Threshold and Multi-Signature Schemes from Linear Hash Functions*

Stefano Tessaro and Chenzhi Zhu

Paul G. Allen School of Computer Science & Engineering
University of Washington, Seattle, US
{tessaro,zhucz20}@cs.washington.edu

Abstract. This paper gives new constructions of two-round multi-signatures and threshold signatures for which security relies solely on either the hardness of the (plain) discrete logarithm problem or the hardness of RSA, in addition to assuming random oracles. Their signing protocol is partially non-interactive, i.e., the first round of the signing protocol is independent of the message being signed.
We obtain our constructions by generalizing the most efficient discrete-logarithm based schemes, MuSig2 (Nick, Ruffing, and Seurin, CRYPTO '21) and FROST (Komlo and Goldberg, SAC '20), to work with suitably defined linear hash functions. While the original schemes rely on the stronger and more controversial one-more discrete logarithm assumption, we show that suitable instantiations of the hash functions enable security to be based on either the plain discrete logarithm assumption or on RSA. The signatures produced by our schemes are equivalent to those obtained from Okamoto's identification schemes (CRYPTO '92).
More abstractly, our results suggest a general framework to transform schemes secure under OMDL into ones secure under the plain DL assumption and, with some restrictions, under RSA.

1 Introduction

Many novel applications, such as digital wallets [GGN16], are re-energizing a multi-decade agenda aimed at developing new efficient multi-signatures [Ita83b] and threshold signatures [Des88, DF90] from a variety of assumptions. Threshold signatures are also at the center of standardization efforts by NIST [Natnt] and IETF [CKGW22]. Both signature types are relatively straightforward to obtain from pairings (using, e.g., BLS [BLS01, Bol03]); however, specific implementation constraints make pairing-free schemes, which are based on either variants of the discrete logarithm or RSA problems, appealing in several contexts.

This paper aims to build the best possible pairing-free multi-signatures and threshold signatures under the weakest possible assumptions. As our main contribution, we develop new two-round protocols that are secure under the (1) discrete logarithm assumption and (2) the RSA assumption. In both cases, we also assume the random oracle model (ROM) [BR93]. Our RSA multi-signatures require a trusted setup to produce a public RSA modulus with unknown factorization. The signatures produced by both schemes resemble those proposed by Okamoto [Oka93]. Furthermore, our signing protocols are partially non-interactive, i.e., the first round messages do not depend on the message being signed, which is a desirable property in practice.

Significance. Our DL-based schemes are the first partially non-interactive 2-round schemes based solely on the hardness of the discrete logarithm assumption. For threshold signatures, in particular, no two-round scheme is known from only the discrete logarithm assumption. For RSA, the landscape is more complex, and our main contribution is to provide a viable multi-signature scheme, as all prior solutions impose restrictions.

* An extended abstract of this paper appears in the proceedings of EUROCRYPT 2023. This is the full version.

OUR APPROACH. Our schemes are the outcome of the same paradigm applied to the two most efficient DL-based schemes, FROST [KG20, BCK+22] and MuSig2 [NRS21]. It is not known how to prove the security of either scheme under the plain discrete logarithm assumption, and they are instead proved secure under the (stronger) *one-more discrete logarithm assumption* (OMDL) [BNPS03], an assumption that has been the subject of criticism [KM07, KM08]. As we explain next, our paradigm can be seen as a general recipe to remove the OMDL assumption from these schemes.

The main ingredient of our approach are *linear hash functions*, which have also been used in recent works [BBSS18, HKL19, HKLN20] to abstract identification schemes from which signature variants are derived. Here, we observe that both FROST ad MuSig2 can naturally be generalized by replacing the exponentiation map $x \mapsto g^x$ with a linear hash function $\mathsf{F} : \mathcal{D} \to \mathcal{R}$, where $\mathcal{D}, \mathcal{R}$ are $\mathcal{S}$-modules for a field $\mathcal{S}$. We generically refer to these instantiations as FROST-H and MuSig2-H. (In fact, we present two variants for FROST-H but make no distinction in the introduction.) In particular, we require that:

- F is an epimorphism of $\mathcal{S}$-modules from $\mathcal{D}$ to $\mathcal{R}$, i.e., F is a surjection from $\mathcal{D}$ to $\mathcal{R}$ such that for any $r \in \mathcal{S}$ and $x, y \in \mathcal{D}$, $\mathsf{F}(x + r \cdot y) = \mathsf{F}(x) + r \cdot \mathsf{F}(y)$.
- F is *not a monomorphism*, which is equivalent to postulating that there exists $z^* \in \mathcal{D}$ such that $z^* \neq 0$ and $\mathsf{F}(z^*) = 0$.

We then define a natural analogue of the OMDL assumption, which we refer to as the *Algebraic One-More Preimage Resistance* (AOMPR). Roughly speaking, the corresponding security game allows the attacker to obtain multiple *challenges* $X_i = \mathsf{F}(x_i)$ for a random element $x_i \leftarrow\!\!\$\, \mathcal{D}$, and the attacker also gets access to an *inversion oracle* which, on input $X \in \mathcal{R}$, returns a element in the preimage set of X under F. The restriction here, and hence the term *algebraic*, is that X must be an affine combination of previously obtained X_i's, and this affine combination is given to the inversion oracle, along with X. (This makes the assumption falsifiable since the oracle can efficiently answer such inversion queries.) To win the game, the attacker is then asked to invert $q + 1$ challenges after querying the inversion oracle at most q times.

Our results then follow from the combination of the following two theorems, which we state here informally:

> **Theorem (informal).** The security of FROST-H and MuSig2-H follows from the AOMPR assumption on the underlying linear hash function.

> **Theorem (informal).** If F is collision-resistant, then the AOMPR assumption holds with respect to F.

The proof of the first theorem is, on its own, not particularly surprising and mostly generalizes the prior proofs in the literature, in particular those of [NRS21] and [BCK+22]. Our main contribution here is to notice that these proofs, and the resulting schemes, can be abstracted in terms of linear hash functions. In particular, for threshold signatures, as in [BCK+22], we consider an abstract setting with an ideal distributed key generation, and we target the security notions of TS-SUF-2 and TS-SUF-3, which were shown to be achieved by two variants of FROST, both of which we model here abstractly. Since we are targeting feasibility, we are less concerned with the concrete round complexity of distributed key generation and could use any secure multi-party computation protocol for this task.

In contrast, the rough intuition behind a proof of the latter theorem is that for any execution of a (wlog deterministic) adversary $\mathcal{A}$ playing the AOMPR game with challenges $\boldsymbol{X} = \mathsf{F}(\boldsymbol{x})$, since

F is not a monomorphism, there exists another execution with challenges $\boldsymbol{X} = \mathsf{F}(\boldsymbol{x}')$ such that $\boldsymbol{x} \neq \boldsymbol{x}'$, but the views of $\mathcal{A}$ are identical in the two executions. Then, if $\mathcal{A}$ wins the game given $\boldsymbol{x}$ by outputting $\boldsymbol{y}$ such that $\mathsf{F}(\boldsymbol{y}) = \boldsymbol{X}$, $\mathcal{A}$ also wins the game given $\boldsymbol{x}'$ by outputting $\boldsymbol{y}$. Therefore, we have $\mathsf{F}(\boldsymbol{x}) = \mathsf{F}(\boldsymbol{y}) = \mathsf{F}(\boldsymbol{x}') \ \wedge \ (\boldsymbol{x} \neq \boldsymbol{y} \ \vee \ \boldsymbol{x}' \neq \boldsymbol{y})$, which implies that we can find a collision in at least one of the executions. Indeed, special cases of this technique already underlie several works, including Okamoto's [Oka93], but our main challenges are to prove the concrete mapping of $\boldsymbol{x}'$ from $\boldsymbol{x}$ and to package this in terms of the AOMPR abstraction.

1.1 DL-based Instantiations

To obtain an instantiation of FROST-H and MuSig2-H based on the hardness of the discrete logarithm (DL) problem, we can use the Pedersen linear hash function [Ped92]

$$\mathsf{F}(x_1, x_2) = g^{x_1} Z^{x_2} \ ,$$

which is well known to be collision-resistant under the hardness of DL whenever g, Z are generators of a group with prime size p. While MuSig2 and FROST produce valid Schnorr signatures [Sch90], the signatures produced by our DL-based instantiations of FROST-H and MuSig2-H are slightly less efficient, and effectively compatible with Okamoto's signatures [Oka93]. Here, as in Schnorr signatures, the secret signing key is $x \in \mathbb{Z}_p$, and the public verification key $\mathsf{pk} = g^x$, and a signature for a message $m \in \{0,1\}^*$ has format

$$\sigma = (R = g^a Z^b, a + \mathrm{H}(\mathsf{pk}, m, R) \cdot x, b) \ ,$$

where H is a hash function that is modeled as a random oracle in our proofs. To verify a signature (R, a, b), we check that $g^a Z^b = R \cdot \mathsf{pk}^{\mathrm{H}(\mathsf{pk}, M, R)}$. The only difference from Okamoto's scheme [Oka93] is that the latter uses a secret key $(x_1, x_2) \in \mathbb{Z}_p^2$, and a signature has form $(R = g^a Z^b, a + c \cdot x_1, b + c \cdot x_2)$, where $c = \mathrm{H}(\mathsf{pk}, m, R)$, i.e., here, we restrict the scheme to the case where $(x_1, x_2) = (x, 0)$. This optimization is generic and could have been applied to Okamoto's scheme directly; however, it is particularly advantageous for threshold signatures since it lets us leverage any distributed key generation protocol for Schnorr signatures. Here, we need a trusted setup to generate Z as a random group element independent of g, but we note that this is a minimal setup since it can be made transparent, e.g., g, Z can be generated as outputs of a hash function.

Related work (DL). Our DL-based threshold signatures are the first two-round scheme with security proved based solely on the discrete logarithm assumption in the ROM. The most efficient protocol is FROST [KG20, BCK+22], which is slightly more efficient than our scheme since it generates plain Schnorr signatures; however, FROST relies on the stronger OMDL assumption. Though schemes based solely on the discrete logarithm assumption exist [SS01, GJKR07, Lin22], they use more rounds. We stress that not all schemes achieve the same security goals, and here we target the notions of [BCK+22], whereas Lindell [Lin22] targets UC security.

Our DL-based scheme gives the first partially non-interactive two-round multi-signatures based on plain DL and the ROM. It is almost as efficient as MuSig2 [NRS21], which is based on OMDL. Drijvers *et al.* [DEF+19] proposed a less efficient two-round scheme, called mBCJ, based on DL and ROM only, and it repairs a prior scheme by Bagherzandi, Cheon, and Jarecki [BCJ08]. mBCJ signatures, less efficient than ours, consist of two group elements and three scalars, and public keys also consist of one group element and two scalars. Moreover, mBCJ is not partially non-interactive

(i.e., the first round does depend on the message being signed). Another option is the MuSig-DN scheme [NRSW20], but it relies on heavy machinery from zero-knowledge proofs.

A more efficient DL-based alternative is the HBMS scheme by Bellare and Dai [BD21], but HBMS is not partially non-interactive. Further, our security reduction is tighter than that of HBMS. Most relevant to us, Lee and Kim [LK22] gave a multi-signature scheme based on Okamoto signatures that, however, is proved secure only in the AGM [FKL18]; their signing is also not partially non-interactive.

More recently, Pan and Wagner [PW23] proposed a two-round multi-signature scheme based only on the Decisional Diffie-Hellman (DDH) assumption with a tight reduction, but their scheme is also not partially non-interactive.

Finally, the work of Drijvers *et al.* [DEF+19], as well as recent ROS attacks [BLL+21], also surfaced several security issues in earlier DL-based proposals that we do not discuss here.

1.2 RSA-based instantiation

The situation with RSA is slightly more complex since the above framework, *as is*, does not appear to support an RSA instantiation directly: no natural RSA-based linear hash function realizes an appropriate $\mathcal{S}$-module where $\mathcal{S}$ is a field, which is of critical importance for our constructions and proofs of theorems. However, we show that the framework can be adapted to support the RSA-based linear hash function

$$\mathsf{F}(x_1, x_2) = x_1^e w^{x_2} \ ,$$

based on public parameters $par = (N, e, w)$, where N is an RSA modulus, $e \in \mathbb{Z}_N^*$ is a prime such that $\gcd(e, \phi(N)) = 1$ and $w \in \mathbb{Z}_N^*$. We refer to this linear hash function as RLHF. Here, it is important to note that the supported scalar space is set to $\mathcal{S} := \mathbb{Z}$, which is only a ring. (We refer to such hash functions as *weak* linear hash functions.)

RSA-SPECIFIC CHALLENGES. We now describe the problems caused by the lack of inversion in $\mathcal{S}$, and briefly explain how we fix them for the specific case of RLHF. We stress that these fixes are very ad-hoc for RSA, and *do not work* in general for weak linear hash functions.

- FROST-H generates signing keys using Shamir secret sharing [Sha79], which requires the scalar space to be a field in order to compute the Lagrange coefficients. This is a common problem for RSA-based threshold schemes [Sho00, DK01], and we address it via the standard trick of multiplying the Lagrange coefficients with a large number to make them integers.
- One place in the proof of our first informal theorem above (reducing the security of MuSig2-H and FROST-H to AOMPR) where the scalar space needs to be a field is to invert challenges $\boldsymbol{X} \in \mathcal{R}^n$, given a linear equation $A\boldsymbol{X} = \mathsf{F}(\boldsymbol{b})$, where A in $\mathcal{S}^{n \times n}$, $\boldsymbol{X}$ in $\mathcal{R}^n$, and $\boldsymbol{b}$ in $\mathcal{D}^n$. Since $\mathcal{S}$ is a field in our original proof, we show that A has full rank; thus, one can compute $\boldsymbol{x}$ such that $\boldsymbol{X} = \mathsf{F}(\boldsymbol{x})$ by multiplying the inverse of A on both sides of the equation. Clearly, this fails if $\mathcal{S}$ is not a field. Fortunately, to instantiate RLHF, we find that this equation can be solved efficiently whenever A has full rank modulo e (which, recall, is a prime), and we show this condition holds whenever we need to solve the equation in the proof for the special case of RLHF. In addition, for MuSig2-H, we require one of the prime factors of N to be a safe prime in order to make the reduction go through. We also show how to remove this safe-prime requirement by minimally modifying the key aggregation algorithm.

- For our second informal theorem (reducing AOMPR to the collision-resistance of the linear hash function), we need the scalar space to be a field upon showing that, for any matrix $B \in \mathcal{S}^{\ell \times q}$ for $\ell < q$, there exists $\boldsymbol{u} \in \mathcal{S}^q$ such that
  1. $B\boldsymbol{u} = 0$;
  2. $u_i z^* \neq 0$ for some $i \in [q]$, where z^* is an a prior fixed non-zero element in $\mathcal{D}$ such that $\mathsf{F}(z^*) = 0$.

  Again, if $\mathcal{S}$ is a ring, such an $\boldsymbol{u}$ might not exist. However, for the RSA-based linear hash function, since $\mathcal{S} = \mathbb{Z}$, we can always find a non-zero $\boldsymbol{u} \in \mathbb{Z}^q$ such that $B\boldsymbol{u} = 0$. Showing the second condition involves some technical details of RLHF, but roughly, we need to show that there exists $i \in [q]$ such that $u_i \not\equiv 0 \bmod e$.

Resulting schemes. Our RSA-based instantiations of FROST-H and MuSig2-H produce signatures that also resemble the RSA-based signatures by Okamoto [Oka93]. Given public parameters $par = (N, e, w)$, where $e \in \mathbb{Z}_N^*$ is a prime such that $\gcd(e, \phi(N)) = 1$ and $w \in \mathbb{Z}_N^*$, the secret signing key is $x \in \mathbb{Z}_N^*$, and the public verification key $\mathsf{pk} = x^e$, and a signature for a message $m \in \{0,1\}^*$ has format

$$\sigma = (R = a^e w^b, a \cdot x^{\mathrm{H}(\mathsf{pk}, m, R)}, b) \ .$$

To verify a signature (R, s, b), one checks whether $s^e w^b = R \cdot \mathsf{pk}^{\mathrm{H}(\mathsf{pk}, m, R)}$. We give a simpler scheme that assumes that N is the product of safe primes, but we then drop this restriction in a slightly less efficient scheme.

We note that this scheme's drawback is that the public parameters *par* must be generated honestly. In the multi-signature case, this requires a trusted setup, whereas in the threshold signature case, *par* could be generated as part of the distributed key generation process. An important open question is whether we can remove a trusted setup, but we note that no better construction without a trusted setup is known, as we discuss next. Another unusual aspect of our use of the RSA assumption is that we require e to be large and prime, but this does not appear to weaken the assumption in any way.

Related work (RSA). Threshold signatures based on RSA go back to the work of Shoup [Sho00], whose scheme is more efficient than ours since it is round optimal. Shoup's basic scheme guarantees only the inability to come up with a signature for messages for which no party has issued a signature share. A stronger notion would require that the only way to issue a valid signature is for sufficiently many honest parties to contribute, i.e., if k signature shares are needed for a valid signature to be created, and t parties can be corrupted, no valid signature should be generated *unless* at least $k - t$ parties create shares. (This notion is referred to as TS-UF-1 in [BCK+22, BTZ22].) To achieve this stronger notion, Shoup [Sho00] modifies the scheme and relies on a variant of the DDH assumption, which we do not need here. All previous works on RSA-based threshold signatures [DDFY94, GJKR96, FMY98, Rab98, DK01, FS01, ADN06, GHKR08] do not consider this stronger security goal, although some of these works consider properties such as proactivity [Rab98, ADN06], robustness [GJKR96, FMY98, Rab98], removing trusted dealers [DK01, FS01], and adaptive-security [ADN06], which we do not consider.

Our RSA-based instantiation of MuSig2-H improves upon the state-of-the-art even further. Indeed, only a few works on RSA multi-signatures, e.g., [DF92, AA05], support fully non-interactive signing, but they all assume a trusted third party that distributes *all* signing keys and that the number of signers is fixed. Others [Ita83a, Oka88, HK89, KH90, Oka93, PPKW97, MO+00, MM00, PLL02] support only sequential signing, i.e., all signers engage in the signing process one by one.

Another relevant line of works addresses identity-based multi-signatures [BN07, BJ10] (IBMS). IBMS can be viewed as multi-signature schemes where each ID plays the role of the public key for each signer. However, if used as a multi-signature scheme, these schemes require a trusted dealer to generate the keys for each signer. Also, they do not support key aggregation, which our scheme supports.

2 Preliminaries

2.1 Notations

For any positive integers $k < n$, $[n]$ denotes $\{1, \ldots, n\}$, and $[k..n]$ denotes $\{k, \ldots, n\}$. We use κ to denote the security parameter. For a finite set S, $|S|$ denotes the size of S, and $x \leftarrow\!\!\$\, S$ denotes sampling an element uniformly from S and assigning it to x.

2.2 Basic Algebra

MODULES. For any ring R with multiplicative identity 1 and any abelian group $(M, +)$, we say M is an R-module if there exists an operation $\cdot : R \times M \to M$ such that for any $a, b \in R$ and any $x, y \in M$, (i) $a \cdot (x + y) = a \cdot x + a \cdot y$, (ii) $(a + b) \cdot x = a \cdot x + b \cdot x$, (iii) $(ab) \cdot x = a \cdot (b \cdot x)$, (iv) $1 \cdot x = x$. Also, we use 0 to denote the identity of M.

MODULE HOMOMORPHISMS. For any R-modules M and N, a map $f : M \to N$ is a homomorphism of R-modules if for any $r \in R$ and $x, y \in M$, $f(x + r \cdot y) = f(x) + r \cdot f(y)$. We say a homomorphism f is an epimorphism if f is a surjection. We say a homomorphism f is a monomorphism if f is an injection.

CHARACTERISTIC OF A FIELD. For any field $\mathbb{F}$, the characteristic of $\mathbb{F}$, denoted by $\mathrm{char}(\mathbb{F})$, is the smallest positive number k such that $k \cdot \mathbf{1} = \sum_{i=1}^{k} \mathbf{1} = \mathbf{0}$, where $\mathbf{1}$ denotes the multiplicative identity of $\mathbb{F}$ and $\mathbf{0}$ denotes the additive identity of $\mathbb{F}$. If k does not exist, we say the characteristic of F is 0.

3 Algebraic One-more Preimage Resistance

In this section, we first give the definition of linear hash functions, then define collision resistance and algebraic one-more preimage resistance (AOMPR) of a linear hash function family, and finally show AOMPR is implied by collision resistance.

3.1 Linear Hash Functions

The notion of linear hash functions is introduced in [HKL19, HKLN20], which is in turn adapted from [BBSS18]. We adapt the definition from [HKL19] by additionally requiring the scalar set $\mathcal{S}$ to be a field and $\mathcal{D}$ and $\mathcal{R}$ to be $\mathcal{S}$-modules, which is necessary for the reduction from collision resistance to AOMPR and for our constructions in Section 4 to work.

Definition 1. *A linear hash function family* LHF *is a pair of algorithms* $(\mathsf{PGen}, \mathsf{F})$ *such that*

a) PGen *is a randomized algorithm that takes as input the security parameter* 1^κ *and returns the system parameter* par *that defines three sets* $\mathcal{S} = \mathcal{S}(par), \mathcal{D} = \mathcal{D}(par)$ *and* $\mathcal{R} = \mathcal{R}(par)$*, where* $\mathcal{S}$ *is a field, and* $\mathcal{D}$ *and* $\mathcal{R}$ *are* $\mathcal{S}$*-modules. Moreover, we require* $|\mathcal{S}| \geqslant 2^\kappa, |\mathcal{D}| \geqslant 2^\kappa$*, and* $|\mathcal{R}| \geqslant 2^\kappa$*.*

Game $\mathrm{CR}^{\mathcal{A}}_{\mathsf{LHF}}(\kappa)$:
$par \leftarrow \mathsf{PGen}(1^\kappa)$
$(x_1, x_2) \leftarrow\!\!\$\ \mathcal{A}(par)$
If $x_1 \neq x_2$ and $\mathsf{F}(x_1) = \mathsf{F}(x_2)$ then
  Return 1
Return 0

Fig. 1. The CR security game for a linear hash family $\mathsf{LHF} = (\mathsf{PGen}, \mathsf{F})$.

Game $\mathrm{AOMPR}^{\mathcal{A}}_{\mathsf{LHF}}(\kappa)$:
$par \leftarrow\!\!\$\ \mathsf{PGen}(1^\kappa)$
$\mathrm{cid} \leftarrow 0$; $\ell \leftarrow 0$
$\{y_i\}_{i\in[\mathrm{cid}]} \leftarrow \mathcal{A}^{\textsc{Chal},\mathrm{PI}}(par)$
If $\ell \geqslant \mathrm{cid}$ then return 0
If $\forall\ i \in [\mathrm{cid}]\ \mathsf{F}(y_i) = X_i$ then
  Return 1
Return 0

Oracle $\textsc{Chal}()$:
$\mathrm{cid} \leftarrow \mathrm{cid} + 1$
$x_{\mathrm{cid}} \leftarrow\!\!\$\ \mathcal{D}$; $X_{\mathrm{cid}} \leftarrow \mathsf{F}(x_{\mathrm{cid}})$
Return X_{cid}

Oracle $\mathrm{PI}(Y, \alpha, \{\beta_i\}_{i\in[\mathrm{cid}]})$:
Require: $Y = \mathsf{F}(\alpha) + \sum_{i\in[\mathrm{cid}]} \beta_i X_i$
$\ell \leftarrow \ell + 1$
Return $\alpha + \sum_{i\in[\mathrm{cid}]} \beta_i x_i$

Fig. 2. The AOMPR game for a linear hash function family $\mathsf{LHF} = (\mathsf{PGen}, \mathsf{F})$. For the inputs of PI, X is in $\mathcal{R}$, α is in $\mathcal{D}$, and each β_i is in $\mathcal{S}$.

b) F *is a deterministic function that takes as input the system parameter par and an element* $x \in \mathcal{D}$ *and returns an element in* $\mathcal{R}$ *such that* $\mathsf{F}(par, \cdot) : \mathcal{D} \to \mathcal{R}$ *is a* epimorphism *of* $\mathcal{S}$*-modules. Moreover,* F *is* not a monomorphism*, which is equivalent to there exists* $z^* \in \mathcal{D}$ *such that* $z^* \neq 0$ *and* $\mathsf{F}(par, z^*) = 0$*. For simplicity, we omit par from the input of* F *from now on.*

Collision Resistance. Collision resistance of linear hash functions is analogous to collision resistance of cryptographic hash functions, which ensures that it is hard to find two distinct inputs that map to the same output. The $\mathrm{CR}^{\mathcal{A}}_{\mathsf{LHF}}$ game is defined in Figure 1. The corresponding advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{LHF}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\mathrm{CR}^{\mathcal{A}}_{\mathsf{LHF}} = 1\right]$.

