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 [25], are re-energizing a multi-decade agenda aimed at developing new efficient multi-signatures [33] and threshold signatures [18,19] from a variety of assumptions. Threshold signatures are also at the center of standardization efforts by NIST [42] and IETF [15]. Both signature types are relatively straightforward to obtain from pairings (using, e.g., BLS [14,13]); 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) [10]. 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 [46]. 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.

Our approach. Our schemes are the outcome of the same paradigm applied to the two most efficient DL-based schemes, FROST [37,6] and $\mathsf{MuSig2}$ [43]. 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) [8], an assumption that has been the subject of criticism [36,35]. 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 [3,30,31] to abstract identification schemes from which signature variants are derived. Here, we observe that both FROST ad $\mathsf{MuSig2}$ can naturally be generalized by replacing the exponentiation map $x \mapsto g^x$ with a linear hash function $\mathsf{F} : \mathcal{D} \rightarrow \mathcal{R}$, where $\mathcal{D}, \mathcal{R}$ are $\mathcal{S}$-modules for a field $\mathcal{S}$. We generically refer to these instantiations as $\mathsf{FROST\text{-}H}$ and $\mathsf{MuSig2\text{-}H}$. (In fact, we present two variants for $\mathsf{FROST\text{-}H}$ 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 $\mathsf{FROST\text{-}H}$ and $\mathsf{MuSig2\text{-}H}$ 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 [43] and [6]. 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 [6], 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 [46], 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 [49]

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

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 [52], 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 [46]. 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 [46] 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 [37,6], 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 [55,28,39], they use more rounds. We stress that not all schemes achieve the same security goals, and here we target the notions of [6], whereas Lindell [39] 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 [43], which is based on OMDL. Drijvers *et al.* [21] 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 [4]. 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 [44], but it relies on heavy machinery from zero-knowledge proofs.

A more efficient DL-based alternative is the HBMS scheme by Bellare and Dai [7], but HBMS is not partially non-interactive. Further, our security reduction is tighter than that of HBMS. Most relevant to us, Lee and Kim [38] gave a multi-signature scheme based on Okamoto signatures that, however, is proved secure only in the AGM [24]; their signing is also not partially non-interactive.

More recently, Pan and Wagner [47] 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.* [21], as well as recent ROS attacks [12], 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.)

<u>RSA-specific challenges.</u> 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 [53], 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 [54,16], 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 $\mathsf{MuSig2\text{-}H}$ and $\mathsf{FROST\text{-}H}$ 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 $\mathsf{MuSig2\text{-}H}$, 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 $u_i \not\equiv 0 \bmod e$.

<u>Resulting schemes.</u> Our RSA-based instantiations of $\mathsf{FROST\text{-}H}$ and $\mathsf{MuSig2\text{-}H}$ produce signatures that also resemble the RSA-based signatures by Okamoto [46]. 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 [54], 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 [6,11].) To achieve this stronger notion, Shoup [54] 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 [17,27,23,51,16,22,2,26] do not consider this stronger security goal, although some of these works consider properties such as proactivity [51,2], robustness [27,23,51], removing trusted dealers [16,22], and adaptive-security [2], 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., [20,1], 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 [32,45,29,34,46,48,40,41,50] 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 [9,5] (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 [30,31], which is in turn adapted from [3]. We adapt the definition from [30] 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* $|\mathcal{R}| \geqslant 2^\kappa$*.*

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.*

```
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 $X_{\mathrm{cid}}$

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 $\mathcal{S}$.

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 $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{LHF}}(\mathcal{A}, \kappa) := \Pr\left[\mathrm{CR}^{\mathcal{A}}_{\mathsf{LHF}} = 1\right]$.

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 [43], 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 $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{LHF}}(\mathcal{B},\kappa) = \mathsf{Pr}\,[\mathsf{WIN}_{\mathcal{B}}]$.

It is left to show that $\mathsf{Pr}\,[\mathsf{WIN}_{\mathcal{B}}] \geqslant \frac{1}{2}\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa)$. 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 $\varPhi : \mathcal{W}_{\mathcal{A}} \to \mathcal{W}_{\mathcal{A}}$ *such that for any* $\boldsymbol{x} \in \mathcal{W}_{\mathcal{A}}$*, we have* $\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}} \ \vee\ \varPhi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}$.

From the above claim, we can conclude the proof since $\mathsf{Pr}\,[\mathsf{WIN}_{\mathcal{B}}] = \mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}}] = \frac{1}{2}\left(\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}}] + \mathsf{Pr}[\varPhi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}]\right) \geqslant \frac{1}{2}\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{B}} \ \vee\ \varPhi(\boldsymbol{x}) \in \mathcal{W}_{\mathcal{B}}] \geqslant \frac{1}{2}\mathsf{Pr}[\boldsymbol{x} \in \mathcal{W}_{\mathcal{A}}] = \frac{1}{2}\mathsf{Pr}[\mathsf{WIN}_{\mathcal{A}}] = \frac{1}{2}\mathsf{Adv}^{\mathrm{aompr}}_{\mathsf{LHF}}(\mathcal{A},\kappa)$. □

Proof (of Claim 1). We construct $\varPhi$ 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* $B_{i,j} = \beta_i^{(j)}$ *for* $i \in [\mathrm{cid}^{(j)}]$ *and* $B_{i,j} = 0$ *otherwise, 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 [43], a multi-signature scheme, and FROST [37], a threshold signature scheme.

4.1 Multi-Signatures

MuSig2 [43] 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 [43], 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 $m \in \{0,1\}^*$, $\Pr[\mathsf{Ver}(\mathsf{KeyAgg}(\{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}), m, \sigma)] = 1$, 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 [43] 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

