How to Prove Schnorr Assuming Schnorr: Security of Multi- and Threshold Signatures

Elizabeth Crites[1], Chelsea Komlo[2], and Mary Maller[3]

[1] University of Edinburgh
[2] University of Waterloo, Zcash Foundation
[3] Ethereum Foundation

Abstract. This work investigates efficient multi-party signature schemes in the discrete logarithm setting. We focus on a concurrent model, in which an arbitrary number of signing sessions may occur in parallel. Our primary contributions are: (1) a modular framework for proving the security of Schnorr multisignature and threshold signature schemes, (2) an optimization of the two-round threshold signature scheme FROST that we call FROST2, and (3) the application of our framework to prove the security of FROST2 as well as a range of other multi-party schemes.

We begin by demonstrating that our framework is applicable to multisignatures. We prove the security of a variant of the two-round MuSig2 scheme with proofs of possession and a three-round multisignature SimpleMuSig. We introduce a novel three-round threshold signature SimpleTSig and propose an optimization to the two-round FROST threshold scheme that we call FROST2. FROST2 reduces the number of scalar multiplications required during signing from linear in the number of signers to constant. We apply our framework to prove the security of FROST2 under the one-more discrete logarithm assumption and SimpleTSig under the discrete logarithm assumption in the programmable random oracle model.

Table of Contents

1 Introduction

Schnorr signatures are one of the most widely used and studied primitives in public key cryptography [41]. In this work, we are interested in proving the security of multi-party signature schemes whose output is a Schnorr signature. In particular, we focus on *multisignatures* and *threshold signatures*. A *multisignature scheme* allows a group of n signers, each in possession of a public/private key pair, to jointly compute a signature σ on a message m. A *threshold signature scheme* defines a t-out-of-n access structure of a private key that is shared by a set of n parties, at least t of whom are required to cooperate in order to issue a valid signature. Each multi- and threshold signature scheme in this work produces a standard (single-party) Schnorr signature, which is a Sigma-protocol zero-knowledge proof of knowledge of the discrete logarithm of the group public key, made non-interactive and bound to the message m by the Fiat-Shamir transform [20]. All are suitable as drop-in replacements in systems that implement Schnorr signatures.

Great care is required when proving the concurrent security of multi- and threshold signatures, wherein an adversary that corrupts some number of signers may open an arbitrary number of signing sessions with any honest signer simultaneously. Even a seemingly intuitive analysis can contain subtle errors that render the proof completely invalid. Indeed, Drijver's et al. [18] demonstrated that a wide range of multisignature schemes cannot be proven secure under the one-more discrete logarithm assumption. Benhamouda et al. [12] later confirmed that there exists a polynomial-time ROS attack against these multisignatures as well as against various blind and threshold signature schemes. The attack assumes a concurrent adversary; prior security reductions either did not consider concurrency or had an incorrect arguments arising from the complexity of using forking lemmas in reductions.

Our methods for proving security concentrate on cleanly separating the parts of the reduction that involve complex interactions between multiple parties, and the parts of the reduction that require rewinding or forking the adversary. By modularizing our reductions in this manner, we avoid having to argue about rewinding adversaries and concurrency at the same time. An immediate consequence is that concurrent security comes for free, and our reductions hold against adversaries that can open multiple signing sessions at the same time.

In this work, we prove the security of four different schemes: (1) a three-round multisignature scheme SimpleMuSig with "proofs of possession" (to be discussed shortly); (2) a more efficient version of the two-round multisignature scheme MuSig2 [36] with proofs of possession, which we call SpeedyMuSig; (3) a novel three-round threshold signature SimpleTSig; and (4) an optimized version of the two-round threshold signature FROST [31] that we call FROST2. SimpleTSig and FROST2 are both proven in combination with an adaptation of the Pedersen distributed key generation protocol by Komlo and Goldberg [31] that uses proofs of possession, which we call PedPoP. The security models and assumptions on which our schemes rely are outlined in Figure 1.

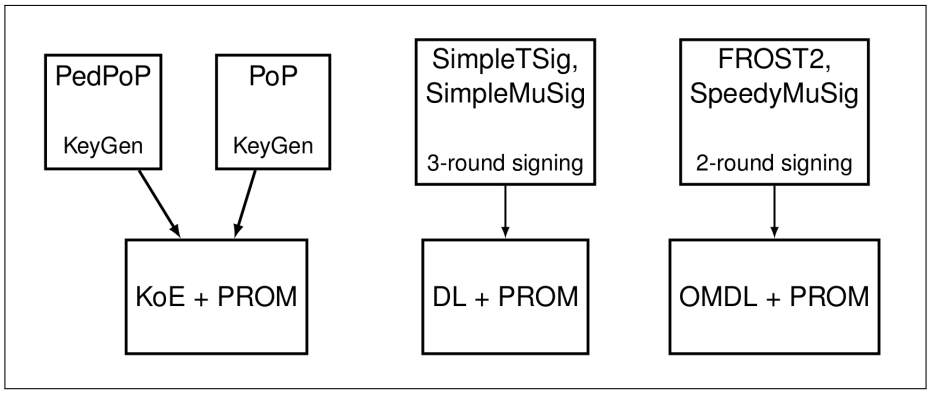

Fig. 1. Comparison of security models and assumptions underpinning the multisignature and threshold signature constructions in this work. PROM is the Programmable Random Oracle Model, KoE is Knowledge of Exponent Assumption, DL is the Discrete Logarithm Problem, and OMDL is the One-More Discrete Logarithm Problem. PoP is the Proof-of-Possession key generation protocol employed for our multisignature schemes. PedPoP is the distributed key generation protocol employed for our threshold schemes.

