ▷ $G_9$–$G_{11}$
        $\mathsf{sinf} := \mathsf{sinf}_{k,\mathsf{sid}}$
        $z_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sinf},k} + c_{\mathcal{S},\mathsf{sinf}}\lambda_{\mathcal{S},k}x_k$   ▷ $G_{10}$–$G_{11}$
        $\Delta_{\mathcal{S},\mathsf{sinf},k} := \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} - \begin{bmatrix} z_{\mathcal{S},\mathsf{sinf},k} \\ y_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix}$   ▷ $G_9$–$G_{11}$
        Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$   ▷ $G_9$–$G_{11}$
$\mathsf{sk}_k := (x_k, \{\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k}\}_{i\in[n]\setminus\{k\}})$   ▷ $G_4$–$G_{11}$
return $(\mathsf{sk}_k, \{\mathsf{st}_{k,\mathsf{sid}}\}_{\mathsf{sid}\in[\mathsf{ctr}_i]})$

$\underline{\text{Initialize related } d_{\mathcal{S},\mathsf{sinf},i,j} \text{ conditioned on } \Delta_{\mathcal{S},\mathsf{sinf},k}}$
Let $i^*$ be the smallest number in $\mathcal{S} \setminus \{k\}$ such that $d_{k,i} = \perp$
for $(i',j') \in \{(k,i)\}_{i\in\mathcal{S}\setminus\{k\}} \cup \{(i,k)\}_{i\in\mathcal{S}\setminus\{k\}} \setminus \{(k,i^*)\}$ do   ▷ Sample all but one
    if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then
        $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p^2$
$d_{\mathcal{S},\mathsf{sinf},k,i^*} := \Delta_{\mathcal{S},\mathsf{sinf},k} - \sum_{i\in\mathcal{S}\setminus\{k\}\setminus\{i^*\}} d_{\mathcal{S},\mathsf{sinf},k,i} + \sum_{i\in\mathcal{S}\setminus\{k\}} d_{\mathcal{S},\mathsf{sinf},i,k}$   ▷ Calculate the last one
```

Figure 14: Corr in proof of Theorem 3.

```
$\underline{\textsc{Sign}'(k, \mathsf{sid}, \mathcal{M})}$
if $k \in \mathcal{C} \vee \mathsf{round}_{k,\mathsf{sid}} \neq 1$ then
    return $\perp$
$(\mathcal{S}, \mu) := (\mathcal{S}_{k,\mathsf{sid}}, \mu_{k,\mathsf{sid}})$
$\{\mathsf{msg}_i\}_{i \in \mathcal{S}} := \mathcal{M}$
for $i \in \mathcal{S}$ do
    $R_i := \mathsf{msg}_i$
$R := \prod_{i \in \mathcal{S}} R_i$
$\mathsf{sinf} := (\mu, \mathcal{M})$
$(r_{\mathcal{S},\mathsf{sinf},k}, y_{\mathcal{S},\mathsf{sinf},k}) := \mathsf{st}_{k,\mathsf{sid}}$                        ▷ Global variables
$c_{\mathcal{S},\mathsf{sinf}} := \mathsf{H}_{\mathrm{chal}}(\mu, R)$
$z_{\mathcal{S},\mathsf{sinf},k} := r_{\mathcal{S},\mathsf{sinf},k} + c_{\mathcal{S},\mathsf{sinf}} \lambda_{\mathcal{S},k} x_k$                        ▷ $\mathsf{G}_0$–$\mathsf{G}_9$

$\underline{\mathsf{G}_0\text{–}\mathsf{G}_5}$
$\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S}\setminus\{k\}} \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{k,i}) - \sum_{i \in \mathcal{S}\setminus\{k\}} \mathsf{H}_{\mathrm{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}_{i,k})$        ▷ $\mathsf{G}_0$–$\mathsf{G}_2$
for $(i', j') \in \{(k,i)\}_{i \in \mathcal{S}\setminus\{k\}} \cup \{(i,k)\}_{i \in \mathcal{S}\setminus\{k\}}$ do        ▷ Initialize for the first reference, $\mathsf{G}_3$–$\mathsf{G}_5$
    if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then                        ▷ $\mathsf{G}_3$–$\mathsf{G}_5$
        $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p^2$                        ▷ $\mathsf{G}_3$–$\mathsf{G}_5$
$\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S}\setminus\{k\}} (d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$                        ▷ $\mathsf{G}_5$
$\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} := \begin{bmatrix} z_{\mathcal{S},\mathsf{sinf},k} \\ y_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} + \Delta_{\mathcal{S},\mathsf{sinf},k}$                        ▷ $\mathsf{G}_5$–$\mathsf{G}_7$

$\underline{\mathsf{G}_6\text{–}\mathsf{G}_7}$
if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},i} = \perp$ then                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{11}$
    $\Delta_{\mathcal{S},\mathsf{sinf},k} \leftarrow \mathbb{Z}_p^2$                        ▷ $\mathsf{G}_6$–$\mathsf{G}_7$
else                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{11}$
    $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i \in \mathcal{S}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$                        ▷ $\mathsf{G}_6$
    for $i \in \mathcal{S} \setminus \mathcal{C} \setminus \{k\}$ do                        ▷ $\mathsf{G}_7$–$\mathsf{G}_{18}$
    $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} ([\begin{smallmatrix} x \\ 0 \end{smallmatrix}] - \sum_{i \in \mathcal{S}\cap\mathcal{C}\cup\{k\}} \begin{bmatrix} \lambda_{\mathcal{S},i} x_i \\ 0 \end{bmatrix})$
        $- \sum_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}$                        ▷ $\mathsf{G}_7$
$\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} := \begin{bmatrix} z_{\mathcal{S},\mathsf{sinf},k} \\ y_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} + \Delta_{\mathcal{S},\mathsf{sinf},k}$                        ▷ $\mathsf{G}_5$–$\mathsf{G}_7$
Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$                        ▷ $\mathsf{G}_6$–$\mathsf{G}_8$