```
Game $\text{MS-UF-CMA}^{\mathcal{A}}_{\mathsf{MS}}(\kappa)$ :
$par \leftarrow \mathsf{Setup}(1^\kappa)$
$\mathrm{H} \leftarrow_\$ \mathsf{MS.HF}$
$(\mathsf{sk}, \mathsf{pk}) \leftarrow_\$ \mathsf{KeyGen}()$
$\mathrm{sid} \leftarrow 0$
$S \leftarrow \varnothing$ ; $S' \leftarrow \varnothing$ ; $Q \leftarrow \varnothing$
$(L, m, \sigma) \leftarrow \mathcal{A}^{\textsc{Sign},\textsc{Sign}',\mathrm{RO}}(par, \mathsf{pk})$
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 $\textsc{PreSign}()$ :
$\mathrm{sid} \leftarrow \mathrm{sid} + 1$ ; $S \leftarrow S \cup \{\mathrm{sid}\}$
$(pp, \mathsf{st}^{(\mathrm{sid})}) \leftarrow \mathsf{PreSign}()$
Return $pp$

Oracle $\textsc{Sign}(k, app, m, L)$ :
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 $\mathrm{RO}(x)$ :
Return $\mathrm{H}(x)$
```

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

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 $\mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MS}}(\mathcal{A}, \kappa) := \Pr\left[\text{MS-UF-CMA}^{\mathcal{A}}_{\mathsf{MS}}(\kappa) = 1\right]$.

Our Scheme. Figure 4 shows the scheme $\mathsf{MuSig2\text{-}H}$, which is transformed from $\mathsf{MuSig2}$ [43] 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 $\mathsf{MuSig2\text{-}H}$. 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 $\mathsf{MuSig2}$ under the AOMDL assumption. Analogous to that, the following theorem shows that the security of $\mathsf{MuSig2\text{-}H[LHF]}$ is implied by AOMPR of the underlying linear hash function family LHF in the random oracle model.

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

```
$\underline{\mathsf{Setup}(1^\kappa):}$
$par \leftarrow \mathsf{PGen}(1^\kappa)$
Return $par$

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

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

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

$\underline{\mathsf{PreSign}():}$
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 $(pp, \mathsf{st})$
```

```
$\underline{\mathsf{PreAgg}(\{pp_1, \ldots, pp_n\}):}$
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 $app \leftarrow (R_1, \ldots, R_4)$

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

$\underline{\mathsf{SignAgg}(\{out_1, \ldots, out_n\}):}$
$(R, s) \leftarrow out_1$
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 $\mathrm{H} : \{0,1\}^* \rightarrow \mathcal{S}_{\mathrm{hash}}$, $\mathcal{S}_{\mathrm{hash}} \subseteq \mathcal{S}$, and $|\mathcal{S}_{\mathrm{hash}}| \geqslant 2^\kappa$. Moreover, we require $\mathrm{char}(\mathcal{S}) \geqslant 2^\kappa$.

We prove the above theorem using the same techniques as used in the security proof of MuSig2 [43] to construct $\mathcal{B}$ given an adversary $\mathcal{A}$. Here, we briefly highlight the differences:

- We need to show that $\mathcal{B}$ simulates the $\mathrm{MS\text{-}UF\text{-}CMA}^{\mathsf{MuSig2\text{-}H[LHF]}}$ 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 $\mathcal{D}_{\mathrm{key}}$ 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 $A \in \mathcal{S}^{\ell \times \ell}$

```
$\mathsf{Fork}^{\mathcal{A}}(x, v_1, v_1', \ldots, v_{q'}, v_{q'}')$ :
Pick the random coin $\rho$ 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.

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 [43] 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 $\mathsf{Fork}^{\mathcal{C}'}$.

<u>The adversary $\mathcal{C}$.</u> 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 $\textsc{Chal}$ 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 $\textsc{Chal}$ 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.

$\widetilde{\textbf{RO}}$ **query** $\mathrm{H}_{\mathrm{agg}}(x)$**:** If $\mathrm{H}_{\mathrm{agg}}(x) \neq \perp$, $\mathcal{C}$ 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 $\mathrm{H}_{\mathrm{agg}}(x)$.

$\widetilde{\textbf{RO}}$ **query** $\mathrm{H}_{\mathrm{non}}(x)$**:** If $\mathrm{H}_{\mathrm{non}}(x) \neq \perp$, $\mathcal{C}$ 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 $\mathrm{H}_{\mathrm{sig}}(\widetilde{\mathsf{apk}}, R, m) = \perp$, $\mathcal{C}$ 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 $\mathrm{H}_{\mathrm{non}}(x)$.

$\widetilde{\textbf{RO}}$ **query** $\mathrm{H}_{\mathrm{sig}}(x)$: If $\mathrm{H}_{\mathrm{sig}}(x) \neq \perp$, $\mathcal{C}$ 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 $\mathrm{H}_{\mathrm{sig}}(x)$.

$\widetilde{\textbf{PreSign}}(i)$ **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 $i \in [4]$.

$\widetilde{\textbf{Sign}}(k, app, m, L)$ **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 $\perp$ 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 $\mathcal{C}$.

A<span style="font-variant:small-caps"></span>NALYSIS OF $\mathcal{C}$. To use Lemma 1, we define IG as the algorithm that sets $par \leftarrow\!\!\$\ \mathsf{PGen}(1^\kappa)$, uniformly samples $h_1^{(\mathrm{agg})}, \dots, h_{\mathsf{q}}^{(\mathrm{agg})} \in \mathcal{S}_{\mathrm{hash}}$, and returns $(par, h_1^{(\mathrm{agg})}, \dots, h_{\mathsf{q}}^{(\mathrm{agg})})$. Also, $(h_1^{(\mathrm{sig})}, \dots, h_{\mathsf{q}}^{(\mathrm{sig})})$ plays the role of $(h_1, \dots, h_q)$, and $(h_1^{(\mathrm{non})}, \dots, h_{\mathsf{q}}^{(\mathrm{non})})$ plays the role of $(v_1, \dots, v_{q'})$.

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, $\mathrm{acc}(\mathcal{C}) \geqslant \mathsf{Adv}^{\text{ms-uf-cma}}_{\mathsf{MuSig2\text{-}H[LHF]}}(\mathcal{A}) - \Pr[\mathsf{BadKey}]$, where $\Pr[\mathsf{BadKey}]$ 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 $\Pr[\mathsf{BadKey}] \leqslant \frac{2\mathsf{q}^2+1}{2^\kappa}$.

Proof (of Claim 3). Consider a $\widetilde{\mathrm{RO}}$ query $\mathrm{H_{agg}}(L, \widetilde{\mathsf{pk}})$ from $\mathcal{A}$ such that $X \in L$ and $\mathrm{H_{agg}}(L, X)$ is not assigned prior to the query. The aggregated key from L can be represented as $\mathsf{apk} = (X)^{t\cdot h^{(\mathrm{agg})}_{\mathrm{ctr_{agg}}}} Z$, where t is the number of times X appears in L and $Z := \sum_{\mathsf{pk}\in L, \mathsf{pk}\neq X} \mathsf{pk}^{\mathrm{H_{agg}}(L,\mathsf{pk})}$, which is independent of $h^{(\mathrm{agg})}_{\mathrm{ctr_{agg}}}$. 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 $\Pr_{X \leftarrow\!\$\, \mathcal{R}}[X$ is not* 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 $\Pr[\mathsf{BadKey}] \leqslant \Pr[X \text{ is not } \mathsf{Good}] + \Pr[\mathsf{BadKey} \wedge X \text{ is } \mathsf{Good}] \leqslant \frac{2\mathsf{q}^2+1}{2^\kappa}$. □

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, $|C(t,X)| = |\mathcal{S}_{\text{hash}}|$, 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 $h_1^{(\text{agg})},\dots,h_{\mathsf{q}}^{(\text{agg})}$ $h_1^{(\text{non})}$, $h_1^{(\text{non})'}$, $\dots$, $h_{\mathsf{q}}^{(\text{non})}$, $h_{\mathsf{q}}^{(\text{non})'} \in \mathcal{S}_{\text{hash}}$. Also, $\mathcal{C}'$ can access oracles Chal and PI defined the same way as those in the AOMPR game. To begin, $\mathcal{C}'$ runs $\mathsf{Fork}^{\mathcal{C}}(par, h_1^{(\text{agg})},\dots,h_{\mathsf{q}}^{(\text{agg})}, h_1^{(\text{non})}, h_1^{(\text{non})'},\dots,h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'})$. 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 $\bot$, 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 $par \leftarrow\!\!{\scriptstyle\$}\; \mathsf{PGen}(1^\kappa)$ and returns *par*. Also, $(h_1^{(\text{agg})},\dots,h_{\mathsf{q}}^{(\text{agg})})$ plays the role of $(h_1,\dots,h_q)$, and $((h_1^{(\text{non})}, h_1^{(\text{non})'}),\dots,(h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'}))$ plays the role of $(v_1,\dots,v_{q'})$. 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 $h_1^{(\text{non})}$, $h_1^{(\text{non})'}$, $h_1^{(\text{non})''}$, $h_1^{(\text{non})'''}$, $\dots$, $h_{\mathsf{q}}^{(\text{non})}$, $h_{\mathsf{q}}^{(\text{non})'}$, $h_{\mathsf{q}}^{(\text{non})''}$, $h_{\mathsf{q}}^{(\text{non})'''} \in \mathcal{S}_{\text{hash}}$. Then, $\mathcal{B}$ runs $\mathsf{Fork}^{\mathcal{C}'}$ on input *par*, $(h_1^{(\text{non})}, h_1^{(\text{non})'})$, $(h_1^{(\text{non})''}, h_1^{(\text{non})'''})$, $\dots$, $(h_{\mathsf{q}}^{(\text{non})}, h_{\mathsf{q}}^{(\text{non})'})$, $(h_{\mathsf{q}}^{(\text{non})''}, h_{\mathsf{q}}^{(\text{non})'''})$, 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 $\mathsf{BadHash}$ as any two of the scalars $h_1^{(\mathrm{non})}$, $h_1^{(\mathrm{non})'}$, $h_1^{(\mathrm{non})''}$, $h_1^{(\mathrm{non})'''}$, …, $h_{\mathsf{q}}^{(\mathrm{non})}$, $h_{\mathsf{q}}^{(\mathrm{non})'}$, $h_{\mathsf{q}}^{(\mathrm{non})''}$, $h_{\mathsf{q}}^{(\mathrm{non})'''}$ are same. Since they are sampled uniformly from $\mathcal{S}_{\mathrm{hash}}$, we know $\Pr[\mathsf{BadHash}] \leqslant (4\mathsf{q})^2/|\mathcal{S}_{\mathrm{hash}}| \leqslant \frac{16\mathsf{q}^2}{2^\kappa}$. Then, we can conclude the proof with the following claim, which implies $\mathsf{Adv}_{\mathsf{LHF}}^{\mathrm{aompr}}(\mathcal{B}) \geqslant \mathrm{acc}(\mathsf{Fork}^{\mathcal{C}}) - \Pr[\mathsf{BadHash}] \geqslant (\mathsf{Adv}_{\mathsf{MuSig2\text{-}H[LHF]}}^{\mathrm{ms\text{-}uf\text{-}cma}}(\mathcal{A}))^4/\mathsf{q}^3 - \frac{16\mathsf{q}^2+15}{2^\kappa}$.

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

□

Proof (proof of Claim 5). Denote $(\widetilde{x}, \mathrm{Out}^{(1)}, \mathrm{Out}^{(2)}) \leftarrow \mathrm{Out}$ and $(\widetilde{x}'$, $\mathrm{Out}^{(3)}$, $\mathrm{Out}^{(4)}) \leftarrow \mathrm{Out}'$, and we use $(\cdot)^{(i)}$ to denote the variables in $\mathrm{Out}^{(i)}$. 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 $\mathrm{H}_{\mathrm{agg}}^{(1)}(L^{*(1)}, X) = h_{I_{\mathrm{agg}}}^{(\mathrm{agg})} \neq h_{I_{\mathrm{agg}}}^{(\mathrm{agg})'} = \mathrm{H}_{\mathrm{agg}}^{(3)}(L^{*(3)}, X)$. 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

$\mathsf{FROST1}$ [37] and a more efficient version $\mathsf{FROST2}$ [6] of $\mathsf{FROST1}$ are (partially) non-interactive threshold signature schemes as formalized in [6]. We first give the syntax and security definitions of non-interactive threshold signature schemes following [6], then present new schemes based on LHF that are transformed from $\mathsf{FROST1/2}$, 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 $\mathsf{LPP}(i, \mathit{pp}, \mathsf{st}_0)$ to update its state st_0 to incorporate token pp. In a signing round, for any signer set $\mathit{SS} \subseteq [n]$ with size t and message $m \in \{0,1\}^*$, the leader runs $\mathsf{LR}(m, \mathit{SS}, \mathsf{st}_0)$ to generate a leader request lr with $\mathit{lr}.\mathsf{msg} = m$ and $\mathit{lr}.\mathsf{SS} = \mathit{SS}$ and sends lr to each signer $i \in \mathit{SS}$. Then, each signer i runs $\mathsf{PS}(\mathit{lr}, i, \mathsf{st}_i)$ to generate its partial signature psig_i. Finally, the leader computes a signature σ for m by running $\mathsf{Agg}(\{\mathit{psig}_i\}_{i\in \mathit{SS}})$. 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:

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

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 $\mathit{SS} \subseteq [n]$ and any $m \in \{0,1\}^*$, $\Pr[\mathsf{Vf}(\mathsf{pk}, m, \sigma)] = 1$, 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 [6]. Here, we focus on two of them, TS-SUF-2 and TS-SUF-3, which are achieved by $\mathsf{FROST2}$ and $\mathsf{FROST1}$, 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 $\mathsf{Adv}^{\text{ts-suf-}\mathsf{X}}_{\mathsf{TS}}(\mathcal{A}, \kappa) := \mathsf{Pr}\left[\text{TS-SUF-}\mathsf{X}^{\mathcal{A}}_{\mathsf{TS}}(\kappa) = 1\right]$ for $\mathsf{X} \in \{2, 3\}$.

Our Schemes. Figure 7 shows the protocols $\mathsf{FROST1\text{-}H}$ and $\mathsf{FROST2\text{-}H}$ that are transformed from $\mathsf{FROST1}$ and $\mathsf{FROST2}$, 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 $\lambda^S_i := \prod_{j \in S \setminus \{i\}} \frac{\mathsf{x}_j}{\mathsf{x}_i - \mathsf{x}_j}$. We analyse the correctness of the scheme in the full version of this paper. 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, $\mathsf{FROST2\text{-}H}$ is TS-SUF-2-secure and $\mathsf{FROST1\text{-}H}$ is TS-SUF-3-secure in the random oracle model. We prove the theorems in the full version of this paper.

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 *running in time roughly equal two times that of $\mathcal{A}$ such that* $\mathsf{Adv}^{\text{ts-suf-2}}_{\mathsf{FROST2\text{-}H[LHF]}}(\mathcal{A}, \kappa) \leqslant \sqrt{\mathsf{q} \cdot (\mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}, \kappa) + (3\mathsf{q}^2)/2^\kappa)}$ *, 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* $\mathsf{Adv}^{\text{ts-suf-3}}_{\mathsf{FROST1\text{-}H[LHF]}}(\mathcal{A}, \kappa) \leqslant 4n \cdot \mathsf{q} \cdot \sqrt{(\mathsf{Adv}^{\text{aompr}}_{\mathsf{LHF}}(\mathcal{B}, \kappa) + 6\mathsf{q}/2^\kappa)}$ *, where* $\mathsf{q} = \mathsf{q}_h + \mathsf{q}_s + 1$.