3.2 Algebraic One-more Preimage Resistance

We introduce the notion of algebraic one-more preimage resistance (AOMPR) for linear hash functions, which is formally defined via the game $\mathrm{AOMPR}^{\mathcal{A}}_{\mathsf{LHF}}$, as described in Figure 2. It guarantees that any adversary given a description of a linear hash function $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$ cannot invert $q+1$ challenges $X_1, \ldots, X_{q+1}$, where $X_i = \mathsf{F}(x_i)$ for $x_i \leftarrow\!\!\$\ \mathcal{D}$, by making at most q queries to the PI oracle that, on any input $Y \in \mathcal{R}$ that is an affine combination of the challenges, outputs an element in the preimage of Y. It is syntactically analogous to the algebraic one-more discrete logarithm (AOMDL) problem [NRS21], where the adversary wants to compute the discrete logarithms of $q+1$ random challenges in $\mathbb{G}$ by making at most q queries to the DLog oracle, which outputs the discrete logarithm of the input Y only when Y is an affine combination of the challenges and the combination is known to the adversary.

The following theorem, our main result on AOMPR, shows that AOMPR of a linear hash function family is implied by its collision resistance.

Theorem 1. *For any linear hash function family* LHF *and any* AOMPR *adversary* $\mathcal{A}$ *making at most* q *queries to* CHAL*, there exists an adversary* $\mathcal{B}$ *for the* $\mathrm{CR}^{\mathsf{LHF}}$ *game running in a similar running time as* $\mathcal{A}$ *such that* $\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa) \leqslant 2\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{LHF}}(\mathcal{B},\kappa)$.

Proof (of Theorem 1). Given an adversary $\mathcal{A}$ for the $\mathrm{AOMPR}^{\mathsf{LHF}}$ game, without loss of generality, we assume that $\mathcal{A}$ is deterministic, queries CHAL exactly q times, and queries PI exactly $\mathsf{q}-1$ times. The construction of $\mathcal{B}$ is straightforward. After receiving *par* from the $\mathrm{CR}^{\mathsf{LHF}}$ game, $\mathcal{B}$ runs $\mathcal{A}$ on input *par* by simulating the oracles CHAL and PI exactly the same as in the $\mathrm{AOMPR}^{\mathsf{LHF}}$ game. After $\mathcal{A}$ outputs $\{y_i\}_{i\in[\mathsf{q}]}$, if

$$\exists\, i \in [\mathsf{q}] \text{ such that } \mathsf{F}(y_i) = X_i \text{ and } y_i \neq x_i, \tag{1}$$

where x_i and X_i are generated in the oracle CHAL, then $\mathcal{B}$ outputs (x_i, y_i). Otherwise, $\mathcal{B}$ aborts.

ANALYSIS OF $\mathcal{B}$. Denote the event $\mathsf{WIN}_{\mathcal{B}}$ as after $\mathcal{A}$ returns, the condition (1) holds. If $\mathsf{WIN}_{\mathcal{B}}$ occurs, $\mathcal{B}$ wins the $\mathrm{CR}^{\mathsf{LHF}}$ game since $\mathsf{F}(x_i) = X_i = \mathsf{F}(y_i)$, which implies $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{LHF}}(\mathcal{B},\kappa) = \mathsf{Pr}\left[\mathsf{WIN}_{\mathcal{B}}\right]$.

It is left to show that $\mathsf{Pr}\left[\mathsf{WIN}_{\mathcal{B}}\right] \geqslant \frac{1}{2}\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa)$. Since $\mathcal{A}$ is deterministic, the execution of $\mathcal{A}$ is fixed given the pair $(par, \boldsymbol{x})$, where $\boldsymbol{x} \in \mathcal{D}^{\mathsf{q}}$ denotes the randomness generated in the oracle CHAL. Denote the event $\mathsf{WIN}_{\mathcal{A}}$ as $\mathcal{A}$ wins the $\mathrm{AOMPR}^{\mathsf{LHF}}$ game simulated by $\mathcal{B}$. Since $\mathcal{B}$ simulate the game perfectly, we know $\mathsf{Pr}[\mathsf{WIN}_{\mathcal{A}}] = \mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa)$. For each *par*, denote

$$\mathcal{W}_{\mathcal{A}} := \{\boldsymbol{x} \mid \mathsf{WIN}_{\mathcal{A}} \text{ occurs given } (par, \boldsymbol{x})\} ,$$

$$\mathcal{W}_{\mathcal{B}} := \{\boldsymbol{x} \mid \mathsf{WIN}_{\mathcal{B}} \text{ occurs given } (par, \boldsymbol{x})\} .$$

Claim 1 *For each par, there exists a* bijection $\Phi : \mathcal{W}_{\mathcal{A}} \to \mathcal{W}_{\mathcal{A}}$ *such that for any* $\boldsymbol{x} \in \mathcal{W}_{\mathcal{A}}$*, we have* $\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}} \ \vee\ \Phi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}$.

From the above claim, we can conclude the proof since

$$\begin{aligned}
\mathsf{Pr}\left[\mathsf{WIN}_{\mathcal{B}}\right] = \mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}}] &= \frac{1}{2}\left(\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}}] + \mathsf{Pr}[\Phi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}]\right) \\
&\geqslant \frac{1}{2}\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}} \ \vee\ \Phi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}] \geqslant \frac{1}{2}\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{A}}] \\
&= \frac{1}{2}\mathsf{Pr}[\mathsf{WIN}_{\mathcal{A}}] = \frac{1}{2}\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa) .
\end{aligned}$$

□

Proof (of Claim 1). We construct Φ as follows. For each $\boldsymbol{x} \in \mathcal{W}_{\mathcal{A}}$, consider the execution of $\mathcal{A}$ given $(par, \boldsymbol{x})$. Denote $B \in \mathcal{S}^{(\mathsf{q}-1)\times\mathsf{q}}$ as the query matrix of the execution, which is defined as follows.

Definition 2. *Given an execution of an adversary* $\mathcal{A}$ *for the* AOMPR *game, where* $\mathcal{A}$ *makes* q *queries to* CHAL *and* ℓ *queries to* PI*, define the* query matrix *of the execution as* $B \in \mathcal{S}^{\ell\times q}$ *such that*

$$B_{i,j} = \begin{cases} \beta_i^{(j)} , & i \in [\mathrm{cid}^{(j)}] \\ 0 , & o.w. \end{cases} ,$$

where $\beta_i^{(j)}$ *and* $\mathrm{cid}^{(j)}$ *are the values of* β_i *and* cid *when* $\mathcal{A}$ *makes the* j*-th query to* PI.

We now define

$$\Phi(\boldsymbol{x}) := \boldsymbol{x} + \boldsymbol{u}^{(B)} z^* \ ,$$

where $z^* \in \mathcal{D}$ and $\boldsymbol{u}^{(B)} \in \mathcal{S}^{\mathsf{q}}$ are defined in the following claim.

Claim 2 *There exists $z^* \in \mathcal{D}$ such that $\mathsf{F}(z^*) = 0$ and for any matrix $A \in \mathcal{S}^{\ell \times q}$ where $0 < \ell < q$, there exists a vector $\boldsymbol{u}^{(A)} \in \mathcal{S}^q$ and $i \in [q]$ such that*

$$A\boldsymbol{u}^{(A)} = 0 \ \wedge \ \exists\, i \in [\mathsf{q}] \ : \ u_i^{(A)} z^* \neq 0 \ . \tag{2}$$

Proof (of Claim 2). Since F is not a monomorphism from $\mathcal{D}$ to $\mathcal{R}$, there exists a non-zero element $z^* \in \mathcal{D}$ such that $\mathsf{F}(z^*) = 0$. Since $\mathcal{S}$ is a field and A has rank at most $\ell < q$, there exists a non-zero vector $\boldsymbol{u}^{(A)} \in \mathcal{S}^q$ such that $A\boldsymbol{u}^{(A)} = 0$. Also, since $\boldsymbol{u}^{(A)}$ is non-zero, there exists $i \in [q]$ such that $u_i^{(A)} \neq 0$, and since $\mathcal{S}$ is a field and $z^* \neq 0$, we have $u_i^{(A)} z^* \neq 0$.

Analysis of Φ. For simplicity, we use $\boldsymbol{u}$ to denote $\boldsymbol{u}^{(B)}$ in the following analysis. We first show that the executions of $\mathcal{A}$ given $(par, \boldsymbol{x})$ and given $(par, \Phi(\boldsymbol{x}))$ are identical. Since $\mathsf{F}(\Phi(\boldsymbol{x})) = \mathsf{F}(\boldsymbol{x}) + \boldsymbol{u} \cdot \mathsf{F}(z^*) = \mathsf{F}(\boldsymbol{x}) + \boldsymbol{u} \cdot 0 = \mathsf{F}(\boldsymbol{x})$, the challenges output by Chal are the same in the two executions. For the j-th query to PI, suppose the prior views of $\mathcal{A}$ are identical. Then, $\mathcal{A}$ must make the same query $\left(X^{(j)}, \alpha^{(j)}, \{\beta_i^{(j)}\}_{i \in [\mathrm{cid}^{(j)}]}\right)$ in both executions. Since $B\boldsymbol{u} = 0$, we have $\alpha^{(j)} + \sum_{i \in [\mathrm{cid}^{(j)}]} \beta_i^{(j)} x_i = \alpha^{(j)} + \left(\boldsymbol{\beta}^{(j)}\right)^T \boldsymbol{x} = \alpha^{(j)} + \left(\boldsymbol{\beta}^{(j)}\right)^T (\boldsymbol{x} + \boldsymbol{u}z^*) = \alpha^{(j)} + \sum_{i \in [\mathrm{cid}^{(j)}]} \beta_i^{(j)} \left(\Phi(\boldsymbol{x})\right)_i$, where $\boldsymbol{\beta}^{(j)}$ denotes the j-th row of B. Therefore, $\mathcal{A}$ receives the same value from PI in both executions. By induction, the views of $\mathcal{A}$ are identical in both executions and thus $\mathcal{A}$ outputs the same values in both executions, which implies $\Phi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{A}}$ and thus Φ is a map from $\mathcal{W}_{\mathcal{A}}$ to $\mathcal{W}_{\mathcal{A}}$.

Then, it is not hard to see that $\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}} \ \vee \ \Phi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}$. Since the executions of $\mathcal{A}$ given $\boldsymbol{x}$ and $\Phi(\boldsymbol{x})$ are identical, the outputs $y_1, \ldots, y_{\mathsf{q}}$ of $\mathcal{A}$ are also identical in the two executions. Since there exists $i \in [\mathsf{q}]$ such that $u_i z^* \neq 0$, we have either $y_i \neq x_i$ or $y_i \neq x_i + u_i \cdot z^*$, which means $\mathsf{WIN}_{\mathcal{B}}$ occurs either in the execution given $\boldsymbol{x}$ or $\Phi(\boldsymbol{x})$.

It is left to show that Φ is a bijection. Since both the domain and range of Φ are $\mathcal{W}_{\mathcal{A}}$, which is a finite set, it is enough to show that Φ is an injection. For any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{W}_{\mathcal{A}}$ such that $\Phi(\boldsymbol{x}_1) = \Phi(\boldsymbol{x}_2)$, since the execution of $\mathcal{A}$ given $\boldsymbol{x}_1$ is identical to that given $\Phi(\boldsymbol{x}_1)$ and the execution of $\mathcal{A}$ given $\boldsymbol{x}_2$ is identical to that given $\Phi(\boldsymbol{x}_2)$, we know the executions of $\mathcal{A}$ given $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are identical, which implies the query matrix B in the two executions are identical. Therefore, we have $\Phi(\boldsymbol{x}_1) = \boldsymbol{x}_1 + \boldsymbol{u}z^*$ and $\Phi(\boldsymbol{x}_2) = \boldsymbol{x}_2 + \boldsymbol{u}z^*$ for the same $\boldsymbol{u} \in \mathcal{S}^{\mathsf{q}}$, which implies $\boldsymbol{x}_1 = \boldsymbol{x}_2$. This shows that Φ is an injection. □

4 Schemes Based on Linear Hash Functions

For a cyclic group $\mathbb{G}$ with prime size p and generator g, we can view the description of a linear hash function with description $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$ as an analogue to $(\mathbb{G}, p, g)$, where $\mathcal{R}$ corresponds to the group $\mathbb{G}$, the preimage under the function F corresponds to the discrete logarithm to base g, and $\mathcal{S}$ corresponds to the field of scalar $\mathbb{Z}_p$. Also, the AOMPR game is analogous to the AOMDL game. This suggests a general way of transforming any scheme that is secure under the AOMDL assumption into a scheme that is constructed from linear hash functions and is secure under the AOMPR assumption. In this section, we discuss how this idea is applied to two specific examples: MuSig2 [NRS21], a multi-signature scheme, and FROST [KG20], a threshold signature scheme.

4.1 Multi-Signatures

MuSig2 [NRS21] is a two-round multi-signature scheme with key aggregation. Moreover, the first signing round is message-independent. We first give the syntax and security definition of two-round multi-signatures following [NRS21], then present a new scheme MuSig2-H based on LHF that is transformed from MuSig2, and finally show the security of the new scheme under the AOMPR assumption.

Syntax. A two-round multi-signature scheme with key aggregation is a tuple of efficient (randomized) algorithms $\mathsf{MS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyAgg}, \mathsf{PreSign}, \mathsf{PreAgg}, \mathsf{Sign}, \mathsf{SignAgg}, \mathsf{Ver})$ that behave as follows. The setup algorithm $\mathsf{Setup}(1^\kappa)$ returns a system parameter par, and we assume par is given to all other algorithms implicitly. The key generation algorithm $\mathsf{KeyGen}()$ returns a pair of secret and public keys $(\mathsf{sk}, \mathsf{pk})$. The (deterministic) key aggregation algorithm KeyAgg takes as input a multiset of public keys L with size at most 2^κ and returns an aggregate public key apk. For n signers, where the i-th signer has key-pair $(\mathsf{sk}_i, \mathsf{pk}_i)$, the signing protocol between them and an aggregator node to sign a message $m \in \{0,1\}^*$ is defined by the following experiment:

$$\begin{aligned}
&(pp_i, \mathsf{st}_i) \leftarrow \mathsf{PreSign}() \ , \ \text{for each } i \in SS \ , \\
&app \leftarrow \mathsf{PreAgg}(\{pp_1, \ldots, pp_n\}) \ , \\
&(out_i, \mathsf{st}_i) \leftarrow \mathsf{Sign}(\mathsf{st}_i, app, \mathsf{sk}_i, \mathsf{pk}_i, m, \{\mathsf{pk}_j\}_{j \in [n] \setminus \{i\}}) \ , \ \text{for each } i \in SS \ , \\
&\sigma \leftarrow \mathsf{SignAgg}(\{out_1, \ldots, out_n\}) \ ,
\end{aligned} \tag{3}$$

where each signer runs the algorithms PreSign and Sign; the aggregator node runs the algorithms PreAgg and SignAgg and outputs the signature σ. The aggregator node can be one of the signers and is untrusted in our security model. The (deterministic) verification algorithm $\mathsf{Ver}(\mathsf{apk}, m, \sigma)$ outputs a bit that indicates whether or not σ is valid for apk and m or not. We say that MS is (perfectly) *correct* if, for any $m \in \{0,1\}^*$, $\mathsf{Pr}[\mathsf{Ver}(\mathsf{KeyAgg}(\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}), m, \sigma)] = 1$, where σ is generated in the experiment in (3) and the probability is taken over the sampling of the system parameter par, all key-pairs $\{(\mathsf{sk}_i, \mathsf{pk}_i)\}_{i \in [n]}$.

Security. The security notion of multi-signatures considered in the prior work [NRS21] is referred to as MS-UF-CMA, which guarantees that it is not possible to forge a valid multi-signature that involves at least one honest party. The MS-UF-CMA game for a multi-signature scheme MS is defined in Figure 3, where MS.HF denotes the space of the hash functions used in MS from which the random oracle is drawn. In the game, we assume the adversary corrupts the aggregator node and all signers except one and can engage in any number of (concurrent) signing sessions with the honest party. The corresponding advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MS}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\text{MS-UF-CMA}^{\mathcal{A}}_{\mathsf{MS}}(\kappa) = 1\right]$.

Our Scheme. Figure 4 shows the scheme MuSig2-H, which is transformed from MuSig2 [NRS21] with the parameter $\nu = 4$, where ν denotes the number of nonces generated in the first round of the signing protocol. In addition to the general transformation, we do two optimizations to MuSig2-H. First, in $\mathsf{KeyGen}()$, the secret key sk is not sampled from $\mathcal{D}$ but from a subset $\mathcal{D}_{\text{key}} \subseteq \mathcal{D}$ such that F is a bijection from $\mathcal{D}_{\text{key}}$ to $\mathcal{R}$. It can reduce the size of the secret key to the size of the public key. Also, the range of each hash function is set to $\mathcal{S}_{\text{hash}}$ instead of $\mathcal{S}$, where $\mathcal{S}_{\text{hash}}$ is an arbitrary subset of $\mathcal{S}$ with size at least 2^κ. Further, we require the characteristic of the field $\mathcal{S}$ to be at least 2^κ.

The original paper shows the unforgeability of MuSig2 under the AOMDL assumption. Analogous to that, the following theorem shows that the security of MuSig2-H[LHF] is implied by AOMPR of the underlying linear hash function family LHF in the random oracle model.

Game MS-UF-CMA$^{\mathcal{A}}_{\mathsf{MS}}(\kappa)$:

$par \leftarrow \mathsf{Setup}(1^\kappa)$
$\mathrm{H} \leftarrow\!\!\$\ \mathsf{MS.HF}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow\!\!\$\ \mathsf{KeyGen}()$
$\mathrm{sid} \leftarrow 0$
$S \leftarrow \varnothing\ ;\ S' \leftarrow \varnothing\ ;\ Q \leftarrow \varnothing$
$(L, m, \sigma) \leftarrow \mathcal{A}^{\textsc{Sign},\textsc{Sign}',\mathrm{RO}}(par, \mathsf{pk})$
If $\mathsf{pk} \notin L\ \wedge\ (L, m) \notin Q$
  $\wedge\ \mathsf{Ver}(\mathsf{KeyAgg}(L), m, \sigma) = 1$ then
  Return 1
Return 0

Oracle PreSign() :

$\mathrm{sid} \leftarrow \mathrm{sid} + 1\ ;\ S \leftarrow S \cup \{\mathrm{sid}\}$
$(pp, \mathsf{st}^{(\mathrm{sid})}) \leftarrow \mathsf{PreSign}()$
Return pp

Oracle Sign(k, app, m, L) :

If $k \notin S$ then return $\perp$
$out \leftarrow \mathsf{Sign}(\mathsf{st}^{(k)}, app, \mathsf{sk}, m, L)$
$L \leftarrow L \cup \{\mathsf{pk}\}$
$Q \leftarrow Q \cup \{(L, m)\}$
$S \leftarrow S \backslash \{k\}\ ;\ S' \leftarrow S' \cup \{k\}$
Return out

Oracle RO(x) :

Return $\mathrm{H}(x)$

Fig. 3. The MS-UF-CMA game for a mutil signature scheme MS.

$\mathsf{Setup}(1^\kappa)$:

$par \leftarrow \mathsf{PGen}(1^\kappa)$
Return par

$\mathsf{KeyGen}()$:

$\mathsf{sk} \leftarrow\!\!\$\ \mathcal{D}_{\mathrm{key}}\ ;\ \mathsf{pk} \leftarrow \mathsf{F}(sk)$
Return $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{KeyAgg}(L)$:

$\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\} \leftarrow L$
For $i \in [n]$ do
  $a_i \leftarrow \mathrm{H}_{\mathrm{agg}}(L, \mathsf{pk}_i)$
Return $\mathsf{apk} \leftarrow \sum_{i\in[n]} a_i \mathsf{pk}_i$

$\mathsf{Ver}(\mathsf{apk}, m, \sigma)$:

$c \leftarrow \mathrm{H}_{\mathrm{sig}}(\mathsf{apk}, R, m)\ ;\ (R, s) \leftarrow \sigma$
If $\mathsf{F}(s) = R + c\mathsf{apk}$ then return 1
Return 0

$\mathsf{PreSign}()$:

For $j \in [4]$ do
  $r_j \leftarrow\!\!\$\ \mathcal{D}\ ;\ R_j \leftarrow \mathsf{F}(r_j)$
$pp \leftarrow (R_1, \ldots, R_4)$
$\mathsf{st} \leftarrow (r_1, \ldots, r_4)$
Return (pp, st)

$\mathsf{PreAgg}(\{pp_1, \ldots, pp_n\})$:

For $i \in [n]$ do
  $(R_{i,1}, \ldots, R_{i,4}) \leftarrow pp_i$
For $j \in [4]$ do
  $R_j \leftarrow \sum_{i\in[n]} R_{i,j}$
Return $app \leftarrow (R_1, \ldots, R_4)$

$\mathsf{Sign}(\mathsf{st}, app, \mathsf{sk}, \mathsf{pk}, m, L)$:

$(r_1, \ldots, r_4) \leftarrow \mathsf{st}$
$L \leftarrow L \cup \{\mathsf{pk}\}$
$\mathsf{apk} \leftarrow \mathsf{KeyAgg}(L)$
$a \leftarrow \mathrm{H}_{\mathrm{agg}}(L, \mathsf{pk})$
$(R_1, \ldots, R_4) \leftarrow app$
$b \leftarrow \mathrm{H}_{\mathrm{non}}(\mathsf{apk}, (R_1, \ldots, R_4), m)$
$R \leftarrow \sum_{j\in[4]} b^{j-1} R_j$
$c \leftarrow \mathrm{H}_{\mathrm{sig}}(\mathsf{apk}, R, m)$
$s \leftarrow \sum_{j\in[4]} b^{j-1} r_j + ca \cdot \mathsf{sk}$
Return $out \leftarrow (R, s)$

$\mathsf{SignAgg}(\{out_1, \ldots, out_n\})$:

$(R, s) \leftarrow out_1$
For $i \in [2..n]$ do
  $(R_i, s_i) \leftarrow out_i$
  If $R_i \neq R$ then return $\perp$
  $s \leftarrow s + s_i$
Return $\sigma \leftarrow (R, s)$

Fig. 4. The multi-signature scheme $\mathsf{MuSig2\text{-}H[LHF]}$, where $\mathsf{LHF} = (\mathsf{PGen}, \mathsf{F})$ is a linear hash function family. We assume $n \leqslant 2^\kappa$ and $|L| \leqslant 2^\kappa$. $\mathcal{D}_{\mathrm{key}}$ is a subset of $\mathcal{D}$ such that F is a bijection from $\mathcal{D}_{\mathrm{key}}$ to $\mathcal{R}$. Further, $\mathrm{H}_{\mathrm{agg}}(\cdot) := \mathrm{H}(1, \cdot)$, $\mathrm{H}_{\mathrm{non}}(\cdot) := \mathrm{H}(2, \cdot)$, $\mathrm{H}_{\mathrm{sig}}(\cdot) := \mathrm{H}(3, \cdot)$, where $\mathrm{H} : \{0,1\}^* \to \mathcal{S}_{\mathrm{hash}}$, $\mathcal{S}_{\mathrm{hash}} \subseteq \mathcal{S}$, and $|\mathcal{S}_{\mathrm{hash}}| \geqslant 2^\kappa$. Moreover, we require $\mathrm{char}(\mathcal{S}) \geqslant 2^\kappa$.

