Partially Non-Interactive Two-Round Threshold and Multi-Signatures with Tighter and Adaptive Security

Yanbo Chen*

September 9, 2026

Abstract

Threshold and multi-signature schemes enable a group of signers to jointly sign messages. In the pairing-free discrete-logarithm setting, there have been extensive research efforts on two-round threshold and multi-signature schemes. A prominent category is partially non-interactive schemes, where the first round is message-independent and can be preprocessed offline. However, there exist gaps in security properties between partially non-interactive schemes and fully online schemes. While there are fully online schemes that achieve rewinding-free or fully adaptive security based on standard assumptions, existing partially non-interactive schemes require the adversary to act algebraically or rely on non-standard assumptions for these properties.

We bridge this gap by proposing a family of new constructions of partially non-interactive threshold and multi-signature schemes. Central to our constructions is a technique that renders HBMS (Bellare and Dai, Asiacrypt 2021) and its variants partially non-interactive. Specifically, we present the following instances:

– The first partially non-interactive two-round multi-signature scheme with a rewinding-free reduction under standard assumptions. This scheme provides a tighter security guarantee against non-algebraic adversaries than all prior partially non-interactive schemes.

– The first partially non-interactive two-round threshold signature scheme achieving fully adaptive security against non-algebraic adversaries under standard assumptions.

1 Introduction

Threshold signature schemes [Des88] and multi-signature schemes [IN83] are protocols for distributed signature generation. Threshold signatures distribute the signing key among a group of users, such that any subgroup of users that exceeds a threshold can sign on behalf of the group. Multi-signatures allow users with individual key pairs to dynamically form a group and jointly sign. Threshold and multi-signatures have both received renewed attention for applications in cryptocurrencies (e.g., [GGN16, MPSW19]).

This work focuses on threshold and multi-signature schemes over pairing-free groups. Unlike pairing-based signatures [BLS01] which typically support efficient, non-interactive distributed signing [Bol03], practical pairing-free threshold and multi-signatures require the signers to interact. Naturally, the number of rounds becomes a central criterion, and a number of schemes have realized two-round signing protocols.

Among the two-round schemes, a family has the property that their first round can be preprocessed without knowing the message and the signer set. Therefore, with preprocessing, the online signing phase after the message and signer set are determined is effectively non-interactive. Such schemes enjoy some advantages of non-interactive protocols and are called *partially non-interactive*. This family consists of the threshold Schnorr scheme FROST [KG20], Schnorr multi-signature schemes

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

DWMS/MuSig2 [AB21, NRS21], and their variants [KG20, BCK+22, RRJ+22, HDL+25, BCL+26, BGC+25].

On the other hand, there are a number of interesting two-round schemes where both rounds are online, yet they achieve more favorable security results:

- Tighter security: [PW23, PW24, TSS+24, BW25, BC26, LÖ26] proposed two-round multi-signature schemes with rewinding-free security reductions in the ROM. Such reductions avoid invoking the forking lemma [PS96, BN06] and hence enjoy tighter or even fully tight security bounds. There also exist two-round threshold signature schemes [Che25a, BC26] with rewinding-free tighter security. In comparison, the security proofs for existing partially non-interactive threshold and multi-signature schemes rely on either the algebraic group model (AGM) [FKL18] or rewinding. The AGM requires the adversary to act algebraically, while rewinding results in a quadratic security loss. Multi-signatures with public key aggregation [NRS21, TZ23] even require nested rewinding and thus suffer from a quartic loss.
- Adaptive security: The adaptive security model considers an adversary that can adaptively corrupt users at any time to obtain their secret key shares and full history. There have been fully online two-round threshold signatures with full adaptive security in the ROM from standard assumptions [Che25a, Che25b, GCRS26].
  By contrast, FROST and its variants [KG20, CKK+25, BCL+26, BGC+25] are only proved fully adaptively secure in the AGM in addition to the ROM. Indeed, several works provide meta-reductions [CKM25, BCL+26] that formally establish barriers to proving their adaptive security without the AGM.

1.1 Contributions

A Family of New Partially Non-Interactive Schemes.

We propose a family of new constructions of partially non-interactive two-round threshold and multi-signatures. The key to our constructions is an elegant modification to the two-round scheme HBMS by Bellare and Dai [BD21] that makes the first-round preprocessable with only minor overhead. Particularly, our modification does not increase the signature size. All our schemes produce signatures consisting of three field/group elements.

While HBMS is based on DL, the scheme TSSHO [TSS+24] can be viewed as a DDH-based variant of HBMS. Moreover, HBMS has a straightforward threshold signature version called HBTS [Che25b]. Therefore, our modification on HBMS provides a family of threshold and multi-signature schemes offering both DL-based and DDH-based instantiations. We call them HBMS2[DL], HBMS2[DDH], HBTS2[DL], HBTS2[DDH]. We can also apply the masking technique from [KRT24, Che25b] to our threshold signature schemes to obtain adaptively secure schemes HBTS2[DL]-Mask, HBTS2[DDH]-Mask. Among these new schemes, we present HBMS2[DDH] and HBTS2[DDH]-Mask as primary contributions, which achieve the desirable security results that we mentioned and previously only achieved by schemes with two online rounds.

Multi-Signatures with Tighter Security.

Our multi-signature construction HBMS2[DDH] supports non-interactive public-key aggregation. Like TSSHO [TSS+24], our scheme can be proved secure in the ROM by a rewinding-free security reduction. Consequently, our scheme has a security loss of $\Theta(q_h)$, where q_h is the number of hash queries made by the adversary. Namely, the security reduction that solves DDH succeeds with probability $\Theta(\varepsilon/q_h)$, where ε is the success probability of any adversary that breaks the security of HBMS2[DDH]. In comparison, existing partially non-interactive schemes with key aggregation all rely on a nested rewinding security reduction. Their security bounds are $\Theta(\varepsilon^4/q_h^3)$. Even without key aggregation, they still rely on rewinding and have a bound of $\Theta(\varepsilon^2/q_h)$ that suffers from a quadratic loss.

Scheme	Round	Sig.	PK.	Comm.	Assumption	Bound
Musig-DN [NRSW20]	2	$1\|\mathbb{G}\|+1\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$2\|\mathbb{G}\|+1\|\mathbb{Z}_p\|+\|\pi\|$	DDH	ε^4/q_h^3
HBMS [BD21]	2	$1\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$1\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	DL	$\varepsilon^4/(q_s^4 q_h^3)$
Chopsticks I [PW23]	2	$3\,\|\mathbb{G}\|+4\,\|\mathbb{Z}_p\|$	$2\,\|\mathbb{G}\|$	$3\,\|\mathbb{G}\|+1\,\|\mathbb{Z}_p\|+\lambda$	DDH	ε/q_s
Toothpicks I [PW24]	2	$3\,\|\mathbb{Z}_p\|+2\lambda$	$2\,\|\mathbb{G}\|$	$2\,\|\mathbb{G}\|+1\,\|\mathbb{Z}_p\|+\lambda$	DDH	ε/q_s
TSSHO [TSS+24]	2	$3\|\mathbb{Z}_p\|$	$2\|\mathbb{G}\|$	$2\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	DDH	ε/q_s
T-Spoon [BW25]	2	$2\|\mathbb{G}\|+9\|\mathbb{Z}_p\|$	$2\|\mathbb{G}\|$	$3\|\mathbb{G}\|+1\|\mathbb{Z}_p\|+\lambda+1$	DDH	ε
Earpick [BC26]	2	$3\|\mathbb{Z}_p\|+1$	$2\|\mathbb{G}\|$	$5\|\mathbb{G}\|+4\|\mathbb{Z}_p\|$	DDH	ε
MuSig2 [NRS21]	1+1	$1\|\mathbb{G}\|+1\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$4\|\mathbb{G}\|+1\|\mathbb{Z}_p\|$	AOMDL	ε^4/q_h^3
TZ [TZ23]	1+1	$1\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$4\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	DL	ε^4/q_h^3
HBMS2[DDH]	1+1	$3\|\mathbb{Z}_p\|$	$2\|\mathbb{G}\|$	$4\|\mathbb{G}\|+2\|\mathbb{Z}_p\|$	DDH	ε/q_h

Table 1: Comparison between pairing-free two-round multi-signature schemes with key aggregation. We compare the round complexity of the signing protocol, the signature size, the public-key size, the per-signer communication complexity, the underlying hardness assumption, and the security bound. For round complexity, "1+1" means that the first round is offline preprocessable. We let $|\mathbb{G}|$ and $|\mathbb{Z}_p|$ denote the sizes of a group element and a field element, respectively. We use λ to denote the security parameter. For Musig-DN, $|\pi|$ denotes the size of a zk-SNARK that proves correct computation of deterministic nonces. The security bound is the success probability of the reduction that solves the underlying hard problem, expressed as a function of the adversary's success probability ε, ignoring Θ factors. For the security bound, we only consider proofs in the ROM without the AGM.

We provide some concrete numbers to give some sense about these bounds. Consider standard cryptographic groups of order p with $p \approx 2^{256}$. Assume a τ-time algorithm can solve the DL-related hard problems with probability at most τ^2/p (ignoring Θ factors). Consider an adversary with $\tau \approx q_h \approx 2^{60}$. Then our bound guarantees a success probability of about 2^{-76}. The bound $\Theta(\varepsilon^2/q_h)$ for previous schemes without key aggregation only guarantees 2^{-38}, while the bound $\Theta(\varepsilon^4/q_h^3)$ for previous schemes with key aggregation offers no meaningful guarantee. In Section A, we provide concrete efficiency comparison between partially non-interactive multi-signature schemes under parameter settings that guarantee 128-bit security. Notably, our HBMS2[DDH] can safely use a 384-bit standard group, such as NIST P-384, while MuSig2 and TZ require 640-bit groups, which lacks standardized support.

We remark that our HBMS2[DDH] is not fully tight. Furthermore, our scheme has looser security than its fully online counterpart [TSS+24]. Our scheme has a security loss of $\Theta(q_h)$, while [TSS+24] achieves a security loss of $\Theta(q_s)$, where the number of hash queries q_h is typically considered larger than the number of signing queries q_s. We consider further improving the tightness for partially non-interactive schemes an interesting future direction.

Table 1 compares our HBMS2[DDH] with existing pairing-free two-round multi-signature schemes with key aggregation.[1]

Threshold Signatures with Adaptive (and Tighter) Security.

Our DDH-based threshold signature scheme HBTS2[DDH] also achieves a tighter reduction for static security than existing partially non-interactive schemes in the ROM. Moreover, when combining with the masking technique [KRT24, Che25b], we obtain HBTS2[DDH]-Mask with full adaptive security in the ROM. This is the *first* partially non-interactive threshold signature scheme proved to achieve full adaptive security in the ROM from standard assumptions.

Previously, only FROST [KG20, CKK+25] and ms-FROST [BCL+26] are proved fully adaptively secure, based on the algebraic one-more discrete logarithm (AOMDL) assumption, but both in the AGM. In addition, FROST *necessarily* requires a new hardness assumption unrelated to DL called

[1] We did not list [AB21, HDL+25, LÖ25]. DWMS [AB21] is highly similar to MuSig2 but is only proved secure in the AGM. [HDL+25] is a privacy-preserving variant of MuSig2. Skewer-PF [LÖ26] is a fully online two-round scheme that achieves tight multi-user security, yet it requires proofs of possession.

Scheme	Round	Sig.	PK.	Comm.	Assumption	AGM?	Bound
Dazzle [Che25a]	2	$1\|\mathbb{G}\| + 3\|\mathbb{Z}_p\|$	$2\|\mathbb{G}\|$	$4\|\mathbb{G}\| + 2\|\mathbb{Z}_p\|$	DDH	✗	ε/q_s
HBTS-Mask [Che25b]	2	$1\|\mathbb{G}\| + 2\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$1\|\mathbb{G}\| + 2\|\mathbb{Z}_p\|$	DL	✗	$\varepsilon^2/(q_s^2 q_h)$
GCRS [GCRS26]*	2	$1\|\mathbb{G}\| + 1\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$\Theta(\log d)(\|\mathbb{G}\| + \|\mathbb{Z}_p\|)$	DDH	✗	ε^2/q_h
FROST [KG20, CKK+25]	1+1	$1\|\mathbb{G}\| + 1\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$2\|\mathbb{G}\| + 1\|\mathbb{Z}_p\|$	AOMDL, LDVR	✓	ε†
ms-FROST [BCL+26, BGC+25]	1+1	$1\|\mathbb{G}\| + 1\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$2\|\mathbb{G}\| + 1\|\mathbb{Z}_p\|$	AOMDL	✓	ε
HBTS2[DDH]-Mask	1+1	$3\|\mathbb{Z}_p\|$	$2\|\mathbb{G}\|$	$4\|\mathbb{G}\| + 2\|\mathbb{Z}_p\|$	DDH	✗	ε/q_h
HBTS2[DL]-Mask	1+1	$3\|\mathbb{Z}_p\|$	$1\|\mathbb{G}\|$	$2\|\mathbb{G}\| + 2\|\mathbb{Z}_p\|$	DL	✗	ε^2/q_h^3

Table 2: Comparison between pairing-free two-round threshold signature schemes with full adaptive security. We compare the round complexity of the signing protocol, the signature size, the public-key size, the per-signer communication complexity, the underlying hardness assumption, whether the proof of adaptive security relies on the AGM, and the security bound. The security bound is the success probability of the reduction that solves the underlying hard problem, expressed as a function of the adversary's success probability ε, ignoring Θ factors.
*: After issuing a pre-specified bound of d signatures, GCRS requires the signers to refresh their keys.
†: We remark that these tight bounds are in the AGM. In general, proofs for threshold signatures in the AGM are usually tight.

Low-Dimensional Vector Representation (LDVR) [CS25, CKK+25]. Our HBTS2[DDH]-Mask avoids both AGM and LDVR.

The masking technique by [KRT24] is somewhat generic and can be applied to several different schemes to gain adaptive security. Chen [Che25b] applies it to FROST to achieve non-AGM full adaptive security, but that results in a three-round or stateful two-round scheme. ms-FROST [BCL+26] is another way to apply the masking technique to FROST, which yields a partially non-interactive two-round scheme but requires the AGM.

As observed by [Che25b], HBTS is more masking-friendly, in the sense that it can be masked to achieve non-AGM full adaptive security without increasing the round number or making the scheme stateful. This turns out to be also true for our HBTS2. We remark that being rewinding-free is not necessary for the masking technique to be applicable. Masking our HBTS2[DL] would give a fully adaptively secure scheme HBTS2[DL]-Mask based on DL (but with a looser reduction).

Table 2 compares our HBTS2[DDH]-Mask and HBTS2[DL]-Mask with existing pairing-free two-round threshold signature schemes with full adaptive security.

Concurrent Work.

Concurrent to this work, Jiang, Wee, and Zhu [JWZ26] proposed a lattice-based partially non-interactive two-round threshold signature scheme with adaptive security. The construction uses lattice trapdoors, so their techniques do not readily adapt to the pairing-free discrete logarithm setting.

1.2 Related Work

Multi-Signatures and Tighter Security.

Provably secure multi-signatures in early literature rely on interactive key generation protocols [MOR01] or the knowledge of secret key (KOSK) assumption [Bol03, LOS+06]. Bellare and Neven proposed the first multi-signature scheme in the *plain public-key model* [BN06], where each signer locally generates its own key and is not assumed to have proved the knowledge of its secret key. This model is now widely accepted as the most robust and realistic one for multi-signatures. We remark

that there is a substantial body of work considering the Proof of Possession (PoP) model [RY07], similar to yet weaker than KOSK.

Recent years have seen a resurgence of interest in multi-signatures, largely driven by their application in cryptocurrency as highlighted in [MPSW19]. The work also formalized *public-key aggregation*, which allows a group of signers to only publish a compact aggregated key for verification. Support for public-key aggregation has become a common practice in designing multi-signature schemes.

In the pairing-free discrete logarithm setting, early three-round schemes were proposed in [MOR01, BN06, BJ08, BDN18, MPSW19]. While some early two-round proposals [BCJ08, MWLD10, STV+16, MPSW19] were shown to be insecure [DEF+19, BLL+21], a number of provably secure two-round schemes later emerged [DEF+19, NRSW20, AB21, NRS21, BD21, TZ23, PW23, PW24, RSY24, TSS+24, HDL+25, BW25, LÖ25], where [AB21, NRS21, TZ23, HDL+25] are partially non-interactive. [BDPT25] also proposed an ordered multi-signature scheme.

However, a major challenge in this domain, as well as in threshold signatures, remains the tightness of security reductions. The security of cryptographic schemes is usually proved by building a reduction that given access to an adversary breaking the security with probability ε_a, breaks the underlying hard problem with probability ε_r. All the previously known partially non-interactive two-round threshold and multi-signature schemes over pairing-free groups are proved secure in the AGM or with rewinding reductions. Such reductions establish a loose bound $\varepsilon_r \geq \varepsilon_a^2/\Theta(q_h)$. For multi-signature schemes with public-key aggregation, the reductions apply nested rewinding and obtain an even looser bound $\varepsilon_r \geq \varepsilon_a^4/\Theta(q_h^3)$.

Such loose security bounds constitute a significant drawback for concrete security and parameter choices. To address this, a line of research has successfully constructed multi-signatures [FH21, PW23, PW24, BW25, BC26, LÖ26] and threshold signatures [Che25a, BW26, BC26] with fully tight security. Other rewinding-free yet non-tight schemes were proposed in [PW23, PW24, TSS+24]. Furthermore, tightness of security proofs has also been studied in [BD21, RSY24, SS24]. Despite these advances, none of the threshold or multi-signature schemes achieving tighter bounds or rewinding-free security are partially non-interactive.

Threshold Signatures and Adaptive Security.

Initial work in pairing-free discrete logarithm-based threshold signatures [GJKR07, CGJ+99, SS01] focused on robust schemes. Since the proposal of FROST [KG20], which compromises on robustness for a lightweight protocol, many works considered threshold signatures in fewer rounds [BCK+22, RRJ+22, TZ23, CGRS23, Lin24].

A critical aspect of these schemes is their security model. A basic model is the *static* model, where the adversary first statically corrupts t users before key generation. A stronger, more realistic model is the *adaptive* model, allowing corruption after initiation. Adaptive security was considered in early robust schemes [CGJ+99, FMY99, JL00, LP01, AF04], and its study for lightweight schemes was initiated by [BL22, Mak22, CKM23]. A generic guessing argument reduces adaptive security to static security at the cost of a factor of $\binom{n}{t}$. While equivalent for small groups, this loss grows exponentially with n. For instance, for $n = 128$ and $t = 42 \approx 128/3$, the loss exceeds 2^{113}, meaning a scheme with 128 bits of static security essentially provides no confidence against adaptive attacks.

This discrepancy highlights the need for natively adaptively secure schemes. FROST is a state-of-the-art threshold Schnorr scheme with a partially non-interactive two-round signing protocol. However, recent progress [CS25, CKK+25] showed that its full adaptive security necessarily relies on an additional, unusual assumption called Low-Dimensional Vector Representation (LDVR), unrelated to DL. Moreover, even assuming LDVR, its full adaptive security is proved in the AGM [CKK+25], with a meta-reduction [CKM25] establishing a barrier to a non-AGM proof.

There has been a subsequent line of work constructing adaptively secure threshold signature schemes from standard, well-understood assumptions [BLT+24, KRT24, BDLR25b, Che25a,

BDLR25a, Che25b, GCRS26, NRT26, BCL+26, BGC+25], mostly without the AGM. Among these, [Che25a, Che25b, GCRS26, BCL+26, BGC+25] proposed fully adaptively secure two-round schemes. Unfortunately, the only partially non-interactive scheme ms-FROST, independently proposed in [BCL+26] and [BGC+25], is indeed the only one relying on the AGM. Similar to FROST, a barrier to a non-AGM proof for it is established by a meta-reduction [BCL+26]. Prior to our work, there exists no partially non-interactive scheme achieving non-AGM fully adaptive security.

Finally, we mention that other important research directions in the pairing-free discrete-logarithm setting include threshold ECDSA [GGN16] and robustness in asynchronous networks [RRJ+22]. Such schemes are not as lightweight as the works mentioned above. Recent progress pushed the round complexity of threshold ECDSA down to two rounds [LLZD25, DGNP25], but uses substantially heavier cryptographic tools (class groups) and considers only static security. For asynchronous settings, [BLSW24] considers adaptive security, but requires more rounds, a suboptimal corruption threshold, and the AGM.

2 Technical Overview

In this section, we provide an overview of our techniques. Essentially, we are not developing any new techniques toward rewinding-free and adaptive security. We are using the techniques from [TSS+24] and [KRT24, Che25b], which can be understood as making HBMS [BD21] rewinding-free and HBTS adaptively secure, respectively. The core of our contribution is a simple yet effective modification to HBMS that renders it partially non-interactive. Meanwhile, we preserve most of the structures, so the techniques from [TSS+24] and [KRT24, Che25b] remain perfectly applicable. This brings rewinding-free and adaptive security to partially non-interactive schemes for the first time.

2.1 A Recap on HBMS

Let us recall the construction of HBMS [BD21].