```
Game $\boxed{\text{TS-SUF-2}^{\mathcal{A}}_{\mathsf{TS}}(\kappa)}$ , ¦ $\text{TS-SUF-3}^{\mathcal{A}}_{\mathsf{TS}}(\kappa)$ ¦ :
$par \leftarrow \mathsf{Setup}(1^\kappa)$ ; $\mathrm{H} \leftarrow_{\$} \mathsf{TS.HF}$
$\mathrm{L} \leftarrow \varnothing$ ; $\mathrm{S} \leftarrow ()$ ; $\mathrm{S}' \leftarrow ()$
$(m, \sigma) \leftarrow \mathcal{A}^{\textsc{Init},\mathrm{PPO},\mathrm{PSignO},\mathrm{RO}}(par)$
If $(\mathsf{Vf}(\mathsf{pk}, m, \sigma) \neq 1)$ then return 0
+------------------------------------------------------------------+
| Return (not $\exists lr$ : $lr.\mathsf{msg} = m \wedge \mathsf{SVf}(\mathsf{pk}, lr, \sigma)$
|             $\wedge\ |\mathrm{S}(lr)| \geqslant t - |CS|$)       |
+------------------------------------------------------------------+
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +
¦ For $lr \in \mathrm{L}$ do
¦   $\mathrm{S}'(lr) \leftarrow \{i \in HS \cap lr.\mathsf{SS} : lr.\mathsf{PP}(i) \in \mathrm{PP}_i\}$
¦ Return (not $\exists lr$ : $lr.\mathsf{msg} = m \wedge \mathsf{SVf}(\mathsf{pk}, lr, \sigma)$
¦             $\wedge\ |\mathrm{S}(lr)| \geqslant \max\{\mathrm{S}'(lr), t - |CS|\}$)
+- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - +

Oracle $\textsc{Init}(CS)$ :
$HS \leftarrow [n] \backslash CS$
$(\mathsf{pk}, \mathsf{aux}, \mathsf{sk}_1, \ldots, \mathsf{sk}_n) \leftarrow \mathsf{KeyGen}()$
For $i \in HS$ do
  $\mathsf{st}_i.\mathsf{sk} \leftarrow \mathsf{sk}_i$ ; $\mathsf{st}_i.\mathsf{pk} \leftarrow \mathsf{pk}$ ; $\mathsf{st}_i.\mathsf{aux} = \mathsf{aux}$
Return $\mathsf{pk}, \mathsf{aux}, \{\mathsf{sk}_i\}_{i \in CS}$

Oracle $\mathrm{PPO}(i)$ :
Require: $i \in HS$
$(pp, \mathsf{st}_i) \leftarrow_{\$} \mathsf{SPP}(\mathsf{st}_i)$
$\mathrm{PP}_i \leftarrow \mathrm{PP}_i \cup \{pp\}$
Return $pp$

Oracle $\mathrm{PSignO}(i, lr)$ :
$m \leftarrow lr.\mathsf{msg}$
Require: $lr.\mathsf{SS} \subseteq [n]$ and $i \in HS$
$\mathrm{L} \leftarrow \mathrm{L} \cup \{lr\}$
$(psig, \mathsf{st}'_i) \leftarrow_{\$} \mathsf{PS}(lr, i, \mathsf{st}_i)$
If $(psig \neq \perp)$ then
  $\mathrm{S}(lr) \leftarrow \mathrm{S}(lr) \cup \{i\}$
Return $psig$

Oracle $\mathrm{RO}(x)$ :
Return $\mathrm{H}(x)$
```

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.

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 8. 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 $\mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{A}, \kappa) := \Pr\left[\mathrm{DLog}^{\mathcal{A}}_{\mathsf{GGen}} = 1\right]$.

INSTANTIATION Following the instantiation from [30], 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 $\mathbb{G}$.

```
Setup(1^κ):
$par \leftarrow\!\!\$\ \mathsf{PGen}(1^\kappa)$
For $i \in [n]$ do
    $\mathsf{st}_0.\mathrm{curPP}_i \leftarrow \varnothing$
    $\mathsf{st}_i.\mathrm{mapPP} \leftarrow ()$
Return $par$

KeyGen():
For $i \in [0..t-1]$ do
    $a_i \leftarrow\!\!\$\ \mathcal{D}_{\mathrm{key}}$
For $i \in [n]$ do
    $\mathsf{sk}_i \leftarrow\!\!\$\ \sum_{j=0}^{t-1} a_j \cdot \mathsf{x}_i^j$ ; $\mathsf{pk}_i \leftarrow \mathsf{F}(\mathsf{sk}_i)$
$\mathsf{pk} \leftarrow \mathsf{F}(a_0)$
$\mathsf{aux} \leftarrow (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$
Return $\mathsf{pk}, \mathsf{aux}, \{\mathsf{sk}_i\}_{i \in [1..n]}$

SPP(st_i):
$r \leftarrow\!\!\$\ \mathcal{D}$ ; $s \leftarrow\!\!\$\ \mathcal{D}$
$pp \leftarrow (\mathsf{F}(r), \mathsf{F}(s))$
$\mathsf{st}_i.\mathrm{mapPP}(pp) \leftarrow (r, s)$
Return $(pp, \mathsf{st}_i)$

LPP(i, pp, st_0):
$\mathsf{st}_0.\mathrm{curPP}_i \leftarrow \mathsf{st}_0.\mathrm{curPP}_i \cup \{pp\}$
Return $\mathsf{st}_0$

LR(M, SS, st_0):
If $\exists\, i \in SS : \mathsf{st}_0.\mathrm{curPP}_i = \varnothing$ then
    Return $\perp$
$lr.\mathsf{msg} \leftarrow M$ ; $lr.\mathsf{SS} \leftarrow SS$
For $i \in SS$ do
    Pick $pp_i$ from $\mathsf{st}_0.\mathrm{curPP}_i$
    $lr.\mathsf{PP}(i) \leftarrow pp_i$
    $\mathsf{st}_0.\mathrm{curPP}_i \leftarrow \mathsf{st}_0.\mathrm{curPP}_i \backslash \{pp_i\}$
Return $(lr, \mathsf{st}_0)$

Vf(pk, m, σ):
$(R, s) \leftarrow \sigma$
$c \leftarrow \mathrm{H}_2(\mathsf{pk}, m, R)$
Return $(\mathsf{F}(s) = R + c \cdot \mathsf{pk})$
```

```
CompPar(pk, lr):
$m \leftarrow lr.\mathsf{msg}$ ; $(R^*, s^*) \leftarrow \sigma$
For $i \in lr.\mathsf{SS}$ do
    $\boxed{d_i \leftarrow \mathrm{H}_1(\mathsf{pk}, lr, i)}$
    $d_i \leftarrow \mathrm{H}_1(\mathsf{pk}, lr)$
    $(R_i, S_i) \leftarrow lr.\mathsf{PP}(i)$
$R \leftarrow \sum_{i \in lr.\mathsf{SS}} (R_i + d_i S_i)$
$c \leftarrow \mathrm{H}_2(\mathsf{pk}, M, R)$
Return $(R, c, \{d_i\}_{i \in lr.\mathsf{SS}})$