Theorem 2. *For any* MS-UF-CMA *adversary* $\mathcal{A}$ *making at most* q_s *queries to* PreSign *and* q_h *queries to* RO*, there exists an* AOMPR *adversary* $\mathcal{B}$ *making at most* $4\mathsf{q}_s + 1$ *queries to* Chal

$\mathsf{Fork}^{\mathcal{A}}(x, v_1, v'_1, \ldots, v_{q'}, v'_{q'})$:

Pick the random coin ρ of $\mathcal{A}$ at random
$h_1, h'_1, \ldots, h_q, h'_q \leftarrow H$
$(I, J, \mathrm{Out}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q, v_1, \ldots, v_{q'}; \rho)$
If $I = \perp$ or $J = \perp$ then return $\perp$
$(I', J', \mathrm{Out}') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q, v_1 \ldots, v_{J-1}, v'_J, \ldots, v'_{q'}; \rho)$
If $I \neq I'$ or $h_I = h'_I$ then return $\perp$
Return $(I, \mathrm{Out}, \mathrm{Out}')$

Fig. 5. The forking algorithm built from $\mathcal{A}$ for Lemma 1.

running in time roughly four times that of $\mathcal{A}$ such that

$$\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}, \kappa) \leqslant \sqrt[4]{\mathsf{q}^3 \cdot \mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}, \kappa) + (16\mathsf{q}^2 + 15)/2^{\kappa}} \,,$$

where $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.

We prove the above theorem using the same techniques as used in the proof of MuSig2 [NRS21] to construct $\mathcal{B}$ given an adversary $\mathcal{A}$. Here, we briefly highlight the differences:

- We need to show that $\mathcal{B}$ simulates the MS-UF-CMA$^{\mathsf{MuSig2\text{-}H[LHF]}}$ game perfectly when no bad event occurs and that the bad events occur with a negligible probability (Claim 3 and Lemma 2) when the secret key is sampled from $\mathcal{D}_{\text{key}}$ instead of $\mathbb{Z}_p$, and the randomness r_j is sampled from $\mathcal{D}$ instead of $\mathbb{Z}_p$.
- We need to show that $\mathcal{B}$ can compute a preimage for each challenge (Claim 4 and Claim 5) instead of the discrete logarithm to the base element. More precisely, the problem can be described as follows. Denote the challenges by $U_1, \ldots, U_\ell \in \mathcal{R}$. After the interaction with $\mathcal{A}$, $\mathcal{B}$ computes a matrix $A \in \mathcal{S}^{\ell \times \ell}$ and a vector $\boldsymbol{b} \in \mathcal{D}^\ell$ such that $A \cdot \boldsymbol{U} = \mathsf{F}(\boldsymbol{b})$, we need to show that A has full rank and thus $\mathcal{B}$ can compute a vector $\boldsymbol{u} = A^{-1}\boldsymbol{b}$ such that $\mathsf{F}(\boldsymbol{u}) = \boldsymbol{U}$.

Before turning to the proof, we first recall the following variant of the forking lemma from [NRS21] that will be used in the proof.

Lemma 1. *Let $q, q' \geqslant 1$ be integers and H, V be two sets. Let $\mathcal{A}$ be a randomized algorithm that, on input $x, h_1, \ldots, h_q, v_1, \ldots, v_{q'}$, outputs a tuple (I, J, Out), where $I \in \{\perp\} \cup [q]$, $J \in \{\perp\} \cup [q'+1]$ and* Out *is a side output. Let* IG *be a randomized algorithm that generates x. The accepting probability of $\mathcal{A}$ is defined as* $\mathrm{acc}(\mathcal{A}) = \mathsf{Pr}[(I, J, \mathrm{Out}) \leftarrow\!\!\$\ \mathcal{A}(x, h_1, \ldots, h_q, v_1, \ldots, v_{q'}) \ : \ I \neq \perp \ \wedge \ J \neq \perp]$ *, where the probability is over $x \leftarrow\!\!\$\ \mathsf{IG}, h_1, \ldots, h_q \leftarrow\!\!\$\ H, v_1, \ldots, v_{q'} \leftarrow\!\!\$\ V$ and the random coins of $\mathcal{A}$. Consider algorithm $\mathsf{Fork}^{\mathcal{A}}$ described in Figure 5. The accepting probability of $\mathsf{Fork}^{\mathcal{A}}$ is defined as*

$$\mathrm{acc}(\mathsf{Fork}^{\mathcal{A}}) = \mathsf{Pr}[\alpha \leftarrow\!\!\$\ \mathsf{Fork}^{\mathcal{A}}(x, v_1, v'_1, \ldots, v_{q'}, v'_{q'}) \ : \ \alpha \neq \perp] \,,$$

where the probability is over $x \leftarrow\!\!\$\ \mathsf{IG}, v_1, v'_1, \ldots, v_{q'}, v'_{q'} \leftarrow\!\!\$\ V$. Then,

$$\mathrm{acc}(\mathsf{Fork}^{\mathcal{A}}) \geqslant \mathrm{acc}(\mathcal{A}) \left(\frac{\mathrm{acc}(\mathcal{A})}{q} - \frac{1}{H} \right) .$$

Proof (of Theorem 2). Let $\mathcal{A}$ be an adversary as described in the theorem. Denote the output message-signature pair of $\mathcal{A}$ as $(L^*, m^*, \sigma^* = (R^*, z^*))$. Without loss of generality, we assume $\mathcal{A}$ always queries RO on $\mathrm{H}_{\mathrm{sig}}(\mathsf{apk}^*, m^*, R^*)$ before $\mathcal{A}$ returns, where $\mathsf{apk}^* = \mathsf{KeyAgg}(L^*)$, and always queries RO on $\mathrm{H}_{\mathrm{non}}(\mathsf{apk}, (R_1, \ldots, R_4), m)$ prior to each $\textsc{Sign}(k, (R_1, \ldots, R_4), m, L)$ query, where $\mathsf{apk} = \mathsf{KeyAgg}(L)$. (This adds up to $\mathsf{q}_s + 1$ additional RO queries, and we let $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.)

We first construct an algorithm $\mathcal{C}$ compatible with the syntax in Lemma 1, then construct an algorithm $\mathcal{C}'$ from $\mathsf{Fork}^{\mathcal{C}}$, and finally construct $\mathcal{B}$ from $\mathsf{Fork}^{\mathcal{C}'}$.

THE ADVERSARY $\mathcal{C}$. The input of $\mathcal{C}$ consists of par, which defines a linear hash function $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$, and uniformly random elements $h_1^{(\mathrm{agg})}, \ldots, h_{\mathsf{q}}^{(\mathrm{agg})}, h_1^{(\mathrm{sig})}, \ldots, h_{\mathsf{q}}^{(\mathrm{sig})}, h_1^{(\mathrm{non})}, \ldots, h_{\mathsf{q}}^{(\mathrm{non})} \in \mathcal{S}_{\mathrm{hash}}$. Also, $\mathcal{C}$ can access oracles CHAL and PI, defined the same way as those in the $\mathrm{AOMPR}^{\mathsf{LHF}}$ game. (We can think of this oracle as part of $\mathcal{C}$ in the context of the Forking Lemma.) For simplicity, when $\mathcal{C}$ makes a query $(X, \alpha, \{\beta_i\})$ to PI, we omit the coefficients $\alpha, \{\beta_i\}$ whenever they are clear from the context.

To start with, $\mathcal{C}$ makes $4\mathsf{q}_s+1$ queries to CHAL and denotes the challenges as $X, U_1, \ldots, U_{4\mathsf{q}_s} \in \mathcal{D}$. Then, $\mathcal{C}$ initializes H to an empty table. In addition, it initializes counters ctr_s,$\mathrm{ctr}_{\mathrm{agg}}$, $\mathrm{ctr}_{\mathrm{sig}}$, $\mathrm{ctr}_{\mathrm{non}}$ to 0 and a function dt to an empty table, which are used to record the PI query related to each U_j.

We also use a flag BadKey, initially set to $\mathtt{false}$, to denote whether a bad event occurs. Then, $\mathcal{C}$ sets $\mathsf{pk} \leftarrow X$ and runs $\mathcal{A}(par, \mathsf{pk})$ with access to the oracles $\widetilde{\textsc{PreSign}}, \widetilde{\textsc{Sign}}, \widetilde{\mathrm{RO}}$, which are simulated as follows.

$\widetilde{\mathrm{RO}}$ query $\mathrm{H}_{\mathrm{agg}}(x)$: If $\mathrm{H}_{\mathrm{agg}}(x) \neq \bot$, $\mathcal{C}$ returns $\mathrm{H}_{\mathrm{agg}}(x)$. Otherwise, parse x as $(L, \widetilde{\mathsf{pk}})$. If the parsing fails, or $X \notin L$, $\mathcal{C}$ sets $\mathrm{H}_{\mathrm{agg}}(x) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ and returns $\mathrm{H}_{\mathrm{agg}}(x)$. Otherwise, $\mathcal{C}$ increases $\mathrm{ctr}_{\mathrm{agg}}$ by 1, sets $\mathrm{H}_{\mathrm{agg}}(L, X) \leftarrow h^{(\mathrm{agg})}_{\mathrm{ctr}_{\mathrm{agg}}}$ and $\mathrm{H}_{\mathrm{agg}}(L, \mathsf{pk}') \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ for each $\mathsf{pk}' \in L$ and $\mathsf{pk}' \neq X$. Let $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(L)$. If $\mathsf{apk} \in K$, $\mathcal{B}$ sets $\mathsf{BadKey} \leftarrow \mathtt{true}$. Otherwise, $\mathcal{C}$ sets $K \leftarrow K \cup \{\mathsf{apk}\}$ and returns $\mathrm{H}_{\mathrm{agg}}(x)$.

$\widetilde{\mathrm{RO}}$ query $\mathrm{H}_{\mathrm{non}}(x)$: If $\mathrm{H}_{\mathrm{non}}(x) \neq \bot$, $\mathcal{C}$ returns $\mathrm{H}_{\mathrm{non}}(x)$. Otherwise, parse x as $(\widetilde{\mathsf{apk}}, (R_1, \ldots, R_4), m)$. If the parsing fails, $\mathcal{C}$ sets $\mathrm{H}_{\mathrm{non}}(x) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ and returns $\mathrm{H}_{\mathrm{non}}(x)$. Otherwise, $\mathcal{C}$ increases $\mathrm{ctr}_{\mathrm{non}}$ by 1 and sets $\mathrm{H}_{\mathrm{non}}(x) \leftarrow h^{(\mathrm{non})}_{\mathrm{ctr}_{\mathrm{non}}}$. Also, $\mathcal{C}$ computes $R \leftarrow \sum_{i\in[4]} (h^{(\mathrm{non})}_{\mathrm{ctr}_{\mathrm{non}}})^{j-1} R_j$. If $\mathrm{H}_{\mathrm{sig}}(\widetilde{\mathsf{apk}}, R, m) = \bot$, $\mathcal{C}$ increases $\mathrm{ctr}_{\mathrm{sig}}$ by 1 and sets $\mathrm{H}_{\mathrm{sig}}(\widetilde{\mathsf{apk}}, R, m) = h^{(\mathrm{sig})}_{\mathrm{ctr}_{\mathrm{sig}}}$. Finally, $\mathcal{C}$ returns $\mathrm{H}_{\mathrm{non}}(x)$.

$\widetilde{\mathrm{RO}}$ query $\mathrm{H}_{\mathrm{sig}}(x)$: If $\mathrm{H}_{\mathrm{sig}}(x) \neq \bot$, $\mathcal{C}$ returns $\mathrm{H}_{\mathrm{sig}}(x)$. Otherwise, parse x as $(\widetilde{\mathsf{apk}}, m, R)$. If the parsing fails, $\mathcal{C}$ sets $\mathrm{H}_{\mathrm{sig}}(x) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ and returns $\mathrm{H}_{\mathrm{sig}}(x)$. Otherwise, $\mathcal{C}$ increases $\mathrm{ctr}_{\mathrm{sig}}$ by 1 and sets $\mathrm{H}_{\mathrm{sig}}(x) \leftarrow h^{(\mathrm{sig})}_{\mathrm{ctr}_{\mathrm{sig}}}$. Finally, $\mathcal{C}$ sets $K \leftarrow K \cup \{\widetilde{\mathsf{apk}}\}$ and returns $\mathrm{H}_{\mathrm{sig}}(x)$.

$\widetilde{\mathbf{PreSign}}(i)$ query: Same as in the game MS-UF-CMA$^{\mathsf{MuSig2\text{-}H}}$, except in the simulation of algorithm Sign, $\mathcal{C}$ first increases ctr_s by 1 and sets $R_{1,i} \leftarrow U_{i+4(\mathrm{ctr}_s-1)}$ for $i \in [4]$.

$\widetilde{\mathbf{Sign}}(k, app, m, L)$ query: Same as in the game MS-UF-CMA$^{\mathsf{MuSig2\text{-}H}}$, except in the simulation of algorithm Sign', $\mathcal{C}$ sets $s \leftarrow \mathrm{PI}(\sum_{j\in[4]} b^{j-1} U_{i+4(k-1)} + ca \cdot \mathsf{pk})$, and sets $\mathsf{dt}(k) \leftarrow (b, c, a, s)$.

After receiving the output $(L^*, m^*, \sigma^* = (R^*, s^*))$ from $\mathcal{A}$, $\mathcal{C}$ returns $\bot$ if $\mathsf{BadKey} = \mathtt{true}$ or $\mathcal{A}$ does not win the game. Otherwise, $\mathcal{C}$ computes $\mathsf{apk}^* \leftarrow \mathsf{KeyAgg}(L^*)$ and:

- I_{sig} as the index such that $\mathrm{H}_{\mathrm{sig}}(\mathsf{apk}^*, m^*, R^*)$ is set to $h^{(\mathrm{sig})}_{I_{\mathrm{sig}}}$;
- J_{sig} as the value of $\mathrm{ctr}_{\mathrm{non}}$ when $\mathrm{H}_{\mathrm{sig}}(\mathsf{apk}^*, m^*, R^*)$ is assigned;
- I_{agg} as the index such that $\mathrm{H}_{\mathrm{agg}}(L^*, X)$ is set to $h^{(\mathrm{agg})}_{I_{\mathrm{agg}}}$;
- J_{agg} as the value of $\mathrm{ctr}_{\mathrm{non}}$ when $\mathrm{H}_{\mathrm{agg}}(\mathsf{apk}^*, m^*, R^*)$ is assigned.

Since $\mathcal{A}$ wins the game by our simulation, we know such I_{agg} and I_{sig} must exist. Then, $\mathcal{C}$ returns $(I_{\mathrm{sig}}, J_{\mathrm{sig}}, \mathrm{Out})$, where Out consists of all variables received or generated by $\mathcal{C}$.

ANALYSIS OF $\mathcal{C}$. To use Lemma 1, we define IG as the algorithm that sets $par \leftarrow\!\!\$\ \mathsf{PGen}(1^\kappa)$, uniformly samples $h_1^{(\mathrm{agg})}, \ldots, h_{\mathsf{q}}^{(\mathrm{agg})} \in \mathcal{S}_{\mathrm{hash}}$, and returns $(par, h_1^{(\mathrm{agg})}, \ldots, h_{\mathsf{q}}^{(\mathrm{agg})})$. Also, $(h_1^{(\mathrm{sig})}, \ldots, h_{\mathsf{q}}^{(\mathrm{sig})})$ plays the role of $(h_1, \ldots, h_q)$, and $(h_1^{(\mathrm{non})}, \ldots, h_{\mathsf{q}}^{(\mathrm{non})})$ plays the role of $(v_1, \ldots, v_{q'})$.

We now show that $\mathcal{C}$ simulates the game MS-UF-CMA perfectly. In the real game, sk is uniformly sampled from $\mathcal{D}_{\mathrm{key}}$, and, since F is a bijection from $\mathcal{D}_{\mathrm{key}}$ to $\mathcal{S}$, pk is uniformly distributed over $\mathcal{S}$, which is identical to the simulation. Also, it is clear that the output distributions of each $\widetilde{\mathrm{RO}}$ query and each $\widetilde{\textsc{PreSign}}$ query are identical to those of the real game. For the simulation of $\widetilde{\textsc{Sign}}$, from the MS-UF-CMA game, we know that $\mathcal{C}$ makes at most one query to PI for each session k. Therefore, from the AOMPR game, we know s_1 is uniformly distributed over the preimage of $\sum_{j\in[4]} b^{j-1}U_{i+4(k-1)} + ca_1 \cdot \mathsf{pk}$ given the view of the adversary, which is identical to the real game.

Therefore, since $\mathcal{C}$ simulates the game MS-UF-CMA perfectly, $\mathrm{acc}(\mathcal{C}) \geqslant \mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}) - \mathsf{Pr}[\mathsf{BadKey}]$, where $\mathsf{Pr}[\mathsf{BadKey}]$ is the probability that $\mathsf{BadKey} = \mathtt{true}$ at the end of $\mathcal{C}$'s execution. By the following claim and Lemma 1,

$$\begin{aligned}\mathrm{acc}(\mathsf{Fork}^{\mathcal{C}}) &\geqslant (\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}) - (2\mathsf{q}^2+1)/2^\kappa)^2/\mathsf{q} - \frac{1}{|\mathcal{S}_{\mathrm{hash}}|} \\ &\geqslant \frac{(\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}))^2}{\mathsf{q}} - \frac{4\mathsf{q}+3}{2^\kappa}\,.\end{aligned} \tag{4}$$

Claim 3 $\mathsf{Pr}[\mathsf{BadKey}] \leqslant \frac{2\mathsf{q}^2+1}{2^\kappa}$.

Proof (of Claim 3). Consider a $\widetilde{\mathrm{RO}}$ query $\mathrm{H}_{\mathrm{agg}}(L, \widetilde{\mathsf{pk}})$ from $\mathcal{A}$ such that $X \in L$ and $\mathrm{H}_{\mathrm{agg}}(L, X)$ is not assigned prior to the query. The aggregated key from L can be represented as $\mathsf{apk} = (X)^{t \cdot h^{(\mathrm{agg})}_{\mathrm{ctr}_{\mathrm{agg}}}} Z$, where t is the number of times X appears in L and $Z := \sum_{\mathsf{pk}\in L, \mathsf{pk}\neq X} \mathsf{pk}^{\mathrm{H}_{\mathrm{agg}}(L,\mathsf{pk})}$, which is independent of $h^{(\mathrm{agg})}_{\mathrm{ctr}_{\mathrm{agg}}}$. BadKey is set to true if and only if $\mathsf{apk} \in K$. We use the following lemma, which we show later, to bound the probability that $\mathsf{apk} \in K$.

Lemma 2. *For any $X \in \mathcal{R}$ and any integer t, denote $C(t, X) := \{(ts) \cdot X \mid s \in \mathcal{S}_{\mathrm{hash}}\}$. We say X is Good if and only if $|C(t, X)| = |\mathcal{S}_{\mathrm{hash}}|$ for any $1 \leqslant t \leqslant 2^\kappa$. Then, we have $\mathsf{Pr}_{X \leftarrow\!\!\$\, \mathcal{R}}[X$ is not $\mathsf{Good}] \leqslant 1/2^\kappa$.*

Suppose X is Good. Given Z, since $t \leqslant 2^\kappa$, we have that apk is uniformly distributed over the set $\{YZ \mid Y \in C(t, X)\}$, which has size $|\mathcal{S}_{\mathrm{hash}}|$. Also, from the execution, we have that $|K| \leqslant \mathsf{q}+\mathsf{q}_s \leqslant 2\mathsf{q}$, and thus the probability that BadKey is set to true after the query is at most $|K|/|\mathcal{S}_{\mathrm{hash}}| \leqslant 2\mathsf{q}/2^\kappa$. Since there are at most q RO queries, the probability that BadKey is set to true during the simulation is at most $2\mathsf{q}^2/2^\kappa$. Therefore, we have that $\mathsf{Pr}[\mathsf{BadKey}] \leqslant \mathsf{Pr}[X \text{ is not } \mathsf{Good}] + \mathsf{Pr}[\mathsf{BadKey} \wedge X \text{ is } \mathsf{Good}] \leqslant \frac{2\mathsf{q}^2+1}{2^\kappa}$. □

Proof (of Lemma 2). For any $1 \leqslant t \leqslant 2^\kappa$, $s_1, s_2 \in \mathcal{S}$, and $X \in \mathcal{R}$ such that $s_1 \neq s_2$ and $X \neq 0$, since $\mathrm{char}(\mathcal{S}) \geqslant 2^\kappa$, we know that $t \cdot (s_1 - s_2) \neq 0$ and thus $t \cdot (s_1 - s_2) \cdot X \neq 0$, which implies $ts_1 \cdot X \neq ts_2 \cdot X$. Therefore, $|C(t, X)| = |\mathcal{S}_{\mathrm{hash}}|$, which means that X is Good. Thus, we have that $\mathsf{Pr}_{X \leftarrow \mathcal{R}}[X \text{ is not } \mathsf{Good}] \leqslant \mathsf{Pr}_{X \leftarrow \mathcal{R}}[X = 0] = \frac{1}{|\mathcal{R}|} \leqslant 1/2^\kappa$. □

CONSTRUCT $\mathcal{C}'$ FROM $\mathsf{Fork}^{\mathcal{C}}$. The input of $\mathcal{C}'$ consists of par, which defines a linear hash function $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$ and uniformly random elements $h_1^{(\mathrm{agg})}, \ldots, h_{\mathsf{q}}^{(\mathrm{agg})}$ $h_1^{(\mathrm{non})}$, $h_1^{(\mathrm{non})'}$, $\ldots$, $h_{\mathsf{q}}^{(\mathrm{non})}$, $h_{\mathsf{q}}^{(\mathrm{non})'} \in$

$\mathcal{S}_{\text{hash}}$. Also, $\mathcal{C}'$ can access oracles CHAL and PI defined the same way as those in the AOMPR game. To begin, $\mathcal{C}'$ runs $\mathsf{Fork}^{\mathcal{C}}(par, h_1^{(\text{agg})}, \ldots, h_{\mathsf{q}}^{(\text{agg})}, h_1^{(\text{non})}, h_1^{(\text{non})'}, \ldots, h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'})$. All queries to oracle CHAL from the first execution of $\mathcal{C}'$ are relayed by $\mathcal{B}$ to its own CHAL oracle, and for all CHAL queries from the second execution of $\mathcal{C}'$, $\mathcal{B}$ answers them with the same challenges as in the first execution. All PI queries from $\mathsf{Fork}^{\mathcal{C}'}$ are relayed by $\mathcal{B}$ to its own PI oracle.

After $\mathsf{Fork}^{\mathcal{C}}$ returns $(I_{\text{sig}}, J_{\text{sig}}, \text{Out}, \text{Out}')$, by the following claim, $\mathcal{C}'$ computes $\widetilde{x}$ such that $\mathsf{F}(\widetilde{x}) = \mathsf{apk}^*$ and returns $(I_{\text{agg}}, J_{\text{agg}}, (\widetilde{x}, \text{Out}, \text{Out}'))$, where I_{agg}, J_{agg}, and apk^* are from Out.

Claim 4 *If* $\mathsf{Fork}^{\mathcal{C}}$ *returns* $(I_{\text{sig}}, J_{\text{sig}}, \text{Out}, \text{Out}')$, $\mathcal{C}'$ *can compute* $\widetilde{x}$ *such that* $\mathsf{F}(\widetilde{x}) = \mathsf{apk}^*$, *where* apk^* *is from* Out.