- Setup: HBMS runs over a cyclic group $\mathbb{G}$ of prime order p with generator g.
- Key generation: The secret key $x \leftarrow \mathbb{Z}_p$ is chosen uniformly at random. The public key is $X := g^x$.
- Key aggregation: The key aggregation algorithm KAgg outputs $\mathsf{apk} := \prod_{i\in[n]} X_i^{a_i}$, where $a_i := \mathsf{H}_a(\mathcal{L}, X_i)$, and $\mathcal{L}$ is the public-key (multi-)set.
- Signing: Suppose n users are going to jointly sign on a message μ. They sign as follows:
  - In the first round, each signer (without loss of generality, indexed by 1) uniformly chooses $r_1, w_1 \leftarrow \mathbb{Z}_p$ and broadcasts $R_1 := g^{r_1}h^{w_1}$, where $h := \mathsf{H}_h(\mathsf{apk}, \mu)$.
  - In the second round, each signer receives $\{R_i\}_{i\in[n]}$ from other signers. It computes $(z_1, y_1) := (r_1 + a_1 c x_1, w_1)$ for $c := \mathsf{H}_c(\mathsf{apk}, \mu, R)$, where $R := \prod_{i\in[n]} R_i$. It broadcasts (z_1, y_1).
  - Given $\{(z_i, y_i)\}_{i\in[n]}$ from the signers, the final multi-signature is generated as $\sigma := (c, z, y)$, where $z := \sum_{i\in[n]} z_i$ and $y := \sum_{i\in[n]} y_i$.
- Verification: The verifier recovers $R := g^z h^y \mathsf{apk}^{-c}$, where $h := \mathsf{H}_h(\mathsf{apk}, \mu)$, and checks if $c = \mathsf{H}_c(\mathsf{apk}, \mu, R)$.

One way to understand HBMS is a multi-signature version of the Okamoto signature scheme [Oka93], where h is a hash of (apk, μ) rather than a global parameter. Any (α, β) such that $g^\alpha h^\beta = X$ can be used to generate Okamoto signatures. In HBMS, the signer uses $(x, 0)$ as the Okamoto signing key, which works regardless of the choice of h.

Security Proof.

HBMS is proved secure under the DL assumption. The goal of the adversary in the security game is to forge a multi-signature under a public-key set that contains the honest target public key X_*. In the security proof, the reduction sets the target public key X_* to be the input group element.

The reduction sets $\mathsf{H}_h(\mathsf{apk}, \mu)$ to be g^α with $\alpha \leftarrow \mathbb{Z}_p$ for a partition of (apk, μ), which we call the "extractable" partition. It sets $\mathsf{H}_h(\mathsf{apk}, \mu)$ to be $g^\alpha X_*^\beta$ with $\alpha \leftarrow \mathbb{Z}_p$ and $\beta \neq 0$ for the other partition, which we call the "signable" partition.

For every (apk, μ) from the signable partition, it holds that $g^{-\alpha/\beta} h^{1/\beta} = g^{-\alpha/\beta} \cdot (g^\alpha X_*^\beta)^{1/\beta} = X_*$. Thus, the reduction knows an (apk, μ)-specific Okamoto signing key $(-\alpha/\beta, 1/\beta)$ and can sign (apk, μ). The reduction expects that the honest signer is only queried to sign (apk, μ) from the signable partition, so it can simulate the security game.

The reduction expects that, on the other hand, the adversary returns a forged signature (c_*, z_*, y_*) for (apk_*, μ_*) from the extractable partition. Let $R_* = g^{z_*} h^{y_*} \mathsf{apk}_*^{-c_*}$, where $h = \mathsf{H}_h(\mathsf{apk}_*, \mu_*)$. The validity of the forged signature requires that $\mathsf{H}_c(\mathsf{apk}_*, \mu_*, R_*) = c_*$. Since (apk_*, μ_*) lies in the extractable partition, the reduction knows the discrete logarithm α of h. Therefore, we have $R_* = g^{z_* + \alpha y_*} \mathsf{apk}_*^{-c_*}$. Essentially, for the extractable partition, the signature degrades to a Schnorr signature $(R_*, z_* + \alpha y_*)$. The reduction can apply the same (nested) rewinding as in Schnorr multi-signatures [MPSW19, NRS21] to extract the discrete logarithm of X_*.

2.2 From HBMS to HBMS2[DL]: Our Core Technique

The first round of HBMS cannot be preprocessed, as the signer needs to compute h from apk and μ. We resolve this by using another way to derive h from a hash function:

> Let u and v be public group elements. We let $h := uv^b$ for $b := \mathsf{H}_b(\mathsf{apk}, \mu)$.

The main idea of the security proof for HBMS still applies. The reduction sets $u := g^\alpha X_*^\gamma$ and $v := g^\beta X_*^{-1}$ with $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$. It follows that $h = uv^b = g^{\alpha + b\beta} X_*^{\gamma - b}$. The reduction guesses the query $\mathsf{H}_b(\mathsf{apk}_*, \mu_*)$ that will correspond to the adversary's forgery and programs $\mathsf{H}_b(\mathsf{apk}_*, \mu_*) := \gamma$. Consequently, we have $h = g^{\alpha + \beta\gamma}$, which puts (apk_*, μ_*) to the extractable partition. For any other (apk, μ), we have $\mathsf{H}_b(\mathsf{apk}, \mu) \neq \gamma$ with high probability. Consequently, we have $h = g^{\alpha + b\beta} X_*^{\gamma - b}$ with $\gamma - b \neq 0$, which puts (apk, μ) to the signable partition with a (apk, μ)-specific Okamoto signing key $(-(\alpha + b\beta)/(\gamma - b), 1/(\gamma - b))$. We note that such a partitioning method is inspired by the Boneh-Boyen identity-based encryption and signature scheme [BB04].

Now let us see the magic: h becomes a linear combination of public elements u and v. In the first round, each signer (indexed by 1) can just compute two partial nonces R_1 and S_1 over u and v, respectively. It can defer computing b until the second round and then linearly combine R_1 and S_1 to get its real nonce $R_1 S_1^b$. As a result, the first round becomes independent of the message and the set of signers! Specifically, our HBMS2[DL] is described as follows:

- Setup: HBMS2[DL] runs over a cyclic group $\mathbb{G}$ of prime order p with generator g. The public parameters additionally contain uniformly random group elements u and v.
- Key generation: Same as in HBMS, the secret key $x \leftarrow \mathbb{Z}_p$ is chosen uniformly at random, and the public key is set to be $X := g^x$.
- Key aggregation: Same as in HBMS, the key aggregation algorithm KAgg outputs $\mathsf{apk} := \prod_{i \in [n]} X_i^{a_i}$, where $a_i := \mathsf{H}_a(\mathcal{L}, X_i)$, and $\mathcal{L}$ is the public-key (multi-)set.
- Signing:
  - In the first round, each signer (without loss of generality, indexed by 1) uniformly chooses $r_1, s_1, w_1 \leftarrow \mathbb{Z}_p$ and broadcasts $R_1 := g^{r_1} u^{w_1}$ and $S_1 := g^{s_1} v^{w_1}$.
  - In the second round, each signer receives $\{(R_i, S_i)\}_{i \in [n]}$ from other signers. The signer computes $R := \prod_{i \in [n]} (R_i S_i^b)$ for $b := \mathsf{H}_b(\mathsf{apk}, \mu)$. Then it computes $(z_1, y_1) := (r_1 + b s_1 + a_1 c x_1, w_1)$ for $c := \mathsf{H}_c(\mathsf{apk}, \mu, R)$. It broadcasts (z_1, y_1).
  - Same as in HBMS, given $\{(z_i, y_i)\}_{i \in [n]}$ from the signers, the final multi-signature is generated as $\sigma := (c, z, y)$, where $z := \sum_{i \in [n]} z_i$ and $y := \sum_{i \in [n]} y_i$.
- Verification: The verifier recovers $R := g^z (uv^b)^y \mathsf{apk}^{-c}$, where $b := \mathsf{H}_b(\mathsf{apk}, \mu)$, and checks if $c = \mathsf{H}_c(\mathsf{apk}, \mu, R)$.

Comparison to MuSig2.

Interestingly, HBMS2[DL] has a similar form to MuSig2 [NRS21]. In MuSig2, each signer also outputs multiple partial nonces in the first round. In the second round, the signers derive a coefficient from a hash function to combine the partial nonces into the aggregated nonce.

The key difference is that the combination of partial nonces in MuSig2 is only used within the group of signers but not during verification. In HBMS2[DL], however, b is a public hash of (apk, μ) used in the verification. The hash has the effect of partitioning the space of (apk, μ) in HBMS2[DL] but not in MuSig2. In the next subsection, we will see that partitioning turns out to be important for the rewinding-free technique from [TSS+24] to be applicable.

2.3 Toward Rewinding-Free Security

Starting from HBMS2[DL], we obtain our HBMS2[DDH] in a manner analogous to how HBMS is transformed into TSSHO [TSS+24]. Specifically, the public parameters g, u, v are replaced with pairs of elements $\mathbf{g} := (g, \hat{g})$, $\mathbf{u} := (u, \hat{u})$, $\mathbf{v} := (v, \hat{v})$. Throughout the scheme, exponentiation over g, u, v is replaced with element-wise exponentiation over $\mathbf{g}$, $\mathbf{u}$, $\mathbf{v}$, respectively. Consequently, now the public key consists of two elements $\mathbf{X} = (X, \hat{X}) := (g^x, \hat{g}^x)$. Similarly, the aggregated public key and the nonces also consist of two elements. The signature is still in the form of (c, z, y). To verify it, one recovers $\mathbf{R} := \mathbf{g}^z(\mathbf{u}\mathbf{v}^b)^y\mathsf{apk}^{-c}$ and checks if $c = \mathsf{H}_c(\mathsf{apk}, \mu, \mathbf{R})$, where multiplication is element-wise.

In the security proof, we still set $\mathbf{u} := \mathbf{g}^\alpha\mathbf{X}_*^\gamma$ and $\mathbf{v} := \mathbf{g}^\beta\mathbf{X}_*^{-1}$ with $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$, where $\mathbf{X}_*$ is the target public key. By DDH, we indistinguishably switch to a game where the target public key $\mathbf{X}_*$ is independent of $\mathbf{g}$, which makes a valid secret key x unlikely to exist, i.e., $\mathbf{X}_* \notin \mathsf{Span}\{\mathbf{g}\}$. For any (apk, μ) in the signable partition, the reduction can still sign using the (apk, μ)-specific Okamoto signing key $(-(\alpha + b\beta)/(\gamma - b), 1/(\gamma - b))$.

On the other hand, the extractable partition now becomes the "unsignable partition". A valid forgery (c_*, z_*, y_*) for (apk_*, μ_*) requires $\mathsf{H}_c(\mathsf{apk}_*, \mu_*, \mathbf{R}_*) = c_*$ for $\mathbf{R}_* := \mathbf{g}^{z_*}(\mathbf{u}\mathbf{v}^b)^{y_*}\mathbf{X}^{-c_*}$ and $b := \mathsf{H}_b(\mathsf{apk}_*, \mu_*)$. In this partition, we have $b = \gamma$ and therefore $\mathbf{u}\mathbf{v}^b = \mathbf{g}^{\alpha+\beta\gamma}$. Consequently, a valid forgery has to satisfy $\mathbf{R}_*\mathbf{X}_*^{c_*} \in \mathsf{Span}\{\mathbf{g}\}$, since $\mathbf{R}_*\mathbf{X}_*^{c_*} = \mathbf{g}^{z_*}(\mathbf{u}\mathbf{v}^b)^{y_*} = \mathbf{g}^{z_*+(\alpha+\beta\gamma)y_*}$. However, since $\mathbf{X}_* \notin \mathsf{Span}\{\mathbf{g}\}$ and c_* is the uniformly random value of $\mathsf{H}_c(\mathsf{apk}_*, \mu_*, \mathbf{R}_*)$, $\mathbf{R}_*\mathbf{X}_*^{c_*} \in \mathsf{Span}\{\mathbf{g}\}$ is unlikely to hold. This means that, in this game with a fake public key $\mathbf{X}_*$, making a forgery for (apk_*, μ_*) is unconditionally hard with a bounded number of queries to H_c. Thus, making a forgery is computationally hard under DDH in the original security game. We have obtained a rewinding-free security proof, with a security loss of only $\Theta(q_h)$ due to guessing the H_b query $\mathsf{H}_b(\mathsf{apk}_*, \mu_*)$ that will correspond to the forgery.

The Challenge of Applying Tight Security Techniques to Previous Schemes.

We discuss a question one may ask: why we cannot directly apply the technique in [TSS+24] to existing partially non-interactive schemes?

Tracing back, this technique for tighter security proofs follows the high-level idea by Katz and Wang [KW03], which gives a Schnorr-like tightly secure signature scheme based on DDH. Briefly speaking, this idea is to let the scheme have a "lossy" mode in addition to a normal mode, such that these two modes are computationally indistinguishable, and signing is unconditionally hard in the lossy mode. The idea has been generalized to give signature schemes from different assumptions [AFLT16] and underlies all the previous rewinding-free threshold and multi-signature schemes in pairing-free groups [FH21, PW23, PW24, TSS+24, Che25a, BW25, BW26, LÖ25].

A key point in the security proofs is that, the efficient reduction needs to be able to perform the security game in the lossy mode, where it is requested to sign, despite the fact that the lossy mode is "unsignable". This requires the reduction to have a special capability in signing beyond that of a normal user. For (non-interactive) standard signature schemes, the special capability is the honest-verifier zero-knowledge (HVZK) simulators enabled by programming the random oracle. However,

such simulators can only output a whole signature in one shot. They are not directly applicable to two-round threshold and multi-signature schemes, where the two components of signatures are output in two rounds.

Therefore, all the rewinding-free two-round threshold and multi-signature schemes [PW23, PW24, TSS+24, Che25a, BW25] crucially rely on partitioning. Some parameters are generated by hashing (apk, μ), such that the scheme is only lossy for a partition of (apk, μ) (corresponding to our unsignable partition) while still signable for other (apk, μ). The reduction only signs the signable partition.

In contrast, existing partially non-interactive two-round schemes [KG20, AB21, NRS21, TZ23] do not do partitioning. The reductions rely on interactive assumptions and use their own oracles to simulate signing. This requires that, in the game that the reduction executes, signing is only computationally hard. If we apply the Katz-Wang technique and switch to a lossy mode where signing is unconditionally hard, then even oracles cannot be helpful. We successfully overcome this barrier by introducing a novel partitioning technique that allows to delay hashing to the second round.

2.4 Toward Adaptive Security

We first briefly explain how to turn our HBMS2[DDH] into a (statically secure) threshold scheme HBTS2[DDH]. Suppose that we have a group of n signers with threshold t. Instead of letting each signer generate its own key, we now have a centralized key generation. It samples the main secret key $x \leftarrow \mathbb{Z}_p$ and distributes it using Shamir's secret sharing as $\{x_i\}_{i\in[n]}$. For every signing set $\mathcal{S} \subseteq [n]$ where $|\mathcal{S}| = t + 1$, the main secret key can be recovered as $x = \sum_{i\in\mathcal{S}} \lambda_{\mathcal{S},i} x_i$, where $\lambda_{\mathcal{S},i}$ is the corresponding Lagrange coefficient. The main public key is $\mathbf{X} := \mathbf{g}^x$. To jointly generate a signature under the main key, signers in $\mathcal{S}$ run the same protocol as in the multi-signature scheme, except that the key aggregation coefficient a_i is replaced with $\lambda_{\mathcal{S},i}$.

In the security proof, again, $\mathbf{X}$ is uniformly chosen such that $\mathbf{X} \notin \mathsf{Span}\{\mathbf{g}\}$ with high probability. Without a valid secret key, the key generation can be simulated by performing Shamir's secret sharing on exponents, which outputs public key shares $\{\mathbf{X}_i\}_{i\in[n]}$ such that $\mathbf{X} = \prod_{i\in\mathcal{S}} \mathbf{X}_i^{\lambda_{\mathcal{S},i}}$ for every $\mathcal{S} \subseteq [n]$, $|\mathcal{S}| = t+1$. As a result, there exist at most t public key shares $\mathbf{X}_i$ such that the corresponding secret key shares x_i exist.

This causes the main issue for adaptive security. An adaptive adversary can corrupt any t signers during the security game. Since there exist at most t secret key shares, the reduction has to guess who the adversary will corrupt, which introduces an exponential security loss of $\binom{n}{t}$.

We now introduce our HBTS2[DDH]-Mask, which applies the masking technique from [KRT24, Che25b] to HBTS2[DDH]. In HBTS2[DDH], the public key share $\mathbf{X}_i$ becomes public once the adversary requests a signature share from signer i. The public key share $\mathbf{X}_i$ binds the corresponding secret key share x_i, so the reduction has to correctly output x_i to simulate the game. The main idea of masking is to mask the signature shares by one-time shares of zero. As a result, the public key shares are never revealed, and the secret key shares become perfectly hidden until being output for corruptions. The reduction does not need to simulate the key sharing and can simply sample one uniformly when asked to output a secret key share.

[Che25b] observed that for HBTS to achieve non-AGM adaptive security, only the second round needs to be masked. This turns out to be also true for our HBTS2. To make sure the masks are indeed one-time, they are generated by hashing the signers' view in the signing protocol, which is of high entropy.

In comparison, if we only mask the second round of FROST, we only achieve adaptive security in the AGM [BCL+26, BGC+25]. If we mask both rounds of FROST, we indeed obtain non-AGM adaptive security [Che25b]; however, for the first-round masks to be one-time, we either require the signers to be stateful or to spend one more round to jointly generate some entropy first.

2.5 Open Directions

HBMS2[DDH] and HBTS2[DDH]-Mask both have a security loss of $\Theta(q_h)$. As we have mentioned in Section 2.2, computing h as uv^b follows the approach in Boneh–Boyen identity-based encryption and signature schemes. It seems that the technique from Waters' signature scheme [Wat05, HJK12] may give a scheme with security loss $\Theta(q_s)$. In particular, the public parameters contain random group elements $u_0, \ldots, u_l$, and the scheme instead computes h as $u_0 \prod_{i\in[l]} u_i^{b_i}$, where $(b_1, \ldots, b_l)$ is some l-bit encoding of the hashed message $\mathsf{H}(\mu)$. This comes at the cost of also increasing the number of first-round nonces to $l + 1 = O(\lambda)$. Is it possible to construct a partially non-interactive two-round scheme with fully or almost tight security under standard assumptions?

Is it possible to construct a scheme with the same or better properties as ours under search assumptions such as CDH, or even under interactive (still search) assumptions such as AOMCDH?

3 Preliminaries

Notations.

For a positive integer n, $[n]$ denotes $\{1, 2, \ldots, n\}$. For a randomized algorithm A, $y \in \mathsf{A}(x)$ denotes that y is a possible output of A on input x. For vectors $\mathbf{g}$, $\mathbf{h}$ of group elements, exponentiation $\mathbf{g}^x$ and multiplication $\mathbf{gh}$ are both element-wise.

Shamir's Secret Sharing.

We use Shamir's secret sharing scheme [Sha79] to share a secret over $\mathbb{Z}_p$. On input the number of users $n < p$, the threshold $t \le n$, and the secret $x \in \mathbb{Z}_p$, the sharing algorithm $\mathsf{Share}(n, t, x)$ sets $\alpha_0 := x$ and samples the other t coefficients $\alpha_1, \ldots, \alpha_t \leftarrow \mathbb{Z}_p$ of polynomial $\alpha(\cdot)$. It returns the secret shares $\{x_i\}_{i\in[n]}$ where $x_i = \alpha(i) = \sum_{j=0}^{t} \alpha_j i^j$. For any $(t+1)$-subset $\mathcal{S}$ of $[n]$, we have $x = \sum_{i\in\mathcal{S}} \lambda_{\mathcal{S},i} x_i$ by polynomial interpolation, where the Lagrange coefficient for $\mathcal{S}$ is defined as $\lambda_{\mathcal{S},i} = \prod_{j\in\mathcal{S}, j\neq i} j/(j-i)$. The security of Shamir's secret sharing guarantees that the collection of any t shares of x is independent of x.

DDH Assumption.

Let GrGen be a group generation algorithm that takes the security parameter 1^λ as input and outputs group parameters $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group of prime order p and g is a generator.

Definition 1 (Decisional Diffie-Hellman (DDH))**.** We say that DDH is (τ, ε)-hard with respect to GrGen if for every positive integer λ, every τ-time algorithm A,

$$\begin{aligned}&|\Pr[\mathsf{A}(\mathbb{G}, p, g, \hat{g}, g^a, \hat{g}^a) = 1 : (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda), \hat{g} \leftarrow \mathbb{G}, a \leftarrow \mathbb{Z}_p]\\ &- \Pr[\mathsf{A}(\mathbb{G}, p, g, \hat{g}, g^a, \hat{g}^b) = 1 : (\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda), \hat{g} \leftarrow \mathbb{G}, a, b \leftarrow \mathbb{Z}_p] = 1| \le \varepsilon.\end{aligned}$$

Let $(g, \hat{g}, h, \hat{h})$ be a tuple of four group elements over $\mathbb{G}$ where g is a generator. We call it a DH-*tuple* if $(h, \hat{h}) \in \mathsf{Span}\{(g, \hat{g})\}$.