PS(lr, i, st_i):
$pp_i \leftarrow lr.\mathsf{PP}(i)$
If $\mathsf{st}_i.\mathrm{mapPP}(pp_i) = \perp$ then
    Return $(\perp, \mathsf{st}_i)$
$(r_i, s_i) \leftarrow \mathsf{st}_i.\mathrm{mapPP}(pp_i)$
$\mathsf{st}_i.\mathrm{mapPP}(pp_i) \leftarrow \perp$
$(R, c, \{d_j\}_{j \in lr.\mathsf{SS}})$
    $\leftarrow \mathsf{CompPar}(\mathsf{st}_i.\mathsf{pk}, lr)$
$z_i \leftarrow r_i + d_i \cdot s_i + c \cdot \lambda_i^{lr.\mathsf{SS}} \cdot \mathsf{st}_i.\mathsf{sk}$
Return $((R, z_i), \mathsf{st}_i)$

Agg(PS, st_0):
$R \leftarrow \perp$ ; $z \leftarrow 0$
For $(R', z') \in \mathrm{PS}$ do
    If $R = \perp$ then $R \leftarrow R'$
    If $R \neq R'$ then return $(\perp, \mathsf{st}_0)$
    $z \leftarrow z + z'$
Return $((R, z), \mathsf{st}_0)$

SVf(pk, lr, σ):
$(R^*, z^*) \leftarrow \sigma$
$(R, c, \{d_j\}_{j \in lr.\mathsf{SS}})$
    $\leftarrow \mathsf{CompPar}(\mathsf{st}_i.\mathsf{pk}, lr)$
Return $(R = R^*) \wedge (\mathsf{F}(z^*) = R + c \cdot \mathsf{pk}$
```

Fig. 7. The protocol FROST1-H[LHF] and FROST1-H[LHF], where LHF = (PGen, 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 $\lambda_i^{lr.\mathsf{SS}} := \prod_{j \in S \backslash \{i\}} \frac{\mathsf{x}_j}{\mathsf{x}_j - \mathsf{x}_i}$. $\mathcal{D}_{\mathrm{key}}$ 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 $\mathrm{H} : \{0,1\}^* \rightarrow \mathcal{S}$.