Proof (of Claim 4). We directly use the notations in the description of $\mathcal{C}$ to denote the variables in Out and use $(\cdot)'$ to denote the variables in Out′. Since $\mathsf{Fork}^{\mathcal{C}}$ does not return $\perp$, we have $\mathrm{H}_{\text{sig}}(\mathsf{apk}^*, m^*, R^*) = h_I \neq h'_I = \mathrm{H}'_{\text{sig}}(\mathsf{apk}^*, m^*, R^*)$. Since the two executions of $\mathcal{C}$ are identical before $\mathrm{H}_{\text{sig}}(\mathsf{apk}^*, m^*, R^*)$ is assigned h_I, we know $(\mathsf{apk}^*, m^*, R^*) = (\mathsf{apk}^{*\prime}, m^{*\prime}, R^{*\prime})$. Therefore, we have $\mathsf{F}(s^*) = R^* + h_I \mathsf{apk}^*$ and $\mathsf{F}(s^{*\prime}) = R^* + h'_I \mathsf{apk}^*$, and $\mathcal{C}'$ computes $\widetilde{x} \leftarrow \frac{s^* - s^{*\prime}}{h_I - h'_I}$. □

ANALYSIS OF $\mathcal{C}'$. To use Lemma 1, we define IG as the algorithm that sets $par \leftarrow\!\!\$\ \mathsf{PGen}(1^\kappa)$ and returns par. Also, $(h_1^{(\text{agg})}, \ldots, h_{\mathsf{q}}^{(\text{agg})})$ plays the role of $(h_1, \ldots, h_q)$, and $((h_1^{(\text{non})}, h_1^{(\text{non})'}), \ldots, (h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'}))$ plays the role of $(v_1, \ldots, v_{q'})$. It is clear that $\text{acc}(\mathcal{C}') = \text{acc}(\mathsf{Fork}^{\mathcal{C}})$. Therefore, by Lemma 1 and (4), $\text{acc}(\mathsf{Fork}^{\mathcal{C}'}) \geqslant (\text{acc}(\mathsf{Fork}^{\mathcal{C}}))^2/\mathsf{q} - \frac{1}{|\mathcal{S}_{\text{hash}}|} \geqslant (\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}))^4/\mathsf{q}^3 - \frac{15}{2^\kappa}$.

CONSTRUCT $\mathcal{B}$ FROM $\mathsf{Fork}^{\mathcal{C}'}$. We now give a construct of the AOMPR adversary $\mathcal{B}$ using $\mathsf{Fork}^{\mathcal{C}'}$ and the available CHAL and PI oracles. To start with, $\mathcal{B}$ receives par from the $\text{AOMPR}^{\mathsf{LHF}}$ game and uniformly samples $h_1^{(\text{non})}$, $h_1^{(\text{non})'}$, $h_1^{(\text{non})''}$, $h_1^{(\text{non})'''}$, …, $h_{\mathsf{q}}^{(\text{non})}$, $h_{\mathsf{q}}^{(\text{non})'}$, $h_{\mathsf{q}}^{(\text{non})''}$, $h_{\mathsf{q}}^{(\text{non})'''} \in \mathcal{S}_{\text{hash}}$. Then, $\mathcal{B}$ runs $\mathsf{Fork}^{\mathcal{C}'}$ on input par, $(h_1^{(\text{non})}, h_1^{(\text{non})'})$, $(h_1^{(\text{non})''}, h_1^{(\text{non})'''})$, …, $(h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'})$, $(h_{\mathsf{q}}^{(\text{non})''}, h_{\mathsf{q}}^{(\text{non})'''})$, where $(h_i^{(\text{non})}, h_i^{(\text{non})'})$ plays the role of v_i and $(h_i^{(\text{non})''}, h_i^{(\text{non})'''})$ plays the role of v'_i. All CHAL queries from the first execution of $\mathcal{C}'$ are relayed by $\mathcal{B}$ to its own CHAL oracle, and, for all CHAL queries from the second execution of $\mathcal{C}'$, $\mathcal{B}$ answers them with the same challenges as the first execution. All PI queries from $\mathsf{Fork}^{\mathcal{C}'}$ are relayed by $\mathcal{B}$ to its own PI oracle. Without loss of generality, we can assume all challenges are different since otherwise $\mathcal{B}$ can solve them trivially. Denote the event BadHash as any two of the scalars $h_1^{(\text{non})}$, $h_1^{(\text{non})'}$, $h_1^{(\text{non})''}$, $h_1^{(\text{non})'''}$, …, $h_{\mathsf{q}}^{(\text{non})}$, $h_{\mathsf{q}}^{(\text{non})'}$, $h_{\mathsf{q}}^{(\text{non})''}$, $h_{\mathsf{q}}^{(\text{non})'''}$ are same. Since they are sampled uniformly from $\mathcal{S}_{\text{hash}}$, we know $\mathsf{Pr}[\mathsf{BadHash}] \leqslant (4\mathsf{q})^2/|\mathcal{S}_{\text{hash}}| \leqslant \frac{16\mathsf{q}^2}{2^\kappa}$. Then, we can conclude the proof with the following claim, which implies $\mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}) \geqslant \text{acc}(\mathsf{Fork}^{\mathcal{C}}) - \mathsf{Pr}[\mathsf{BadHash}] \geqslant (\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}))^4/\mathsf{q}^3 - \frac{16\mathsf{q}^2+15}{2^\kappa}$.

Claim 5 *If* $\mathsf{Fork}^{\mathcal{C}'}$ *returns* $(I_{\text{agg}}, J_{\text{agg}}, \text{Out}, \text{Out}')$ *and* BadHash *does not occur,* $\mathcal{B}$ *can win the game* $\text{AOMPR}_{\mathsf{LHF}}$.

□

Proof (proof of Claim 5). Denote $(\widetilde{x}, \text{Out}^{(1)}, \text{Out}^{(2)}) \leftarrow \text{Out}$ and $(\widetilde{x}', \text{Out}^{(3)}, \text{Out}^{(4)}) \leftarrow \text{Out}'$, and we use $(\cdot)^{(i)}$ to denote the variables in $\text{Out}^{(i)}$. The total number of CHAL queries is $4\mathsf{q}_s + 1$, and the corresponding challenges are $X, U_1, \ldots, U_{4\mathsf{q}_s}$.

We first show how to compute x^* such that $\mathsf{F}(x^*) = X$. Since $\mathsf{Fork}^{\mathcal{C}'}$ returns I_{agg}, we have $\mathrm{H}^{(1)}_{\text{agg}}(L^{*(1)}, X) = h^{(\text{agg})}_{I_{\text{agg}}} \neq h^{(\text{agg})'}_{I_{\text{agg}}} = \mathrm{H}^{(3)}_{\text{agg}}(L^{*(3)}, X)$. Since the two executions of $\mathcal{C}$ are identical

before $\mathrm{H}_{\mathrm{sig}}$ is assigned $h_{I_{\mathrm{agg}}}^{(\mathrm{agg})}$, we have $L^{*(1)} = L^{*(3)}$ (we denote $L^{*(1)}$ as L^* from here forward) and $\mathrm{H}_{\mathrm{agg}}^{(1)}(L^*, \mathsf{pk}') = \mathrm{H}_{\mathrm{agg}}^{(3)}(L^*, \mathsf{pk}')$ for any $\mathsf{pk}' \in L^*$ and $\mathsf{pk}' \neq X$. Therefore, the aggregated keys from L^* in the two execution can be represented as $\mathsf{apk}^{*(1)} = t \cdot h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} \cdot X + Z$, $\mathsf{apk}^{*(3)} = t \cdot h_{I_{\mathrm{agg}}}^{(\mathrm{agg})'} \cdot X + Z$, where t is the number of times X appears in L^* and $Z := \sum_{\mathsf{pk}' \in L^*, \mathsf{pk}' \neq X} \mathrm{H}_{\mathrm{agg}}^{(1)}(L^*, \mathsf{pk}) \cdot \mathsf{pk}'$. By Claim 4, $\mathsf{F}(\widetilde{x}) = \mathsf{apk}^{*(1)}$ and $\mathsf{F}(\widetilde{x}') = \mathsf{apk}^{*(3)}$. Therefore, $\mathcal{B}$ computes $x^* = \frac{\widetilde{x} - \widetilde{x}'}{t(h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} - h_{I_{\mathrm{agg}}}^{(\mathrm{agg})'})}$.

We now show how to compute $u_1, \ldots, u_{4\mathsf{q}_s}$ such that $\mathsf{F}(u_i) = U_i$. For $k \in [\mathsf{q}_s]$, $\mathsf{dt}^{(i)}(k) = (b, c, a, s) \neq \perp$ if and only if $\mathcal{C}$ queries PI on $\sum_{j \in [4]} b^{j-1} U_{i+4(k-1)} + ca \cdot X$. Define a set $T := \{(b, c \cdot a, s) \ : \ i \in [4], \mathsf{dt}^{(i)}(k) = (b, c, a, s)\}$. The total number of PI queries for simulating those PI queries from $\mathcal{C}$ is equal to $|T|$. From the execution of $\mathcal{B}$, we know for any $i_1, i_2 \in [4]$ and $i_1 \neq i_2$, where $(b, c, a, s) = \mathsf{dt}^{(i_1)}(k)$ and $(b', c', a', s') = \mathsf{dt}^{(i_2)}(k)$, if $b = b'$, then we have $(b, c, a, s) = (b', c', a', s')$. Therefore, we know for any distinct $(b, v, s), (b', v', s') \in T$, it holds that $b \neq b'$. Also, we have $|T| \leqslant 4$. If $|T| < 4$, $\mathcal{B}$ picks an arbitrary $b' \in \mathcal{S}_{\mathrm{hash}} \backslash \{b \ : \ (b, v, s) \in T\}$ and sets $s' \leftarrow \mathrm{PI}\left(\sum_{j \in [4]} b'^{j-1} U_{i+(4-1)k}\right)$. Then, $\mathcal{B}$ adds $(b', 0, s')$ to T and repeats this until T has size 4. Denote the elements in T as $(b_1, v_1, s_1), \ldots, (b_4, v_4, s_4)$, and we have $A\boldsymbol{U} = \mathsf{F}(\boldsymbol{s})$, where

$$A = \begin{pmatrix} 1 & b_1 & b_1^2 & b_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & b_4 & b_4^2 & b_4^3 \end{pmatrix}, \ \boldsymbol{U} = \begin{pmatrix} U_{1+4(k-1)} \\ \vdots \\ U_{4k} \end{pmatrix}, \ \boldsymbol{s} = \begin{pmatrix} s_1 - v_1 x^* \\ \vdots \\ s_4 - v_4 x^* \end{pmatrix}.$$

Since A is a Vandermonde matrix over the field $\mathcal{S}$, A has full rank. Therefore, $\mathcal{B}$ can compute $(u_{1+(k-1)4}, \ldots, u_{k4})^T = A^{-1}\boldsymbol{s}$. Also, the number of PI queries for simulating the PI queries from $\mathcal{C}$ and computing T is equal to 4. Therefore, the total number of PI queries made by $\mathcal{B}$ is $4\mathsf{q}_s$, which implies $\mathcal{B}$ wins the game $\mathrm{AOMPR}^{\mathsf{LHF}}$. □

4.2 Threshold Signatures

FROST1 [KG20] and a more efficient version FROST2 [BCK+22] of FROST1 are (partially) non-interactive threshold signature schemes as formalized in [BCK+22]. We first give the syntax and security definitions of non-interactive threshold signature schemes following [BCK+22], then present new schemes based on LHF that are transformed from FROST1/2, and finally show the security of the new schemes under the AOMPR assumption.

Syntax. A (partially) non-interactive threshold signature schemes for n signers and threshold t is a tuple of efficient (randomized) algorithms $\mathsf{TS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{SPP}, \mathsf{LPP}, \mathsf{LR}, \mathsf{PS}, \mathsf{Agg}, \mathsf{Vf})$ that behave as follows. Parties involved are a leader and n signers. The setup algorithm $\mathsf{Setup}(1^\kappa)$ initializes the state st_i for each signer $i \in [n]$ and st_0 for the leader and returns a system parameter par. We assume par is given to all other algorithms implicitly. The key generation algorithm $\mathsf{KeyGen}()$ returns a public verification key pk, public auxiliary information aux, and a secret key sk_i for each signer i.

The signing protocol consists of two rounds: a message-independent pre-processing round and a signing round. In the pre-processing round, any signer i can run $\mathsf{SPP}(\mathsf{st}_i)$ to generate a pre-processing token pp, which is sent to the leader, and the leader runs $\mathsf{LPP}(i, pp, \mathsf{st}_0)$ to update its state st_0 to incorporate token pp. In a signing round, for any signer set $SS \subseteq [n]$ with size t and message $m \in \{0,1\}^*$, the leader runs $\mathsf{LR}(m, SS, \mathsf{st}_0)$ to generate a leader request lr with

$lr.\mathsf{msg} = m$ and $lr.\mathsf{SS} = SS$ and sends lr to each signer $i \in SS$. Then, each signer i runs $\mathsf{PS}(lr, i, \mathsf{st}_i)$ to generate its partial signature $psig_i$. Finally, the leader computes a signature σ for m by running $\mathsf{Agg}(\{psig_i\}_{i \in SS})$. In summary, the signing protocol between signers in SS and the leader to sign a message $m \in \{0,1\}^*$ is represented by the following experiment:

$$\begin{aligned} &(pp_i, \mathsf{st}_i) \leftarrow \mathsf{SPP}() \ , \ \mathsf{st}_0 \leftarrow \mathsf{LPP}(i, pp_i, \mathsf{st}_0) \ , \text{ for each } i \in SS \ , \\ &(lr, \mathsf{st}_0) \leftarrow \mathsf{LR}(m, SS, \mathsf{st}_0) \ , \\ &(psig_i, \mathsf{st}_i) \leftarrow \mathsf{PS}(lr, i, \mathsf{st}_i) \ , \text{ for each } i \in SS \ , \\ &\sigma \leftarrow \mathsf{Agg}(\{psig_i\}_{i \in SS}) \ . \end{aligned} \tag{5}$$

The (deterministic) verification algorithm $\mathsf{Vf}(\mathsf{pk}, m, \sigma)$ outputs a bit that indicates whether or not σ is valid for pk and m or not. We say that TS is (perfectly) *correct* if for any $SS \subseteq [n]$ and any $m \in \{0,1\}^*$, $\mathsf{Pr}[\mathsf{Vf}(\mathsf{pk}, m, \sigma)] = 1$, where σ is output from the experiment in (5) and the probability is taken over the sampling of the system parameter par and the randomness of KeyGen.

Security. A hierarchy for security notions of threshold signatures is proposed in [BCK+22]. Here, we focus on two of them, TS-SUF-2 and TS-SUF-3, which are achieved by FROST2 and FROST1, respectively. TS-SUF-2 and TS-SUF-3 require that there exists an efficient strong verification algorithm SVf that takes as input a public key pk, a leader request lr, and a signature σ and outputs a bit that indicates whether σ is obtained legitimately for lr. SVf satisfies that for each (pk, lr), there exists at most one signature σ such that $\mathsf{SVf}(\mathsf{pk}, lr, \sigma) = 1$ and for any $SS \subseteq [n]$ and any $m \in \{0,1\}^*$, $\mathsf{Pr}[\mathsf{SVf}(\mathsf{pk}, lr, \sigma)] = 1$, where lr and σ are generated in the experiment in (5) and the probability is taken over the sampling of the system parameter par and the randomness of KeyGen. TS-SUF-2 guarantees that an adversary can generate a valid signature σ for m only if it receives partial signatures from at least $t - |CS|$ honest parties for the same leader request lr such that $lr.\mathsf{msg} = m$ and $\mathsf{SVf}(\mathsf{pk}, lr, \sigma) = 1$, where CS denotes the set of corrupted signers.

TS-SUF-3 is defined only for schemes where lr additionally specifies a function $lr.\mathsf{PP}$ that maps each $i \in lr.\mathsf{SS}$ to a pre-processing token generated by signer i. TS-SUF-3 guarantees that an adversary can generate a valid signature σ for m only if, in addition to the condition of TS-SUF-2, it receives partial signatures from each honest signer i such that $lr.\mathsf{PP}(i)$ is honestly generated by signer i for lr. Formally, the TS-SUF-2 game and the TS-SUF-3 game are defined in Figure 6, where $\mathsf{TS.HF}$ denotes the space of the hash functions used in TS from which the random oracle is drawn. The advantage of $\mathcal{A}$ for the TS-SUF-X game is defined as $\mathsf{Adv}^{\text{ts-suf-X}}_{\mathsf{TS}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\text{TS-SUF-X}^{\mathcal{A}}_{\mathsf{TS}}(\kappa) = 1\right]$ for $\mathsf{X} \in \{2, 3\}$.

Our Schemes. Figure 7 shows the protocols FROST1-H and FROST2-H that are transformed from FROST1 and FROST2, respectively. In addition to the general transformation, we need to pick an injection $\mathsf{x}_{(\cdot)} : [n] \to \mathcal{S}$. The choice of $\mathsf{x}_{(\cdot)}$ can be arbitrary, and the corresponding Lagrange coefficient for a set of index $S \subseteq [n]$ and $i \in S$ is defined as $\lambda_i^S := \prod_{j \in S \setminus \{i\}} \frac{\mathsf{x}_j}{\mathsf{x}_i - \mathsf{x}_j}$. We analyse the correctness of the scheme in Appendix A. Also, similar to the multi-signature case, we optimize the schemes by sampling key shares from $\mathcal{D}_{\text{key}} \subseteq \mathcal{D}$ and setting the hash range to be $\mathcal{S}_{\text{hash}} \subseteq \mathcal{S}$.

The following theorems show that, under the AOMPR assumption, FROST2-H is TS-SUF-2-secure and FROST1-H is TS-SUF-3-secure in the random oracle model. We prove the theorems using the same techniques from [BCK+22]. The differences from the previous proofs are the same as those highlighted in the case of multi-signatures in Section 4.1.

Theorem 3. *For any* TS-SUF-2 *adversary* $\mathcal{A}$ *game making at most* q_s *queries to* PPO *and* q_h *queries to* RO*, there exists an* AOMPR *adversary* $\mathcal{B}$ *making at most* $2\mathsf{q}_s + t$ *queries to* Chal

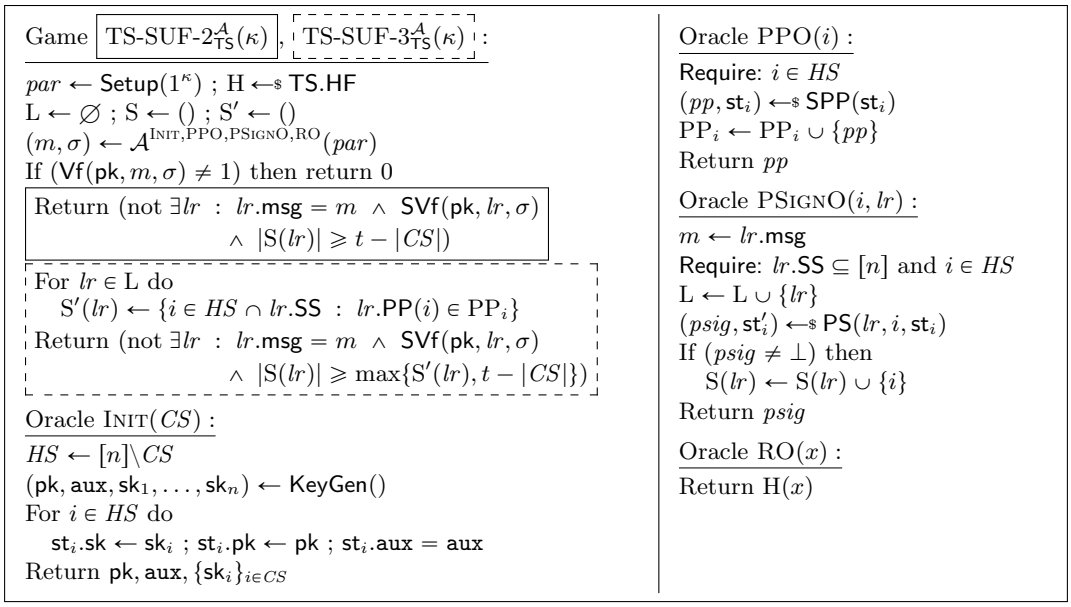

Fig. 6. The TS-SUF-2 game and the TS-SUF-3 game for a threshold signature scheme TS. The TS-SUF-2 game contains all but the dashed box, and the TS-SUF-3 game contains all but the solid box.

running in time roughly equal two times that of $\mathcal{A}$ such that

$$\mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{FROST2\text{-}H[LHF]}}(\mathcal{A}, \kappa) \leqslant \sqrt{\mathsf{q} \cdot \left(\mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}, \kappa) + (5\mathsf{q}^2)/2^\kappa\right)} \ ,$$

where $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.

Theorem 4. *For any* TS-SUF-3 *adversary $\mathcal{A}$ making at most q_s queries to* PPO *and q_h queries to* RO*, there exists an* AOMPR *adversary $\mathcal{B}$ making at most $2\mathsf{q}_s + t$ queries to* CHAL *running in time roughly equal two times that of $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{ts-suf-3}}_{\mathsf{FROST1\text{-}H[LHF]}}(\mathcal{A}, \kappa) \leqslant 4n \cdot \mathsf{q} \cdot \sqrt{\left(\mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}, \kappa) + 6\mathsf{q}/2^\kappa\right)} \ ,$$

where $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.

4.3 Proof of Theorem 3

Let $\mathcal{A}$ be an adversary as described in the theorem. Denote the output message-signature pair of $\mathcal{A}$ as $(m^*, \sigma^* = (R^*, z^*))$. Without loss of generality, we assume $\mathcal{A}$ always queries RO on $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$ before $\mathcal{A}$ returns and always queries RO on $\mathrm{H}_1(\mathsf{pk}, lr)$ prior to the query $\mathrm{PSIGNO}(i, lr)$ for some i and lr. (This adds up to q_s additional RO queries, and we let $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.) Denote lr^* as the leader query such that $\mathrm{H}_1(\mathsf{pk}, lr^*)$ is the first query prior to the query $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$ satisfying $\mathsf{SVf}(\mathsf{pk}, lr^*, \sigma^*) = \mathtt{true}$. If such lr^* does not exists, lr^* is set to $\perp$. Denote the event E_1 as

$$\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) \ \wedge\ (lr^* = \perp \ \vee\ \mathrm{S}_2(lr^*) < t - |CS|) \ .$$

It is clear that if $\mathcal{A}$ wins the game TS-SUF-2$^{\mathsf{FROST2\text{-}H[LHF]}}$, then E_1 must occur, which implies $\mathsf{Pr}[E_1] \geqslant \mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{FROST2\text{-}H[LHF]}}(\mathcal{A})$. Therefore, the theorem will follow from the following lemma. (We

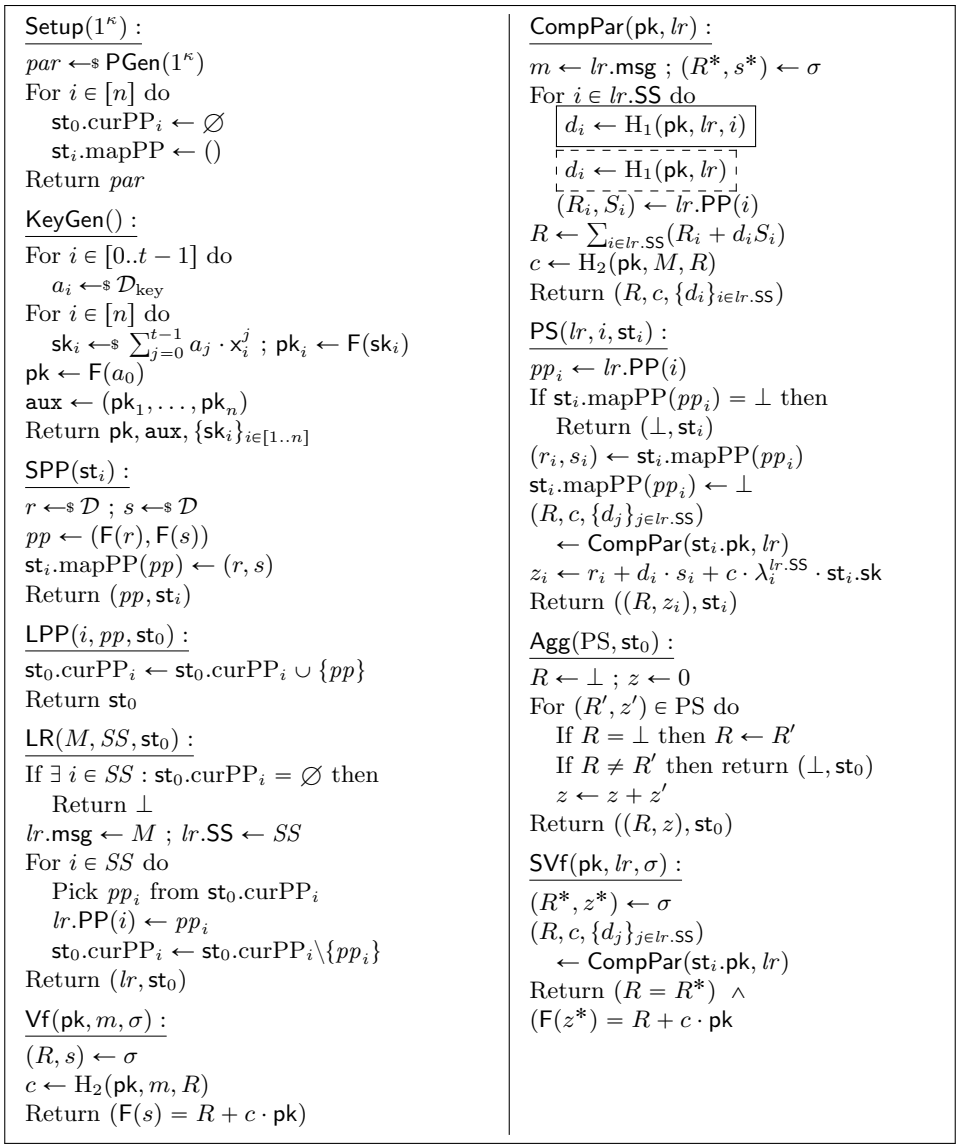

Fig. 7. The protocol FROST1-H[LHF] and FROST1-H[LHF], where $\mathsf{LHF} = (\mathsf{PGen}, \mathsf{F})$ is a linear hash function family. The protocol FROST1-H contains all but the dashed box, and the protocol FROST2-H contains all but the solid box. Further, n is the number of parties, and t is the threshold of the schemes. $\mathsf{x}_{(\cdot)}$ is an injection from $[n]$ to $\mathcal{S}$ and $\lambda_i^{lr.\mathsf{SS}}$ denotes the Lagrange coefficient which is computed as $\lambda_i^{lr.\mathsf{SS}} := \prod_{j\in S\setminus\{i\}} \frac{\mathsf{x}_j}{\mathsf{x}_j - \mathsf{x}_i}$. $\mathcal{D}_{\mathrm{key}}$ is a subset of $\mathcal{D}$ such that F is a bijection between $\mathcal{D}_{\mathrm{key}}$ and $\mathcal{S}$. The function $\mathrm{H}_i(\cdot)$ is computed as $\mathrm{H}(i, \cdot)$ for $i = 1, 2$, where $\mathrm{H} : \{0,1\}^* \to \mathcal{S}$.

isolate this statement as its own lemma also because it will be helpful in the proof of Theorem 4 below.)