3.1 Multi-Signature Schemes

Syntax. We define partially non-interactive two-round multi-signature schemes with key aggregation. We follow the plain public-key model [BN06]. Each user's key pair is individually generated by the key generation algorithm KGen. A deterministic, non-interactive key aggregation KAgg can aggregate a set of individual public keys into a single aggregated key, such that the multi-signature verification algorithm Vf can take only the aggregated public key rather than the whole public key list as input.

$\underline{\mathsf{MSJointSign}(\{\mathsf{pk}_i\}_{i\in[n]}, \mu, \{\mathsf{sk}_i\}_{i\in[n]})}$

1: **for** $k \in [n]$ **do**
2: $\quad (\mathsf{st}_k, \mathsf{msg}_{1,k}) \leftarrow \mathsf{Sign}_1(\mathsf{sk}_k)$
3: **for** $k \in [n]$ **do**
4: $\quad \mathsf{msg}_{2,k} \leftarrow \mathsf{Sign}_2(\{\mathsf{pk}_i\}_{i\in[n]}, \mu, \mathsf{sk}_k, \mathsf{st}_k, \{\mathsf{msg}_{1,i}\}_{i\in[n]})$
5: **Return** $\sigma \leftarrow \mathsf{Combine}(\mathcal{L}, \mu, \{\mathsf{msg}_{1,i}\}_{i\in[n]}, \{\mathsf{msg}_{2,i}\}_{i\in[n]})$

Figure 1: Procedure MSJointSign used to define the completeness of multi-signature schemes.

The two-round signing protocol consists of three algorithms: each signer runs Sign_1 and Sign_2 to generate its first and second-round protocol messages to broadcast, respectively. The last algorithm Combine generates the final multi-signature from the whole protocol transcript, which is deterministic and can be executed by any designated user. The partially non-interactive property is reflected in that Sign_1 does not take the public key set and the message as inputs. We use procedure MSJointSign in Fig. 1 to describe an honest execution of the signing protocol and also to define the completeness.

We remark that the protocol is independent of the indexing of signers, public keys, and protocol messages. That is, the signers do not need to agree on an order among them, as any order will yield the same output. Without loss of generality, we assume that each user indexes itself by 1.

Definition 2 (Partially non-interactive two-round multi-signatures with key aggregation)**.** A *partially non-interactive two-round multi-signature scheme* MS *with key aggregation* consists of algorithms with syntax defined as follows:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: The setup algorithm takes the security parameter 1^λ as input and outputs the public parameter pp. All algorithms related to MS implicitly take pp as input.
- $\mathsf{KGen}() \to (\mathsf{pk}, \mathsf{sk})$: The key generation algorithm outputs a secret key sk and a public key pk.
- $\mathsf{KAgg}(\mathcal{L}) \to \mathsf{apk}$: The deterministic key aggregation algorithm takes a set of public keys $\mathcal{L} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}$ as input and outputs an aggregated public key apk.
- The signing protocol consists of three algorithms:
  - $\mathsf{Sign}_1(\mathsf{sk}_1) \to (\mathsf{st}_1, \mathsf{msg}_{1,1})$: The first-round signing algorithm takes a secret key sk_1 as input and outputs a state st_1 and a first-round protocol message $\mathsf{msg}_{1,1}$.
  - $\mathsf{Sign}_2(\mathcal{L}, \mu, \mathsf{sk}_1, \mathsf{st}_1, \mathcal{M}_1) \to \mathsf{msg}_{2,1}$: The second-round signing algorithm takes a set of public keys $\mathcal{L}$, a message μ, and a secret key sk_1, a state st_1, and a set of first-round protocol messages $\mathcal{M}_1$ as inputs and outputs a second-round protocol message $\mathsf{msg}_{2,1}$.
  - $\mathsf{Combine}(\mathcal{L}, \mu, \mathcal{M}_1, \mathcal{M}_2) \to \sigma$: The combining algorithm takes a set of public key $\mathcal{L}$, a message μ, a set of first-round protocol messages $\mathcal{M}_1$ and a set of second-round protocol messages $\mathcal{M}_2$ as inputs and outputs a multi-signature σ.
- $\mathsf{Vf}(\mathsf{apk}, \mu, \sigma) \to 0/1$: The deterministic verification algorithm takes an aggregated public key apk, a message μ, and a multi-signature σ as inputs and outputs 0 or 1.

For completeness, we require that for every λ, every $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, every n, every $(\mathsf{pk}_1, \mathsf{sk}_1), \ldots, (\mathsf{pk}_n, \mathsf{sk}_n) \in \mathsf{KGen}()$, and every message μ, we have

$$\Pr[\mathsf{Vf}(\mathsf{KAgg}(\{\mathsf{pk}_i\}_{i\in[n]}), \mu, \sigma) = 1 : \\ \sigma \leftarrow \mathsf{MSJointSign}(\{\mathsf{pk}_i\}_{i\in[n]}, \mu, \{\mathsf{sk}_i\}_{i\in[n]})] = 1,$$

where the procedure MSJointSign, modeling an honest execution of MS, is defined in Fig. 1.

Security. The security of two-round multi-signature schemes with key aggregation is defined by the security game MS-UF described in Fig. 2. The adversary A is given the public parameter pp and a target public key pk_*. Its goal is to forge a multi-signature on a message μ_* under a public-key set $\mathcal{L}_*$ that contains pk_*, both of its own choice. A can concurrently open multiple signing sessions

$\underline{\mathsf{MS\text{-}UF}^{\mathsf{A}}_{\mathsf{MS}}(\lambda)}$
$\mathcal{Q} := \emptyset$
$\mathsf{ctr} := 0$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}_*, \mathsf{sk}_*) \leftarrow \mathsf{KGen}()$
$(\mathcal{L}_*, \mu_*, \sigma_*) \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(\mathsf{pk}_*)$
Return $[\![(\mathcal{L}_*, \mu_*) \notin \mathcal{Q} \wedge \mathsf{pk}_* \in \mathcal{L}_* \wedge \mathsf{Vf}(\mathsf{KAgg}(\mathcal{L}_*), \mu_*, \sigma_*) = 1]\!]$

$\underline{\textsc{Sign}_1()}$
$\mathsf{ctr} := \mathsf{ctr} + 1$
$\mathsf{sid} := \mathsf{ctr}$
$(\mathsf{st}_{\mathsf{sid}}, \mathsf{msg}_{1,\mathsf{sid}}) \leftarrow \mathsf{Sign}_1(\mathsf{sk}_*)$
$\mathsf{round}_{\mathsf{sid}} := 1$
Return $\mathsf{msg}_{1,\mathsf{sid}}$

$\underline{\textsc{Sign}_2(\mathsf{sid}, \mathcal{L}, \mu, \mathcal{M}_1)}$
Assert $\mathcal{L}[1] = \mathsf{pk}_* \wedge \mathsf{round}_{\mathsf{sid}} = 1 \wedge \mathcal{M}_1[1] = \mathsf{msg}_{1,\mathsf{sid}}$
$\mathcal{Q} := \mathcal{Q} \cup (\mathcal{L}, \mu)$
$\mathsf{msg}_2 \leftarrow \mathsf{Sign}_2(\mathcal{L}, \mu, \mathsf{sk}_*, \mathsf{st}_{\mathsf{sid}}, \mathcal{M}_1)$
$\mathsf{round}_{\mathsf{sid}} := 2$
Return msg_2

Figure 2: The MS-UF security game for multi-signature scheme MS against A.

$\underline{\mathsf{TSJointSign}(\mathsf{pk}, \mathcal{S}, \mu, \{\mathsf{sk}_i\}_{i \in \mathcal{S}})}$
for $k \in \mathcal{S}$ **do**
  $(\mathsf{st}_k, \mathsf{msg}_{1,k}) \leftarrow \mathsf{Sign}_1(\mathsf{pk}, k, \mathsf{sk}_k)$
for $k \in \mathcal{S}$ **do**
  $\mathsf{msg}_{2,k} \leftarrow \mathsf{Sign}_2(\mathsf{pk}, \mathcal{S}, \mu, k, \mathsf{sk}_k, \mathsf{st}_k, \{\mathsf{msg}_{1,i}\}_{i \in \mathcal{S}})$
Return $\sigma \leftarrow \mathsf{Combine}(\mathsf{pk}, \mathcal{S}, \mu, \{\mathsf{msg}_{1,i}\}_{i \in \mathcal{S}}, \{\mathsf{msg}_{2,i}\}_{i \in \mathcal{S}})$

Figure 3: Procedure TSJointSign used to define the completeness of threshold signature schemes.

with an honest signer possessing pk_*. In each session, A chooses the message μ to sign, and it plays the part of all other signers, with public-key set $\mathcal{L}$ of its choice. This is formalized by two signing oracles $\textsc{Sign}_1$, $\textsc{Sign}_2$, where we use a session identifier sid to distinguish the concurrent sessions. It is required that no signing session on $(\mathcal{L}_*, \mu_*)$ has been completed for the forgery to be non-trivial.

Definition 3 (Security of Two-Round Multi-Signature Schemes with Key Aggregation)**.** A two-round multi-signature scheme MS with key aggregation is (q_s, τ, ε)-secure if for every λ, every adversary A that makes at most $q_s = q_s(\lambda)$ queries to $\textsc{Sign}_1$ and runs in time at most $\tau = \tau(\lambda)$, $\Pr[\mathsf{MS\text{-}UF}^{\mathsf{A}}_{\mathsf{MS}}(\lambda) = 1] \leq \varepsilon = \varepsilon(\lambda)$.

3.2 Threshold Signature Schemes

Syntax. We define partially non-interactive two-round threshold signature schemes. A centralized key generation algorithm KGen generates a public key and n secret key shares. The two-round signing protocol consists of three algorithms Sign_1, Sign_2, Combine as in multi-signature schemes. An honest execution of the signing protocol is described as the procedure TSJointSign in Fig. 3, which we use

to define completeness.

Definition 4 (Partially non-interactive two-Round Threshold Signature Schemes)**.** A *partially non-interactive two-round threshold signature scheme* TS consists of the following algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input the security parameter 1^λ, the setup algorithm Setup outputs global parameters pp. All algorithms related to TS implicitly take pp as input.
- $\mathsf{KGen}(n, t) \to (\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]})$: On inputs a number of users n and a threshold t, the key generation algorithm KGen outputs a public key pk and n secret key shares $\{\mathsf{sk}_i\}_{i\in[n]}$.
- The signing protocol consists of three algorithms:
  - $\mathsf{Sign}_1(\mathsf{pk}, k, \mathsf{sk}_k) \to (\mathsf{st}_k, \mathsf{msg}_{1,k})$: On inputs a public key pk, a signer index k, a secret key share sk_k, the first-round signing algorithm Sign_1 outputs a state st_k and a first-round protocol message $\mathsf{msg}_{1,k}$.
  - $\mathsf{Sign}_2(\mathsf{pk}, \mathcal{S}, \mu, k, \mathsf{sk}_k, \mathsf{st}_k, \mathcal{M}_1) \to \mathsf{msg}_{2,k}$: On inputs a public key pk, a set of signer indices $\mathcal{S}$, a message μ, a signer index k, a secret key share sk_k, a state st_k, and a set of first-round protocol messages $\mathcal{M}_1$, the second-round signing algorithm Sign_2 outputs a second-round protocol message $\mathsf{msg}_{2,k}$.
  - $\mathsf{Combine}(\mathsf{pk}, \mathcal{S}, \mu, \mathcal{M}_1, \mathcal{M}_2) \to \sigma$: On inputs a public key pk, a set of signer indices $\mathcal{S}$, a message μ, a set of first-round protocol messages $\mathcal{M}_1$, and a set of second-round protocol messages $\mathcal{M}_2$, 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 deterministic verification algorithm Vf outputs 0 or 1.

For completeness, we require that for every λ, every $\mathsf{pp} \in \mathsf{Setup}(1^\lambda)$, every n and t that TS supports, every $(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \in \mathsf{KGen}(n, t)$, every subset of signers $\mathcal{S} \subseteq [n]$ with $|\mathcal{S}| = t + 1$, and every message μ, we have

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

where the procedure $\mathsf{TSJointSign}$, modeling an honest execution of TS, is defined in Fig. 3.

Security. The fully adaptive security of two-round threshold signature schemes is defined by the security game $\mathsf{adp\text{-}TS\text{-}UF}$ described in Fig. 4. Our definition is the adaptive version of the $\mathsf{TS\text{-}UF\text{-}0}$ security in [BCK+22]. The adversary A is given the public parameter pp, the public key pk, and the access to the signing and corruption oracles. Its goal is to forge a signature on a message that it has never queried to the signing oracle. It can concurrently open multiple signing sessions with each users, and we use a session identifier sid to distinguish the concurrent sessions with each user. A can corrupt at most t users in total, each corruption at any time and adaptively based on its previous view.

Our model does not assume secure erasure. When a signer is corrupted, the adversary not only learns its secret key share but also the *whole computation history*. Our corruption oracle returns all randomness that the signer used in previous signing sessions, which allows the adversary to recover the whole history.

Our model considers fully corrupted communication channels. As the inputs to the second-round signing oracles, the adversary decides every protocol message even on behalf of honest parties. In fact, honest signers will not realize the signing sessions of others. There is no consistency requirement on the protocol messages chosen by the adversary.

Definition 5 (Adaptive Security of Threshold Signature Schemes.)**.** A two-round threshold signature scheme TS is (q_s, τ, ε)-adaptively secure if for every λ, every adversary A that makes at most $q_s = q_s(\lambda)$ signing queries and runs in time at most $\tau = \tau(\lambda)$, for every n and t that TS supports, $\Pr[\mathsf{adp\text{-}TS\text{-}UF}^{\mathsf{A}}_{\mathsf{TS}}(\lambda, n, t) = 1] \le \varepsilon = \varepsilon(\lambda)$, where the security game $\mathsf{adp\text{-}TS\text{-}UF}$ is defined in Fig. 4.

$\underline{\mathsf{adp\text{-}TS\text{-}UF}^{\mathsf{A}}_{\mathsf{TS}}(\lambda, n, t)}$

$\mathcal{Q} := \emptyset$, $\mathcal{C} := \emptyset$
for $i \in [n]$ **do**
  $\mathsf{ctr}_i := 0$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$(\mathsf{pk}, \{\mathsf{sk}_i\}_{i\in[n]}) \leftarrow \mathsf{KGen}(n, t)$
$(\mu_*, \sigma_*) \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2, \textsc{Corr}}(\mathsf{pp}, \mathsf{pk})$
Return $[\![\mu_* \notin \mathcal{Q} \wedge \mathsf{Vf}(\mathsf{pk}, \mu_*, \sigma_*) = 1]\!]$

$\underline{\textsc{Corr}(k)}$
Assert $|\mathcal{C}| < t$
$\mathcal{C} := \mathcal{C} \cup \{k\}$
Return $(\mathsf{sk}_k, \{\rho_{1,k,i}\}_{i\in[\mathsf{ctr}_k]}, \{\rho_{2,k,i}\}_{i\in[\mathsf{ctr}_k]})$

$\underline{\textsc{Sign}_1(k)}$
Assert $k \notin \mathcal{C}$
$\mathsf{ctr}_k := \mathsf{ctr}_k + 1$
$\mathsf{sid} := \mathsf{ctr}_k$
$(\mathsf{st}_{k,\mathsf{sid}}, \mathsf{msg}_{1,k,\mathsf{sid}}) \leftarrow \mathsf{Sign}_1(\mathsf{pk}, k, \mathsf{sk}_k; \rho_{1,k,\mathsf{sid}})$
$\mathsf{round}_{k,\mathsf{sid}} := 1$
Return $\mathsf{msg}_{1,k,\mathsf{sid}}$

$\underline{\textsc{Sign}_2(k, \mathsf{sid}, \mathcal{S}, \mu, \mathcal{M}_1)}$
Assert $k \notin \mathcal{C} \wedge |\mathcal{S}| = t + 1 \wedge k \in \mathcal{S} \wedge \mathsf{round}_{k,\mathsf{sid}} = 1$
$\{\mathsf{msg}_{1,i}\}_{i\in\mathcal{S}} := \mathcal{M}_1$
Assert $\mathsf{msg}_{1,k} = \mathsf{msg}_{1,k,\mathsf{sid}}$
$\mathcal{Q} := \mathcal{Q} \cup \{\mu\}$
$\mathsf{msg}_{2,k,\mathsf{sid}} \leftarrow \mathsf{Sign}_2(\mathsf{pk}, \mathcal{S}, \mu, k, \mathsf{sk}_k, \mathsf{st}_{k,\mathsf{sid}}, \mathcal{M}_1; \rho_{2,k,\mathsf{sid}})$
$\mathsf{round}_{k,\mathsf{sid}} := 2$
Return $\mathsf{msg}_{2,k,\mathsf{sid}}$

Figure 4: The adp-TS-UF security game.

4 HBMS2[DDH]: Multi-Signature Scheme with Tighter Security

4.1 Scheme

We describe HBMS2[DDH], defined with respect to group generation algorithm GrGen, in Fig. 5.

$\underline{\mathsf{Setup}(1^\lambda)}$
1: $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
2: $\hat{g}, u, \hat{u}, v, \hat{v} \leftarrow \mathbb{G}$
3: $\mathbf{g} := (g, \hat{g}), \mathbf{u} := (u, \hat{u}), \mathbf{v} := (v, \hat{v})$
4: Sample $\mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c : \{0,1\}^* \to \mathbb{Z}_p$
5: **Return** $(\mathbb{G}, p, \mathbf{g}, \mathbf{u}, \mathbf{v}, \mathsf{H}_a, \mathsf{H}_b, \mathsf{H}_c)$

$\underline{\mathsf{KGen}()}$
6: $x \leftarrow \mathbb{Z}_p$
7: $\mathbf{X} := \mathbf{g}^x$
8: **Return** $(x, \mathbf{X})$

$\underline{\mathsf{KAgg}(\mathcal{L})}$
9: $\{\mathbf{X}_i\}_{i\in[n]} := \mathcal{L}$
10: **for** $i \in [n]$ **do**
11: $\quad a_i := \mathsf{H}_a(\mathcal{L}, \mathbf{X}_i)$
12: $\tilde{\mathbf{X}} := \prod_{i\in[n]} \mathbf{X}_i^{a_i}$
13: **Return** $\tilde{\mathbf{X}}$

$\underline{\mathsf{Sign}_1(\mathsf{sk}_1)}$
14: $r_1, s_1, w_1 \leftarrow \mathbb{Z}_p$
15: $\mathbf{R}_1 := \mathbf{g}^{r_1}\mathbf{u}^{w_1}$
16: $\mathbf{S}_1 := \mathbf{g}^{s_1}\mathbf{v}^{w_1}$
17: **Return** $((r_1, s_1, w_1), (\mathbf{R}_1, \mathbf{S}_1))$

$\underline{\mathsf{Sign}_2(\mathcal{L}, \mu, \mathsf{sk}_1, \mathsf{st}_1, \mathcal{M}_1)}$
18: $\mathsf{apk} := \mathsf{KAgg}(\mathcal{L})$
19: $\{\mathbf{X}_i\}_{i\in[n]} := \mathcal{L}$
20: $a_1 := \mathsf{H}_a(\mathcal{L}, \mathbf{X}_1)$
21: $x_1 := \mathsf{sk}_1$
22: $(r_1, s_1, w_1) := \mathsf{st}_1$
23: $\{(\mathbf{R}_i, \mathbf{S}_i)\}_{i\in[n]} := \mathcal{M}_1$
24: $b := \mathsf{H}_b(\mathsf{apk}, \mu)$
25: $\mathbf{R} := \prod_{i\in[n]}(\mathbf{R}_i\mathbf{S}_i^b)$
26: $c := \mathsf{H}_c(\mathsf{apk}, \mu, \mathbf{R})$
27: $z_1 := r_1 + bs_1 + a_1cx_1$
28: $y_1 := w_1$
29: **Return** (z_1, y_1)

$\underline{\mathsf{Combine}(\mathcal{L}, \mu, \mathcal{M}_1, \mathcal{M}_2)}$
30: $\mathsf{apk} := \mathsf{KAgg}(\mathcal{L})$
31: $\{(\mathbf{R}_i, \mathbf{S}_i)\}_{i\in[n]} := \mathcal{M}_1$
32: $\{(z_i, y_i)\}_{i\in[n]} := \mathcal{M}_2$
33: $b := \mathsf{H}_b(\mathsf{apk}, \mu)$
34: $\mathbf{R} := \prod_{i\in[n]}(\mathbf{R}_i\mathbf{S}_i^b)$
35: $c := \mathsf{H}_c(\mathsf{apk}, \mu, \mathbf{R})$
36: $(z, y) := \sum_{i\in[n]}(z_i, y_i)$
37: **Return** (c, z, y)

$\underline{\mathsf{Vf}(\mathsf{apk}, \mu, \sigma)}$
38: $\tilde{\mathbf{X}} := \mathsf{apk}$
39: $(c, z, y) := \sigma$
40: $b := \mathsf{H}_b(\mathsf{apk}, \mu)$
41: $\mathbf{R} := \mathbf{g}^z(\mathbf{u}\mathbf{v}^b)^y\tilde{\mathbf{X}}^{-c}$
42: **Return** $[\![c = \mathsf{H}_c(\mathsf{apk}, \mu, \mathbf{R})]\!]$

Figure 5: HBMS2[DDH], defined with respect to group generation algorithm GrGen.