$\underline{\mathsf{G}_8\text{–}\mathsf{G}_{11}}$
if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},k} = \perp$ then                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{11}$
    $\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} \leftarrow \mathbb{Z}_p^2$                        ▷ $\mathsf{G}_8$–$\mathsf{G}_{11}$
else                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{11}$
    $\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} := \sum_{i \in \mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} ([\begin{smallmatrix} x \\ 0 \end{smallmatrix}] - \sum_{\mathcal{S}\cap\mathcal{C}} \begin{bmatrix} \lambda_{\mathcal{S},i} x_i \\ 0 \end{bmatrix})$
        $- \sum_{i \in \mathcal{S}\setminus\mathcal{C}\setminus\{k\}} \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \Delta_{\mathcal{S},\mathsf{sinf},i}$                        ▷ Eq. (15), $\mathsf{G}_8$–$\mathsf{G}_{11}$
$\Delta_{\mathcal{S},\mathsf{sinf},k} := \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix} - \begin{bmatrix} z_{\mathcal{S},\mathsf{sinf},k} \\ y_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix}$                        ▷ $\mathsf{G}_8$
$\Delta_{\mathcal{S},\mathsf{sinf},k} :=?$                        ▷ $\mathsf{G}_9$–$\mathsf{G}_{11}$
Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$                        ▷ $\mathsf{G}_6$–$\mathsf{G}_8$

$\mathsf{sinf}_{k,\mathsf{sid}} := \mathsf{sinf}$
$\mathsf{round}_{k,\mathsf{sid}} := 2$
return $\begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},k} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},k} \end{bmatrix}$
```

Figure 15: $\textsc{Sign}'$ in the proof of Theorem 3.

```
$\underline{\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_0$–$\mathsf{G}_1$
if $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p^2$
return $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_2$–$\mathsf{G}_4$
if $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        $d_{\mathcal{S},\mathsf{sinf},i,j} \leftarrow \mathbb{Z}_p^2$
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p^2$
return $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_5$
if $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            for $(i', j') \in \{(k, i)\}_{\mathcal{S}\setminus\{k\}} \cup \{(i, k)\}_{i\in\mathcal{S}\setminus\{k\}}$ do     ▷ Initialize for the first reference
                if $d_{\mathcal{S},\mathsf{sinf},i',j'} = \perp$ then
                    $d_{\mathcal{S},\mathsf{sinf},i',j'} \leftarrow \mathbb{Z}_p^2$
            $\Delta_{\mathcal{S},\mathsf{sinf},k} := \sum_{i\in\mathcal{S}\setminus\{k\}}(d_{\mathcal{S},\mathsf{sinf},k,i} - d_{\mathcal{S},\mathsf{sinf},i,k})$
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p^2$
return $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$

$\underline{\mathsf{H}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})}$                                        ▷ $\mathsf{G}_6$–$\mathsf{G}_{11}$
if $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) = \perp$ then
    if $\exists i \neq j \in \mathcal{S} : \mathsf{seed} = \mathsf{seed}_{i,j}$ then
        for $k \in \{i, j\}$ do
            if $\exists i \in \mathcal{S} \setminus \{k\} : \Delta_{\mathcal{S},\mathsf{sinf},i} = \perp$ then
                if $\Delta_{\mathcal{S},\mathsf{sinf},k} = \perp$ then
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} \leftarrow \mathbb{Z}_p^2$
                else
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} := -\sum_{i\in\mathcal{S}\setminus k} \Delta_{\mathcal{S},\mathsf{sinf},i}$                        ▷ $\mathsf{G}_6$
                    $\Delta_{\mathcal{S},\mathsf{sinf},k} = \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} r_{\mathcal{S},\mathsf{sinf},i} \\ y_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} + c_{\mathcal{S},\mathsf{sinf}} \cdot \left(\begin{bmatrix} x \\ 0 \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \begin{bmatrix} \lambda_{\mathcal{S},i} \cdot x_i \\ 0 \end{bmatrix}\right)$
                    $- \sum_{i\in\mathcal{S}\setminus\mathcal{C}} \begin{bmatrix} \tilde{z}_{\mathcal{S},\mathsf{sinf},i} \\ \tilde{y}_{\mathcal{S},\mathsf{sinf},i} \end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}\setminus\{k\}} \Delta_{\mathcal{S},\mathsf{sinf},i}$      ▷ Eq. (14), $\mathsf{G}_7$–$\mathsf{G}_{11}$
                Initialize related $d_{\mathcal{S},\mathsf{sinf},i,j}$ conditioned on $\Delta_{\mathcal{S},\mathsf{sinf},k}$
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) := d_{\mathcal{S},\mathsf{sinf},i,j}$
    else
        $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed}) \leftarrow \mathbb{Z}_p^2$
return $\mathcal{T}_{\text{mask}}(\mathcal{S}, \mathsf{sinf}, \mathsf{seed})$
```

Figure 16: H_{mask} in the proof of Theorem 3.