Lemma 3. *There exists an adversary $\mathcal{B}$ for the* $\mathrm{AOMPR}^{\mathsf{LHF}}$ *game making at most* $2\mathsf{q}_s + t$ *queries to* Chal *such that*

$$\mathsf{Pr}[E_1] \leqslant \sqrt{\mathsf{q} \cdot (\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{B}) + 5\mathsf{q}^2/2^\kappa)}\,.$$

Moreover, $\mathcal{B}$ runs in time roughly twice that of $\mathcal{A}$.

$\underline{\mathsf{Fork}^{\mathcal{A}}(x)}$:
Pick the random coin ρ of $\mathcal{A}$ at random
$h_1, h'_1, \ldots, h_q, h'_q \leftarrow H$
$(I, \mathrm{Out}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q; \rho)$
If $I = \perp$ then return $\perp$
$(I', \mathrm{Out}') \leftarrow \mathcal{A}(x, h_1, \ldots, h_{I-1}, h'_I, \ldots, h'_q; \rho)$
If $I \neq I'$ then return $\perp$
Return $(I, \mathrm{Out}, \mathrm{Out}')$

Fig. 8. The forking algorithm build from $\mathcal{A}$.

To prove Lemma 3, we use the following variant of the forking lemma from [BTZ22].

Lemma 4. *Let $q \geqslant 1$ be an integer, $S \subseteq [1..q]$ be a set, and H be a set. Let $\mathcal{A}$ be a randomized algorithm that on input $x, h_1, \ldots, h_q$ outputs a pair (I, Out), where $I \in \{\perp\} \cup S$ and Out is a side output. Let IG be a randomized algorithm that generates x. The accepting probability of $\mathcal{A}$ is defined as*

$$\mathrm{acc}(\mathcal{A}) = \mathsf{Pr}_{x \leftarrow\$ \mathsf{IG}, h_1, \ldots, h_q \leftarrow\$ H}[(I, \mathrm{Out}) \leftarrow\$ \mathcal{A}(x, h_1, \ldots, h_q) \ : \ I \neq \perp] \ .$$

Consider algorithm $\mathsf{Fork}^{\mathcal{A}}$ described in Figure 8. The accepting probability of $\mathsf{Fork}^{\mathcal{A}}$ is defined as

$$\mathrm{acc}(\mathsf{Fork}^{\mathcal{A}}) = \mathsf{Pr}_{x \leftarrow\$ \mathsf{IG}}[\alpha \leftarrow\$ \mathsf{Fork}^{\mathcal{A}}(x) \ : \ \alpha \neq \perp] \ .$$

Then, $\mathrm{acc}(\mathsf{Fork}^{\mathcal{A}}) \geqslant \mathrm{acc}(\mathcal{A})^2/|S|$.

Proof (of Lemma 3). We first construct an algorithm $\mathcal{C}$ compatible with the syntax in Lemma 4 and then construct $\mathcal{B}$ from $\mathsf{Fork}^{\mathcal{C}}$. The input of $\mathcal{C}$ consists of *par* that defines a linear hash function $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$ and uniformly random elements $h_1, \ldots, h_{2\mathsf{q}} \in \mathcal{S}_{\mathrm{hash}}$. Also, $\mathcal{C}$ can access oracles CHAL and PI defined the same as those in the AOMPR$^{\mathsf{LHF}}$ game. (We can think of the oracles as part of the input of $\mathcal{C}$ in the context of the Forking Lemma.) For simplicity, when $\mathcal{C}$ makes a query $(X, \alpha, \{\beta_i\})$ to PI, we omit the coefficients $\alpha, \{\beta_i\}$ which are clear from the context. To start with, $\mathcal{C}$ makes $2\mathsf{q}_s + t$ queries to CHAL and denotes the challenges as $A_0, \ldots, A_{t-1}$, U_1, V_1, $\ldots$, U_{q_s}, $V_{\mathsf{q}_s} \in \mathcal{D}$. Then, $\mathcal{C}$ initializes all the states $\mathsf{st}_0, \ldots, \mathsf{st}_n$. In addition, it initializes counters ctr_s,ctr_h to 0 and a function dt to an empty table, which are used to record the PI query related to each (U_j, V_j). $\mathcal{C}$ also initializes curLR $\leftarrow \varnothing$ to record all leader requests that appears during the game and initializes ctrPP to an empty table, which are used to record the counter corresponding to each token generated by honest parties. We also use a flag BadPPO to denote whether a bad event occurs, which are initially set to `false`. Then, $\mathcal{C}$ runs $\mathcal{A}$ with access to the oracles $\widetilde{\text{INIT}}$, $\widetilde{\text{PPO}}$, $\widetilde{\text{PSIGNO}}$, $\widetilde{\text{RO}}$, which are simulated as follows.

$\widetilde{\text{Init}}(CS)$: $\mathcal{C}$ initializes H to an empty table and sets $\overline{\mathsf{pk}} \leftarrow A_0$, $\overline{\mathsf{pk}}_i = \prod_{j=0}^{t-1} A_j \cdot \mathsf{x}_i^j$ for $i \in [n]$, and $\overline{\mathsf{sk}}_i \leftarrow \mathrm{PI}(\overline{\mathsf{pk}}_i)$ for $i \in CS$. $\mathcal{C}$ samples $\tilde{a}_i \leftarrow\$ \mathcal{D}_{\mathrm{key}}$ for $i \in [0..(t-1)]$ and sets $\mathsf{sk}_i \leftarrow \sum_{j=0}^{t-1} \tilde{a}_j \mathsf{x}_i^j$ for $i \in CS$. Then, $\mathcal{C}$ computes a polynomial $f(x) = \sum_{i=0}^{t-1} \mu_i x^i$ such that $\mu_i \in \mathcal{D}$ for $i \in [0..(t-1)]$, $f(\mathsf{x}_i) = \mathsf{sk}_i - \overline{\mathsf{sk}}_i$ for $i \in CS$, and $f(\mathsf{x}_i) = 0$ for $i \in S'$, where $S' \subseteq [n]$ denotes the set of the first $(t - |CS|)$ honest parties. [1] $\mathcal{C}$ sets $\mathsf{pk}_i \leftarrow \overline{\mathsf{pk}}_i + \mathsf{F}(f(\mathsf{x}_i))$ for $i \in [n]$. Finally, $\mathcal{C}$ returns $(\mathsf{pk}, \mathsf{aux} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \{\mathsf{sk}_i\}_{i \in CS})$.

[1] Since the degree of f is t and we fix t points of f, f is fixed and we can compute the coefficients of f by solving a linear equation.

$\widetilde{\mathbf{RO}}$ query $\mathrm{H}_1(x)$: If $\mathrm{H}_1(x) \neq \bot$, $\mathcal{C}$ returns $\mathrm{H}_1(x)$. Otherwise, parse x as $(\widetilde{\mathsf{pk}}, lr)$. If the parsing fails or $\widetilde{\mathsf{pk}} \neq \mathsf{pk}$, $\mathcal{C}$ sets $\mathrm{H}_1(x) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ and returns $\mathrm{H}_1(x)$. Otherwise, $\mathcal{C}$ increases ctr_h by 1, sets $\mathrm{H}_1(x) \leftarrow h_{2\mathrm{ctr}_h-1}$, and adds lr to curLR. Also, $\mathcal{C}$ computes $R \leftarrow \sum_{i\in lr.\mathsf{SS}}(R_i + h_{2\mathrm{ctr}_h-1} \cdot S_i)$, where $(R_i, S_i) \leftarrow lr.\mathsf{PP}(i)$. If $\mathrm{H}_2(\mathsf{pk}, lr.\mathsf{msg}, R) = \bot$, $\mathcal{C}$ sets $\mathrm{H}_2(\mathsf{pk}, lr.\mathsf{msg}, R) = h_{2\mathrm{ctr}_h}$. In addition, define $\mathrm{mapLR}(\mathrm{ctr}_h) := lr$ and set $\mathrm{curLR} \leftarrow \mathrm{curLR} \cup \{lr\}$. Finally, $\mathcal{C}$ returns $\mathrm{H}_1(x)$.

$\widetilde{\mathbf{RO}}$ query $\mathrm{H}_2(x)$: If $\mathrm{H}_2(x) \neq \bot$, $\mathcal{C}$ returns $\mathrm{H}_2(x)$. Otherwise, parse x as $(\widetilde{\mathsf{pk}}, m, R)$. If the parsing fails or $\widetilde{\mathsf{pk}} \neq \mathsf{pk}$, $\mathcal{C}$ sets $\mathrm{H}_2(x) \leftarrow\!\!\$\ \mathcal{S}_{\mathrm{hash}}$ and returns $\mathrm{H}_2(x)$. Otherwise, $\mathcal{C}$ increases ctr_h by 1 and sets $\mathrm{H}_2(x) \leftarrow h_{2\mathrm{ctr}_h}$. Finally, $\mathcal{C}$ returns $\mathrm{H}_2(x)$.

$\widetilde{\mathbf{PPO}}(i)$ query: Same as in the game TS-SUF-2$^{\mathsf{FROST2\text{-}H}}$, except in the simulation of algorithm SPP, $\mathcal{C}$ first increases ctr_s by 1 and sets $pp \leftarrow (U_{\mathrm{ctr}_s}, V_{\mathrm{ctr}_s})$, $\mathsf{st}_i.\mathrm{mapPP}(pp) \leftarrow (0,0)$, and $\mathrm{ctrPP}(i, pp) \leftarrow \mathrm{ctr}_s$. In addition, BadPPO is set to $\mathtt{true}$ if there exists $lr \in \mathrm{curLR}$ such that $lr.\mathsf{PP}(i) = (U_{\mathrm{ctr}_s}, V_{\mathrm{ctr}_s})$.

$\widetilde{\mathbf{PSignO}}(i, lr)$ query: Same as in the game TS-SUF-2$^{\mathsf{FROST2\text{-}H}}$, except in the simulation of algorithm PS, if $\mathsf{st}_i.\mathrm{mapPP}(pp) \neq \bot$, $\mathcal{C}$ sets

$$z_i \leftarrow \mathrm{PI}\left(U_j + d_i V_j + c \cdot \lambda_i^{lr.\mathsf{SS}} \cdot \mathsf{pk}_i\right) ,$$

where $j \leftarrow \mathrm{ctrPP}(i, lr.\mathsf{PP}(i))$. In addition, $\mathcal{C}$ sets $\mathsf{dt}(j) \leftarrow (i, k, d_i, c\lambda_i^{lr.\mathsf{SS}}, z_i)$, where k denotes the index such that $\mathrm{H}_1(\mathsf{pk}, lr)$ is set to h_{2k-1} during the simulation.

After receiving the output $(m^*, \sigma^* = (R^*, z^*))$ from $\mathcal{A}$, $\mathcal{C}$ returns $\bot$ if $\mathsf{BadPPO} = \mathtt{true}$ or E_1 does not occur. Otherwise, $\mathcal{C}$ finds the index I such that $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$ is set to h_I during the simulation. By our assumption of $\mathcal{A}$, we know such I must exist. Then, $\mathcal{C}$ returns (I, Out), where Out consists of all variables received or generated by $\mathcal{C}$.

Analysis of $\mathcal{C}$. To use Lemma 4, we define $S := \{2k\}_{k\in[1..\mathsf{q}]}$ and IG as the algorithm that runs $\mathsf{PGen}(1^\kappa)$ and outputs par. From the simulation, we know the output index I of $\mathcal{C}$ is always in S.

It is not hard to see $\mathcal{C}$ simulates the game TS-SUF-2$^{\mathsf{FROST2\text{-}H}}$ perfectly, which implies $\mathrm{acc}(\mathcal{C}) \geqslant \Pr[E_1] - \Pr[\mathsf{BadPPO}]$, where $\Pr[E_1]$ refers to the probability in the original TS-SUF-2$^{\mathsf{FROST2\text{-}H[LHF]}}$ game with $\mathcal{A}$ (as in the lemma statement), whereas $\Pr[\mathsf{BadPPO}]$ is the probability that $\mathsf{BadPPO} = \mathtt{true}$ at the end of $\mathcal{C}$'s execution. Since (U_j, V_j) is sampled uniformly from $\mathcal{R}$. Therefore, for each $\mathrm{PPO}(i)$ query, the probability BadPPO is set to $\mathtt{true}$ is less than $|\mathrm{curLR}|/|\mathcal{R}| \leqslant \mathsf{q}_h/2^\kappa$. Therefore, we have $\Pr[\mathsf{BadPPO}] \leqslant \mathsf{q}_s\mathsf{q}_h/2^\kappa$. By Lemma 4,

$$\begin{aligned}\mathrm{acc}(\mathsf{Fork}^{\mathcal{C}}) &\geqslant (\Pr[E_1] - \mathsf{q}_s\mathsf{q}_h/2^\kappa)^2/\mathsf{q} \geqslant \Pr[E_1]^2/\mathsf{q} - 2\Pr[E_1]\mathsf{q}_s\mathsf{q}_h/(2^\kappa \cdot q)\\ &\geqslant \Pr[E_1]^2/\mathsf{q} - \mathsf{q}/2^\kappa .\end{aligned}$$

Construct $\mathcal{B}$ from $\mathsf{Fork}^{\mathcal{C}}$. We now give a construct of the AOMPR adversary $\mathcal{B}$ using $\mathsf{Fork}^{\mathcal{C}}$, and the available Chal and PI oracles. To start with, $\mathcal{B}$ receives par from the AOMPR$^{\mathsf{LHF}}$ game and runs $\mathsf{Fork}^{\mathcal{C}}(par)$. All the Chal queries from the first execution of $\mathcal{C}$ are relayed by $\mathcal{B}$ to its own Chal oracle, and for all the Chal queries from the second execution of $\mathcal{C}$, $\mathcal{B}$ answers them with the same challenges as the first execution. All the PI queries from $\mathsf{Fork}^{\mathcal{C}}$ are relayed by $\mathcal{B}$ to its own PI oracle. Without loss of generality, we can assume all the challenges are different, since otherwise, $\mathcal{B}$ can solve them trivially. Denote the event $\mathsf{BadHash}$ as $\{h_1, \ldots, h_{2\mathsf{q}}\} \cap \{h'_1, \ldots, h'_{2\mathsf{q}}\} \neq \varnothing$, where $h_1, h'_1, \ldots, h_{2\mathsf{q}}, h'_{2\mathsf{q}}$ generated in the execution of $\mathsf{Fork}^{\mathcal{C}}$ are same. Since the hash values are sampled uniformly from $\mathcal{S}_{\mathrm{hash}}$, we know $\Pr[\mathsf{BadHash}] \leqslant 4\mathsf{q}^2/|\mathcal{S}_{\mathrm{hash}}| \leqslant 4\mathsf{q}^2/2^\kappa$. Then, we can conclude the proof with the following claim, which implies

$$\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{B}) \geqslant \mathrm{acc}(\mathsf{Fork}^{\mathcal{C}}) - \Pr[\mathsf{BadHash}] \geqslant \Pr[E_1]^2/\mathsf{q} - 5\mathsf{q}^2/2^{\kappa} .$$

Claim 6 *$\mathcal{B}$ can win the game* $\mathrm{AOMPR}_{\mathsf{LHF}}$*, if* $\mathsf{Fork}^{\mathcal{C}}$ *returns* $(I, \mathrm{Out}, \mathrm{Out}')$ *and* $\mathsf{BadHash}$ *does not occur.*

□

Proof (of Claim 6). We directly use the notations in the description of $\mathcal{C}$ to denote the variables in Out and use $(\cdot)'$ to denote the variables in Out′. The total number of the CHAL queries is $t + 2\mathsf{q}_s$ and the corresponding challenges are $A_0, \ldots, A_{t-1}, U_1, V_1, \ldots, U_{\mathsf{q}_s}, V_{\mathsf{q}_s}$.

We first show how to compute $a_0, \ldots, a_{t-1}$ such that $\mathsf{F}(a_i) = A_i$. By the execution of $\mathsf{Fork}^{\mathcal{C}}$, we know $(\mathsf{pk}, m^*, R^*) = (\mathsf{pk}', m^{*\prime}, R^{*\prime})$ and $\mathsf{pk} = A_0$. Since $I \in S$, let $k^* = I/2$. It is not hard to see that $\mathrm{mapLR}(k^*) = lr^*$. (If $\mathrm{mapLR}(k^*) = \perp$, lr^* is also $\perp$.) Since $\mathsf{BadHash}$ does not occur, we have $\mathrm{H}_2(\mathsf{pk}, m^*, R^*) = h_I \neq h'_I = \mathrm{H}'_2(\mathsf{pk}, m^*, R^*)$. Since $\mathsf{F}(z^*) = R^* + h_I A_0$, $\mathsf{F}(z^{*\prime}) = R^* + h'_I A_0$, we have $\mathsf{F}(z^* - z^{*\prime}) = (h_I - h'_I)A_0$ and therefore $\mathcal{B}$ computes $a_0 \leftarrow \frac{z^* - z^{*\prime}}{h_I - h'_I}$.

Define $T_{\mathsf{dt}} := \{j \ : \ (i,k,d,c,z) \leftarrow \mathsf{dt}(j), k = k^*\}$. For each $j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}$, let $(i,k,d,c,z) \leftarrow \mathsf{dt}(j)$ and $(i',k',d',c',z') \leftarrow \mathsf{dt}'(j)$, and we have $\mathsf{F}(z) = U_j + dV_j + c \cdot \mathsf{pk}_i$, $\mathsf{F}(z') = U_j + d'V_j^{d'} + c' \cdot \mathsf{pk}_{i'}$, $c = h_I \lambda_i^{lr^*.\mathsf{SS}}$, and $c' = h'_I \lambda_{i'}^{lr^*.\mathsf{SS}}$. Since $\mathsf{BadPPO} = \mathtt{false}$ during both execution of $\mathcal{C}$, we know (U_j, V_j) is returned by a query $\mathrm{PPO}(i)$ prior to the query $\mathrm{H}_2(\mathsf{pk}, M^*, R^*)$ during the first execution of $\mathcal{C}$. Since the two executions of $\mathcal{C}$ are exactly the same prior to the query $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$, we know $i' = i$. Also, we know $d = h_{2k-1} = h_{2k^*-1} = h_{2k'-1} = d'$, which implies $\mathsf{F}(z - z') = \lambda_{i'}^{lr^*.\mathsf{SS}}(h_I - h'_I) \cdot \mathsf{pk}_i$. Since $h_I \neq h'_I$, $\mathcal{B}$ computes $y_i \leftarrow \frac{z - z'}{\lambda_i^{lr^*.\mathsf{SS}}(h_I - h'_I)}$, which satisfies $\mathsf{F}(y_i) = \mathsf{pk}_i$. Denote $D := \{i\}_{j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}, (i,k,d,c,z) \leftarrow \mathsf{dt}(j)}$. Since E_1 occurs in the first execution of $\mathcal{C}$, we know $|T_{\mathsf{dt}}| = |\mathrm{S}_2(lr^*)| < t - |CS|$. Therefore, we know $|D| = |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| < t - |CS|$. Therefore, $\mathcal{B}$ can pick an arbitrary set $D' \in HS \backslash D$ with size $(t - |CS| - |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| - 1)$ and for each $i \in D'$, $\mathcal{B}$ sets $y_i \leftarrow \mathrm{PI}(\mathsf{pk}_i)$. Denote $D_{tot} = CS \cup D \cup D'$, and we have $|D_{tot}| = t - 1$ and for each $i \in D_{tot}$, $\mathcal{B}$ knows y_i such that $\mathsf{F}(y_i) = \mathsf{pk}_i$. Since $\mathsf{pk}_i = \mathsf{F}(f(\mathsf{x}_i)) + A_0 + \sum_{j \in [t-1]} A_j \cdot \mathsf{x}_i^j$, denote $D_{tot} = \{i_1, \ldots, i_{t-1}\}$ and we have

$$M \cdot \begin{pmatrix} A_1 \\ \cdots \\ A_{t-1} \end{pmatrix} = \begin{pmatrix} \mathsf{F}(y_{i_1} - f(\mathsf{x}_{i_1}) - a_0) \\ \cdots \\ \mathsf{F}(y_{i_{t-1}} - f(\mathsf{x}_{i_{t-1}}) - a_0) \end{pmatrix}, \text{ where } M = \begin{pmatrix} \mathsf{x}_{i_1} & \cdots & \mathsf{x}_{i_1}^{t-1} \\ \vdots & \ddots & \vdots \\ \mathsf{x}_{i_{t-1}} & \cdots & \mathsf{x}_{i_{t-1}}^{t-1} \end{pmatrix}. \tag{6}$$

Since M is a Vandermonde matrix, we know M has full rank and thus $\mathcal{B}$ can compute $a_1, \ldots, a_{t-1}$ from (6) such that $\mathsf{F}(a_i) = A_i$ for $i \in [t-1]$. Further, for $i \in [n] \backslash D_{tot}$, $\mathcal{B}$ computes $y_i \leftarrow f(\mathsf{x}_i) + \sum_{j \in [0..(t-1)]} a_j \cdot \mathsf{x}_i^j$, which satisfies $\mathsf{F}(y_i) = \mathsf{pk}_i$.

We now show how to compute $u_1, v_1, \ldots, u_{\mathsf{q}_s}, v_{\mathsf{q}_s}$ such that $\mathsf{F}(u_i) = U_i$ and $\mathsf{F}(v_i) = V_i$. From the execution of $\mathcal{C}$, we know $\mathsf{dt}(j) = (i,k,d,c,z) \neq \perp$ if and only if $\mathcal{C}$ queries PI on $U_j + dV_j + c \cdot \mathsf{pk}_i$. Therefore, denote $U_j + dV_j + c \cdot \mathsf{pk}_i$ as the PI query *associated* with $\mathsf{dt}(j)$. For each $j \in \mathsf{q}_s$, there are the following cases.

Case 0: Both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are $\perp$. In this case, $\mathcal{B}$ computes u_j and v_j by directly querying oracle PI on U_j and V_j.