Completeness. The following equation shows that, for an honestly generated multi-signature, the verification correctly recovers the aggregated commitment. Then the completeness follows directly.

$$\begin{aligned}
\mathbf{g}^z(\mathbf{u}\mathbf{v}^b)^y\tilde{\mathbf{X}}^{-c} &= \mathbf{g}^{\sum_{i\in[n]}(r_i+bs_i+a_icx_i)}(\mathbf{u}\mathbf{v}^b)^{\sum_{i\in[n]}w_i}(\prod_{i\in[n]}\mathbf{X}_i^{a_i})^{-c} \\
&= \prod_{i\in[n]}(\mathbf{g}^{r_i+bs_i+a_icx_i}\mathbf{u}^{w_i}\mathbf{v}^{bw_i}\mathbf{g}^{-a_icx_i}) \\
&= \prod_{i\in[n]}(\mathbf{g}^{r_i}\mathbf{u}^{w_i}\mathbf{g}^{bs_i}\mathbf{v}^{bw_i}) \\
&= \prod_{i\in[n]}(\mathbf{R}_i\mathbf{S}_i^b) = \mathbf{R}.
\end{aligned}$$

4.2 Security

Theorem 1. *If* DDH *is* $(\tau_{\mathrm{ddh}}, \varepsilon_{\mathrm{ddh}})$*-hard for* GrGen*, then scheme* HBMS2[DDH] *is* $(q_s, \tau_{\mathrm{uf}}, \varepsilon_{\mathrm{uf}})$-MS-UF *in the random oracle model against any adversary that makes at most* q_h *hash queries,*

where

$$\varepsilon_{\mathrm{ddh}} = \frac{\varepsilon_{\mathrm{uf}}}{q'+1} - \frac{4q+2}{p},$$

$q = n_{\max}(q_s+1)+q_h$, $q' = q_s+q_h+1$, $n_{\max} < p$ *is the maximal size of public-key sets that* A *returns or queries, and* $\tau_{\mathrm{uf}} \approx \tau_{\mathrm{ddh}}$.

Proof. Let A be any τ_{uf}-time adversary that makes at most q_s queries to $\textsc{Sign}_1$ and q_h hash queries to each hash oracle. We need to prove that A's winning probability is at most $\varepsilon_{\mathrm{uf}}$. To do so, we will build a series of hybrid games. Let G_0 be the original $\mathsf{MS\text{-}UF}^{\mathsf{A}}_{\mathsf{HBMS2[DDH]}}$ game. Eventually, we will reach a game where we can unconditionally bound A's winning probability. Let ε_i denote the probability that A wins G_i. Then we have $\varepsilon_0 = \varepsilon_{\mathrm{uf}}$.

In G_0, let $\mathbf{X}_*$ denote the target public key input to A, and x_* denote the corresponding secret key. Hash functions H_a, H_b, H_c are modeled as random oracles. We assume that A never makes repeated queries to a random oracle. This is without loss of generality, as we can easily build a wrapper to ensure so.

It is not hard to see that the total number of queries to H_a, including external queries from A and internal queries from other oracles and the verification of the final forgery, is bounded by q. The total numbers of queries to H_b and H_c, respectively, are bounded by q'. One can verify that these bounds hold for every hybrid game in the following.

G_1: Rule out collisions. In this game, we exclude collisions in H_b and KAgg. Specifically:

- Before H_b returns, the game aborts if the answer to be returned is repeated with a previous distinct H_b query.
- For each public key $[\mathcal{L}]$ with $\mathbf{X}_* \in \mathcal{L}$, when $\mathsf{KAgg}(\mathcal{L})$ is determined, i.e. $\mathsf{H}_a(\mathcal{L}, \mathbf{X}_i)$ has been defined for every $i \in |\mathcal{L}|$, this game records $\mathsf{KAgg}(\mathcal{L})$. The game aborts when there exist $\mathcal{L} \neq \mathcal{L}'$ that both contain $\mathbf{X}_*$ such that $\mathsf{KAgg}(\mathcal{L}) = \mathsf{KAgg}(\mathcal{L}')$.

The collision probability in H_b is clearly upper-bounded by $(q')^2/p < q^2/p$. Below we bound the collision probability for KAgg.

We analyze the abort probability conditioned on $\mathbf{X}_* \neq \mathbb{1}^2$, which is the case except with probability $1/p$. Suppose $\mathbf{X}_*$ appears n_* times in $\mathcal{L}$, and without loss of generality assume $\mathcal{L}[1] = \mathbf{X}_*$. Then $\mathsf{KAgg}(\mathcal{L}) = \mathbf{X}_*^{n_* a_1} \prod_{\mathbf{X}_i \neq \mathbf{X}_* \in \mathcal{L}} \mathbf{X}_i^{a_i}$. Let $\mathbf{Y} := \prod_{\mathbf{X}_i \neq \mathbf{X}_* \in \mathcal{L}} \mathbf{X}_i^{a_i}$. Since $n_* < p$, we have $\mathbf{X}_*^{n_*} \neq \mathbb{1}$. Then, as a_1 is uniform and independent of $\mathbf{Y}$, we have $\mathsf{KAgg}(\mathcal{L})$ uniformly distributed over the coset $\mathbf{Y} \cdot \mathsf{Span}\{\mathbf{X}_*\}$ conditioned on any $\mathbf{Y}$. This means that $\mathsf{KAgg}(\mathcal{L})$ is uniformly distributed over a set of size at least p. Consequently, $\mathsf{KAgg}(\mathcal{L}) = \mathsf{KAgg}(\mathcal{L}')$ with probability at most $1/p$ for every pair of distinct $\mathcal{L}$, $\mathcal{L}'$. In total, the collision probability of KAgg is at most $(q^2+1)/p$.

Overall, we have $\varepsilon_1 \geq \varepsilon_0 - (2q^2+1)/p$.

G_2: Generate $\mathbf{u}$ and $\mathbf{v}$ as combinations of $\mathbf{g}$ and $\mathbf{X}_*$. This game changes the way to generate $\mathbf{u}$ and $\mathbf{v}$ in Setup as follows. It first samples $\mathbf{g}$ and $\mathbf{X}_*$ as in G_1. Then it samples α, β, $\gamma \leftarrow \mathbb{Z}_p$ and computes $\mathbf{u} := \mathbf{g}^{\alpha}\mathbf{X}_*^{\gamma}$ and $\mathbf{v} := \mathbf{g}^{\beta}\mathbf{X}_*^{-1}$.

In G_1, $\mathbf{u}$ and $\mathbf{v}$ are uniformly distributed over $\mathbb{G}^2$, while in G_2, $\mathbf{u} = \mathbf{g}^{\alpha+\gamma x_*}$ and $\mathbf{v} = \mathbf{g}^{\beta - x_*}$ are uniformly over $\mathsf{Span}\{\mathbf{g}\}$. We can reduce DDH to distinguishing G_1 and G_2 as follows.

On inputs $\mathbb{G}$, p, g, A, B, C, the reduction runs $\mathsf{G}_1/\mathsf{G}_2$ with public parameters $\mathsf{pp} = (\mathbb{G}, p, \mathbf{g}, \mathbf{u}, \mathbf{v})$, where $\mathbf{g} := (g, A)$, $\mathbf{u}$ and $\mathbf{v}$ being uniformly random linear combinations of (g, A) and (B, C). If (g, A, B, C) is a uniform DH tuple, then $\mathbf{u}$ and $\mathbf{v}$ are both uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}$, so the reduction simulates G_2. If (A, B, C) are uniformly distributed over $\mathbb{G}^3$, then (g, A, B, C) is not a DH tuple except with probability $1/p$, in which case the reduction simulates G_1. Therefore, we have $\varepsilon_2 \geq \varepsilon_1 - \varepsilon_{\mathrm{ddh}} - 1/p$.

G_3: **Guess the index of $\mathsf{H}_b(\mathsf{apk}_*, \mu_*)$.** In this game, we change the winning condition of A. In the setup phase, the game uniformly chooses a secret index $i_* \leftarrow [q']$. We perform one more check on A's output $(\mathcal{L}_*, \mu_*, \sigma_*)$ in addition to the previous winning conditions. For A to win, we require that $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}_*), \mu_*)$ is the i_*-th distinct query made to H_b. Since i_* is independent of A's view, we have $\varepsilon_3 = \varepsilon_2/q'$.

G_4: **Abort in advance if impossible to win.** This game aborts whenever it turns out that A is impossible to win. In particular, this is when A queries $(\mathcal{L}, \mu)$ to $\textsc{Sign}_2$ such that $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu)$ is the i_*-th distinct H_b query. Recall that the winning condition requires $(\mathcal{L}_*, \mu_*) \neq (\mathcal{L}, \mu)$, and we have ruled out KAgg collisions in G_1. Thus, we have $(\mathsf{KAgg}(\mathcal{L}_*), \mu_*) \neq (\mathsf{KAgg}(\mathcal{L}), \mu)$. Consequently, if $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu)$ is the i_*-the distinct query, then $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}_*), \mu_*)$ cannot be the i_*-th query, and A is impossible to win.

This can be checked by tracking the indices of H_b queries. Recall that when A queries $(\mathcal{L}, \mu)$ to $\textsc{Sign}_2$, the game makes an internal query $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu)$ when executing Sign_2. At this time, we let the game abort if this $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu)$ is the i_*-th distinct H_b query.

Aborting when A is impossible to win clearly does not affect the winning probability. We have $\varepsilon_4 = \varepsilon_3$.

G_5: **Embed γ into the i_*-th H_b query.** This game uses γ to answer the i_*-th distinct H_b query. As α is uniform and independent of γ, and $\mathbf{X}_* \in \mathsf{Span}\{\mathbf{g}\}$, we know that $\mathbf{u} = \mathbf{g}^{\alpha}\mathbf{X}_*^{\gamma}$ is also independent of γ. Hence, the answer to the i_*-th H_b query is correctly distributed: uniform and independent of the previous view of A. We have $\varepsilon_5 = \varepsilon_4$.

G_6: **Use α, β, γ to sign.** We change $\textsc{Sign}_2$ in this game. Recall that in previous games, $\textsc{Sign}_2$ calls Sign_2 which sets:

$$z_1 := r_1 + bs_1 + a_1 c x_*, \quad y_1 := w_1.$$

In this game, $\textsc{Sign}_2$ sets

$$z_1 := r_1 + bs_1 + a_1 c \cdot \frac{-\alpha - b\beta}{\gamma - b}, \quad y_1 := w_1 + a_1 c \cdot \frac{1}{\gamma - b}.$$

We remark that, since we have excluded collisions in H_b in G_1, only the i_*-th H_b query is answered with γ. Moreover, in G_4, we have made the game abort when A queries $(\mathcal{L}, \mu)$ to $\textsc{Sign}_2$ if $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu)$ is the i_*-th H_b query, so the game never needs to answer $\textsc{Sign}_2$ with $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}), \mu) = \gamma$. Hence, the denominator $\gamma - b$ is always non-zero.

In the following, we argue that G_6 and G_5 are identical in the view of A. We will do so by bridging these two games by a computationally unbounded simulator, which perfectly simulates both of them. Since

$$\begin{aligned}\mathbf{g}^{(-\alpha - b\beta)/(\gamma - b)}(\mathbf{u}\mathbf{v}^b)^{1/(\gamma-b)} &= \mathbf{g}^{(-\alpha-b\beta)/(\gamma-b)}(\mathbf{g}^{\alpha}\mathbf{X}_*^{\gamma})^{1/(\gamma-b)}(\mathbf{g}^{\beta}\mathbf{X}_*^{-1})^{b/(\gamma-b)} \\ &= \mathbf{X}_*,\end{aligned}$$

in G_6, we have

$$\begin{aligned}\mathbf{g}^{z_1}(\mathbf{u}\mathbf{v}^b)^{y_1} &= \mathbf{g}^{r_1}\mathbf{u}^{w_1} \cdot (\mathbf{g}^{s_1}\mathbf{v}^{w_1})^b \cdot (\mathbf{g}^{(-a-b\beta)/(\gamma-b)}(\mathbf{u}\mathbf{v}^b)^{1/(\gamma-b)})^{a_1 c} \\ &= \mathbf{R}_1 \mathbf{S}_1^b \mathbf{X}_*^{a_1 c}\end{aligned}$$

as in G_5.

We first give two intermediate simulators that simulate $\mathsf{G}_5/\mathsf{G}_6$, respectively. Since $\mathbf{u}, \mathbf{v} \in \mathsf{Span}\{\mathbf{g}\}$, we have $\mathbf{R}_1 = \mathbf{g}^{r_1}\mathbf{u}^{w_1}$ and $\mathbf{S}_1 = \mathbf{g}^{s_1}\mathbf{v}^{w_1}$ uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}$ and independent of w_1. It suffices for the simulators to uniformly sample $\mathbf{R}_1, \mathbf{S}_1 \leftarrow \mathsf{Span}\{\mathbf{g}\}$ in $\textsc{Sign}_1$ and defer sampling

$w_1 \leftarrow \mathbb{Z}_p$ to SIGN_2. Then SIGN_2 computes the unique r_1 and s_1 such that $\mathbf{R}_1 = \mathbf{g}^{r_1}\mathbf{u}^{w_1}$ and $\mathbf{S}_1 = \mathbf{g}^{s_1}\mathbf{v}^{w_1}$, and it sets z_1 and y_1 to be as in $\mathsf{G}_5/\mathsf{G}_6$.

Now we can give the single simulator that simulates both G_5 and G_6. Note that for both $y_1 := w_1$ in G_5 and $y_1 := w_1 + a_1 c/(\gamma - b)$ in G_6, uniformly sampling w_1 yields uniform y_1. Thus, it suffices for the simulator to directly sample $y_1 \leftarrow \mathbb{Z}_p$ and compute the unique z_1 satisfying $\mathbf{g}^{z_1}(\mathbf{u}\mathbf{v}^b)^{y_1} = \mathbf{R}_1\mathbf{S}_1^b\mathbf{X}_*^{a_1 c}$ in SIGN_2. Thus, the simulator perfectly simulates the views of A in both G_5 and G_6, showing that the views are identical. We hence have $\varepsilon_6 = \varepsilon_5$.

G_7. Change $\mathbf{X}_*$. In this game, we change the distribution of $\mathbf{X}_*$. Instead of setting $\mathbf{X}_* := \mathbf{g}^{x_*}$ for uniformly random x_*, this game samples $\mathbf{X}_* \leftarrow \mathbb{G}^2$. Thanks to the change in the last game, x_* is only referenced when computing $\mathbf{X}_*$. Hence, we can reduce distinguishing between G_6 and G_7 from DDH, so $\varepsilon_7 \geq \varepsilon_6 - \varepsilon_{\text{ddh}}$.

Bound ε_7. Finally, we bound ε_7. We will be conditioned on $\mathbf{X}_* \notin \mathsf{Span}\{\mathbf{g}\}$, which is true except with probability $1/p$.

We first bound the probability of a bad event: there exists $\mathcal{L}$ with $\mathbf{X}_* \in \mathcal{L}$ and $\mathsf{KAgg}(\mathcal{L})$ having been determined such that $\mathsf{KAgg}(\mathcal{L}) \in \mathsf{Span}\{\mathbf{g}\}$. Recall that $\mathsf{KAgg}(\mathcal{L}) = \prod_{i\in[|\mathcal{L}|]} \mathbf{X}_i^{a_i} = \mathbf{X}_*^{n_* a_1} \prod_{\mathbf{X}_i \neq \mathbf{X}_* \in \mathcal{L}} \mathbf{X}_i^{a_i}$ if $\mathbf{X}_*$ occurs n_* times in $\mathcal{L}$. Fixing the values of a_i for $\mathbf{X}_i \neq \mathbf{X}_* \in \mathcal{L}$, there exists at most one value of a_1 such that $\mathsf{KAgg}(\mathcal{L}) \in \mathsf{Span}\{\mathbf{g}\}$. Otherwise, if there exist $a_1 \neq a_1'$ making $\mathsf{KAgg}(\mathcal{L}) \in \mathsf{Span}\{\mathbf{g}\}$, then the quotient between the two $\mathsf{KAgg}(\mathcal{L})$ values, i.e. $\mathbf{X}_*^{n_*(a_1 - a_1')}$, must be in $\mathsf{Span}\{\mathbf{g}\}$. This contradicts our condition $\mathbf{X}_* \notin \mathsf{Span}\{\mathbf{g}\}$. Thus, the bad event occurs with probability at most q/p.

We now bound the probability that A wins G_7. For A to win, it must output a valid signature σ_* on $\mathcal{L}_*$ and μ_*, where $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}_*), \mu_*)$ is the i_*-th query and hence was set to γ. To be specific, A outputs a forged signature (c_*, z_*, y_*), and the verification recovers $\mathbf{R}_*$ such that $\mathsf{H}_c(\tilde{\mathbf{X}}_*, \mu_*, \mathbf{R}_*) = c_*$ and

$$\mathbf{R}_* \tilde{\mathbf{X}}_*^{c_*} = \mathbf{g}^{z_*}(\mathbf{u}\mathbf{v}^\gamma)^{y_*},$$

where $\tilde{\mathbf{X}}_* = \mathsf{KAgg}(\mathcal{L}_*)$. Recall that from G_2, we have $\mathbf{u} := \mathbf{g}^\alpha \mathbf{X}_*^\gamma$ and $\mathbf{v} := \mathbf{g}^\beta \mathbf{X}_*^{-1}$. Therefore,

$$\mathbf{R}_* \tilde{\mathbf{X}}_*^{c_*} = \mathbf{g}^{z_*}(\mathbf{g}^\alpha \mathbf{X}_*^\gamma (\mathbf{g}^\beta \mathbf{X}_*^{-1})^\gamma)^{y_*} = \mathbf{g}^{z_* + y_*(\alpha + \beta\gamma)}.$$

This implies that $\mathbf{R}_* \tilde{\mathbf{X}}_*^{c_*} \in \mathsf{Span}\{\mathbf{g}\}$.

Excluding the bad event, we have $\tilde{\mathbf{X}}_* \notin \mathsf{Span}\{\mathbf{g}\}$. Consequently, it holds that $\mathsf{H}_c(\tilde{\mathbf{X}}_*, \mu_*, \mathbf{R}_*) = c_*$ is a bad query to be defined as follows. A query $\mathsf{H}_c(\tilde{\mathbf{X}}, \mu, \mathbf{R}) = c$ is said to be bad if $\tilde{\mathbf{X}} \notin \mathsf{Span}\{\mathbf{g}\}$ while $\mathbf{R}\tilde{\mathbf{X}}^c \in \mathsf{Span}\{\mathbf{g}\}$. What remains for us is to bound the probability that a bad query occurs throughout the game.