```
Game $\mathrm{DLog}^{\mathcal{A}}_{\mathsf{GGen}}(\kappa)$ :
$(\mathbb{G}, p, g) \leftarrow\$ \mathsf{GGen}(1^\kappa)$
$Z \leftarrow\$ \mathbb{G}$
$z \leftarrow\$ \mathcal{A}(\mathbb{G}, p, g, Z)$
If $g^z = Z$ then
  Return 1
Return 0
```

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

Fig. 8. The DLog game and the RSA game.

The following theorem shows that GLHF is a linear hash function family and collision resistance of GLHF is implied by the discrete logarithm assumption. [30] shows similar statements, and we also give the proof in the full version of this paper.

Lemma 3. *For any group generation algorithm* GGen*,* GLHF[GGen] *is a linear hash function family (Definition 1). Moreover, for any adversary $\mathcal{A}$ for the* $\mathrm{CR}^{\mathsf{GLHF[GGen]}}$ *game, there exists an adversary $\mathcal{B}$ for the* $\mathrm{DLog}^{\mathsf{GGen}}$ *game such that* $\mathsf{Adv}^{\mathrm{cr}}_{\mathsf{GLHF[GGen]}}(\mathcal{A}, \kappa) \leqslant \mathsf{Adv}^{\mathrm{dlog}}_{\mathsf{GGen}}(\mathcal{B}, \kappa)$.

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 3 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 8. 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 $e \geqslant 2^\kappa$.[1] The corresponding advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathrm{rsa}}_{\mathsf{RGen}}(\mathcal{A}, \kappa) := \Pr\left[\mathrm{RSA}^{\mathcal{A}}_{\mathsf{RGen}} = 1\right]$.

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.

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

- 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 "·" 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 [30]. 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 give the proof in the full version of this paper.

Lemma 4. *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:

> 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 5. *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 MuSig2-H, FROST1-H, and FROST2-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 MuSig2-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 4 shows the security of RLHF-based $\mathsf{MuSig2\text{-}H}$ 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 $s_1, s_2 \in \mathbb{Z}_{2^\kappa} = \mathcal{S}_{\text{hash}}$. Therefore, we have $|C(t,X)| = |\mathcal{S}_{\text{hash}}|$. 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 $h_I, h_I' \in \mathcal{S}_{\text{hash}} = \mathbb{Z}_{2^\kappa} \subseteq \mathbb{Z}_e$, we have $1 \leqslant h_I - h_I' < e$. Therefore, $\mathcal{C}'$ computes $\widetilde{x}$ using Lemma 5 for the case $n = 1$. □

Proof (of Claim 5 for RLHF*).* The total number of $\textsc{Chal}$ 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 $\mathsf{F}(\widetilde{x}) = t \cdot h_{I_{\text{agg}}}^{(\text{agg})} \cdot X + Z$, $\mathsf{F}(\widetilde{x}') = t \cdot h_{I_{\text{agg}}}^{(\text{agg})\prime} \cdot X + Z$, where $h_{I_{\text{agg}}}^{(\text{agg})} \neq h_{I_{\text{agg}}}^{(\text{agg})\prime} \in \mathcal{S}_{\text{hash}} = \mathbb{Z}_{2^\kappa}$, $1 \leqslant t \leqslant 2^\kappa$, and $Z \in \mathcal{R}$. Therefore, we have $t(h_{I_{\text{agg}}}^{(\text{agg})} - h_{I_{\text{agg}}}^{(\text{agg})\prime}) \cdot X = \mathsf{F}(\widetilde{x} - \widetilde{x}')$. Assume $h_{I_{\text{agg}}}^{(\text{agg})\prime} < h_{I_{\text{agg}}}^{(\text{agg})}$ without loss of generality. We have $1 \leqslant t \leqslant 2^\kappa < e$ and $1 \leqslant (h_{I_{\text{agg}}}^{(\text{agg})} - h_{I_{\text{agg}}}^{(\text{agg})\prime}) \leqslant 2^\kappa < e$, which implies $t(h_{I_{\text{agg}}}^{(\text{agg})} - h_{I_{\text{agg}}}^{(\text{agg})\prime}) \not\equiv 0 \mod e$. Therefore, $\mathcal{B}$ computes x^* using Lemma 5 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

$$A = \begin{pmatrix} 1 & b_1 & b_1^2 & b_1^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & b_4 & b_4^2 & b_4^3 \end{pmatrix}, \boldsymbol{U} = \begin{pmatrix} U_{1+4(k-1)} \\ \vdots \\ U_{4k} \end{pmatrix}, \boldsymbol{s} = \begin{pmatrix} s_1 \\ \vdots \\ s_4 \end{pmatrix}. \tag{6}$$

Also, $b_i \in \mathcal{S}_{\text{hash}} = \mathbb{Z}_{2^\kappa} \subseteq \mathbb{Z}_e$ 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 5 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 $\mathrm{H}'(L) : \{0,1\}^* \to \mathcal{D}_{\text{key}}$, 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 MuSig2-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 $\mathrm{H}_{\text{agg}}(L, \widetilde{\mathsf{pk}})$, 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 MuSig2-HR since, for any new L in the random oracle model, $\mathrm{H}'(L)$ is uniformly random over $\mathcal{D}_{\text{key}}$; 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 FROST1-H and FROST2-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 FROST1-HR and FROST2-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. FROST1-HR/FROST2-HR is identical to FROST1-H/ FROST2-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 $k = \lfloor r\Delta/e \rfloor$, 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 FROST2-HR and FROST1-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 4. We give a more detailed analysis in the full version of this paper.

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

1. Aboud, S.J., Al-Fayoumi, M.A.: Two efficient RSA digital multisignature and blind multisignature schemes. In: Hamza, M.H. (ed.) IASTED International Conference on Computational Intelligence, Calgary, Alberta, Canada, July 4-6, 2005. pp. 359–362. IASTED/ACTA Press (2005)
2. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold RSA with adaptive and proactive security. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 593–611. Springer, Heidelberg (May / Jun 2006). https://doi.org/10.1007/11761679_35
3. Backendal, M., Bellare, M., Sorrell, J., Sun, J.: The fiat-shamir zoo: relating the security of different signature variants. In: Nordic Conference on Secure IT Systems. pp. 154–170. Springer (2018)
4. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM CCS 2008. pp. 449–458. ACM Press (Oct 2008). https://doi.org/10.1145/1455770.1455827
5. Bagherzandi, A., Jarecki, S.: Identity-based aggregate and multi-signature schemes based on RSA. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 480–498. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7_28
6. Bellare, M., Crites, E.C., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part IV. LNCS, vol. 13510, pp. 517–550. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15985-5_18
7. Bellare, M., Dai, W.: Chain reductions for multi-signatures and the HBMS scheme. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 650–678. Springer, Heidelberg (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_22
8. Bellare, M., Namprempre, C., Pointcheval, D., Semanko, M.: The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. Journal of Cryptology **16**(3), 185–215 (Jun 2003). https://doi.org/10.1007/s00145-002-0120-1
9. Bellare, M., Neven, G.: Identity-based multi-signatures from RSA. In: Abe, M. (ed.) CT-RSA 2007. LNCS, vol. 4377, pp. 145–162. Springer, Heidelberg (Feb 2007). https://doi.org/10.1007/11967668_10
10. Bellare, M., Rogaway, P.: Random oracles are practical: A paradigm for designing efficient protocols. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) ACM CCS 93. pp. 62–73. ACM Press (Nov 1993). https://doi.org/10.1145/168588.168596
11. Bellare, M., Tessaro, S., Zhu, C.: Stronger security for non-interactive threshold signatures: Bls and frost. Cryptology ePrint Archive (2022)

12. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Canteaut, A., Standaert, F.X. (eds.) EUROCRYPT 2021, Part I. LNCS, vol. 12696, pp. 33–53. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-77870-5_2
13. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Heidelberg (Jan 2003). https://doi.org/10.1007/3-540-36288-6_3
14. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_30
15. Connolly, D., Komlo, C., Goldberg, I., Wood, C.A.: Two-Round Threshold Schnorr Signatures with FROST. Internet-Draft draft-irtf-cfrg-frost-10, Internet Engineering Task Force (Sep 2022), https://datatracker.ietf.org/doc/draft-irtf-cfrg-frost/10/, work in Progress
16. Damgård, I., Koprowski, M.: Practical threshold RSA signatures without a trusted dealer. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 152–165. Springer, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44987-6_10
17. De Santis, A., Desmedt, Y., Frankel, Y., Yung, M.: How to share a function securely. In: 26th ACM STOC. pp. 522–533. ACM Press (May 1994). https://doi.org/10.1145/195058.195405
18. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2_8
19. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 307–315. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_28
20. Desmedt, Y., Frankel, Y.: Shared generation of authenticators and signatures (extended abstract). In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 457–469. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_37
21. Drijvers, M., Edalatnejad, K., Ford, B., Kiltz, E., Loss, J., Neven, G., Stepanovs, I.: On the security of two-round multi-signatures. In: 2019 IEEE Symposium on Security and Privacy. pp. 1084–1101. IEEE Computer Society Press (May 2019). https://doi.org/10.1109/SP.2019.00050
22. Fouque, P.A., Stern, J.: Fully distributed threshold RSA under standard assumptions. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 310–330. Springer, Heidelberg (Dec 2001). https://doi.org/10.1007/3-540-45682-1_19
23. Frankel, Y., MacKenzie, P.D., Yung, M.: Robust efficient distributed RSA-key generation. In: Coan, B.A., Afek, Y. (eds.) 17th ACM PODC. p. 320. ACM (Jun / Jul 1998). https://doi.org/10.1145/277697.277779
24. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Heidelberg (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2
25. Gennaro, R., Goldfeder, S., Narayanan, A.: Threshold-optimal DSA/ECDSA signatures and an application to bitcoin wallet security. In: Manulis, M., Sadeghi, A.R., Schneider, S. (eds.) ACNS 16. LNCS, vol. 9696, pp. 156–174. Springer, Heidelberg (Jun 2016). https://doi.org/10.1007/978-3-319-39555-5_9

26. Gennaro, R., Halevi, S., Krawczyk, H., Rabin, T.: Threshold RSA for dynamic and ad-hoc groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 88–107. Springer, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_6
27. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust and efficient sharing of RSA functions. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 157–172. Springer, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_13
28. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
29. Harn, L., Kiesler, T.: New scheme for digital multisignatures. Electronics letters **25**(15), 1002–1003 (1989)
30. Hauck, E., Kiltz, E., Loss, J.: A modular treatment of blind signatures from identification schemes. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part III. LNCS, vol. 11478, pp. 345–375. Springer, Heidelberg (May 2019). https://doi.org/10.1007/978-3-030-17659-4_12
31. Hauck, E., Kiltz, E., Loss, J., Nguyen, N.K.: Lattice-based blind signatures, revisited. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 500–529. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_18
32. Itakura, K.: A public-key cryptosystem suitable for digital multisignatures (1983)
33. Itakura, K; Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC research & development (1983)
34. Kiesler, T., Harn, L.: Rsa blocking and multisignature schemes with no bit expansion. Electronics letters **18**(26), 1490–1491 (1990)
35. Koblitz, N., Menezes, A.: Another look at non-standard discrete log and diffiehellman problems. J. Math. Cryptol. **2**(4), 311–326 (2008). https://doi.org/10.1515/JMC.2008.014, https://doi.org/10.1515/JMC.2008.014
36. Koblitz, N., Menezes, A.J.: Another look at "provable security". Journal of Cryptology **20**(1), 3–37 (Jan 2007). https://doi.org/10.1007/s00145-005-0432-z
37. Komlo, C., Goldberg, I.: Frost: flexible round-optimized schnorr threshold signatures. In: International Conference on Selected Areas in Cryptography. pp. 34–65. Springer (2020)
38. Lee, K., Kim, H.: Two-round multi-signatures from okamoto signatures. Cryptology ePrint Archive, Report 2022/1117 (2022), https://eprint.iacr.org/2022/1117
39. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Paper 2022/374 (2022), https://eprint.iacr.org/2022/374, https://eprint.iacr.org/2022/374
40. Mambo, M., Okamoto, E., et al.: On the security of the rsa-based multisignature scheme for various group structures. In: Australasian Conference on Information Security and Privacy. pp. 352–367. Springer (2000)
41. Mitomi, S., Miyaji, A.: A multisignature scheme with message flexibility, order flexibility and order verifiability. In: Australasian Conference on Information Security and Privacy. pp. 298–312. Springer (2000)
42. National Institute of Standards and Technology: Multi-Party Threshold Cryptography (2018–Present), https://csrc.nist.gov/Projects/threshold-cryptography
43. Nick, J., Ruffing, T., Seurin, Y.: MuSig2: Simple two-round Schnorr multisignatures. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part I. LNCS,

vol. 12825, pp. 189–221. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84242-0_8

44. Nick, J., Ruffing, T., Seurin, Y., Wuille, P.: MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1717–1731. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417236
45. Okamoto, T.: A digital multisignature scheme using bijective public-key cryptosystems. ACM Transactions on Computer Systems (TOCS) **6**(4), 432–441 (1988)
46. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 31–53. Springer, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_3
47. Pan, J., Wagner, B.: Chopsticks: Fork-free two-round multi-signatures from non-interactive assumptions. EUROCRYPT 2023 (2023)
48. Park, S., Park, S., Kim, K., Won, D.: Two efficient rsa multisignature schemes. In: International Conference on Information and Communications Security. pp. 217–222. Springer (1997)
49. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO'91. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (Aug 1992). https://doi.org/10.1007/3-540-46766-1_9
50. Pon, S.F., Lu, E.H., Lee, J.Y.: Dynamic reblocking rsa-based multisignatures scheme for computer and communication networks. IEEE Communications Letters **6**(1), 43–44 (2002)
51. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 89–104. Springer, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055722
52. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 239–252. Springer, Heidelberg (Aug 1990). https://doi.org/10.1007/0-387-34805-0_22
53. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979)
54. Shoup, V.: Practical threshold signatures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 207–220. Springer, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_15
55. Stinson, D.R., Strobl, R.: Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In: Varadharajan, V., Mu, Y. (eds.) ACISP 01. LNCS, vol. 2119, pp. 417–434. Springer, Heidelberg (Jul 2001). https://doi.org/10.1007/3-540-47719-5_33