Case 1: Exactly one of $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ is not $\perp$. Without loss of generality, assume $\mathsf{dt}(j) = (i,k,d,c,z)$, which implies $\mathsf{F}(z) = U_j + dV_j + c \cdot \mathsf{pk}_i$. $\mathcal{B}$ computes v_j by directly querying oracle PI and computes $u_j \leftarrow z - d \cdot v_j - c \cdot y_i$.

For all the following cases, both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are not $\perp$ and we denote $(i,k,d,c,z) \leftarrow \mathsf{dt}(j)$ and $(i',k',d',c',z') \leftarrow \mathsf{dt}'(j)$.

Case 2: $k \neq k'$ or $k = k' > k^*$. In this case, we know $d = h_{2k-1} \neq h'_{2k'-1} = d'$ and $\mathsf{F}(z) = U_j + dV_j + c \cdot \mathsf{pk}_i$, $\mathsf{F}(z') = U_j + d'V_j + c' \cdot \mathsf{pk}_{i'}$. Therefore, $\mathcal{B}$ computes $v_j \leftarrow \frac{z - c \cdot y_i - z' + c' \cdot y_{i'}}{d - d'}$, and $u_j \leftarrow z - d \cdot v_j - c \cdot y_i$.

Case 3: $k = k' = k^*$. In this case, $\mathcal{B}$ computes v_j, u_j the same as Case 1.

Case 4: $k = k' < k^*$. $\mathcal{B}$ computes v_j, u_j the same as Case 1. Also, in this case, we have $d = d'$ and $c = c'$. Therefore, $\mathcal{B}$ queries PI oracle only once in order to simulate the PI queries associated with $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$.

We now count the number of PI queries made by $\mathcal{B}$.

- $\mathcal{B}$ queries PI oracle $|CS|$ times queries for simulating query $\mathrm{PI}(\mathsf{pk}_i)$ made by $\mathcal{C}$ for each $i \in |CS|$.
- $\mathcal{B}$ queries PI oracle $|D'|$ times queries for computing $a_0, \ldots, a_{t-1}$.
- For each $j \in \mathsf{q}_s$, $\mathcal{B}$ queries PI twice for simulating query associated with $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ and computing u_j and v_j in case 0, 1, 2, 4 and queries 3 times in case 3.

Since the condition of case 3 is equivalent to $j \in T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}$, the total number of PI queries made by $\mathcal{B}$ is equal to $2\mathsf{q}_s + |T_{\mathsf{dt}} \cap T_{\mathsf{dt}'}| + |CS| + |D'| = 2\mathsf{q}_s + t - 1$. Therefore, $\mathcal{B}$ wins the game $\mathrm{AOMPR}_{\mathsf{LHF}}$. □

4.4 Proof of Theorem 4

Let $\mathcal{A}$ be the adversary described in the theorem. Denote the output message-signature pair of $\mathcal{A}$ as $(m^*, \sigma^* = (R^*, z^*))$. Without loss of generality, we assume $\mathcal{A}$ always queries RO on $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$ before $\mathcal{A}$ returns and always queries RO on $\mathrm{H}_1(\mathsf{pk}, lr, i)$ prior to the query $\mathrm{PSIGNO}(i, lr)$ for some i and lr. (This adds up to q_s additional RO queries, and we let $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.) Denote lr^* as the leader query such that $\mathrm{H}_1(\mathsf{pk}, lr^*, i)$ is the first RO query prior to the $\mathrm{H}_2(\mathsf{pk}, m^*, R^*)$ query for some i satisfying $\mathsf{SVf}[\mathrm{H}](\mathsf{pk}, lr^*, \sigma^*) = \mathtt{true}$. If such lr^* does not exist, lr^* is set to $\perp$. Denote the event E_1 as

$$\mathsf{Vf}[\mathrm{H}](\mathsf{pk}, m^*, \sigma^*) \ \wedge \ (lr^* = \perp \ \vee \ \mathrm{S}_2(lr^*) < t - |CS|) \ .$$

Denote the event E_2 as

$$\mathsf{Vf}[\mathrm{H}](\mathsf{pk}, m^*, \sigma^*) \ \wedge \ lr^* \neq \perp \ \wedge \ \mathrm{S}_2(lr^*) \neq \mathrm{S}_3(lr^*) \ .$$

If $\mathcal{A}$ wins the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$ and $lr^* \neq \perp$, we know either $\mathrm{S}_2(lr^*) < t - |CS|$ or $\mathrm{S}_2(lr^*) \neq \mathrm{S}_3(lr^*)$. Therefore, if $\mathcal{A}$ wins the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$, then either E_1 or E_2 occurs, which implies

$$\mathsf{Adv}^{\text{ts-suf-3}}_{\mathsf{FROST1\text{-}H[LHF]}}(\mathcal{A}) \leqslant \mathsf{Pr}[E_1] + \mathsf{Pr}[E_2] \leqslant 2\max\{\mathsf{Pr}[E_1], \mathsf{Pr}[E_2]\} \ .$$

Thus, we conclude the theorem with the following two lemmas.

Lemma 5. *There exists an adversary $\mathcal{B}$ for the* $\mathrm{AOMPR}^{\mathsf{LHF}}$ *game making at most* $2\mathsf{q}_s + t$ *queries to* CHAL *such that*

$$\mathsf{Pr}[E_1] \leqslant \sqrt{\mathsf{q} \cdot (\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{B}) + 3\mathsf{q}^2(n+1)^2/2^\kappa)} \ ,$$

Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$.

$\underline{\mathsf{Fork}_2^{\mathcal{A}}(x)}$:
Pick the random coin ρ of $\mathcal{A}$ at random
$h_1, \ldots, h_q \leftarrow H$
$(I, J, \mathrm{Out}) \leftarrow \mathcal{A}(x, h_1, \ldots, h_q; \rho)$
If $I = \bot$ then return $\bot$
$h'_I \leftarrow H$
$(I', J', \mathrm{Out}') \leftarrow$
$\mathcal{A}(x, h_1, \ldots, h_{I-1}, h'_I, h_{I+1}, \ldots, h_q; \rho)$
If $I \neq I'$ or $J \neq J'$ then return $\bot$
Return $(I, J, \mathrm{Out}, \mathrm{Out}')$

Fig. 9. The forking algorithm build from $\mathcal{A}$.

Lemma 6. *There exists an adversary $\mathcal{B}$ for the* $\mathrm{AOMPR}^{\mathsf{LHF}}$ *making at most* $2\mathsf{q}_s$ *queries to* CHAL *such that*

$$\mathsf{Pr}[E_2] \leqslant n \cdot \mathsf{q}\sqrt{2(\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{B}) + 1/2^{\kappa})} \; .$$

Moreover, $\mathcal{B}$ runs in time roughly equal two times that of $\mathcal{A}$.

This completes the proof of the theorem, subject to proofs of the lemmas that we discuss next.

The proof of Lemma 5 is almost the same as Lemma 3, so we omit the full proof. The only difference is that $\mathcal{C}$ takes as input $h_1, \ldots, h_{(n+1)\mathsf{q}}$ in order to simulate all RO queries. For a RO query $\mathrm{H}_1(\mathsf{pk}, lr, i)$, $\mathcal{C}$ first enumerates all $i' \in [n]$ and assigns $h_{(\mathrm{ctr}_h - 1)(n+1)+i'}$ to $\mathrm{H}_1(\mathsf{pk}, lr, i')$. Then, $\mathcal{C}$ computes the nonce R for lr and assigns $h_{\mathrm{ctr}_h(n+1)}$ to $\mathrm{H}_2(\mathsf{pk}, lr.\mathsf{msg}, R)$ if it is not assigned any value yet. Similarly, for a new RO query $\mathrm{H}_1(\mathsf{pk}, M, R)$, its value is set to $h_{\mathrm{ctr}_h(n+1)}$. The rest follows by a similar analysis.

To prove Lemma 6, we need the following variant of the forking lemma.

Lemma 7. *Let $q \geqslant 1$ be an integer and H and Q be two sets. Let $\mathcal{A}$ be a randomized algorithm that on input $x, h_1, \ldots, h_q$ outputs a tuple (I, J, Out), where $I \in \{\bot\} \cup [1..q]$, $J \in Q$, and* Out *is a side output. Let* IG *be a randomized algorithm that generates x. The accepting probability of $\mathcal{A}$ is defined as*

$$\mathrm{acc}(\mathcal{A}) := \mathsf{Pr}_{x \leftarrow\!\$\, \mathsf{IG}, h_1, \ldots, h_q \leftarrow\!\$\, H}[(I, J, \mathrm{Out}) \leftarrow\!\$\, \mathcal{A}(x, h_1, \ldots, h_q) \;:\; I \neq \bot] \; .$$

Consider algorithm $\mathsf{Fork}_2^{\mathcal{A}}$ described in Figure 9. The accepting probability of $\mathsf{Fork}_2^{\mathcal{A}}$ is defined as

$$\mathrm{acc}(\mathsf{Fork}_2^{\mathcal{A}}) := \mathsf{Pr}_{x \leftarrow\!\$\, \mathsf{IG}}[\alpha \leftarrow\!\$\, \mathsf{Fork}^{\mathcal{A}}(x) \;:\; \alpha \neq \bot] \; .$$

Then, $\mathrm{acc}(\mathsf{Fork}_2^{\mathcal{A}}) \geqslant \mathrm{acc}(\mathcal{A})^2/(q \cdot |Q|)$.

Proof (of Lemma 6). We first construct an algorithm $\mathcal{C}$ following the syntax of the algorithm described in Lemma 7 and then construct $\mathcal{B}$ from $\mathsf{Fork}^{\mathcal{C}}$. The input of $\mathcal{C}$ consists of *par* that defines a linear hash function $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$ and uniform random elements $h_1, \ldots, h_{n \cdot q} \in \mathcal{S}_{\mathrm{hash}}$. Similarly to the proof of Lemma 3, $\mathcal{C}$ can oracles CHAL and PI oracle and at the beginning, start with, $\mathcal{C}$ makes $2\mathsf{q}_s$ queries to CHAL and denotes the challenges as U_1, V_1, …, U_{q_s}, $V_{\mathsf{q}_s} \in \mathcal{D}$. Then, $\mathcal{C}$ initializes all the states $\mathsf{st}_0, \ldots, \mathsf{st}_n$ as in the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$, and initializes the counters ctr_s,ctr_h to 0 and the function dt to an empty table. $\mathcal{C}$ also initializes ctrPP to an empty table, which are used

to record the counter corresponding to each token generated by honest parties. Then, $\mathcal{C}$ runs $\mathcal{A}$ with access to the oracles $\widetilde{\text{INIT}}, \widetilde{\text{PPO}}, \widetilde{\text{PSIGNO}}, \widetilde{\text{RO}}$, which are simulated as follows. In the following description, we use i to denote the index of parties, j to denote the index of $U_1, V_1, \ldots, U_{\mathsf{q}_s}, V_{\mathsf{q}_s}$, and k to denote the index of $h_1, \ldots, h_{n\cdot\mathsf{q}}$.

$\widetilde{\text{Init}}(CS)$: $\mathcal{C}$ initializes H to an empty table and samples $a_0, \ldots, a_{t-1}$ uniformly from $\mathcal{D}_{\text{key}}$. Define $f(x) := \sum_{i=0}^{t-1} a_i x^i$. Then, $\mathcal{C}$ sets $\mathsf{pk} \leftarrow \mathsf{F}(a_0)$, $\mathsf{pk}_i \leftarrow \mathsf{F}(f(\mathsf{x}_i))$ for $i \in [n]$, and $\mathsf{sk}_i \leftarrow f(i)$ for $i \in CS$. Finally, $\mathcal{C}$ returns pk,$\mathsf{aux} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, $\{\mathsf{sk}_i\}_{i\in CS}$.

$\widetilde{\text{RO}}$ query $\text{H}_1(x)$**:** If $\text{H}_1(x) \neq \bot$, $\mathcal{C}$ returns $\text{H}_1(x)$. Otherwise, $\mathcal{C}$ parses x as $(\widetilde{\mathsf{pk}}, lr, \tilde{i})$ for some $\tilde{i} \in [1..n]$. If the parsing fails or $\widetilde{\mathsf{pk}} \neq \mathsf{pk}$, $\mathcal{C}$ sets $\text{H}_1(x) \leftarrow\!\!\$\ \mathcal{S}_{\text{hash}}$ and returns $\text{H}_1(x)$. Otherwise, $\mathcal{C}$ increases ctr_h by 1 and sets $\text{H}_1(\mathsf{pk}, lr, i) \leftarrow h_{n(\text{ctr}_h-1)+i}$ for each $i \in [n]$. In addition, let $\text{mapLR}(\text{ctr}_h) := lr$. Then, $\mathcal{C}$ computes $R \leftarrow \sum_{i\in lr.\mathsf{SS}}(R_i + S_i^{d_i})$, where $(R_i, S_i) \leftarrow lr.\mathsf{PP}(i)$ and $d_i = \text{H}_1(\mathsf{pk}, lr, i)$. If $\text{H}_2(\mathsf{pk}, lr.\mathsf{msg}, R) = \bot$, $\mathcal{C}$ sets $\text{H}_2(\mathsf{pk}, lr.\mathsf{msg}, R) \leftarrow\!\!\$\ \mathbb{Z}_p$. Finally, $\mathcal{C}$ returns $\text{H}_1(x)$.

$\widetilde{\text{RO}}$ query $\text{H}_2(x)$**:** If $\text{H}_2(x) = \bot$, $\mathcal{C}$ sets $\text{H}_2(x) \leftarrow\!\!\$\ \mathcal{S}_{\text{hash}}$. Then, $\mathcal{C}$ returns $\text{H}_2(x)$.

$\widetilde{\text{PPO}}(i)$ query: Same as in the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$, except in the simulation of algorithm SPP, $\mathcal{C}$ first increases ctr_s by 1 and sets $pp \leftarrow (U_{\text{ctr}_s}, V_{\text{ctr}_s})$, $\mathsf{st}_i.\text{mapPP}(pp) \leftarrow (0,0)$, and $\text{ctrPP}(i, pp) \leftarrow \text{ctr}_s$.

$\widetilde{\text{PSignO}}(i, lr)$ query: Same as in the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$, except in the simulation of algorithm PS, if $\mathsf{st}_i.\text{mapPP}(pp) \neq \bot$, $\mathcal{C}$ computes

$$z_i \leftarrow \text{PI}\left(U_j + d_i V_j\right) + c \cdot \lambda_i^{lr.SS} \cdot f(i)\ ,$$

where $j \leftarrow \text{ctrPP}(i, pp)$. In addition, $\mathcal{C}$ sets $\mathsf{dt}(j) \leftarrow (k, d_i, z_i - c\lambda_i^{lr.\mathsf{SS}} \cdot f(i))$, where k denotes the index such that $\text{H}_1(\mathsf{pk}, lr, i)$ is set to h_k during the simulation.

After receiving the output $(m^*, \sigma^* = (R^*, z^*))$ from $\mathcal{A}$, $\mathcal{C}$ returns $(\bot, \bot, \bot)$ if E_2 does not occur. Otherwise, we know $\text{S}_2(lr^*) > 0$ and $\text{S}_2(lr^*) \neq \text{S}_3(lr^*)$. Therefore, there exists k^* and i^* such that $i^* \in \text{S}_3(lr^*)\backslash\text{S}_2(lr^*)$ and $\text{mapLR}(k^*) = lr^*$. (Since $\text{S}_2(lr^*) \subseteq \text{S}_3(lr^*)$, we must have $\text{S}_3(lr^*)\backslash\text{S}_2(lr^*) \neq \varnothing$.) Since $i^* \in \text{S}_3(lr^*)$, there exists $j^* \in [1..\mathsf{q}_s]$ such that $lr^*.\mathsf{PP}(i^*) = (U_{j^*}, V_{j^*})$. If $\mathsf{dt}(j^*) = \bot$, $\mathcal{C}$ sets $J \leftarrow \bot$. Otherwise, let $(k, d, z) \leftarrow \mathsf{dt}(j^*)$ and $\mathcal{C}$ sets $J = k$. Then, $\mathcal{C}$ returns $(n(k^*-1) + i^*, J, \text{Out})$, where Out consists of all variables received or generated by $\mathcal{C}$, including i^*, j^*, k^*, lr^*.

Analysis of $\mathcal{C}$. To use Lemma 7, we define IG as the algorithm that runs $\mathsf{PGen}(1^\kappa)$ and outputs par. The output J is either $\bot$ or in $[1..(n \cdot \mathsf{q})]$. It is not hard to see that $\mathcal{C}$ simulates the game TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$ perfectly, which implies $\text{acc}(\mathcal{C}) \geqslant \mathsf{Pr}[E_2]$, where $\mathsf{Pr}[E_2]$ refers to the probability in the original TS-SUF-3$^{\mathsf{FROST1\text{-}H}}$ game with $\mathcal{A}$ (as in the lemma statement). By Lemma 7,

$$\text{acc}(\mathsf{Fork}_2^{\mathcal{C}}) \geqslant \frac{\mathsf{Pr}[E_2]^2}{n \cdot \mathsf{q}(n \cdot \mathsf{q} + 1)} \leqslant \frac{\mathsf{Pr}[E_2]^2}{2n^2\mathsf{q}^2}\ .$$

Construct $\mathcal{B}$ from $\mathsf{Fork}^{\mathcal{C}}$. We now give a construct of the AOMPR adversary $\mathcal{B}$ using $\mathsf{Fork}^{\mathcal{C}}$, and the available Chal and PI oracles. To start with, $\mathcal{B}$ receives par from the AOMPR$^{\mathsf{LHF}}$ game and runs $\mathsf{Fork}^{\mathcal{C}}(par)$. All the Chal queries from the first execution of $\mathcal{C}$ are relayed by $\mathcal{B}$ to its own Chal oracle, and for all the Chal queries from the second execution of $\mathcal{C}$, $\mathcal{B}$ answers them with the same challenges as the first execution. All the PI queries from $\mathsf{Fork}^{\mathcal{C}}$ are relayed by $\mathcal{B}$ to its own PI oracle. Without loss of generality, we can assume all the challenges are different, since otherwise, $\mathcal{B}$

can solve them trivially. Denote the event $\mathsf{BadHash}$ as $h_I \neq h'_I$, where I are outputted by the first execution of $\mathcal{C}$. Since h_I, I are independent of h'_I, we know $\mathsf{Pr}[\mathsf{BadHash}] \leqslant 1/|\mathcal{S}_{\mathrm{hash}}| \leqslant 1/2^{\kappa}$. Then, we can conclude the proof with the following claim, which implies

$$\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{B}) \geqslant \mathrm{acc}(\mathsf{Fork}^{\mathcal{C}}_2) - \mathsf{Pr}[\mathsf{BadHash}] \geqslant \frac{\mathsf{Pr}[E_2]^2}{2n^2\mathsf{q}^2} - 1/2^{\kappa} \ .$$

Claim 7 *$\mathcal{B}$ can win the game* $\mathrm{AOMPR}_{\mathsf{LHF}}$*, if* $\mathsf{Fork}^{\mathcal{C}}$ *returns* $(I, \mathrm{Out}, \mathrm{Out}')$ *and* $\mathsf{BadHash}$ *does not occur.*

□

Proof (of Claim 7). We directly use the notations in the description of $\mathcal{C}$ to denote the variables in Out and use $(\cdot)'$ to denote the variables in Out′. The total number of the CHAL queries is $2\mathsf{q}_s$ and the corresponding challenges are $U_1, V_1, \ldots, U_{\mathsf{q}_s}, V_{\mathsf{q}_s}$.

We now show how to compute u_j, v_j for each $j \in [\mathsf{q}_s]$ such that $\mathsf{F}(u_i) = U_i$ and $\mathsf{F}(v_i) = V_i$. There are the following cases.

Case 0: Both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are $\perp$. In this case, $\mathcal{B}$ computes u_j, v_j by directly querying oracle $\mathrm{PI}(U_j)$ and $\mathrm{PI}(V_j)$.

Case 1: Exactly one of $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ is not $\perp$. Without loss of generality, assume $\mathsf{dt}(j) = (k, d, z)$, which implies $\mathsf{F}(z) = U_j + dV_j$. $\mathcal{B}$ computes v_j by directly querying oracle $\mathrm{PI}(V_j)$ and computes $u_j \leftarrow z - d \cdot v_j$.

For all the following cases, both $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ are not $\perp$ and we denote $(k, d, z) \leftarrow \mathsf{dt}(j)$ and $(k', d', z') \leftarrow \mathsf{dt}'(j)$.

Case 2: $d \neq d'$. In this case, $\mathcal{B}$ computes $v_j = \frac{z - z'}{d - d'}$, $u_j = z - d \cdot v_j$.

Case 3: $d = d'$. In this case, $\mathcal{B}$ computes v_j, u_j the same as Case 1. Also, since $d = d'$, $\mathcal{B}$ queries PI oracle only once in order to answer queries $\mathrm{PI}(U_j + dV_j)$ and $\mathrm{PI}(U_j + d'V_j)$ from $\mathsf{Fork}^{\mathcal{C}}$.

From the execution of $\mathcal{C}$, we know $\mathsf{dt}(j) = (k, d, z) \neq \perp$ if and only if $\mathcal{C}$ queries PI on $(U_j + dV_j)$. Therefore, denote $(U_j + dV_j)$ as the PI query associated with $\mathsf{dt}(j)$. For all the above cases, $\mathcal{B}$ queries PI oracle twice for simulating PI queries associated with $\mathsf{dt}(j)$ and $\mathsf{dt}'(j)$ and computing u_j, v_j.

We now show how to compute u_{j^*} and v_{j^*}. From the execution of $\mathsf{Fork}^{\mathcal{C}}_2$, we know $\mathsf{pk} = \mathsf{pk}'$ and $\mathrm{mapLR}(k) = \mathrm{mapLR}'(k)$ for all $k \leqslant I$, which implies $lr^* = \mathrm{mapLR}(I) = \mathrm{mapLR}'(I) = lr^{*\prime}$. Since E_2 occurs in both executions of $\mathcal{C}$, we know $\mathsf{SVf}(\mathsf{pk}, lr^*, (R^*, z^*)) = \mathtt{true}$ and $\mathsf{SVf}(\mathsf{pk}, lr^*, (R^{*\prime}, z^{*\prime})) = \mathtt{true}$ are valid. Therefore, $\mathsf{F}(z^*) = R^* + \mathsf{F}(a_0 c)$, $R^* = \sum_{i \in lr^*.\mathsf{SS}}(R_i + d_i S_i)$, $g^{z^{*\prime}} = R^{*\prime} + \mathsf{F}(a_0 c')$, $R^{*\prime} = \sum_{i \in lr^*.\mathsf{SS}}(R_i + d'_i S_i)$, where $(R_i, S_i) = lr.\mathsf{PP}(i)$, $c = \mathrm{H}_2(\mathsf{pk}, M^*, R^*)$, $c' = \mathrm{H}'_2(\mathsf{pk}, M^*, R^{*\prime})$, and $d_i = \mathrm{H}_1(\mathsf{pk}, lr^*, i)$, $d'_i = \mathrm{H}'_1(\mathsf{pk}, lr^*, i)$. Since for each $i \neq i^*$ we have $d_i = h_{n(k^*-1)+i} = d'_i$, we have

$$\mathsf{F}(z^* - z^{*\prime}) = R^* - R^{*\prime} + \mathsf{F}(a_0(c - c')) = (d_{i^*} - d'_{i^*})S_{i^*} + \mathsf{F}(a_0(c - c')) \ .$$

Therefore, $\mathcal{C}$ can compute $v_{j^*} = \frac{z^* - z^{*\prime} - a_0(c - c')}{d_{i^*} - d'_{i^*}}$. If $J = \perp$, $\mathcal{B}$ computes u_{j^*} by querying $\mathrm{PI}(U_{j^*})$ directly. In this case, $\mathcal{B}$ queries PI only once to compute u_{j^*} and v_{j^*}. If $J \neq \perp$, let $(k, d, z) \leftarrow \mathsf{dt}(j^*) =$ and $(k', d', z') \leftarrow \mathsf{dt}(j^*)$. Then, $\mathcal{B}$ computes $u_{j^*} = z - d \cdot v_{j^*}$. Since $i^* \notin S_2(lr^*)$, we know $k \neq I$.(Otherwise, suppose $k = I$. Since $I = n(k^* - 1) + i^*$ and $\mathrm{mapLR}(k^*) = lr^*$, we know a $\mathrm{PSIGNO}(i^*, lr^*)$ is made and does not return $\perp$ during the simulation, which implies $i^* \in S_2(lr^*)$.)