For each query $\mathsf{H}_c(\tilde{\mathbf{X}}, \mu, \mathbf{R})$ with $\tilde{\mathbf{X}} \notin \mathsf{Span}\{\mathbf{g}\}$, there exists at most one answer c that makes the query bad, i.e. $\mathbf{R}\tilde{\mathbf{X}}^c \in \mathsf{Span}\{\mathbf{g}\}$. Otherwise, there exists another $c' \neq c$ such that $\mathbf{R}\tilde{\mathbf{X}}^{c'} \in \mathsf{Span}\{\mathbf{g}\}$. It follows that $\tilde{\mathbf{X}}^{c-c'} \in \mathsf{Span}\{\mathbf{g}\}$, a contradiction. Therefore, a bad query occurs throughout the game with probability at most q/p. We then conclude that $\varepsilon_7 \leq (2q+1)/p$. Putting everything together completes the proof. □

5 HBTS2[DDH]-Mask: Threshold Signature Scheme with Adaptive Security

5.1 Scheme

We provide a pseudocode description of HBTS2[DDH]-Mask with respect to group generation GrGen in Fig. 6. The scheme is obtained by applying the one-time masking technique [KRT24, Che25b] to HBTS2[DDH], which is very similar to HBMS2[DDH]. In addition to sharing x, the key generation

$\underline{\mathsf{Setup}(1^\lambda)}$
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\hat{g}, u, \hat{u}, v, \hat{v} \leftarrow \mathbb{G}$
$\mathbf{g} := (g, \hat{g}), \mathbf{u} := (u, \hat{u}), \mathbf{v} := (v, \hat{v})$
Sample $\mathsf{H}_b, \mathsf{H}_c : \{0,1\}^* \to \mathbb{Z}_p$
Sample $\mathsf{H}_m : \{0,1\}^* \to \mathbb{Z}_p^2$
Return $(\mathbb{G}, p, \mathbf{g}, \mathbf{u}, \mathbf{v}, \mathsf{H}_b, \mathsf{H}_c, \mathsf{H}_m)$

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

$\underline{\mathsf{Sign}_1(\mathsf{pk}, k, \mathsf{sk}_k)}$
$r_k, s_k, w_k \leftarrow \mathbb{Z}_p$
$\mathbf{R}_k := \mathbf{g}^{r_k}\mathbf{u}^{w_k}$
$\mathbf{S}_k := \mathbf{g}^{s_k}\mathbf{v}^{w_k}$
Return $(r_k, s_k, w_k), (\mathbf{R}_k, \mathbf{S}_k)$

$\underline{\mathsf{Sign}_2(\mathsf{pk}, \mathcal{S}, \mu, k, \mathsf{sk}_k, \mathsf{st}_k, \mathcal{M}_1)}$
$(x_k, \{(\mathsf{seed}_{k,i}, \mathsf{seed}_{i,k})\}_{i\in[n]\setminus\{k\}}) := \mathsf{sk}_k$
$(r_k, s_k, w_k) := \mathsf{st}_k$
$\{\mathbf{R}_i, \mathbf{S}_i\}_{i\in\mathcal{S}} := \mathcal{M}_1$
$\mathsf{sinf} := (\mathcal{S}, \mu, \mathcal{M}_1)$
$b := \mathsf{H}_b(\mathsf{pk}, \mu)$
$\mathbf{R} := \prod_{i\in\mathcal{S}} \mathbf{R}_i\mathbf{S}_i^b$
$c := \mathsf{H}_c(\mathsf{pk}, \mu, \mathbf{R})$
$z_k := r_k + bs_k + c\lambda_{\mathcal{S},k}x_k$
$y_k := w_k$
$\Delta_k :=$
  $\sum_{i\in\mathcal{S}\setminus\{k\}} \mathsf{H}_m(\mathsf{sinf}, k, i, \mathsf{seed}_{k,i}) -$
  $\sum_{i\in\mathcal{S}\setminus\{k\}} \mathsf{H}_m(\mathsf{sinf}, i, k, \mathsf{seed}_{i,k})$
$(\tilde{z}_k, \tilde{y}_k) := (z_k, y_k) + \Delta_k$
Return $(\tilde{z}_k, \tilde{y}_k)$

$\underline{\mathsf{Combine}(\mathsf{pk}, \mathcal{S}, \mu, \mathcal{M}_1, \mathcal{M}_2)}$
$\{(\mathbf{R}_i, \mathbf{S}_i)\}_{i\in\mathcal{S}} := \mathcal{M}_1$
$\{(\tilde{z}_i, \tilde{y}_i)\}_{i\in\mathcal{S}} := \mathcal{M}_2$
$b := \mathsf{H}_b(\mathsf{pk}, \mu)$
$\mathbf{R} := \prod_{i\in\mathcal{S}} \mathbf{R}_i\mathbf{S}_i^b$
$c := \mathsf{H}_c(\mathsf{pk}, \mu, \mathbf{R})$
$(z, y) := \sum_{i\in\mathcal{S}}(\tilde{z}_i, \tilde{y}_i)$
Return (c, z, y)

$\underline{\mathsf{Vf}(\mathsf{pk}, \mu, \sigma)}$
$\mathbf{X} := \mathsf{pk}$
$(c, z, y) := \sigma$
$b := \mathsf{H}_b(\mathsf{pk}, \mu)$
$\mathbf{R} := \mathbf{g}^z(\mathbf{u}\mathbf{v}^b)^y\mathbf{X}^{-c}$
Return $[\![c = \mathsf{H}_c(\mathsf{pk}, \mu, \mathbf{R})]\!]$

Figure 6: HBTS2[DDH]-Mask, defined with respect to a group generation algorithm GrGen.

generates two symmetric keys $\mathsf{seed}_{i,j}$, $\mathsf{seed}_{j,i} \in \{0,1\}^\lambda$ for each pair of users $i \neq j$. In the second round of the signing protocol, each user k uses its symmetric keys to generate the mask as

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

where sinf contains the session information $(\mathcal{S}, \mu, \mathcal{M}_1)$. Then the user masks its protocol message as $(\tilde{z}, \tilde{y}) := (z_k, y_k) + \Delta_k$.

Completeness. First, observe that the masks in a signing session are shares of zero by

$$\begin{aligned}\sum_{k\in\mathcal{S}} \Delta_k &= \sum_{k\in\mathcal{S}} \left(\sum_{i\in\mathcal{S}\setminus\{k\}} \mathsf{H}_m(\mathsf{sinf}, k, i, \mathsf{seed}_{k,i}) - \sum_{i\in\mathcal{S}\setminus\{k\}} \mathsf{H}_m(\mathsf{sinf}, i, k, \mathsf{seed}_{i,k}) \right) \\ &= \sum_{i\neq j\in\mathcal{S}} \mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}) - \sum_{i\neq j\in\mathcal{S}} \mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}) = 0.\end{aligned}$$

Parse $\Delta_i \in \mathbb{Z}_p^2$ as $(\Delta_{0,i}, \Delta_{1,i})$ for every $i \in \mathcal{S}$. By the following equation, we can see that the verification correctly recovers the aggregated commitment. Then the completeness follows.

$$\begin{aligned}
\mathbf{g}^z(\mathbf{u}\mathbf{v}^b)^y\mathbf{X}^{-c} &= \mathbf{g}^{\sum_{i\in\mathcal{S}}(r_i+bs_i+c\lambda_{\mathcal{S},i}x_i+\Delta_{0,i})}(\mathbf{u}\mathbf{v}^b)^{\sum_{i\in\mathcal{S}}(w_i+\Delta_{1,i})}\mathbf{g}^{-cx} \\
&= \mathbf{g}^{\sum_{i\in\mathcal{S}}\Delta_{0,i}}(\mathbf{u}\mathbf{v}^b)^{\sum_{i\in\mathcal{S}}\Delta_{1,i}}\prod_{i\in\mathcal{S}}\left(\mathbf{g}^{r_i+bs_i+c\lambda_{\mathcal{S},i}x_i}(\mathbf{u}\mathbf{v}^b)^{w_i}\mathbf{g}^{-c\lambda_{\mathcal{S},i}x_i}\right) \\
&= \prod_{i\in\mathcal{S}}(\mathbf{g}^{r_i}\mathbf{u}^{w_i}\mathbf{g}^{bs_i}\mathbf{v}^{bw_i}) \\
&= \prod_{i\in S}(\mathbf{R}_i\mathbf{S}_i^b) = \mathbf{R}.
\end{aligned}$$

5.2 Security

Theorem 2. *If* DDH *is* $(\tau_{\text{ddh}}, \varepsilon_{\text{ddh}})$*-hard for* GrGen*, then* HBTS2[DDH]-Mask *is* $(q_s, \tau_{\text{uf}}, \varepsilon_{\text{uf}})$-adp-TS-UF *in the random oracle model against any adversary that makes at most* q_h *hash queries, where*

$$\varepsilon_{\text{ddh}} = \frac{\varepsilon_{\text{uf}}}{q+1} - \frac{3q+1}{p} - \frac{1}{2^\lambda},$$

$q = q_s + q_h + 1$*, and* $\tau_{\text{uf}} \approx \tau_{\text{ddh}}$.

The proof has two parts. The first part is proving the effect of masking. Briefly speaking, the masked signature shares only provide useful information when combined into a threshold signature under the main key. As a result, we can switch to a hybrid game that only performs signing with the main secret key, instead of the secret key shares. Hence, the secret key share is only uniformly sampled when the corresponding user is corrupted. This part closely follows the roadmap from [KRT24, Che25b]. The second part of this proof is very similar to the proof of Theorem 1, where we switch to a hybrid game for which we can unconditionally bound the success probability of the adversary.

Proof. Let A be any τ_{uf}-time adversary that makes at most q_s queries to Sign_1 and q_h hash queries to each hash oracle. We need to prove that A's winning probability is at most ε_{uf}. To do so, we will build a series of hybrid games. Let G_0 be the original $\mathsf{adp\text{-}TS\text{-}UF}^{\mathsf{A}}_{\mathsf{HBTS[DDH]\text{-}Mask}}$ game. Eventually, we will reach a game where we can unconditionally bound A's winning probability. Let ε_i denote the probability that A wins G_i. Then we have $\varepsilon_0 = \varepsilon_{\text{uf}}$.

In G_0, let $\mathbf{X}$ denote the main public key and $\{x_i\}_{i\in[n]}$ denote the secret key shares. Hash functions H_b, H_c, H_m are modeled as random oracles. Without loss of generality, we assume that A does not make repeated queries to a random oracle. Throughout the game, we let $\mathcal{H}$ denote the set of honest signers $[n] \setminus \mathcal{C}$.

G_1: Globalize variables in Sign_2. This game aborts if Sign_1 ever returns repeating protocol messages. Since the first-round message $(\mathbf{R}_k, \mathbf{S}_k)$ is uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}^2$, we have $\varepsilon_1 \geq \varepsilon_0 - q_s^2/p^2$.

In this game, each signing session under user k that proceeds to the second round can be uniquely identified by sinf where $\mathsf{sinf} = (\mathcal{S}, \mu, \mathcal{M}_1)$. This is because $\mathcal{M}_1$ contains the unique first-round protocol message $(\mathbf{R}_k, \mathbf{S}_k)$ that can be only used once in Sign_2 under user k.

We let Sign_2 store its local variables (in the call of Sign_2) as global variables indexed by sinf. Specifically, it stores the local variables

$$r_k,\ s_k,\ w_k,\ b,\ c,\ z_k,\ y_k,\ \Delta_k,\ \tilde{z}_k,\ \tilde{y}_k,\ \mathbf{R}_k,\ \mathbf{S}_k$$

as global variables

$$r_{\mathsf{sinf},k},\ s_{\mathsf{sinf},k},\ w_{\mathsf{sinf},k},\ b_{\mathsf{sinf}},\ c_{\mathsf{sinf}},\ z_{\mathsf{sinf},k},\ y_{\mathsf{sinf},k},\ \Delta_{\mathsf{sinf},k},\ \tilde{z}_{\mathsf{sinf},k},\ \tilde{y}_{\mathsf{sinf},k},\ \mathbf{R}_{\mathsf{sinf},k},\ \mathbf{S}_{\mathsf{sinf},k}.$$

respectively, which can be referenced in the whole game including other oracle calls. Those indexed by (sinf, k) are never rewritten as sinf uniquely identifies the SIGN_2 queries under each user k. On the other hand, we note that b_{sinf} and c_{sinf} are not indexed by k. Multiple queries to SIGN_2 under different users may reference the same sinf. Nevertheless, b_{sinf} and c_{sinf} are still never written by inconsistent values, as b and c in SIGN_2 are fully determined by $\mathsf{sinf} = (\mathcal{S}, \mu, \mathcal{M}_1)$.

G_2: Replace $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}]$ with $\mathsf{T}[\mathsf{sinf}, i, j]$. Consider H_m as a lazy sampled table. In G_1, on a query $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$, the oracle returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{sinf}_{i,j}]$, with each table value uniformly sampled at random when referenced for the first time.

In this game, we introduce a table T. We use items $\mathsf{T}[\mathsf{sinf}, i, j]$ to replace $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}]$. That is, on each $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ query, the game returns $\mathsf{T}[\mathsf{sinf}, i, j]$ if $\mathsf{seed} = \mathsf{seed}_{i,j}$. If the item is undefined, the game uniformly samples it. If $\mathsf{seed} \neq \mathsf{seed}_{i,j}$, the game returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ as in G_0.

Moreover, we let SIGN_2 directly reference $\mathsf{T}[\mathsf{sinf}, i, j]$ instead of internally querying $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$. This changes in this game are only conceptual, so we have $\varepsilon_2 = \varepsilon_1$.

G_3: Defer generating $\mathsf{seed}_{i,j}$. In this game, we defer generating $\mathsf{seed}_{i,j}$ until signer i or j is corrupted. That is, when user k is corrupted, generate every $\mathsf{seed}_{k,i}$ and $\mathsf{seed}_{i,k}$ for $i \in [n] \setminus \{k\}$ that has not been generated.

Recall that we have made SIGN_2 directly reference $\mathsf{T}[\mathsf{sinf}, i, j]$ instead of querying $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$. As a result, SIGN_2 does not reference any $\mathsf{seed}_{i,j}$ any more, so deferring sampling $\mathsf{seed}_{i,j}$ does not affect SIGN_2. The view of A in G_2 differs from that in G_3 only if it has queried $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed})$ for some sinf and $\mathsf{seed} = \mathsf{seed}_{i,j}$ when both signer i and j are uncorrupted: in G_2 the oracle returns $\mathsf{T}[\mathsf{sinf}, i, j]$, while in G_3 it returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ as $\mathsf{seed}_{i,j}$ has not generated at that point. A can query $\mathsf{seed}_{i,j}$ to H_m before it is generated with probability at most $q_h/2^\lambda$, so we have $\varepsilon_3 \geq \varepsilon_2 - q_h/2^\lambda$.

G_4: Pack the generation of $\Delta_{\mathsf{sinf},k}$ and T items. Prior to this game, the mask $\Delta_{\mathsf{sinf},k} = \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{T}[\mathsf{sinf}, k, i] - \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{T}[\mathsf{sinf}, i, k]$ is only generated in SIGN_2. This game additionally generates $\Delta_{\mathsf{sinf},k}$ when $\mathsf{H}_m(\mathsf{sinf}, k, i, \mathsf{seed}_{k,i})$ or $\mathsf{H}_m(\mathsf{sinf}, i, k, \mathsf{seed}_{i,k})$ is queried, if user k is corrupt. That is, when answering such a query, the game does not only generate $\mathsf{T}[\mathsf{sinf}, k, i]$ or $\mathsf{T}[\mathsf{sinf}, i, k]$. Instead, it samples $\mathsf{T}[\mathsf{sinf}, k, j]$ and $\mathsf{T}[\mathsf{sinf}, j, k]$ that have not been generated for every $j \in \mathcal{S} \setminus \{k\}$ for the $\mathcal{S}$ contained in sinf, and then it computes $\Delta_{\mathsf{sinf},k}$. This change is invisible to A, so $\varepsilon_4 = \varepsilon_3$.

As a result of this change, $\mathsf{T}[\mathsf{sinf}, i, j]$ is never generated individually. Its generation must be packed with the generation of $\Delta_{\mathsf{sinf},i}$ or $\Delta_{\mathsf{sinf},j}$. This is because when a query in the form of $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$ is made, user i or j must be corrupt, as otherwise $\mathsf{seed}_{i,j}$ has never been generated.

G_5: Reverse generating $\Delta_{\mathsf{sinf},k}$ and T items, and defer sampling T items. In this game, we reverse the order of generating $\Delta_{\mathsf{sinf},k}$ and the related T item (i.e., $\mathsf{T}[\mathsf{sinf}, k, i]$ and $\mathsf{T}[\mathsf{sinf}, i, k]$ for every $i \in \mathcal{S}$, where $\mathcal{S}$ is contained in sinf) when they are generated, including in SIGN_2 and H_m. Previously, we first uniformly sample each undefined related T item then compute $\Delta_{\mathsf{sinf},k}$. In this game, we first sample $\Delta_{\mathsf{sinf},k}$ from its distribution, and then we sample all related T items conditioned on the value of $\Delta_{\mathsf{sinf},k}$.

The distribution of $\Delta_{\mathsf{sinf},k}$ is as follows.

- If for each $i \in \mathcal{S} \setminus \{k\}$, $\Delta_{\mathsf{sinf},i}$ has been defined, then all T items related to $\Delta_{\mathsf{sinf},k}$, i.e. $\mathsf{T}[\mathsf{sinf}, k, i]$, $\mathsf{T}[\mathsf{sinf}, i, k]$ for each $i \in \mathcal{S} \setminus \{k\}$ has been defined. Thus, $\Delta_{\mathsf{sinf},k}$ is determined to be $-\sum_{i \in \mathcal{S} \setminus \{k\}} \Delta_{\mathsf{sinf},i}$.

- Else, there exists some $j \in \mathcal{S} \setminus \{k\}$ such that $\Delta_{\mathsf{sinf},j}$ is undefined. Thus, items $\mathsf{T}[\mathsf{sinf}, k, j]$ and $\mathsf{T}[\mathsf{sinf}, j, k]$ are undefined and should be uniformly generated, which is guaranteed by the "packing" in the last game. Consequently, $\Delta_{\mathsf{sinf},k}$ is uniformly distributed.

After we reverse the order, $\mathsf{T}[\mathsf{sinf}, i, j]$ items are no longer referenced to generate $\Delta_{\mathsf{sinf},i}$ and $\Delta_{\mathsf{sinf},j}$. They are only referenced when answering $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$, which only occurs after user i or j are corrupted. Thus, we can defer the sampling of the related T items until user k is corrupted. These items are sampled conditioned on the value of $\Delta_{\mathsf{sinf},k}$. Specifically, the procedure is to uniformly sample all but one, and then the last related T is determined. The changes in this game do not affect the view of A. We have $\varepsilon_5 = \varepsilon_4$.

G_6: Alternative way to compute the last $\Delta_{\mathsf{sinf},k}$. In this game, we change the way to compute $\Delta_{\mathsf{sinf},k}$ for the last $k \in \mathcal{S}$, i.e. when $\Delta_{\mathsf{sinf},i}$ has been defined for every $i \in \mathcal{S} \setminus \{k\}$. Our goal is to compute $\Delta_{\mathsf{sinf},k}$ without referencing $\Delta_{\mathsf{sinf},i}$ or x_i if $i \in \mathcal{H}$.

Recall that $\Delta_{\mathsf{sinf},k} = -\sum_{i \in \mathcal{S}\setminus\{k\}} \Delta_{\mathsf{sinf},i}$. For $i \in \mathcal{H}$, $\Delta_{\mathsf{sinf},i}$ can only be defined in $\text{SIGN}_2(i, \mathsf{sid}, \mathcal{S}, \mu, \mathcal{M}_1)$ for some sid and $(\mathcal{S}, \mu, \mathcal{M}_1) = \mathsf{sinf}$. We substitute

$$\begin{aligned}\Delta_{\mathsf{sinf},i} &= \begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}z_{\mathsf{sinf},i}\\ y_{\mathsf{sinf},i}\end{bmatrix}\\ &= \begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},i} x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\end{aligned}$$

into $\Delta_{\mathsf{sinf},k} = -\sum_{i \in \mathcal{S}\setminus\{k\}} \Delta_{\mathsf{sinf},i}$. For $k \in \mathcal{H}$, we get

$$\begin{aligned}\Delta_{\mathsf{sinf},k} &= -\sum_{i \in (\mathcal{S}\cap\mathcal{H})\setminus\{k\}} \left(\begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},i} x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\right) - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \Delta_{\mathsf{sinf},i}\\ &= \sum_{i \in (\mathcal{S}\cap\mathcal{H})\setminus\{k\}} \left(\begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} - \tilde{z}_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i} - \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\right) + c_{\mathsf{sinf}} \begin{bmatrix}x - \sum_{i \in (\mathcal{S}\cap\mathcal{C})\cup\{k\}} \lambda_{\mathcal{S},i} x_i\\ 0\end{bmatrix} \qquad (1)\\ &\quad - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \Delta_{\mathsf{sinf},i}.\end{aligned}$$

In the second equation, we have used the reconstruction of Shamir's secret sharing: $x = \sum_{i \in \mathcal{S}} \lambda_{\mathcal{S},i} x_i$. Similarly, for $k \in \mathcal{C}$, we get

$$\begin{aligned}\Delta_{\mathsf{sinf},k} &= -\sum_{i \in \mathcal{S}\cap\mathcal{H}} \left(\begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},i} x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\right) - \sum_{i \in (\mathcal{S}\cap\mathcal{C})\setminus\{k\}} \Delta_{\mathsf{sinf},i}\\ &= \sum_{i \in \mathcal{S}\cap\mathcal{H}} \left(\begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} - \tilde{z}_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i} - \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\right) + c_{\mathsf{sinf}} \begin{bmatrix}x - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \lambda_{\mathcal{S},i} x_i\\ 0\end{bmatrix} \qquad (2)\\ &\quad - \sum_{i \in (\mathcal{S}\cap\mathcal{C})\setminus\{k\}} \Delta_{\mathsf{sinf},i}.\end{aligned}$$

In this game, we just change to an equivalent a way to compute $\Delta_{\mathsf{sinf},k}$ when it is deterministic. We have $\varepsilon_6 = \varepsilon_5$.

G_7. Reverse generating $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ and $\Delta_{\mathsf{sinf},k}$. In this game, we reverse the order of generating $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ and $\Delta_{\mathsf{sinf},k}$ in SIGN_2. That is, we directly sample $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ from its distribution, and then compute $\Delta_{\mathsf{sinf},k} = (\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k}) - (z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$.