Game $\mathrm{DLog}^{\mathcal{A}}_{\mathsf{GGen}}(\kappa)$:	Game $\mathrm{RSA}^{\mathcal{A}}_{\mathsf{RGen}}(\kappa)$:
$(\mathbb{G}, p, g) \leftarrow\!\!\$\ \mathsf{GGen}(1^\kappa)$	$(N, e) \leftarrow\!\!\$\ \mathsf{RGen}(1^\kappa)$
$Z \leftarrow\!\!\$\ \mathbb{G}$	$w \leftarrow\!\!\$\ \mathbb{Z}_N^*$
$z \leftarrow\!\!\$\ \mathcal{A}(\mathbb{G}, p, g, Z)$	$u \leftarrow\!\!\$\ \mathcal{A}(N, e, w)$
If $g^z = Z$ then	If $u^e = w$ then
Return 1	Return 1
Return 0	Return 0

Fig. 10. The DLog game and the RSA game.

Thus, we have $k' = J = k \neq I$ and $d = h_J = d'$, which means $\mathcal{B}$ only needs to query PI once to simulate the PI queries associated with $\mathsf{dt}(j^*)$ and $\mathsf{dt}'(j^*)$. Therefore, the total number of PI queries made by $\mathcal{B}$ is equal to $2\mathsf{q}_s - 1$, which implies $\mathcal{B}$ wins the game $\mathrm{AOMDL}^{\mathsf{LHF}}$. □

5 Instantiations

5.1 Instantiations From the Discrete Logarithm Problem

Discrete Logarithm Problem The discrete logarithm problem is formalized by the DLog game defined in the left side of Figure 10. The group generation algorithm $\mathsf{GGen}(1^\kappa)$ outputs $(\mathbb{G}, p, g)$, where $\mathbb{G}$ is a cyclic group with prime size $p \geqslant 2^\kappa$ and generator g. The corresponding advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\mathrm{DLog}^{\mathcal{A}}_{\mathsf{GGen}} = 1\right]$.

Instantiation Following the instantiation from [HKL19], a linear hash function family GLHF is instantiated from a group generation algorithm GGen as follows.

- On input 1^κ, PGen runs $\mathsf{GGen}(1^\kappa)$ and receives a group description $(\mathbb{G}, p, g)$. Then, PGen uniformly samples $Z \in \mathbb{G}$ and returns $\kappa \leftarrow (\mathbb{G}, p, g, Z)$.
- Given $\kappa = (\mathbb{G}, p, g, Z)$, define $\mathcal{S} := \mathbb{Z}_p$, $\mathcal{D} := \mathbb{Z}_p^2$, $\mathcal{R} := \mathbb{G}$. Also, for any $(x_1, x_2) \in \mathbb{Z}_p^2$, define $\mathsf{F}(x_1, x_2) := g^{x_1} Z^{x_2}$.
- The operation over $\mathcal{D}$ is defined as follows. For any $(x_1, y_1), (x_2, y_2) \in \mathcal{D}$ and $s \in \mathcal{S}$, $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$ and $s \cdot (x_1, y_1) = (sx_1, sy_1)$.
- The operation over $\mathcal{R}$ is defined as follows. For any $x_1, x_2 \in \mathcal{R}$ and $s \in \mathcal{S}$, $x_1 + x_2 = x_1 x_2$, $s \cdot x_1 = x_1^s$, where $x_1 x_2$ and x_1^s are the group operations of $\mathbb{G}$.

The following theorem shows that GLHF is a linear hash function family and collision resistance of GLHF is implied by the discrete logarithm assumption. [HKL19] shows similar statements, and we defer the full proof to Appendix B.1.

Lemma 8. *For any group generation algorithm* GGen, $\mathsf{GLHF}[\mathsf{GGen}]$ *is a linear hash function family (Definition 1). Moreover, for any adversary* $\mathcal{A}$ *for the* $\mathrm{CR}^{\mathsf{GLHF}[\mathsf{GGen}]}$ *game, there exists an adversary* $\mathcal{B}$ *for the* $\mathrm{DLog}^{\mathsf{GGen}}$ *game such that* $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{GLHF}[\mathsf{GGen}]}(\mathcal{A}, \kappa) \leqslant \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}, \kappa)$.

To instantiate MuSig2-H, FROST1-H, and FROST2-H, we set $\mathcal{D}_{\mathrm{key}} := \{(x, 0) \ : \ x \in \mathbb{Z}\}$ and $\mathcal{S}_{\mathrm{hash}} := \mathcal{S}$. It is clear that $\mathrm{char}(\mathcal{S}) = p \geqslant 2^\kappa$, F is a bijection from $\mathcal{D}_{\mathrm{key}}$ to $\mathcal{R}$, and $|\mathcal{S}_{\mathrm{hash}}| = |\mathcal{S}| \geqslant 2^\kappa$. Also, for instantiating FROST1-H and FROST2-H, we set $\mathsf{x}_i := i$.

By combining Theorem 1 and Lemma 8 with the theorems in Section 4, we show the security of MuSig2-H, FROST1-H, and FROST2-H instantiated from GLHF under the discrete logarithm assumption in the random oracle model.

5.2 Instantiations from the RSA Problem

RSA Problem. The RSA problem we use here is formalized by the RSA game defined on the right side of Figure 10. The RSA parameter generation algorithm $\mathsf{RGen}(1^\kappa)$ outputs (N, e), where $N = P \cdot Q$ for two primes P and Q and e is a prime such that $\gcd(N, e) = \gcd(\phi(N), e) = 1$ such that $\phi(N) \geqslant 2^\kappa$ and $e \geqslant 2^\kappa$.[2] The corresponding advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathrm{rsa}}_{\mathsf{RGen}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\mathrm{RSA}^{\mathcal{A}}_{\mathsf{RGen}} = 1\right]$.

Instantiation. To instantiate linear hash function families from the RSA problem, we have to use a weaker notion, referred to as *weak linear hash functions*, which are the same as linear hash functions except that $\mathcal{S}$ is only required to be a ring instead of a field. Formally, we construct a weak linear hash function family, RLHF, from an RSA parameter generation algorithm RGen as follows.

- On input 1^κ, PGen runs $\mathsf{RGen}(1^\kappa)$ and receives (N, e). Then, PGen uniformly samples $w \in \mathbb{Z}^*_N$ and returns $par \leftarrow (N, e, w)$.
- Given $par = (N, e, w)$, define $\mathcal{S} := \mathbb{Z}$, $\mathcal{D} := \mathbb{Z}_e \times \mathbb{Z}^*_N$, $\mathcal{R} := \mathbb{Z}^*_N$. Also, for any $(a, x) \in \mathbb{Z}_e \times \mathbb{Z}^*_N$, define $\mathsf{F}(a, x) := w^a x^e \in \mathbb{Z}^*_N$.
- The operations of $\mathcal{D}$ are defined as follows. For any $(a_1, x_1), (a_2, x_2) \in \mathcal{D}$ and $s \in \mathcal{S}$, $(a_1, x_1) + (a_2, x_2) = (a_1 + a_2, x_1 x_2 w^{\lfloor (a_1+a_2)/e \rfloor})$ and $s \cdot (a_1, x_1) = (sa_1, x_1^s w^{\lfloor sa_1/e \rfloor})$, where $a_1 + a_2$ and sa_1 are computed over $\mathbb{Z}_e$.
- The operations of $\mathcal{R}$ are defined as follows. For any $x_1, x_2 \in \mathcal{R}$ and $s \in \mathcal{S}$, $x_1 + x_2 = x_1 x_2$, $s \cdot x_1 = x_1^s$, where $x_1 x_2$ is the multiplicative operation over $\mathbb{Z}^*_N$ and x_1^s is the exponential operation over $\mathbb{Z}^*_N$. Note here and also in the following discussion, we use "+" to denote the group operation of $\mathcal{R}$ instead of the additive operation over $\mathbb{Z}$ and "$\cdot$" to denote the scalar multiplicative operation of $\mathcal{R}$ instead of the multiplicative operation over $\mathbb{Z}$.

The preceding instantiation is similar to the one from [HKL19]. The only difference is that we set $\mathcal{S}$ to $\mathbb{Z}$ in order to make both $\mathcal{D}$ and $\mathcal{R}$ to be $\mathcal{S}$-modules. The following theorem shows that RLHF is a weak linear hash function family and collision resistance of RLHF is implied by the RSA assumption. We defer the proof to Appendix B.2.

Lemma 9. *For any RSA parameter generation algorithm* RGen, $\mathsf{RLHF}[\mathsf{RGen}]$ *is a weak linear hash function family. Moreover, for any adversary* $\mathcal{A}$ *for the* $\mathrm{CR}^{\mathsf{RLHF}[\mathsf{RGen}]}$ *game, there exists an adversary* $\mathcal{B}$ *for the* $\mathrm{RSA}^{\mathsf{RGen}}$ *game such that* $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{RLHF}[\mathsf{RGen}]}(\mathcal{A}, \kappa) \leqslant \mathsf{Adv}^{\mathrm{rsa}}_{\mathsf{RGen}}(\mathcal{B}, \kappa)$.

Reduction from CR to AOMPR Unfortunately, Theorem 1 does not hold for weak linear hash functions: in the proof of Claim 1, if $\mathcal{S}$ is not a field, it is possible that there does not exist $\boldsymbol{u}$ satisfying the condition in (2). Nonetheless, we can show for RLHF that the reduction still works. Formally, we have the following theorem.

Theorem 5. *For any adversary* $\mathcal{A}$ *for the* $\mathrm{AOMPR}^{\mathsf{RLHF}}$ *game, there exists an adversary* $\mathcal{B}$ *for the* $\mathrm{CR}^{\mathsf{RLHF}}$ *game running in a similar running time as* $\mathcal{A}$ *such that* $\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{RLHF}}(\mathcal{A}, \kappa) \leqslant 2\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{RLHF}}(\mathcal{B}, \kappa)$.

Proof (of Theorem 5). We prove the above theorem following the proof of Theorem 1, where the only difference is that in the proof of Claim 1, we need to show the following fact:

[2] Comparing this to the plain RSA problem, here we additionally require that e is prime such that $\gcd(N, e) = 1$ and $e \geqslant 2^\kappa$.

There exists $z^* \in \mathcal{D}$ such that $\mathsf{F}(z^*) = 0$, and, for any matrix $B \in \mathcal{S}^{\ell \times q}$ with $\ell < q$, there exists a vector $\boldsymbol{u} \in \mathcal{S}^q$ and $i \in [q]$ such that $B\boldsymbol{u} = 0$ and $u_i z^* \neq 0$, where 0 denotes the identity of $\mathcal{D}$ and $\mathcal{R}$ and the additive identity of $\mathcal{S}$.

We prove the above fact for RLHF as follows. Given the parameter (N, e, w) that defines $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$, the identity of $\mathcal{D}$ is $(0, 1)$, and the identity of $\mathcal{R}$ is 1. We first set $z^* = (e - 1, w^{1-1/e})$, where $1/e$ denotes the multiplicative inverse of e over $\mathbb{Z}_{\phi(N)}$. $1/e$ exists since $\gcd(\phi(N), e) = 1$. We can verify that $\mathsf{F}(z^*) = w^{e-1+e(1-1/e)} = 1$. Since $\ell < q$, we can always find a non-zero vector $\boldsymbol{v} \in \mathbb{Z}$ such that $B\boldsymbol{v} = 0$ using Gaussian eliminations. Denote $k := \gcd(\{v_i\}_{i\in[q]})$. Let $\boldsymbol{u} = \boldsymbol{v}/k$, and we have $\gcd(\{u_i\}_{i\in[q]}) = 1$. Therefore, there exists $i \in [q]$ such that $u_i \not\equiv 0 \bmod e$ and thus $u_i z^* \neq (0, 1)$. Since $B\boldsymbol{u} \cdot k = B\boldsymbol{v} = 0$ and $k \neq 0$, we know $B\boldsymbol{u} = 0$. □

Solving Linear Equations. Another issue with weak linear hash functions is that it is unclear how to invert challenges $\boldsymbol{X} \in \mathcal{R}$ given $A\boldsymbol{X} = \mathsf{F}(\boldsymbol{b})$, where $A \in \mathcal{S}^{n\times n}$ and $\boldsymbol{b} \in \mathcal{D}^n$, which is a common problem we encounter in the security proofs in Section 4. In these proofs, to solve this problem, we show A has full rank and then, since $\mathcal{S}$ is a field, we can compute $\boldsymbol{x} \in \mathcal{D}^n$ such that $\mathsf{F}(\boldsymbol{x}) = \boldsymbol{X}$ by multiplying the inverse of A on both sides of the equation. However, in the case of weak linear hash functions, A might not have an inverse.

Fortunately, for RLHF, we show that such linear equations can be solved efficiently if A has full rank modulo e, which is formally stated in the following lemma.

Lemma 10. *For any integer $n \geqslant 1$ and any parameter $par = (N, e, w)$ for RLHF, which defines $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$, given $A \in \mathcal{S}^{n\times n}$, $\boldsymbol{X} \in \mathcal{R}^n$, and $\boldsymbol{b} \in \mathcal{D}^n$ such that A has full rank modulo e and $A\boldsymbol{X} = \mathsf{F}(\boldsymbol{b})$, there exists an efficient algorithm with input $(A, \boldsymbol{X}, \boldsymbol{b})$ that outputs $\boldsymbol{x} \in \mathcal{D}^n$ such that $\mathsf{F}(x_i) = X_i$.*

Proof. We compute $\boldsymbol{x}$ as follows.

1. Since A has full rank modulo e and e is a prime, we can efficiently compute the inverse of A modulo e as A'.
2. Set $C \leftarrow A'A$. Since A' is the inverse of A modulo e, we know for any $i, j \in [n]$, $C_{i,j} \equiv \begin{cases} 1 \bmod e, & \text{for } i = j \\ 0 \bmod e, & o.w. \end{cases}$.
3. Set $\boldsymbol{b}' \leftarrow A'\boldsymbol{b}$ and $x_i \leftarrow b'_i - \sum_{j\in[n]} \lfloor C_{i,j}/e \rfloor \cdot (0, X_j)$ for each $i \in [n]$.

Since $A\boldsymbol{X} = \mathsf{F}(\boldsymbol{b})$, we have $C\boldsymbol{X} = A'A\boldsymbol{X} = A'\mathsf{F}(\boldsymbol{b}) = \mathsf{F}(A'\boldsymbol{b}) = \mathsf{F}(\boldsymbol{b}')$, which implies $\mathsf{F}(b'_i) = \sum_{j\in[n]} C_{i,j}X_j = \prod_{j\in[n]} X_j^{C_{i,j}}$. Therefore, due to the above property of C, for $i \in [n]$, $\mathsf{F}(x_i) = \mathsf{F}(b'_i) - \sum_{j\in[n]} X_j^{e\lfloor C_{i,j}/e \rfloor} = \prod_{j\in[n]} X_j^{C_{i,j} - e\lfloor C_{i,j}/e \rfloor} = X_i$. □

$\mathcal{D}_{\text{key}}$ and $\mathcal{S}_{\text{hash}}$. For instantiating $\mathsf{MuSig2\text{-}H}$, $\mathsf{FROST1\text{-}H}$, and $\mathsf{FROST2\text{-}H}$ from RLHF, we set $\mathcal{D}_{\text{key}} := \{(0, x) \mid x \in \mathbb{Z}_N^*\}$ and $\mathcal{S}_{\text{hash}} := \mathbb{Z}_{2^\kappa}$. It is clear that F is bijection from $\mathcal{D}_{\text{key}}$ to $\mathcal{R}$ and $|\mathcal{S}_{\text{hash}}| \geqslant 2^\kappa$.

5.3 Multi-signatures from RSA

To instantiate $\mathsf{MuSig2\text{-}H}$ from RLHF, we additionally require that for $N = P \cdot Q$, P is a safe prime and $P > 2^{\kappa+1}$ for the security proof to go through. We discuss how to remove this requirement later in this section. To show the security, we prove Theorem 2 holds if LHF is replaced by RLHF.

Combining it with Theorem 5 and Lemma 9 shows the security of RLHF-based MuSig2-H under the RSA assumption in the random oracle model.

We now show the proof of Theorem 2 for the case $\mathsf{LHF} = \mathsf{RLHF}$ by discussing only those places that differ from the original proof of Theorem 2.

Proof (of Theorem 2 for RLHF*).* We follow the original proof of Theorem 2 to construct the adversary $\mathcal{B}$. Then, we just need to show that Claim 3, Claim 4, and Claim 5 hold.

Proof (of Claim 3 for RLHF*).* We only need to show that Lemma 2 holds for RLHF, and the rest is the same as the original proof of Claim 3. Denote $r \in \mathbb{Z}_P^*$ as the primitive root of $\mathbb{Z}_P^*$. For any $X \in \mathbb{Z}_N^* = \mathcal{R}$, there exists $k \in \mathbb{Z}_{P-1}^*$ such that $X \equiv r^k \mod P$. Suppose $k \neq P'$. For any $1 \leqslant t, s \leqslant 2^\kappa < P'$ and any $1 \leqslant s < P'$, we have $(X)^{ts} \equiv r^{kts} \not\equiv r^0 \mod P$, which implies $(X^*)^{t \cdot s_1} \neq (X^*)^{t \cdot s_2}$ for any distinct $s_1, s_2 \in \mathbb{Z}_{2^\kappa} = \mathcal{S}_{\mathrm{hash}}$. Therefore, we have $|C(t, X)| = |\mathcal{S}_{\mathrm{hash}}|$. Therefore, X is Good if $X \not\equiv r^{P'} \mod P$. Therefore, we have $\mathsf{Pr}_{X \leftarrow\!\$\, \mathcal{R}}[X \text{ is not } \mathsf{Good}] \leqslant \mathsf{Pr}_{X \leftarrow\!\$\, \mathbb{Z}_N^*}[X \equiv r^{P'} \mod P] \leqslant 1/(P-1) \leqslant 1/2^\kappa$. □

Proof (of Claim 4 for RLHF*).* Following the original proof of Claim 4, we have $\mathsf{F}(s^*) = R^* + h_I \cdot \mathsf{apk}^*$ and $\mathsf{F}(s^{*\prime}) = R^* + h_I' \mathsf{apk}^*$, which implies $(h_I - h_I') \cdot \mathsf{apk}^* = \mathsf{F}(s^* - s^{*\prime})$. Assume $h_I' < h_I$ without loss of generality. Since $h_I, h_I' \in \mathcal{S}_{\mathrm{hash}} = \mathbb{Z}_{2^\kappa} \subseteq \mathbb{Z}_e$, we have $1 \leqslant h_I - h_I' < e$. Therefore, $\mathcal{C}'$ computes $\widetilde{x}$ using Lemma 10 for the case $n = 1$. □

Proof (of Claim 5 for RLHF*).* The total number of CHAL queries made by $\mathcal{B}$ is $4\mathsf{q}_s + 1$ and the corresponding challenges are $X, U_1, \ldots, U_{4\mathsf{q}_s}$. We follow the original proof to show how $\mathcal{B}$ computes $x^*, u_1, \ldots, u_{4\mathsf{q}_s}$ such that $\mathsf{F}(x^*) = X$ and $\mathsf{F}(u_i) = U_i$ for $i \in [4\mathsf{q}_s]$.

To compute x^*, following the original proof, we have $\mathsf{F}(\widetilde{x}) = t \cdot h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} \cdot X + Z$, $\mathsf{F}(\widetilde{x}') = t \cdot h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime} \cdot X + Z$, where $h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} \neq h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime} \in \mathcal{S}_{\mathrm{hash}} = \mathbb{Z}_{2^\kappa}$, $1 \leqslant t \leqslant 2^\kappa$, and $Z \in \mathcal{R}$. Therefore, we have $t(h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} - h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime}) \cdot X = \mathsf{F}(\widetilde{x} - \widetilde{x}')$. Assume $h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime} < h_{I_{\mathrm{agg}}}^{(\mathrm{agg})}$ without loss of generality. We have $1 \leqslant t \leqslant 2^\kappa < e$ and $1 \leqslant (h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} - h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime}) \leqslant 2^\kappa < e$, which implies $t(h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} - h_{I_{\mathrm{agg}}}^{(\mathrm{agg})\prime}) \not\equiv 0 \mod e$. Therefore, $\mathcal{B}$ computes x^* using Lemma 10 for the case $n = 1$.

For each $k \in [\mathsf{q}_s]$, to compute $u_{1+4(k-1)}, \ldots, u_{4k}$, following the original proof, we have $A\boldsymbol{U} = \mathsf{F}(\boldsymbol{s})$, where

$$A = \begin{pmatrix} 1 & b_1 & b_1^2 & b_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & b_4 & b_4^2 & b_4^3 \end{pmatrix}, \ \boldsymbol{U} = \begin{pmatrix} U_{1+4(k-1)} \\ \vdots \\ U_{4k} \end{pmatrix}, \ \boldsymbol{s} = \begin{pmatrix} s_1 \\ \vdots \\ s_4 \end{pmatrix}. \tag{7}$$

Also, $b_i \in \mathcal{S}_{\mathrm{hash}} = \mathbb{Z}_{2^\kappa} \subseteq \mathbb{Z}_e$ for $i \in [4]$, and $b_1, \ldots, b_4$ differ from each other. Therefore, A is a Vandermonde matrix modulo e, which implies A has full rank modulo e. Therefore, $\mathcal{B}$ can compute $u_{1+4(k-1)}, \ldots, u_{4k}$ using Lemma 10 for the case $n = 4$. Then, the rest follows from the original proof. □

REMOVING THE SAFE-PRIME REQUIREMENT. We briefly mention how to remove the safe-prime requirement by slightly modifying MuSig2-H as follows. Denote the modified schemes as MuSig2-HR. MuSig2-HR is identical to MuSig2-H except:

- In algorithm $\mathsf{KeyAgg}(L)$, it additionally computes $a_0 \leftarrow \mathrm{H}'(L)$, where $\mathrm{H}'(L) : \{0,1\}^* \to \mathcal{D}_{\mathrm{key}}$, and sets $\mathsf{apk} \leftarrow \mathsf{F}(a_0) + \sum_{i \in [n]} a_i \mathsf{pk}_i$.

- In algorithm Sign, after s is assigned, it additionally computes $a_0 \leftarrow c \cdot \mathrm{H}'(L)$ and returns (R, a_0, s).
- In algorithm $\mathsf{SignAgg}(\{(R^{(1)}, a_0^{(1)}, s^{(1)}), \ldots, (R^{(n)}, a_0^{(n)}, s^{(n)})\})$, it checks if $(R^{(1)}, a_0^{(1)}), \ldots, (R^{(n)}, a_0^{(n)})$ are all the same. If not, it aborts. Otherwise, it returns $\sigma \leftarrow (R^{(1)}, a_0^{(1)} + \sum_{i\in[n]} s^{(i)})$.

We can show the security of $\mathsf{MuSig2\text{-}HR}$ following the proof of Theorem 2 for RLHF. The only difference is the proof of Claim 3, which is also the only place where we need the safe-prime condition. Claim 3 essentially shows that for any new RO query $\mathrm{H}_{\mathrm{agg}}(L, \widetilde{\mathsf{pk}})$, the probability that $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(L)$ collides with the set K of existing aggregated keys is small. We can easily show it for $\mathsf{MuSig2\text{-}HR}$ since, for any new L in the random oracle model, $\mathrm{H}'(L)$ is uniformly random over $\mathcal{D}_{\mathrm{key}}$; thus, $\mathsf{apk} \leftarrow \mathsf{KeyAgg}(L)$ is uniformly random over $\mathcal{R}$ even given previous queries, which implies the collision probability is small.

5.4 Threshold Signatures from RSA

To instantiate $\mathsf{FROST1\text{-}H}$ and $\mathsf{FROST2\text{-}H}$ from RLHF, the only difficulty is that the Lagrange coefficient λ_i^S might not be defined in $\mathcal{S} = \mathbb{Z}$ for $S \subseteq [n]$. To fix this, we set $\mathsf{x}_i = i$ for $i \in [n]$ and modify the schemes as follows.

Denote the modified schemes as $\mathsf{FROST1\text{-}HR}$ and $\mathsf{FROST2\text{-}HR}$. Define $\widetilde{\lambda}_i^S := r\Delta \cdot \lambda_i^{lr.\mathsf{SS}}$, where $\Delta = n!$ and $r \in \mathbb{Z}_e^*$ is the multiplicative inverse of Δ modulo e. $\mathsf{FROST1\text{-}HR/FROST2\text{-}HR}$ is identical to $\mathsf{FROST1\text{-}H/\ FROST2\text{-}H}$ except:

- In algorithm PS, the Lagrange coefficient λ_i^S is replaced by $\widetilde{\lambda}_i^S$, and (R, c, z_i) is returned as a partial signature.
- In algorithm Agg, we additionally set $\widetilde{z} \leftarrow z - (ck) \cdot (0, \mathsf{pk})$, where $k = \lfloor r\Delta/e \rceil$, and return $(R, \widetilde{z})$ as the signature.

It is not hard to show the correctness of the schemes. Since the denominator of λ_i^S, which is equal to $\prod_{j\in S}(i-j)$, divides $i!(n-i)!$ and thus divides Δ, we know $\widetilde{\lambda}_i^S \in \mathbb{Z}$. Also, for a leader request lr, if each signer i in $lr.\mathsf{SS}$ follows the protocol to compute the partial signature (R, c, z_i), we have $\mathsf{F}(z) = R + (cr\Delta) \cdot \mathsf{pk}$, where $z = \sum_{i\in lr.\mathsf{SS}} z_i$. Since r is the multiplicative inverse of Δ modulo e, we have $r\Delta = ke + 1$. Since $\mathsf{F}(0, \mathsf{pk}) = \mathsf{pk}^e$, we have $\mathsf{F}(\widetilde{z}) = R + c \cdot \mathsf{pk}$, which implies $(R, \widetilde{z})$ is a valid signature.

We show the security of $\mathsf{FROST2\text{-}HR}$ and $\mathsf{FROST1\text{-}HR}$ under the RSA assumption in the random oracle model by showing Theorem 3 and Theorem 4 hold for RLHF and combining them with Theorem 5 and Lemma 9. We now show the proof of Theorem 3 and Theorem 4 for the case $\mathsf{LHF} = \mathsf{RLHF}$ by discussing only those places that differ from the original proofs.

Proof (of Theorem 3 for RLHF). We construct $\mathcal{B}$ following the proof of Theorem 4, except the Lagrange coefficient $\lambda_i^{lr.\mathsf{SS}}$ is replaced by $\widetilde{\lambda}_i^{lr.\mathsf{SS}}$. We only need to show Claim 6 holds. The rest follows from the original proof of Theorem 3.

Proof (of Claim 6 for RLHF). The total number of the $\textsc{Chal}$ queries made by $\mathcal{B}$ is $t + 2\mathsf{q}_s$ and the corresponding challenges are $A_0, \ldots, A_{t-1}, U_1, V_1, \ldots, U_{\mathsf{q}_s}, V_{\mathsf{q}_s}$. We follow the original proof to show how $\mathcal{B}$ computes $a_0, \ldots, a_{t-1}, u_1, \ldots, u_{\nu\mathsf{q}_s}$ such that $\mathsf{F}(a_i) = A_i$ for $i \in [t]$ and $\mathsf{F}(x^*) = X$ and $\mathsf{F}(u_i) = U_i$ for $i \in [\nu\mathsf{q}_s]$ and only mention the parts that are different.

To compute a_0, following the original proof, we have $\mathsf{F}(z^* - z^{*\prime}) = (h_I - h'_I)A_0$. Since $h_I \neq h'_I$ and $h_I, h'_I \in \mathcal{S}_{\text{hash}} = \mathbb{Z}_{2^\kappa} \subseteq \mathbb{Z}_e$, we have $h_I - h'_I \not\equiv 0 \bmod e$. Therefore, $\mathcal{B}$ computes a_0 using Lemma 10 for the case $n = 1$.

Also, for each $i \in D$, we have $\mathsf{F}(z - z') = \widetilde{\lambda}_{i'}^{lr*,\mathsf{SS}}(h_I - h'_I) \cdot \mathsf{pk}_i$. Since $\widetilde{\lambda}_{i'}^{lr*,\mathsf{SS}} \not\equiv 0 \bmod e$, $\mathcal{B}$ computes y_i such that $\mathsf{F}(y_i) = \mathsf{pk}_i$ using Lemma 10 for the case $n = 1$.

Then, following the original proof, we have (6). Since $\mathsf{x}_i = i < e$, M is Vandermonde matrix modulo e and thus M has full rank modulo e. Therefore, $\mathcal{B}$ can compute $a_1, \ldots, a_{t-1}$ by Lemma 10 for the case $n = t - 1$.

For computing $u_1, v_1, \ldots, u_{\mathsf{q}_\mathsf{s}}, v_{\mathsf{q}_\mathsf{s}}$, the only difference is in case 2. For computing v_j, we have $\mathsf{F}(z) = U_j + dV_j + c \cdot \mathsf{pk}_i$ and $\mathsf{F}(z') = U_j + d'V_j + c' \cdot \mathsf{pk}_{i'}$. Therefore, $\mathsf{F}(z - z' - c \cdot y_i + c' \cdot y_i) = (d - d')V_j$. Since $d \neq d'$ and $d, d' \in \mathbb{Z}_e$, $\mathcal{B}$ computes v_j using Lemma 10 for the case $n = 1$. The rest follows from the original proof. □

Proof (of Theorem 4 for RLHF*).* We construct $\mathcal{B}$ following the original proof of Theorem 4, except the Lagrange coefficient $\lambda_i^{lr,\mathsf{SS}}$ is replaced by $\widetilde{\lambda}_i^{lr,\mathsf{SS}}$. It is not hard to see Claim 7 holds using Lemma 10 and similarly as the proof of Claim 6 in the case of RLHF. The rest follows from the original proof of Theorem 4.

□

Acknowledgments

We thank the EUROCRYPT 2023 reviewers for their useful comments and feedback. This research was partially supported by NSF grants CNS-2026774, CNS-2154174, a JP Morgan Faculty Award, a CISCO Faculty Award, and a gift from Microsoft.

References

AA05. Sattar J. Aboud and Mohammad Ahmed Al-Fayoumi. Two efficient RSA digital multisignature and blind multisignature schemes. In M. H. Hamza, editor, *IASTED International Conference on Computational Intelligence, Calgary, Alberta, Canada, July 4-6, 2005*, pages 359–362. IASTED/ACTA Press, 2005.

ADN06. Jesús F. Almansa, Ivan Damgård, and Jesper Buus Nielsen. Simplified threshold RSA with adaptive and proactive security. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 593–611. Springer, Heidelberg, May / June 2006.

BBSS18. Matilda Backendal, Mihir Bellare, Jessica Sorrell, and Jiahao Sun. The fiat-shamir zoo: relating the security of different signature variants. In *Nordic Conference on Secure IT Systems*, pages 154–170. Springer, 2018.

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.

BCK+22. Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, Heidelberg, August 2022.

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

BJ10. Ali Bagherzandi and Stanislaw Jarecki. Identity-based aggregate and multi-signature schemes based on RSA. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 480–498. Springer, Heidelberg, May 2010.

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

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

BN07. Mihir Bellare and Gregory Neven. Identity-based multi-signatures from RSA. In Masayuki Abe, editor, *CT-RSA 2007*, volume 4377 of *LNCS*, pages 145–162. Springer, Heidelberg, February 2007.

BNPS03. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003.

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

BR93. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

BTZ22. Mihir Bellare, Stefano Tessaro, and Chenzhi Zhu. Stronger security for non-interactive threshold signatures: Bls and frost. *Cryptology ePrint Archive*, 2022.

CKGW22. Deirdre Connolly, Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. Two-Round Threshold Schnorr Signatures with FROST. Internet-Draft draft-irtf-cfrg-frost-10, Internet Engineering Task Force, September 2022. Work in Progress.

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

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.

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, Heidelberg, August 1988.

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

DF92. Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures (extended abstract). In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 457–469. Springer, Heidelberg, August 1992.

DK01. Ivan Damgård and Maciej Koprowski. Practical threshold RSA signatures without a trusted dealer. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 152–165. Springer, Heidelberg, May 2001.

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

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

FS01. Pierre-Alain Fouque and Jacques Stern. Fully distributed threshold RSA under standard assumptions. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 310–330. Springer, Heidelberg, December 2001.

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

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

GJKR96. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 157–172. Springer, Heidelberg, August 1996.

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.

HK89. L Harn and T Kiesler. New scheme for digital multisignatures. *Electronics letters*, 25(15):1002–1003, 1989.

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

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

Ita83a. K. Itakura. A public-key cryptosystem suitable for digital multisignatures. 1983.

Ita83b. K Itakura, K; Nakamura. A public-key cryptosystem suitable for digital multisignatures. *NEC research & development*, 1983.

KG20. Chelsea Komlo and Ian Goldberg. Frost: flexible round-optimized schnorr threshold signatures. In *International Conference on Selected Areas in Cryptography*, pages 34–65. Springer, 2020.

KH90. T Kiesler and L Harn. Rsa blocking and multisignature schemes with no bit expansion. *Electronics letters*, 18(26):1490–1491, 1990.

KM07. Neal Koblitz and Alfred J. Menezes. Another look at "provable security". *Journal of Cryptology*, 20(1):3–37, January 2007.

KM08. Neal Koblitz and Alfred Menezes. Another look at non-standard discrete log and diffie-hellman problems. *J. Math. Cryptol.*, 2(4):311–326, 2008.

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

LK22. Kwangsu Lee and Hyoseung Kim. Two-round multi-signatures from okamoto signatures. Cryptology ePrint Archive, Report 2022/1117, 2022. https://eprint.iacr.org/2022/1117.

MM00. Shirow Mitomi and Atsuko Miyaji. A multisignature scheme with message flexibility, order flexibility and order verifiability. In *Australasian Conference on Information Security and Privacy*, pages 298–312. Springer, 2000.

MO$^+$00. Masahiro Mambo, Eiji Okamoto, et al. On the security of the rsa-based multisignature scheme for various group structures. In *Australasian Conference on Information Security and Privacy*, pages 352–367. Springer, 2000.

Natnt. National Institute of Standards and Technology. Multi-Party Threshold Cryptography, 2018–Present. https://csrc.nist.gov/Projects/threshold-cryptography.

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, Heidelberg.

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.

Oka88. Tatsuaki Okamoto. A digital multisignature scheme using bijective public-key cryptosystems. *ACM Transactions on Computer Systems (TOCS)*, 6(4):432–441, 1988.

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, Heidelberg, August 1993.

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

PLL02. Shun-Fu Pon, Erl-Huei Lu, and Jau-Yien Lee. Dynamic reblocking rsa-based multisignatures scheme for computer and communication networks. *IEEE Communications Letters*, 6(1):43–44, 2002.

PPKW97. Sangjoon Park, Sangwoo Park, Kwangjo Kim, and Dongho Won. Two efficient rsa multisignature schemes. In *International Conference on Information and Communications Security*, pages 217–222. Springer, 1997.

PW23. Jiaxin Pan and Benedikt Wagner. Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. *EUROCRYPT 2023*, 2023.

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

Sch90. Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, August 1990.

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

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

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

Supplementary Materials

A Correctness of FROST1-H and FROST2-H

For the correctness of the schemes, we just need to show Shamir's secret sharing scheme works over $\mathcal{D}$. More precisely, we want to show for any $S \subseteq [n]$ with size t, we have

$$\sum_{i \in S} \lambda_i^S \mathsf{sk}_i = a_0 \ , \tag{8}$$

where

$$\mathsf{sk}_i = \sum_{j=0}^{t-1} a_j \mathsf{x}_i^j \ , \ \lambda_i^S = \prod_{j \in S \setminus \{i\}} \frac{\mathsf{x}_j}{\mathsf{x}_j - \mathsf{x}_i},$$

for each $i \in S$ and some $a_0, \ldots, a_{t-1} \in \mathcal{D}$.

Claim 8 *For any $S \subseteq [n]$ with size t and any integer $0 \leqslant k < t$, we have*

$$\sum_{i \in S} \lambda_i^S \mathsf{x}_i^k = \begin{cases} 1, & k = 0 \\ 0, & k \geqslant 0 \end{cases} . \tag{9}$$

With the above claim, we can show (8) since

$$\begin{aligned} \sum_{i \in S} \lambda_i^S \mathsf{sk}_i &= \sum_{i \in S} \lambda_i^S \sum_{j=0}^{t-1} a_j \mathsf{x}_i^j \\ &= \sum_{j=0}^{t-1} a_j \sum_{i \in S} \lambda_i^S \mathsf{x}_i^j \\ &= \sum_{j=0}^{t-1} a_j \cdot \mathbf{1}\{j = 0\} = a_0 \ . \end{aligned}$$

Proof (of Claim 8). Define $f(x) = x^k$. By the polynomial interpolation over the field $\mathcal{S}$, we have

$$\sum_{i \in S} \lambda_i^S f(\mathsf{x}_i) = f(0) \ ,$$

which proves claim. □

B Proofs for the Instantiations of Linear Hash Functions

B.1 Proof of Lemma 8

Given $par = (\mathbb{G}, p, g, Z)$ output from $\mathsf{PGen}(1^\kappa)$ that defines $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$, we need to show

1. $\mathcal{D}$ and $\mathcal{R}$ are $\mathcal{S}$-modules.
2. F is an epimorphism from $\mathcal{D}$ to $\mathcal{R}$ but not a monomorphism.
3. The collision resistance of RLHF is implied by the RSA assumption.

PART 1. It is clear that $\mathcal{D} = \mathbb{G} \times \mathbb{G}$ and $\mathcal{R} = \mathbb{G}$ are $\mathbb{Z}_p$-modules.

PART 2. It is easy to verify F is a homomorphism of $\mathcal{S}$-modules, since for any $b \in \mathcal{S}$ and $(x_1, y_1), (x_2, y_2) \in \mathcal{D}$,

$$\begin{aligned}\mathsf{F}((x_1, y_1) + b \cdot (x_2, y_2)) &= \mathsf{F}(x_1 + bx_1, y_1 + by_2) \\ &= g^{x_1 + bx_2} Z^{y_1 + by_2} \\ &= g^{x_1} Z^{x_1} (g^{x_2} Z^{y_2})^b \\ &= \mathsf{F}(x_1, y_1) + b\mathsf{F}(x_2, y_2) \ .\end{aligned}$$

F is epimorphism since for any $X \in \mathcal{R}$, we have $\mathsf{F}(x, 0) = X$, where x denotes the discrete log of X to the base g. Denote z as the discrete log of Z to base g, and F is not a monomorphism, since $\mathsf{F}(z, -1) = g^z Z^{-1} = g^0$.

PART 3. For any adversary $\mathcal{A}$ for the $\mathrm{CR}^{\mathsf{GLHF}}$ game, we construct $\mathcal{B}$ for the DLog game as follows. After receiving $(\mathbb{G}, p, g, Z)$, $\mathcal{B}$ runs $\mathcal{A}$ with input $(\mathbb{G}, p, g, Z)$. If $\mathcal{A}$ wins the $\mathrm{CR}^{\mathsf{GLHF}}$ by outputting $(x_1, y_1), (x_2, y_2) \in \mathcal{D}$, such that $(x_1, y_1) \neq (x_2, y_2)$ and $\mathsf{F}(x_1, y_1) = \mathsf{F}(x_2, y_2)$, we have $g^{x_1} Z^{y_1} = g^{x_2} Z^{y_2}$. Therefore, $\mathcal{B}$ can compute $z = \frac{y_2 - y_1}{x_1 - x_2}$ and we have $g^z = Z$.

B.2 Proof of Lemma 9

Given $par = (N, e, w)$ output from $\mathsf{PGen}(1^\kappa)$ that defines $(\mathcal{S}, \mathcal{D}, \mathcal{R}, \mathsf{F})$, we need to show

1. $\mathcal{D}$ and $\mathcal{R}$ are $\mathcal{S}$-modules.
2. F is an epimorphism from $\mathcal{D}$ to $\mathcal{R}$ but not a monomorphism.
3. The collision resistance of RLHF is implied by the RSA assumption.

PART 1. It is clear that $\mathcal{R} = \mathbb{Z}_N^*$ is a $\mathbb{Z}$-module. For $\mathcal{D}$, it is not hard to see $\mathcal{D}$ are abelian groups. The unit of $\mathcal{D}$ is $(0, 1)$ and for any $(a, x) \in \mathcal{D}$, its inverse is $(-a, x^{-1})$. $\mathcal{R}$ is an abelian group since $\mathbb{Z}_N^*$ is an abelian group.

We show $\mathcal{D}$ is $\mathcal{S}$-module, since for any $b_1, b_2 \in \mathcal{S}$ and $(a_1, x_1), (a_2, x_2) \in \mathcal{D}$,

$$\begin{aligned}b_1 \cdot ((a_1, x_1) + (a_2, x_2)) &= b_1 \cdot (a_1 + a_2, x_1 x_2 w^{\lfloor (a_1 + a_2)/e \rfloor}) \\ &= (b_1(a_1 + a_2), (x_1 x_2)^{b_1} w^{b_1 \lfloor (a_1 + a_2)/e \rfloor + \left\lfloor \frac{b_1 \lceil a_1 + a_2 \rceil_e}{e} \right\rfloor}) \\ &= (b_1(a_1 + a_2), (x_1 x_2)^{b_1} w^{\left\lfloor \frac{b_1 (e \lfloor (a_1 + a_2)/e \rfloor + \lceil a_1 + a_2 \rceil_e}{e} \right\rfloor}) \\ &= (b_1(a_1 + a_2), (x_1 x_2)^{b_1} w^{\lfloor b_1 (a_1 + a_2)/e \rfloor}) \\ &= (b_1 a_1 + b_1 a_2, (x_1 x_2)^{b_1} w^{\lfloor b_1 a_1/e \rfloor + \lfloor b_1 a_2/e \rfloor + \left\lfloor \frac{\lceil b_1 a_1 \rceil_e + \lceil b_1 a_2 \rceil_e}{e} \right\rfloor}) \\ &= (b_1 a_1, x_1^{b_1} w^{\lfloor b_1 a_1/e \rfloor}) + (b_1 a_2, x_2^{b_1} w^{\lfloor b_1 a_2/e \rfloor}) \\ &= b_1 \cdot (a_1, x_1) + b_1 \cdot (a_2, x_2) \ ,\end{aligned}$$

$$
\begin{aligned}
(b_1+b_2)\cdot(a_1,x_1) &= ((b_1+b_2)a_1, x_1^{b_1+b_2} w^{\lfloor (b_1+b_2)a_1/e \rfloor}) \\
&= (b_1a_1+b_2a_1, x_1^{b_1+b_2} w^{\left\lfloor \frac{e\lfloor b_1a_1/e\rfloor + \lceil b_1a_1\rceil_e + e\lfloor b_2a_1/e\rfloor + \lceil b_2a_1\rceil_e}{e} \right\rfloor}) \\
&= (b_1a_1+b_2a_1, x_1^{b_1+b_2} w^{\lfloor b_1a_1/e\rfloor + \lfloor b_2a_1/e\rfloor + \left\lfloor \frac{\lceil b_1a_1\rceil_e + \lceil b_2a_1\rceil_e}{e} \right\rfloor}) \\
&= (b_1a_1, x_1^{b_1} w^{\lfloor b_1a_1/e\rfloor}) + (b_2a_1, x_1^{b_2} w^{\lfloor b_2a_1/e\rfloor}) \\
&= b_1\cdot(a_1,x_1) + b_2\cdot(a_1,x_1)\ ,
\end{aligned}
$$

$$
\begin{aligned}
(b_1b_2)\cdot(a_1,x_1) &= (b_1b_2a_1, x_1^{b_1b_2} w^{\lfloor b_1b_2a_1/e\rfloor}) \\
&= (b_1b_2a_1, x_1^{b_1b_2} w^{\left\lfloor \frac{b_1(e\lfloor b_2a_1/e\rfloor + \lceil b_2a_1\rceil_e)}{e} \right\rfloor}) \\
&= (b_1b_2a_1, x_1^{b_1b_2} w^{b_1\lfloor b_2a_1/e\rfloor + \left\lfloor \frac{b_1\lceil b_2a_1\rceil_e}{e} \right\rfloor}) \\
&= b_1\cdot(b_2a_1, x_1^{b_2} w^{\lfloor b_2a_1/e\rfloor}) \\
&= b_1\cdot(b_2\cdot(a_1,x_1))\ ,
\end{aligned}
$$

$$
1\cdot(a_1,x_1) = (a_1, x_1 w^{\lfloor a_1/e\rfloor}) = (a_1,x_1)\ .
$$

PART 2. It is easy to verify F is a homomorphism of $\mathcal{S}$-modules, since for any $b\in\mathcal{S}$ and $(a_1,x_1),(a_2,x_2)\in\mathcal{D}$,

$$
\begin{aligned}
\mathsf{F}((a_1,x_1)+b\cdot(a_2,x_2)) &= \mathsf{F}(a_1+ba_2, x_1x_2^b w^{\lfloor (a_1+ba_2)/e\rfloor}) \\
&= x_1^e x_2^{be} w^{e\lfloor (a_1+ba_2)/e\rfloor} w^{a_1+ba_2-e\lfloor (a_1+ba_2)/e\rfloor} \\
&= x_1^e x_2^{be} w^{a_1+ba_2} \\
&= (x_1^e w^{a_1})(x_2^b w^{a_2})^e \\
&= \mathsf{F}(a_1,x_1) + b\cdot\mathsf{F}(a_2,x_2)\ .
\end{aligned}
$$

We use $1/e$ to denotes the inverse of e modulo $\phi(N)$. The inverse exists since we assume $\gcd(e,\phi(N)) = 0$. Moreover, F is epimorphism since for any $x\in\mathcal{R}$, we have $\mathsf{F}(0,x^{1/e}) = x$. Also, F is not a monomorphism, since $\mathsf{F}(e-1, w^{1-1/e}) = 1$, where 1 is the identity of $\mathcal{R}$.

PART 3. For any adversary $\mathcal{A}$ for the $\mathrm{CR}^{\mathsf{RLHF}}$ game, we construct $\mathcal{B}$ for the RSA game as follows. After receiving (N,e,w), $\mathcal{B}$ runs $\mathcal{A}$ with input (N,e,w). If $\mathcal{A}$ wins the $\mathrm{CR}^{\mathsf{RLHF}}$ by outputing $(a_1,x_1),(a_2,x_2)\in\mathcal{D}$, such that $(a_1,x_1)\neq(a_2,x_2)$ and $\mathsf{F}(a_1,x_1) = \mathsf{F}(a_2,x_2)$. $\mathcal{B}$ can compute $u\in\mathbb{Z}_N^*$ such that $u^e = w$ as follows. Since $\mathsf{F}(a_1,x_1) = \mathsf{F}(a_2,x_2)$, we have $w^{a_1}x_1^e = w^{a_2}x_2^e$, which implies $w^{a_1-a_2} = (x_2/x_1)^e$. If $a_1 = a_2$, we have $x_2 = x_1$, which contradicts with the fact that $(a_1,x_1)\neq(a_2,x_2)$. Therefore, we have $a_1\neq a_2$. Since $a_1,a_2\in\mathbb{Z}_e$, $(a_1-a_2)\not\equiv 0 \bmod e$ and thus there exists $t\in\mathbb{Z}_e$ which is the inverse of (a_1-a_2) modulo e. Then, $\mathcal{B}$ sets $u = w^{-\lfloor t(a_1-a_2)/e\rfloor}(x_2/x_1)^t$. Since $t(a_1-a_2)\equiv 1 \bmod e$, we have $1+e\lfloor t(a_1-a_2)/e\rfloor = t(a_1-a_2)$ and thus

$$
u^e = w^{-e\lfloor t(a_1-a_2)/e\rfloor}(x_2/x_1)^{te} = w^{-e\lfloor t(a_1-a_2)/e\rfloor}(w^{(a_1-a_2)})^t = w\ .
$$