The distribution of $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ is the following:

- If for each $i \in \mathcal{S} \setminus \{k\}$, $\Delta_{\mathsf{sinf},i}$ has been defined, then $\Delta_{\mathsf{sinf},k}$ is determined and to be computed by Eq. (4). Thus, $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ is computed by

$$\begin{aligned}\begin{bmatrix}\tilde{z}_{\mathsf{sinf},k}\\ \tilde{y}_{\mathsf{sinf},k}\end{bmatrix} &= \begin{bmatrix}z_{\mathsf{sinf},k}\\ y_{\mathsf{sinf},k}\end{bmatrix} + \Delta_{\mathsf{sinf},k}\\ &= \begin{bmatrix}r_{\mathsf{sinf},k} + b_{\mathsf{sinf}} s_{\mathsf{sinf},k} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},k} x_k\\ w_{\mathsf{sinf},k}\end{bmatrix} + \Delta_{\mathsf{sinf},k}\\ &= \sum_{i \in \mathcal{S}\cap\mathcal{H}} \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i}\end{bmatrix} - \sum_{i \in (\mathcal{S}\cap\mathcal{H})\setminus\{k\}} \begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\\ &\quad + c_{\mathsf{sinf}} \begin{bmatrix}x - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \lambda_{\mathcal{S},i} x_i\\ 0\end{bmatrix} - \sum_{i \in \mathcal{S}\cap\mathcal{C}} \Delta_{\mathsf{sinf},i}.\end{aligned} \tag{3}$$

- Else, there exists some $j \in \mathcal{S} \setminus \{k\}$ such that $\Delta_{\mathsf{sinf},j}$ is undefined. Then $\Delta_{\mathsf{sinf},k}$ is uniformly distributed, so is $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$.

The change does not affect the view of A. We have $\varepsilon_7 = \varepsilon_6$.

G_8**: Defer generating** x_k**.** In this game, we defer the generation or computation of a series of variables.

- Prior to the last game, the *specific value* of $\Delta_{\mathsf{sinf},k}$ is only referenced when computing $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ or when $\mathsf{T}[\mathsf{sinf}, k, i]$ or $\mathsf{T}[\mathsf{sinf}, i, k]$ needs to be defined after k is corrupt. The former occurs no more after the last game. Therefore, we can defer computing *specific value* of $\Delta_{\mathsf{sinf},k}$ until user k is corrupted by assigning a pending symbol $\Delta_{\mathsf{sinf},k} :=?$ for uncorrupted k.
  We stress that, by assigning ? to $\Delta_{\mathsf{sinf},k}$, we only defer the computation of the value of $\Delta_{\mathsf{sinf},k}$, but not its definition. This is because the state of being defined or undefined is still referenced before k is corrupted, when deciding which way to generate $\Delta_{\mathsf{sinf},j}$ or $(\tilde{z}_{\mathsf{sinf},j}, \tilde{y}_{\mathsf{sinf},j})$ for other $j \in \mathcal{S} \setminus \{k\}$.
- Next, note that in the last game, $(z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$ is only referenced to compute the value of $\Delta_{\mathsf{sinf},k}$. Since the computation has been deferred, we can also defer generating $(z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$ to the corruption of k.
- Next, note that x_k is only referenced when: computing $z_{\mathsf{sinf},k}$; being output in $\textsc{Corr}(k)$; computing Eqs. (5) and (6). We has just deferred the generation of $z_{\mathsf{sinf},k}$, and Eqs. (5) and (6) only reference x_i for $i \in \mathcal{S} \cap \mathcal{C}$. Therefore, $\{x_i\}_{i\in[n]}$ do not need to be generated in the game setup. Instead, we defer sampling x_k to $\textsc{Corr}(k)$. Since $x, x_1, \ldots, x_n$ are $(t+1)$-wise independent, and there are at most t corruptions, x_k is always uniformly sampled when generated.

All the changes here do not affect the view of A. We have $\varepsilon_8 = \varepsilon_7$.

We summarize the first part of this proof by a brief takeaway. Now in G_8, x_k is only uniformly sampled when user k is corrupted. The game never signs with the secret key shares. It only "signs" using the main secret key x, which means computing the term

$$\sum_{i \in \mathcal{S}\cap\mathcal{H}} \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i}\end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix}x\\ 0\end{bmatrix}$$

in Eqs. (5) and (6). Other terms in Eqs. (5) and (6) are publicly known.

The remaining part of this proof is similar to the proof of Theorem 1. We omit the details of those steps that have clear analogues in the proof of Theorem 1. The total number of queries to each hash oracle, including external queries from A and internal queries from other oracles and the verification of the final forgery, is bounded by q. One can verify that this bound holds for every hybrid game in the following.

G_9: **Rule out collisions in H_b.** The game aborts if there exist distinct H_b queries with the same returned values. We have $\varepsilon_9 \geq \varepsilon_8 - q^2/p$.

G_{10}: **Generate u and v as combinations of g and X.** This game generates $\mathbf{u}$ and $\mathbf{v}$ to be $\mathbf{u} := \mathbf{g}^{\alpha}\mathbf{X}^{\gamma}$ and $\mathbf{v} := \mathbf{g}^{\beta}\mathbf{X}^{-1}$ for uniformly random $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$. We have $\varepsilon_{10} \geq \varepsilon_9 - \varepsilon_{\mathrm{ddh}} - 1/p$.

G_{11}: **Guess the index of $\mathsf{H}_b(\mathbf{X}, \mu_*)$.** The game uniformly chooses $i_* \leftarrow [q]$ in the setup phase. For A to win, it is additionally required that $\mathsf{H}_b(\mathbf{X}, \mu_*)$ is the i_*-th distinct query made to H_b. We have $\varepsilon_{11} = \varepsilon_{10}/q$.

G_{12}: **Abort in advance if impossible to win.** The game aborts when A queries μ to $\textsc{Sign}_2$ such that $\mathsf{H}_b(\mathbf{X}, \mu)$ is the i_*-th distinct H_b query. We have $\varepsilon_{12} = \varepsilon_{11}$.

G_{13}: **Embed γ into the i_*-th H_b query.** The game answers the i_*-th H_b query with γ. We have $\varepsilon_{13} = \varepsilon_{12}$.

G_{14}: **Use α, β, γ to sign.** Previous games need to compute the following term in Eqs. (5) and (6):

$$\sum_{i \in \mathcal{S} \cap \mathcal{H}} \begin{bmatrix} r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} \\ w_{\mathsf{sinf},i} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} x \\ 0 \end{bmatrix},$$

where $\mathcal{S}$ is contained in sinf. We use $\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix}$ to denote this term. This game replaces this term in Eqs. (5) and (6) with

$$\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix} := \sum_{i \in \mathcal{S} \cap \mathcal{H}} \begin{bmatrix} r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} \\ w_{\mathsf{sinf},i} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} -(\alpha + b_{\mathsf{sinf}}\beta)/(\gamma - b_{\mathsf{sinf}}) \\ 1/(\gamma - b_{\mathsf{sinf}}) \end{bmatrix}.$$

Eq. (6) is computed in $\textsc{Sign}_2$. Eq. (5) is computed to generate $\Delta_{\mathsf{sinf},k}$ when $\Delta_{\mathsf{sinf},i}$ has been defined for every $i \in \mathcal{S} \setminus \{k\}$. At least one $i \in \mathcal{S}$ is uncorrupt, so at least one of these $\Delta_{\mathsf{sinf},i}$ is (was) generated in $\textsc{Sign}_2$. G_{12} has guaranteed that the game does not need to answer $\textsc{Sign}_2$ query that corresponds to the i_*-th H_b query. Since H_b has no collision, only the i_*-th H_b query returns γ. Therefore, we conclude that $b_{\mathsf{sinf}} \neq \gamma$ is Eqs. (5) and (6), so the denominator $\gamma - b \neq 0$.

We argue that G_{14} is identical to G_{13} in the view of A by giving an (inefficient) simulator that perfectly simulates both games. It can be verified that $\mathbf{g}^{\hat{z}_{\mathsf{sinf}}}(\mathbf{u}\mathbf{v}^{b_{\mathsf{sinf}}})^{\hat{y}_{\mathsf{sinf}}} = \prod_{i \in \mathcal{S} \cap \mathcal{H}} (\mathbf{R}_{\mathsf{sinf},i}\mathbf{S}_{\mathsf{sinf},i}^{b_{\mathsf{sinf}}})\mathbf{X}^{c_{\mathsf{sinf}}}$ in both G_{13} and G_{14}.

Since $\mathbf{u}$, $\mathbf{v} \in \mathsf{Span}\{\mathbf{g}\}$, $\textsc{Sign}_1$ outputs $\mathbf{R}_k = \mathbf{g}^{r_k}\mathbf{u}^{w_k}$ and $\mathbf{S}_k = \mathbf{g}^{s_k}\mathbf{v}^{w_k}$ that are uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}$ and independent of w_k. We first consider two intermediate simulators (for G_{13} and G_{14}, respectively) that uniformly samples every $\mathbf{R}_k$, $\mathbf{S}_k$ in $\textsc{Sign}_1$ and defer the generation of r_k, s_k, w_k until they are referenced. Specifically:

- When user k is corrupted, the simulators need to output r_k, s_k, w_k for each $\textsc{Sign}_1$ query under user k. They uniformly sample $w_k \leftarrow \mathbb{Z}_p$ and computes the unique r_k, s_k satisfying $\mathbf{R}_k = \mathbf{g}^{r_k}\mathbf{u}^{w_k}$ and $\mathbf{S}_k = \mathbf{g}^{s_k}\mathbf{v}^{w_k}$.
- When the game needs to compute the term $\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix}$, the simulators sample $\hat{w}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$ and compute the unique $\hat{r}_{\mathsf{sinf}}$, $\hat{s}_{\mathsf{sinf}}$ satisfying $\prod_{i \in \mathcal{S} \cap \mathcal{H}} \mathbf{R}_{\mathsf{sinf},i} = \mathbf{g}^{\hat{r}_{\mathsf{sinf}}}\mathbf{u}^{\hat{w}_{\mathsf{sinf}}}$ and $\prod_{i \in \mathcal{S} \cap \mathcal{H}} \mathbf{S}_{\mathsf{sinf},i} = \mathbf{g}^{\hat{s}_{\mathsf{sinf}}}\mathbf{v}^{\hat{w}_{\mathsf{sinf}}}$. Namely, $\hat{r}_{\mathsf{sinf}}$, $\hat{s}_{\mathsf{sinf}}$, and $\hat{w}_{\mathsf{sinf}}$ are generated as the sums $\sum_{i \in \mathcal{S} \cap \mathcal{H}} r_{\mathsf{sinf},i}$, $\sum_{i \in \mathcal{S} \cap \mathcal{H}} s_{\mathsf{sinf},i}$, and $\sum_{i \in \mathcal{S} \cap \mathcal{H}} w_{\mathsf{sinf},i}$, respectively. Then the simulator for G_{13} computes

$$\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix} := \begin{bmatrix} \hat{r}_{\mathsf{sinf}} + b_{\mathsf{sinf}}\hat{s}_{\mathsf{sinf}} \\ \hat{w}_{\mathsf{sinf}} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} x \\ 0 \end{bmatrix},$$

and the simulator for G_{14} computes

$$\begin{bmatrix}\hat{z}_{\mathsf{sinf}}\\ \hat{y}_{\mathsf{sinf}}\end{bmatrix} := \begin{bmatrix}\hat{r}_{\mathsf{sinf}} + b_{\mathsf{sinf}}\hat{s}_{\mathsf{sinf}}\\ \hat{w}_{\mathsf{sinf}}\end{bmatrix} + c_{\mathsf{sinf}}\begin{bmatrix}-(\alpha + b_{\mathsf{sinf}}\beta)/(\gamma - b_{\mathsf{sinf}})\\ 1/(\gamma - b_{\mathsf{sinf}})\end{bmatrix}.$$

If the simulators have sampled $\hat{w}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$, they should sample $w_{\mathsf{sinf},i}$ for $i \in \mathcal{S} \cap \mathcal{H}$ (when i is corrupted) under the constraint of $\sum_{i\in\mathcal{S}\cap\mathcal{H}} w_{\mathsf{sinf},i} = \hat{w}_{\mathsf{sinf}}$. We note that this does not really affect the distribution of $w_{\mathsf{sinf},i}$, as users in $\mathcal{S} \cap \mathcal{H}$ cannot be all corrupted, so the simulators can still sample each $w_{\mathsf{sinf},i}$ uniformly from $\mathbb{Z}_p$.

Finally, we reach a single simulator that simulates both G_{13} and G_{14}. Since $\hat{w}_{\mathsf{sinf}}$ is uniformly distributed, we know that $\hat{y}_{\mathsf{sinf}}$ is also uniformly distributed in both G_{13} and G_{14}. The simulator simply samples $\hat{y}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$ and computes the unique $\hat{z}_{\mathsf{sinf}}$ such that $\mathbf{g}^{\hat{z}_{\mathsf{sinf}}}(\mathbf{u}\mathbf{v}^{b_{\mathsf{sinf}}})^{\hat{y}_{\mathsf{sinf}}} = \prod_{i\in\mathcal{S}\cap\mathcal{H}}(\mathbf{R}_{\mathsf{sinf},i}\mathbf{S}_{\mathsf{sinf},i}^{b_{\mathsf{sinf}}})\mathbf{X}^{c_{\mathsf{sinf}}}$.

In conclusion, we have $\varepsilon_{14} = \varepsilon_{13}$.

G_{15}: Change $\mathbf{X}$. This game samples $\mathbf{X}$ uniformly random from $\mathbb{G}^2$. We have $\varepsilon_{15} \geq \varepsilon_{14} - \varepsilon_{\mathrm{ddh}}$.

Bound ε_{15}. This is the same as how we bound ε_7 in the proof of Theorem 1, except that here we do not need to bound the bad event about KAgg. We can get a bound of $\varepsilon_{15} \leq (q+1)/p$. □

5.3 HBTS2[DL]-Mask

In Fig. 6, if we set $\mathbf{g}, \mathbf{u}, \mathbf{v}$ to be single elements g, u, v, we obtain the scheme $\mathsf{HBTS2[DL]}$. Consequently, $\mathbf{X}$, $\mathbf{R}$, $\mathbf{S}$ are all single elements in the scheme. The scheme achieves fully adaptive security under the minimal DL assumption, yet with a looser security bound of $\Theta(\varepsilon^2/q_h^3)$. The proof idea has been mentioned in Section 2.1.

The proof would be virtually the same as the proof of Theorem 2 until the very last hybrid game. Instead of changing the distribution of the public key $\mathbf{X}_*$, we would build a rewinding reduction to DL. At this point, the game does not use the secret key x except for the key generation, so the reduction can embed a DL input into $\mathbf{X}_*$. Also, the adversary is required to forge for $\mathcal{L}_*$ and μ_* such that $\mathsf{H}_b(\mathsf{KAgg}(\mathcal{L}_*), \mu_*)$ is the i_*-th query, which was set to γ. A valid forged signature (c_*, z_*, y_*) allows recovering $\mathbf{R}_*$ which satisfies $\mathsf{H}_c(\mathbf{X}_*, \mu, \mathbf{R}_*) = c_*$ and

$$\mathbf{R}_*\mathbf{X}_*^{c_*} = \mathbf{g}^{z_*}(\mathbf{u}\mathbf{v}^{\gamma})^{y_*} = \mathbf{g}^{z_*}(\mathbf{g}^{\alpha}\mathbf{X}_*^{\gamma}(\mathbf{g}^{\beta}\mathbf{X}_*^{-1})^{\gamma})^{y_*} = \mathbf{g}^{z_*+y_*(\alpha+\beta\gamma)}.$$

A standard rewinding reduction runs a related second execution of the game to obtain (c'_*, z'_*, y'_*) that satisfies $c'_* \neq c_*$ and

$$\mathbf{R}_*\mathbf{X}_*^{c'_*} = \mathbf{g}^{z'_*}(\mathbf{u}\mathbf{v}^{\gamma})^{y'_*} = \mathbf{g}^{z'_*}(\mathbf{g}^{\alpha}\mathbf{X}_*^{\gamma}(\mathbf{g}^{\beta}\mathbf{X}_*^{-1})^{\gamma})^{y'_*} = \mathbf{g}^{z'_*+y'_*(\alpha+\beta\gamma)}.$$

It can then extract the discrete logarithm of $\mathbf{X}_*$ as

$$\frac{z'_* + y'_*(\alpha+\beta\gamma) - (z_* + y_*(\alpha+\beta\gamma))}{c'_* - c_*}.$$

References

[AB21] Handan Kilinç Alper and Jeffrey Burdges. Two-round trip Schnorr multi-signatures via delinearized witnesses. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, August 2021. Springer, Cham. 2, 3, 5, 9

[AF04] Masayuki Abe and Serge Fehr. Adaptively secure feldman VSS and applications to universally-composable threshold cryptography. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 317–334. Springer, Berlin, Heidelberg, August 2004. 5

[AFLT16] Michel Abdalla, Pierre-Alain Fouque, Vadim Lyubashevsky, and Mehdi Tibouchi. Tightly secure signatures from lossy identification schemes. *Journal of Cryptology*, 29(3):597–631, July 2016. 8

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 2004. 7

[BC26] Renas Bacho and Yanbo Chen. Earpicks: Tightly secure two-round multi and threshold signatures. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part I*, volume 16541 of *LNCS*, pages 154–183. Springer, Cham, May 2026. 2, 3, 5

[BCJ08] Ali Bagherzandi, Jung Hee Cheon, and Stanislaw Jarecki. Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pages 449–458. ACM Press, October 2008. 5

[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. 2, 5, 13

[BCL+26] Renas Bacho, Yanbo Chen, Julian Loss, Stefano Tessaro, and Chenzhi Zhu. Adaptively secure partially non-interactive threshold schnorr signatures in the AGM. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part I*, volume 16541 of *LNCS*, pages 33–62. Springer, Cham, May 2026. 2, 3, 4, 6, 9

[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. 2, 3, 5, 6

[BDLR25a] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Adaptively secure three-round threshold Schnorr signatures from DDH. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 390–422. Springer, Cham, August 2025. 6

[BDLR25b] Renas Bacho, Sourav Das, Julian Loss, and Ling Ren. Glacius: Threshold schnorr signatures from DDH with full adaptive security. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part II*, volume 15602 of *LNCS*, pages 304–334. Springer, Cham, May 2025. 6

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International conference on the theory and application of cryptology and information security*, pages 435–464. Springer, 2018. 5

[BDPT25] Carsten Baum, Bernardo David, Elena Pagnin, and Akira Takahashi. Universally composable interactive and ordered multi-signatures. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part II*, volume 15675 of *LNCS*, pages 3–31. Springer, Cham, May 2025. 5

[BGC+25] Ruben Baecker, Paul Gerhart, Davide Li Calsi, Luigi Russo, Dominique Schröder, and Arkady Yerukhimovich. Fully adaptive FROST in the algebraic group model from falsifiable assumptions. Cryptology ePrint Archive, Report 2025/1950, 2025. 2, 4, 6, 9

[BJ08] Ali Bagherzandi and Stanislaw Jarecki. Multisignatures using proofs of secret key possession, as secure as the Diffie-Hellman problem. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 218–235. Springer, Berlin, Heidelberg, September 2008. 5

[BL22] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 193–207. ACM Press, November 2022. 5

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Cham, October 2021. 5

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001. 1

[BLSW24] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. HARTS: High-threshold, adaptively secure, and robust threshold Schnorr signatures. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*, pages 104–140. Springer, Singapore, December 2024. 6

[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. 6

[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. 2, 4, 5, 10

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003. 1, 4

[BW25] Renas Bacho and Benedikt Wagner. T-spoon: Tightly secure two-round multi-signatures with key aggregation. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 256–290. Springer, Cham, August 2025. 2, 3, 5, 8, 9

[BW26] Renas Bacho and Benedikt Wagner. Tightly secure threshold signatures over pairing-free groups. *IACR Communications in Cryptology*, 2(4), 2026. 5, 8

[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. 5

[CGRS23] Hien Chu, Paul Gerhart, Tim Ruffing, and Dominique Schröder. Practical Schnorr threshold signatures without the algebraic group model. In Helena Handschuh and Anna

Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 743–773. Springer, Cham, August 2023. 5

[Che25a] Yanbo Chen. Dazzle: Improved adaptive threshold signatures from DDH. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part III*, volume 15676 of *LNCS*, pages 233–261. Springer, Cham, May 2025. 2, 4, 5, 6, 8, 9

[Che25b] Yanbo Chen. Round-efficient adaptively secure threshold signatures with rewinding. *CiC*, 2(2):16, 2025. 2, 3, 4, 6, 9, 18, 20

[CKK+25] Elizabeth C. Crites, Jonathan Katz, Chelsea Komlo, Stefano Tessaro, and Chenzhi Zhu. On the adaptive security of FROST. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 480–511. Springer, Cham, August 2025. 2, 3, 4, 5

[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. 5

[CKM25] Elizabeth Crites, Chelsea Komlo, and Mary Maller. On the adaptive security of key-unique threshold signatures. Cryptology ePrint Archive, Report 2025/943, 2025. 2, 5

[CS25] Elizabeth C. Crites and Alistair Stewart. A plausible attack on the adaptive security of threshold Schnorr signatures. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VI*, volume 16005 of *LNCS*, pages 457–479. Springer, Cham, August 2025. 4, 5

[DEF+19] Manu Drijvers, Kasra Edalatnejad, Bryan Ford, Eike Kiltz, Julian Loss, Gregory Neven, and Igors Stepanovs. On the security of two-round multi-signatures. In *2019 IEEE Symposium on Security and Privacy*, pages 1084–1101. IEEE Computer Society Press, May 2019. 5

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 120–127. Springer, Berlin, Heidelberg, August 1988. 1

[DGNP25] Hila Dahari-Garbian, Ariel Nof, and Luke Parker. Trout: Two-round threshold ECDSA from class groups. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 380–393. ACM Press, October 2025. 6

[FH21] Masayuki Fukumitsu and Shingo Hasegawa. A tightly secure ddh-based multisignature with public-key aggregation. *Int. J. Netw. Comput.*, 11(2):319–337, 2021. 5, 8

[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. 2

[FMY99] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure distributed public-key systems. In *European Symposium on Algorithms*, pages 4–27. Springer, 1999. 5

[GCRS26] Paul Gerhart, Davide Li Calsi, Luigi Russo, and Dominique Schröder. Fully-adaptive two-round threshold schnorr signatures from DDH. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part II*, volume 16542 of *LNCS*, pages 251–283. Springer, Cham, May 2026. 2, 4, 6

[GGN16] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. Threshold-optimal DSA/ECDSA signatures and an application to Bitcoin wallet security. In Mark Manulis, Ahmad-Reza Sadeghi, and Steve Schneider, editors, *ACNS 2016*, volume 9696 of *LNCS*, pages 156–174. Springer, Cham, June 2016. 1, 6

[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. 5

[HDL+25] Calvin Abou Haidar, Dipayan Das, Anja Lehmann, Cavit Özbay, and Octavio Perez-Kempner. Privacy-preserving multi-signatures: Generic techniques and constructions without pairings. In Tibor Jager and Jiaxin Pan, editors, *PKC 2025, Part II*, volume 15675 of *LNCS*, pages 66–98. Springer, Cham, May 2025. 2, 3, 5

[HJK12] Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 66–83. Springer, Berlin, Heidelberg, May 2012. 10

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

[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. 5

[JWZ26] Kaijie Jiang, Hoeteck Wee, and Chenzhi Zhu. Oriole: Adaptively secure partially non-interactive threshold signatures from lattices. Cryptology ePrint Archive, Paper 2026/793, 2026. 4

[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. 1, 2, 3, 4, 5, 9

[KRT24] 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. 2, 3, 4, 6, 9, 18, 20

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, October 2003. 8

[Lin24] Yehuda Lindell. Simple three-round multiparty Schnorr signing with full simulatability. *CiC*, 1(1):25, 2024. 5

[LLZD25] Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, and Xudong Deng. Threshold ECDSA in two rounds. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier, editors, *ACM CCS 2025*, pages 2937–2950. ACM Press, October 2025. 6

[LÖ25] Anja Lehmann and Cavit Özbay. Putting multi into multi-signatures: Tight security for multiple signers. Cryptology ePrint Archive, Report 2025/2198, 2025. 3, 5, 8

[LÖ26] Anja Lehmann and Cavit Özbay. Putting multi into multi-signatures: Tight security for multiple signers. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part I*, volume 16541 of *LNCS*, pages 184–214. Springer, Cham, May 2026. 2, 3, 5

[LOS+06] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 465–485. Springer, Berlin, Heidelberg, May / June 2006. 4

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Berlin, Heidelberg, December 2001. 5

[Mak22] Nikolaos Makriyannis. On the classic protocol for MPC Schnorr signatures. Cryptology ePrint Archive, Report 2022/1332, 2022. 5

[MOR01] Silvio Micali, Kazuo Ohta, and Leonid Reyzin. Accountable-subgroup multisignatures: Extended abstract. In Michael K. Reiter and Pierangela Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, November 2001. 4, 5

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *DCC*, 87(9):2139–2164, 2019. 1, 5, 7

[MWLD10] Changshe Ma, Jian Weng, Yingjiu Li, and Robert H. Deng. Efficient discrete logarithm based multi-signature scheme in the plain public key model. *DCC*, 54(2):121–133, 2010. 5

[NRS21] Jonas Nick, Tim Ruffing, and Yannick Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, August 2021. Springer, Cham. 2, 3, 5, 7, 8, 9

[NRSW20] Jonas Nick, Tim Ruffing, Yannick Seurin, and Pieter Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, November 2020. 3, 5

[NRT26] Guilhem Niot, Michael Reichle, and Kaoru Takemure. Adaptively-secure three-round threshold schnorr from DL. In Joan Daemen and Emmanuel Thomé, editors, *EUROCRYPT 2026, Part I*, volume 16541 of *LNCS*, pages 3–32. Springer, Cham, May 2026. 6

[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. 6

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Berlin, Heidelberg, May 1996. 2

[PW23] Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 597–627. Springer, Cham, April 2023. 2, 3, 5, 8, 9

[PW24] Jiaxin Pan and Benedikt Wagner. Toothpicks: More efficient fork-free two-round multi-signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part I*, volume 14651 of *LNCS*, pages 460–489. Springer, Cham, May 2024. 2, 3, 5, 8, 9

[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. 2, 5, 6

[RSY24] Lior Rotem, Gil Segev, and Eylon Yogev. From one-time to two-round reusable multi-signatures without nested forking. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 371–399. Springer, Cham, December 2024. 5

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multi-party signatures against rogue-key attacks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 228–245. Springer, Berlin, Heidelberg, May 2007. 5

[Sha79] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979. 10

[SS01] Douglas R Stinson and Reto Strobl. Provably secure distributed schnorr signatures and a (t, n) threshold scheme for implicit certificates. In *Australasian Conference on Information Security and Privacy*, pages 417–434. Springer, 2001. 5

[SS24] Gil Segev and Liat Shapira. An explicit high-moment forking lemma and its applications to the concrete security of multi-signatures. *CiC*, 1(2):2, 2024. 5

[STV+16] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy*, pages 526–545. IEEE Computer Society Press, May 2016. 5

[TSS+24] Kaoru Takemure, Yusuke Sakai, Bagus Santoso, Goichiro Hanaoka, and Kazuo Ohta. More efficient two-round multi-signature scheme with provably secure parameters for standardized elliptic curves. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 107(7):966–988, 2024. 2, 3, 5, 6, 8, 9

[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. 2, 3, 5, 9

[Wat05] Brent R. Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 114–127. Springer, Berlin, Heidelberg, May 2005. 10

A Concrete Efficiency

We compare the concrete efficiency (specifically signature size, public-key size, and communication overhead) of partially non-interactive schemes under parameter choices that guarantee 128-bit security in Table 3. In brief, compared to MuSig2 and TZ with key aggregation, our HBMS2[DDH] offers significant advantages in both signature size and communication, at the cost of larger public keys. Crucially, the loose security bounds of MuSig2 and TZ require too large group orders, which make it hard to choose well-analyzed, standardized elliptic curves.

Target Level	Scheme & Bound	Min. $\log_2(p)$ & Standard Curve	Concrete Sizes in Bytes (Min. / Standard Curve)		
			Signature	Public Key	Communication
$\tau = q_h = 2^{128}$ $\varepsilon = 1$	HBMS2[DDH] $\varepsilon = q_h\tau^2/p$	≥ 384 *NIST P-384*	144 / 144	98 / 98	292 / 292
	MuSig2 $\varepsilon = (q_h^3\tau^2/p)^{1/4}$	≥ 640 —	161 / —	81 / —	404 / —
	TZ $\varepsilon = (q_h^3\tau^2/p)^{1/4}$	≥ 640 —	241 / —	81 / —	484 / —
	MuSig2 (no KA) $\varepsilon = (q_h\tau^2/p)^{1/2}$	≥ 384 *NIST P-384*	97 / 97	49 / 49	146 / 146
	TZ (no KA) $\varepsilon = (q_h\tau^2/p)^{1/2}$	≥ 384 *NIST P-384*	145 / 145	49 / 49	194 / 194
$\tau = q_h = 2^{96}$ $\varepsilon = 2^{-32}$	HBMS2[DDH] $\varepsilon = q_h\tau^2/p$	≥ 320 *BrainpoolP320r1*	120 / 120	82 / 82	244 / 244
	MuSig2 $\varepsilon = (q_h^3\tau^2/p)^{1/4}$	≥ 608 —	153 / —	77 / —	384 / —
	TZ $\varepsilon = (q_h^3\tau^2/p)^{1/4}$	≥ 608 —	229 / —	77 / —	460 / —
	MuSig2 (no KA) $\varepsilon = (q_h\tau^2/p)^{1/2}$	≥ 352 *NIST P-384*	89 / 97	45 / 49	134 / 146
	TZ (no KA) $\varepsilon = (q_h\tau^2/p)^{1/2}$	≥ 352 *NIST P-384*	133 / 145	45 / 49	178 / 194

Table 3: Comparison of concrete efficiency and physical byte sizes for selected multi-signature schemes. For each scheme, we calculate the theoretical minimum bit-length of the group order, $\log_2(p)$, to achieve the security target, and specify the closest accommodating standardized elliptic curve (e.g., NIST P-384, BrainpoolP320r1). A "—" indicates that the required bit-length exceeds broadly supported standard curves. Concrete sizes for signatures, public keys, and per-signer communication are measured in bytes. We assume point compression, where a group element requires $|\mathbb{G}| = |\mathbb{Z}_p| + 1$ bytes, and a scalar requires $|\mathbb{Z}_p| = \lceil \log_2(p)/8 \rceil$ bytes.

Here we outline the setting. We evaluate the schemes based on the following security bounds (where ε_r and ε_a denote the success probabilities of the reduction and the adversary, respectively):
- Our HBMS2[DDH]: $\varepsilon_r \geq \varepsilon_a/q_h$
- MuSig2 and TZ with key aggregation: $\varepsilon_r \geq \varepsilon_a^4/q_h^3$
- MuSig2 and TZ without key aggregation: $\varepsilon_r \geq \varepsilon_a^2/q_h$

Assume that the best known algorithm against the underlying discrete-logarithm-type problems runs in time τ and has a success probability upper-bounded by τ^2/p, where p is the order of the group. That is, we assume $\varepsilon_r \leq \tau^2/p$. From this, the above reduction bounds translate to the following upper bounds on the adversary's success probability:
- HBMS2[DDH]: $q_h\tau^2/p$
- MuSig2 and TZ with key aggregation: $(q_h^3\tau^2/p)^{1/4}$
- MuSig2 and TZ without key aggregation: $(q_h\tau^2/p)^{1/2}$

As q_h counts the number of local hashing operations, we consider $q_h = \tau$. 128-bit security requires that the above upper bounds on the adversary's success probability are below the line $\tau/2^{128}$. It suffices to ensure that the bound is smaller than 1 when $\tau = 2^{128}$.

Let us first focus on the comparison between HBMS2[DDH] and MuSig2 with key aggregation. To achieve the target $\tau = 2^{128}$, $\varepsilon_a = 1$, our scheme requires $\log(p) \geq 384$, whereas MuSig2 requires $\log(p) \geq 640$. Consequently, HBMS2[DDH] has shorter signatures (144B vs 161B), less communication (292B vs 404B), trading off for a larger public key (98B vs 81B). More importantly, there are abundant choices for standardized curves at the 384-bit level (e.g., NIST P-384), but there are no such choices supporting $\log(p) \geq 640$.

When targeting $\tau = 2^{128}$, $\varepsilon_a = 1$, the quadratic/quartic losses have no effect. They come into play when we target smaller τ and ε. As an example, we consider $\tau = 2^{96}$, $\varepsilon = 2^{-32}$. This threshold reflects the view that $\tau = 2^{128}$ represents an impractically large amount of computation, so we only ensure that the adversary's success probability is below the line $\tau/2^{128}$ up to the point $\tau = 2^{96}$. Under this target, all schemes achieve better performance, but quadratic/quartic losses make the improvement slower. The comparison between HBMS2[DDH] and MuSig2 is as follows: signature size (120B vs 153B), communication (244B vs 384B), and public key (82B vs 77B).

We remark that MuSig2 relies on an interactive assumption (AOMDL). If one targets a non-interactive assumption, one would switch to TZ, compared to which our signature size and communication advantages become more significant. We note, however, that our scheme relies on DDH while TZ only relies on DL.

We also compare our scheme with MuSig2 and TZ without key aggregation, which also reflects the concrete efficiency comparison between our threshold scheme HBTS2[DDH] and FROST as well as the TZ version of FROST. At $\tau = 2^{128}$ and $\varepsilon = 1$, the tightness gap does not manifest, and our bound provides no concrete advantage over the quadratic-loss schemes. However, for $\tau = 2^{96}$, $\varepsilon = 2^{-32}$, our tighter bound yields concrete signature size savings over TZ. Overall, when key aggregation is not needed, or in the threshold setting, the concrete efficiency advantage of our scheme is mild. The main advantage we claim for our threshold schemes is adaptive security without the AGM rather than tightness and concrete efficiency.

B Proof of Theorem 2

Proof. Let A be any τ_{uf}-time adversary that makes at most q_s queries to $\textsc{Sign}_1$ and q_h hash queries to each hash oracle. We need to prove that A's winning probability is at most ε_{uf}. To do so, we will build a series of hybrid games. Let G_0 be the original $\mathsf{adp\text{-}TS\text{-}UF}^{\mathsf{A}}_{\mathsf{HBTS[DDH]\text{-}Mask}}$ game. Eventually, we will reach a game where we can unconditionally bound A's winning probability. Let ε_i denote the probability that A wins G_i. Then we have $\varepsilon_0 = \varepsilon_{\text{uf}}$.

In G_0, let $\mathbf{X}$ denote the main public key and $\{x_i\}_{i\in[n]}$ denote the secret key shares. Hash functions H_b, H_c, H_m are modeled as random oracles. Without loss of generality, we assume that A does not

make repeated queries to a random oracle. Throughout the game, we let $\mathcal{H}$ denote the set of honest signers $[n] \setminus \mathcal{C}$.

G_1: Globalize variables in Sign_2. This game aborts if $\textsc{Sign}_1$ ever returns repeating protocol messages. Since the first-round message $(\mathbf{R}_k, \mathbf{S}_k)$ is uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}^2$, we have $\varepsilon_1 \geq \varepsilon_0 - q_s^2/p^2$.

In this game, each signing session under user k that proceeds to the second round can be uniquely identified by sinf where $\mathsf{sinf} = (\mathcal{S}, \mu, \mathcal{M}_1)$. This is because $\mathcal{M}_1$ contains the unique first-round protocol message $(\mathbf{R}_k, \mathbf{S}_k)$ that can be only used once in $\textsc{Sign}_2$ under user k.

We let $\textsc{Sign}_2$ store its local variables (in the call of Sign_2) as global variables indexed by sinf. Specifically, it stores the local variables

$$r_k,\ s_k,\ w_k,\ b,\ c,\ z_k,\ y_k,\ \Delta_k,\ \tilde{z}_k,\ \tilde{y}_k,\ \mathbf{R}_k,\ \mathbf{S}_k$$

as global variables

$$r_{\mathsf{sinf},k},\ s_{\mathsf{sinf},k},\ w_{\mathsf{sinf},k},\ b_{\mathsf{sinf}},\ c_{\mathsf{sinf}},\ z_{\mathsf{sinf},k},\ y_{\mathsf{sinf},k},\ \Delta_{\mathsf{sinf},k},\ \tilde{z}_{\mathsf{sinf},k},\ \tilde{y}_{\mathsf{sinf},k},\ \mathbf{R}_{\mathsf{sinf},k},\ \mathbf{S}_{\mathsf{sinf},k}.$$

respectively, which can be referenced in the whole game including other oracle calls. Those indexed by (sinf, k) are never rewritten as sinf uniquely identifies the $\textsc{Sign}_2$ queries under each user k. On the other hand, we note that b_{sinf} and c_{sinf} are not indexed by k. Multiple queries to $\textsc{Sign}_2$ under different users may reference the same sinf. Nevertheless, b_{sinf} and c_{sinf} are still never written by inconsistent values, as b and c in $\textsc{Sign}_2$ are fully determined by $\mathsf{sinf} = (\mathcal{S}, \mu, \mathcal{M}_1)$.

G_2: Replace $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}]$ with $\mathsf{T}[\mathsf{sinf}, i, j]$. Consider H_m as a lazy sampled table. In G_1, on a query $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$, the oracle returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{sinf}_{i,j}]$, with each table value uniformly sampled at random when referenced for the first time.

In this game, we introduce a table T. We use items $\mathsf{T}[\mathsf{sinf}, i, j]$ to replace $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}_{i,j}]$. That is, on each $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ query, the game returns $\mathsf{T}[\mathsf{sinf}, i, j]$ if $\mathsf{seed} = \mathsf{seed}_{i,j}$. If the item is undefined, the game uniformly samples it. If $\mathsf{seed} \neq \mathsf{seed}_{i,j}$, the game returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ as in G_0.

Moreover, we let $\textsc{Sign}_2$ directly reference $\mathsf{T}[\mathsf{sinf}, i, j]$ instead of internally querying $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$. This changes in this game are only conceptual, so we have $\varepsilon_2 = \varepsilon_1$.

G_3: Defer generating $\mathsf{seed}_{i,j}$. In this game, we defer generating $\mathsf{seed}_{i,j}$ until signer i or j is corrupted. That is, when user k is corrupted, generate every $\mathsf{seed}_{k,i}$ and $\mathsf{seed}_{i,k}$ for $i \in [n] \setminus \{k\}$ that has not been generated.

Recall that we have made $\textsc{Sign}_2$ directly reference $\mathsf{T}[\mathsf{sinf}, i, j]$ instead of querying $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$. As a result, $\textsc{Sign}_2$ does not reference any $\mathsf{seed}_{i,j}$ any more, so deferring sampling $\mathsf{seed}_{i,j}$ does not affect $\textsc{Sign}_2$. The view of A in G_2 differs from that in G_3 only if it has queried $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed})$ for some sinf and $\mathsf{seed} = \mathsf{seed}_{i,j}$ when both signer i and j are uncorrupted: in G_2 the oracle returns $\mathsf{T}[\mathsf{sinf}, i, j]$, while in G_3 it returns $\mathsf{H}_m[\mathsf{sinf}, i, j, \mathsf{seed}]$ as $\mathsf{seed}_{i,j}$ has not generated at that point. A can query $\mathsf{seed}_{i,j}$ to H_m before it is generated with probability at most $q_h/2^\lambda$, so we have $\varepsilon_3 \geq \varepsilon_2 - q_h/2^\lambda$.

G_4: Pack the generation of $\Delta_{\mathsf{sinf},k}$ and T items. Prior to this game, the mask $\Delta_{\mathsf{sinf},k} = \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{T}[\mathsf{sinf}, k, i] - \sum_{i \in \mathcal{S} \setminus \{k\}} \mathsf{T}[\mathsf{sinf}, i, k]$ is only generated in $\textsc{Sign}_2$. This game additionally generates $\Delta_{\mathsf{sinf},k}$ when $\mathsf{H}_m(\mathsf{sinf}, k, i, \mathsf{seed}_{k,i})$ or $\mathsf{H}_m(\mathsf{sinf}, i, k, \mathsf{seed}_{i,k})$ is queried, if user k is corrupt. That is, when answering such a query, the game does not only generate $\mathsf{T}[\mathsf{sinf}, k, i]$ or $\mathsf{T}[\mathsf{sinf}, i, k]$. Instead, it samples $\mathsf{T}[\mathsf{sinf}, k, j]$ and $\mathsf{T}[\mathsf{sinf}, j, k]$ that have not been generated for every $j \in \mathcal{S} \setminus \{k\}$ for the $\mathcal{S}$ contained in sinf, and then it computes $\Delta_{\mathsf{sinf},k}$. This change is invisible to A, so $\varepsilon_4 = \varepsilon_3$.

As a result of this change, $\mathsf{T}[\mathsf{sinf}, i, j]$ is never generated individually. Its generation must be packed with the generation of $\Delta_{\mathsf{sinf},i}$ or $\Delta_{\mathsf{sinf},j}$. This is because when a query in the form of $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$ is made, user i or j must be corrupt, as otherwise $\mathsf{seed}_{i,j}$ has never been generated.

G_5: Reverse generating $\Delta_{\mathsf{sinf},k}$ and T items, and defer sampling T items. In this game, we reverse the order of generating $\Delta_{\mathsf{sinf},k}$ and the related T item (i.e., $\mathsf{T}[\mathsf{sinf}, k, i]$ and $\mathsf{T}[\mathsf{sinf}, i, k]$ for every $i \in \mathcal{S}$, where $\mathcal{S}$ is contained in sinf) when they are generated, including in $\textsc{Sign}_2$ and H_m. Previously, we first uniformly sample each undefined related T item then compute $\Delta_{\mathsf{sinf},k}$. In this game, we first sample $\Delta_{\mathsf{sinf},k}$ from its distribution, and then we sample all related T items conditioned on the value of $\Delta_{\mathsf{sinf},k}$.

The distribution of $\Delta_{\mathsf{sinf},k}$ is as follows.

- If for each $i \in \mathcal{S} \setminus \{k\}$, $\Delta_{\mathsf{sinf},i}$ has been defined, then all T items related to $\Delta_{\mathsf{sinf},k}$, i.e. $\mathsf{T}[\mathsf{sinf}, k, i]$, $\mathsf{T}[\mathsf{sinf}, i, k]$ for each $i \in \mathcal{S}\setminus\{k\}$ has been defined. Thus, $\Delta_{\mathsf{sinf},k}$ is determined to be $-\sum_{i\in\mathcal{S}\setminus\{k\}} \Delta_{\mathsf{sinf},i}$.
- Else, there exists some $j \in \mathcal{S} \setminus \{k\}$ such that $\Delta_{\mathsf{sinf},j}$ is undefined. Thus, items $\mathsf{T}[\mathsf{sinf}, k, j]$ and $\mathsf{T}[\mathsf{sinf}, j, k]$ are undefined and should be uniformly generated, which is guaranteed by the "packing" in the last game. Consequently, $\Delta_{\mathsf{sinf},k}$ is uniformly distributed.

After we reverse the order, $\mathsf{T}[\mathsf{sinf}, i, j]$ items are no longer referenced to generate $\Delta_{\mathsf{sinf},i}$ and $\Delta_{\mathsf{sinf},j}$. They are only referenced when answering $\mathsf{H}_m(\mathsf{sinf}, i, j, \mathsf{seed}_{i,j})$, which only occurs after user i or j are corrupted. Thus, we can defer the sampling of the related T items until user k is corrupted. These items are sampled conditioned on the value of $\Delta_{\mathsf{sinf},k}$. Specifically, the procedure is to uniformly sample all but one, and then the last related T is determined. The changes in this game do not affect the view of A. We have $\varepsilon_5 = \varepsilon_4$.

G_6: Alternative way to compute the last $\Delta_{\mathsf{sinf},k}$. In this game, we change the way to compute $\Delta_{\mathsf{sinf},k}$ for the last $k \in \mathcal{S}$, i.e. when $\Delta_{\mathsf{sinf},i}$ has been defined for every $i \in \mathcal{S} \setminus \{k\}$. Our goal is to compute $\Delta_{\mathsf{sinf},k}$ without referencing $\Delta_{\mathsf{sinf},i}$ or x_i if $i \in \mathcal{H}$.

Recall that $\Delta_{\mathsf{sinf},k} = -\sum_{i\in\mathcal{S}\setminus\{k\}} \Delta_{\mathsf{sinf},i}$. For $i \in \mathcal{H}$, $\Delta_{\mathsf{sinf},i}$ can only be defined in $\textsc{Sign}_2(i, \mathsf{sid}, \mathcal{S}, \mu, \mathcal{M}_1)$ for some sid and $(\mathcal{S}, \mu, \mathcal{M}_1) = \mathsf{sinf}$. We substitute

$$\begin{aligned}\Delta_{\mathsf{sinf},i} &= \begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}z_{\mathsf{sinf},i}\\ y_{\mathsf{sinf},i}\end{bmatrix}\\ &= \begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},i} x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\end{aligned}$$

into $\Delta_{\mathsf{sinf},k} = -\sum_{i\in\mathcal{S}\setminus\{k\}} \Delta_{\mathsf{sinf},i}$. For $k \in \mathcal{H}$, we get

$$\begin{aligned}\Delta_{\mathsf{sinf},k} &= -\sum_{i\in(\mathcal{S}\cap\mathcal{H})\setminus\{k\}} \left(\begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} + c_{\mathsf{sinf}} \lambda_{\mathcal{S},i} x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\right) - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \Delta_{\mathsf{sinf},i}\\ &= \sum_{i\in(\mathcal{S}\cap\mathcal{H})\setminus\{k\}} \left(\begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} - \tilde{z}_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i} - \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\right) + c_{\mathsf{sinf}} \begin{bmatrix}x - \sum_{i\in(\mathcal{S}\cap\mathcal{C})\cup\{k\}} \lambda_{\mathcal{S},i} x_i\\ 0\end{bmatrix} \qquad (4)\\ &\quad - \sum_{i\in\mathcal{S}\cap\mathcal{C}} \Delta_{\mathsf{sinf},i}.\end{aligned}$$

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

Similarly, for $k \in \mathcal{C}$, we get

$$\begin{aligned}\Delta_{\mathsf{sinf},k} &= -\sum_{i\in\mathcal{S}\cap\mathcal{H}}\left(\begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix} - \begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}}s_{\mathsf{sinf},i} + c_{\mathsf{sinf}}\lambda_{\mathcal{S},i}x_i\\ w_{\mathsf{sinf},i}\end{bmatrix}\right) - \sum_{i\in(\mathcal{S}\cap\mathcal{C})\setminus\{k\}}\Delta_{\mathsf{sinf},i}\\ &= \sum_{i\in\mathcal{S}\cap\mathcal{H}}\left(\begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}}s_{\mathsf{sinf},i} - \tilde{z}_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i} - \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\right) + c_{\mathsf{sinf}}\begin{bmatrix}x - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\lambda_{\mathcal{S},i}x_i\\ 0\end{bmatrix}\\ &\quad - \sum_{i\in(\mathcal{S}\cap\mathcal{C})\setminus\{k\}}\Delta_{\mathsf{sinf},i}.\end{aligned} \tag{5}$$

In this game, we just change to an equivalent a way to compute $\Delta_{\mathsf{sinf},k}$ when it is deterministic. We have $\varepsilon_6 = \varepsilon_5$.

G_7. Reverse generating $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ and $\Delta_{\mathsf{sinf},k}$. In this game, we reverse the order of generating $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ and $\Delta_{\mathsf{sinf},k}$ in $\textsc{Sign}_2$. That is, we directly sample $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ from its distribution, and then compute $\Delta_{\mathsf{sinf},k} = (\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k}) - (z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$.

The distribution of $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ is the following:

- If for each $i \in \mathcal{S}\setminus\{k\}$, $\Delta_{\mathsf{sinf},i}$ has been defined, then $\Delta_{\mathsf{sinf},k}$ is determined and to be computed by Eq. (4). Thus, $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ is computed by

$$\begin{aligned}\begin{bmatrix}\tilde{z}_{\mathsf{sinf},k}\\ \tilde{y}_{\mathsf{sinf},k}\end{bmatrix} &= \begin{bmatrix}z_{\mathsf{sinf},k}\\ y_{\mathsf{sinf},k}\end{bmatrix} + \Delta_{\mathsf{sinf},k}\\ &= \begin{bmatrix}r_{\mathsf{sinf},k} + b_{\mathsf{sinf}}s_{\mathsf{sinf},k} + c_{\mathsf{sinf}}\lambda_{\mathcal{S},k}x_k\\ w_{\mathsf{sinf},k}\end{bmatrix} + \Delta_{\mathsf{sinf},k}\\ &= \sum_{i\in\mathcal{S}\cap\mathcal{H}}\begin{bmatrix}r_{\mathsf{sinf},i} + b_{\mathsf{sinf}}s_{\mathsf{sinf},i}\\ w_{\mathsf{sinf},i}\end{bmatrix} - \sum_{i\in(\mathcal{S}\cap\mathcal{H})\setminus\{k\}}\begin{bmatrix}\tilde{z}_{\mathsf{sinf},i}\\ \tilde{y}_{\mathsf{sinf},i}\end{bmatrix}\\ &\quad + c_{\mathsf{sinf}}\begin{bmatrix}x - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\lambda_{\mathcal{S},i}x_i\\ 0\end{bmatrix} - \sum_{i\in\mathcal{S}\cap\mathcal{C}}\Delta_{\mathsf{sinf},i}.\end{aligned} \tag{6}$$

- Else, there exists some $j \in \mathcal{S}\setminus\{k\}$ such that $\Delta_{\mathsf{sinf},j}$ is undefined. Then $\Delta_{\mathsf{sinf},k}$ is uniformly distributed, so is $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$.

The change does not affect the view of A. We have $\varepsilon_7 = \varepsilon_6$.

G_8: Defer generating x_k. In this game, we defer the generation or computation of a series of variables.

- Prior to the last game, the *specific value* of $\Delta_{\mathsf{sinf},k}$ is only referenced when computing $(\tilde{z}_{\mathsf{sinf},k}, \tilde{y}_{\mathsf{sinf},k})$ or when $\mathsf{T}[\mathsf{sinf}, k, i]$ or $\mathsf{T}[\mathsf{sinf}, i, k]$ needs to be defined after k is corrupt. The former occurs no more after the last game. Therefore, we can defer computing *specific value* of $\Delta_{\mathsf{sinf},k}$ until user k is corrupted by assigning a pending symbol $\Delta_{\mathsf{sinf},k} :=?$ for uncorrupted k.
  We stress that, by assigning ? to $\Delta_{\mathsf{sinf},k}$, we only defer the computation of the value of $\Delta_{\mathsf{sinf},k}$, but not its definition. This is because the state of being defined or undefined is still referenced before k is corrupted, when deciding which way to generate $\Delta_{\mathsf{sinf},j}$ or $(\tilde{z}_{\mathsf{sinf},j}, \tilde{y}_{\mathsf{sinf},j})$ for other $j \in \mathcal{S}\setminus\{k\}$.
- Next, note that in the last game, $(z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$ is only referenced to compute the value of $\Delta_{\mathsf{sinf},k}$. Since the computation has been deferred, we can also defer generating $(z_{\mathsf{sinf},k}, y_{\mathsf{sinf},k})$ to the corruption of k.
- Next, note that x_k is only referenced when: computing $z_{\mathsf{sinf},k}$; being output in $\textsc{Corr}(k)$; computing Eqs. (5) and (6). We has just deferred the generation of $z_{\mathsf{sinf},k}$, and Eqs. (5) and (6) only reference x_i for $i \in \mathcal{S}\cap\mathcal{C}$. Therefore, $\{x_i\}_{i\in[n]}$ do not need to be generated in the game setup. Instead, we defer sampling x_k to $\textsc{Corr}(k)$. Since $x, x_1, \ldots, x_n$ are $(t+1)$-wise independent, and there are at most t corruptions, x_k is always uniformly sampled when generated.

All the changes here do not affect the view of A. We have $\varepsilon_8 = \varepsilon_7$.

We summarize the first part of this proof by a brief takeaway. Now in G_8, x_k is only uniformly sampled when user k is corrupted. The game never signs with the secret key shares. It only "signs" using the main secret key x, which means computing the term

$$\sum_{i \in \mathcal{S} \cap \mathcal{H}} \begin{bmatrix} r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} \\ w_{\mathsf{sinf},i} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} x \\ 0 \end{bmatrix}$$

in Eqs. (5) and (6). Other terms in Eqs. (5) and (6) are publicly known.

The remaining part of this proof is similar to the proof of Theorem 1. We omit the details of those steps that have clear analogues in the proof of Theorem 1. The total number of queries to each hash oracle, including external queries from A and internal queries from other oracles and the verification of the final forgery, is bounded by q. One can verify that this bound holds for every hybrid game in the following.

G_9: Rule out collisions in H_b. The game aborts if there exist distinct H_b queries with the same returned values. We have $\varepsilon_9 \geq \varepsilon_8 - q^2/p$.

G_{10}: Generate u and v as combinations of g and X. This game generates $\mathbf{u}$ and $\mathbf{v}$ to be $\mathbf{u} := \mathbf{g}^{\alpha}\mathbf{X}^{\gamma}$ and $\mathbf{v} := \mathbf{g}^{\beta}\mathbf{X}^{-1}$ for uniformly random $\alpha, \beta, \gamma \leftarrow \mathbb{Z}_p$. We have $\varepsilon_{10} \geq \varepsilon_9 - \varepsilon_{\mathrm{ddh}} - 1/p$.

G_{11}: Guess the index of $\mathsf{H}_b(\mathbf{X}, \mu_*)$. The game uniformly chooses $i_* \leftarrow [q]$ in the setup phase. For A to win, it is additionally required that $\mathsf{H}_b(\mathbf{X}, \mu_*)$ is the i_*-th distinct query made to H_b. We have $\varepsilon_{11} = \varepsilon_{10}/q$.

G_{12}: Abort in advance if impossible to win. The game aborts when A queries μ to $\textsc{Sign}_2$ such that $\mathsf{H}_b(\mathbf{X}, \mu)$ is the i_*-th distinct H_b query. We have $\varepsilon_{12} = \varepsilon_{11}$.

G_{13}: Embed γ into the i_*-th H_b query. The game answers the i_*-th H_b query with γ. We have $\varepsilon_{13} = \varepsilon_{12}$.

G_{14}: Use α, β, γ to sign. Previous games need to compute the following term in Eqs. (5) and (6):

$$\sum_{i \in \mathcal{S} \cap \mathcal{H}} \begin{bmatrix} r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} \\ w_{\mathsf{sinf},i} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} x \\ 0 \end{bmatrix},$$

where $\mathcal{S}$ is contained in sinf. We use $\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix}$ to denote this term. This game replaces this term in Eqs. (5) and (6) with

$$\begin{bmatrix} \hat{z}_{\mathsf{sinf}} \\ \hat{y}_{\mathsf{sinf}} \end{bmatrix} := \sum_{i \in \mathcal{S} \cap \mathcal{H}} \begin{bmatrix} r_{\mathsf{sinf},i} + b_{\mathsf{sinf}} s_{\mathsf{sinf},i} \\ w_{\mathsf{sinf},i} \end{bmatrix} + c_{\mathsf{sinf}} \begin{bmatrix} -(\alpha + b_{\mathsf{sinf}}\beta)/(\gamma - b_{\mathsf{sinf}}) \\ 1/(\gamma - b_{\mathsf{sinf}}) \end{bmatrix}.$$

Eq. (6) is computed in $\textsc{Sign}_2$. Eq. (5) is computed to generate $\Delta_{\mathsf{sinf},k}$ when $\Delta_{\mathsf{sinf},i}$ has been defined for every $i \in \mathcal{S} \setminus \{k\}$. At least one $i \in \mathcal{S}$ is uncorrupt, so at least one of these $\Delta_{\mathsf{sinf},i}$ is (was) generated in $\textsc{Sign}_2$. G_{12} has guaranteed that the game does not need to answer $\textsc{Sign}_2$ query that corresponds to the i_*-th H_b query. Since H_b has no collision, only the i_*-th H_b query returns γ. Therefore, we conclude that $b_{\mathsf{sinf}} \neq \gamma$ is Eqs. (5) and (6), so the denominator $\gamma - b \neq 0$.

We argue that G_{14} is identical to G_{13} in the view of A by giving an (inefficient) simulator that perfectly simulates both games. It can be verified that $\mathbf{g}^{\hat{z}_{\mathsf{sinf}}}(\mathbf{u}\mathbf{v}^{b_{\mathsf{sinf}}})^{\hat{y}_{\mathsf{sinf}}} = \prod_{i\in\mathcal{S}\cap\mathcal{H}}(\mathbf{R}_{\mathsf{sinf},i}\mathbf{S}_{\mathsf{sinf},i}^{b_{\mathsf{sinf}}})\mathbf{X}^{c_{\mathsf{sinf}}}$ in both G_{13} and G_{14}.

Since $\mathbf{u}$, $\mathbf{v} \in \mathsf{Span}\{\mathbf{g}\}$, $\textsc{Sign}_1$ outputs $\mathbf{R}_k = \mathbf{g}^{r_k}\mathbf{u}^{w_k}$ and $\mathbf{S}_k = \mathbf{g}^{s_k}\mathbf{v}^{w_k}$ that are uniformly distributed over $\mathsf{Span}\{\mathbf{g}\}$ and independent of w_k. We first consider two intermediate simulators (for G_{13} and G_{14}, respectively) that uniformly samples every $\mathbf{R}_k$, $\mathbf{S}_k$ in $\textsc{Sign}_1$ and defer the generation of r_k, s_k, w_k until they are referenced. Specifically:

- When user k is corrupted, the simulators need to output r_k, s_k, w_k for each $\textsc{Sign}_1$ query under user k. They uniformly sample $w_k \leftarrow \mathbb{Z}_p$ and computes the unique r_k, s_k satisfying $\mathbf{R}_k = \mathbf{g}^{r_k}\mathbf{u}^{w_k}$ and $\mathbf{S}_k = \mathbf{g}^{s_k}\mathbf{v}^{w_k}$.
- When the game needs to compute the term $\begin{bmatrix}\hat{z}_{\mathsf{sinf}}\\ \hat{y}_{\mathsf{sinf}}\end{bmatrix}$, the simulators sample $\hat{w}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$ and compute the unique $\hat{r}_{\mathsf{sinf}}$, $\hat{s}_{\mathsf{sinf}}$ satisfying $\prod_{i\in\mathcal{S}\cap\mathcal{H}}\mathbf{R}_{\mathsf{sinf},i} = \mathbf{g}^{\hat{r}_{\mathsf{sinf}}}\mathbf{u}^{\hat{w}_{\mathsf{sinf}}}$ and $\prod_{i\in\mathcal{S}\cap\mathcal{H}}\mathbf{S}_{\mathsf{sinf},i} = \mathbf{g}^{\hat{s}_{\mathsf{sinf}}}\mathbf{v}^{\hat{w}_{\mathsf{sinf}}}$. Namely, $\hat{r}_{\mathsf{sinf}}$, $\hat{s}_{\mathsf{sinf}}$, and $\hat{w}_{\mathsf{sinf}}$ are generated as the sums $\sum_{i\in\mathcal{S}\cap\mathcal{H}} r_{\mathsf{sinf},i}$, $\sum_{i\in\mathcal{S}\cap\mathcal{H}} s_{\mathsf{sinf},i}$, and $\sum_{i\in\mathcal{S}\cap\mathcal{H}} w_{\mathsf{sinf},i}$, respectively. Then the simulator for G_{13} computes

$$\begin{bmatrix}\hat{z}_{\mathsf{sinf}}\\ \hat{y}_{\mathsf{sinf}}\end{bmatrix} := \begin{bmatrix}\hat{r}_{\mathsf{sinf}} + b_{\mathsf{sinf}}\hat{s}_{\mathsf{sinf}}\\ \hat{w}_{\mathsf{sinf}}\end{bmatrix} + c_{\mathsf{sinf}}\begin{bmatrix}x\\ 0\end{bmatrix},$$

  and the simulator for G_{14} computes

$$\begin{bmatrix}\hat{z}_{\mathsf{sinf}}\\ \hat{y}_{\mathsf{sinf}}\end{bmatrix} := \begin{bmatrix}\hat{r}_{\mathsf{sinf}} + b_{\mathsf{sinf}}\hat{s}_{\mathsf{sinf}}\\ \hat{w}_{\mathsf{sinf}}\end{bmatrix} + c_{\mathsf{sinf}}\begin{bmatrix}-(\alpha + b_{\mathsf{sinf}}\beta)/(\gamma - b_{\mathsf{sinf}})\\ 1/(\gamma - b_{\mathsf{sinf}})\end{bmatrix}.$$

  If the simulators have sampled $\hat{w}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$, they should sample $w_{\mathsf{sinf},i}$ for $i \in \mathcal{S}\cap\mathcal{H}$ (when i is corrupted) under the constraint of $\sum_{i\in\mathcal{S}\cap\mathcal{H}} w_{\mathsf{sinf},i} = \hat{w}_{\mathsf{sinf}}$. We note that this does not really affect the distribution of $w_{\mathsf{sinf},i}$, as users in $\mathcal{S}\cap\mathcal{H}$ cannot be all corrupted, so the simulators can still sample each $w_{\mathsf{sinf},i}$ uniformly from $\mathbb{Z}_p$.

Finally, we reach a single simulator that simulates both G_{13} and G_{14}. Since $\hat{w}_{\mathsf{sinf}}$ is uniformly distributed, we know that $\hat{y}_{\mathsf{sinf}}$ is also uniformly distributed in both G_{13} and G_{14}. The simulator simply samples $\hat{y}_{\mathsf{sinf}} \leftarrow \mathbb{Z}_p$ and computes the unique $\hat{z}_{\mathsf{sinf}}$ such that $\mathbf{g}^{\hat{z}_{\mathsf{sinf}}}(\mathbf{u}\mathbf{v}^{b_{\mathsf{sinf}}})^{\hat{y}_{\mathsf{sinf}}} = \prod_{i\in\mathcal{S}\cap\mathcal{H}}(\mathbf{R}_{\mathsf{sinf},i}\mathbf{S}_{\mathsf{sinf},i}^{b_{\mathsf{sinf}}})\mathbf{X}^{c_{\mathsf{sinf}}}$.

In conclusion, we have $\varepsilon_{14} = \varepsilon_{13}$.

G_{15}: Change X. This game samples $\mathbf{X}$ uniformly random from $\mathbb{G}^2$. We have $\varepsilon_{15} \geq \varepsilon_{14} - \varepsilon_{\mathrm{ddh}}$.

Bound ε_{15}. This is the same as how we bound ε_7 in the proof of Theorem 1, except that here we do not need to bound the bad event about KAgg. We can get a bound of $\varepsilon_{15} \leq (q+1)/p$. □