All of the constructions in this work have an explicit dependence on proofs of possession; that is, parties prove in zero knowledge that they know the discrete logarithm of some group element or public key. They are independent from signing and appear only during key generation. While proofs of possession exist in the standard model [21], we work directly with Schnorr signatures instead. This choice was motivated by a desire to prioritize scheme simplicity. As a result, we introduce a non-falsifiable assumption, called the *Schnorr knowledge of exponent assumption* (schnorr-koe). The schnorr-koe assumption says that an adversary that forges a Schnorr proof with respect to a public key of its choosing can extract the corresponding secret key. We prove that schnorr-koe holds without any tightness loss in the algebraic group model (AGM) [22] using similar methods to [1]. It is similar in style to knowledge of exponent assumptions that are widely used in the SNARK literature [16]. While non-falsifiable, we argue that this assumption has already stood the test of time in sense that Schnorr signatures are one of the most widely used and studied proofs of knowledge in the cryptographic literature. The schnorr-koe assumption allows us to use Schnorr signatures as proofs of possession that cost no more to store, verify, and implement than a standard Schnorr signature (512 bit proofs verify in 0.5 milliseconds [46]) and are practical when the same key may be used multiple times.

Our three-round schemes SimpleMuSig and SimpleTSig rely on the discrete logarithm assumption (as well as proofs of possession for key generation). The two-round schemes SpeedyMuSig and FROST2 rely on the one-more discrete logarithm assumption (as well as proofs of possession as part of key generation).

Our proofs of security encompass nuanced attacks that prior security models did not consider, such as ROS-style attacks that emerge in the concurrent setting. Our security reductions are in the programmable random oracle model and use two iterations of the adversary. To increase confidence in our arguments, we implement our FROST2 reduction in python and see that the algorithm succeeds when the adversary does and that the oracle responses are structured correctly.

Our variant of the MuSig2 multisignature scheme with proofs of possession, SpeedyMuSig, is likely of independent interest, as it offers significant efficiency improvements over alternative schemes in the literature. This is because proofs of possession allow the aggregate public key under which the multisignature verifies to be simply the product of the signers' individual public keys. It involves group multiplications instead of costly group exponentiations and remains secure against rogue-key attacks. See Table 2 for a breakdown of costs.

1.1 Our Contributions

The contributions of this paper are as follows:

- We present a new, modular framework for proving the concurrent security of multi- and threshold signatures and demonstrate its applicability to a variety of multi-party schemes. Concurrency applies to signing, but not key generation.
- We introduce and prove the tight security of the Schnorr knowledge of exponent assumption in the algebraic group model. This is of independent interest and useful for any application involving proofs of possession.
- We introduce and prove secure a three-round multisignature SimpleMuSig.
- We present and prove secure a three-round threshold signature SimpleTSig that is the threshold analogue of SimpleMuSig.
- We introduce and prove secure a variant of the two-round MuSig2 multisignature scheme with proofs of possession, called SpeedyMuSig. To the best of our knowledge, SpeedyMuSig is the most efficient Schnorr multisignature in the literature.
- We propose and prove secure an optimized variant of the two-round threshold signature FROST that we call FROST2. FROST2 allows for improved efficiency during signing over FROST, reducing the number of exponentiations from linear in the number of signers to constant.
- We provide an open source python implementation of our reduction for FROST2.[4]

History of this paper. Parts of this work appear in the CRYPTO 2022 paper "*Better than Advertised Security for Non-Interactive Threshold Signatures*" by Bellare, Crites, Komlo, Maller, Tessaro and Zhu [5]. It introduces the optimization FROST2 (Section 5.4) and includes the proof of security for FROST2 together with distributed key generation (Section 5.5). We thank Bellare, Tessaro, and Zhu for their invaluable feedback, which led to the current version of this paper. SimpleTSig is a new contribution to this edition; see Appendix F for details.

[4] `https://github.com/mmaller/multi_and_threshold_signature_reductions`

Scheme	KeyGen			KeyVerify	Sign				Combine	Verify
	exp	$\mathbb{G}$	$\mathbb{F}$	exp	rounds	exp	$\mathbb{G}$	$\mathbb{F}$	exp	exp
Multisignatures										
BN06 [8]	1	1	0	-	3	1	1	1	0	$n+1$
mBCJ [18]	2	2	1	2	2	4	2	3	0	8
MuSig [34]	1	1	0	-	3	$n+1$	1	2	n	$n+1$
MuSig2 [36]	1	1	0	-	2	$n+3$	2	1	$n+1$	$n+2$
DWMS [3]	1	1	0	-	2	$3n+2$	2	1	$n+1$	$n+1$
SimpleMuSig	2	2	1	2	3	1	1	2	0	2
SpeedyMuSig	2	2	1	2	2	3	2	1	1	2
Threshold signatures										
FROST [31]	$3n+nt+t+1$	$t+2$	1	-	2	$t+2$	2	1	t	2
FROST2	$3n+nt+t+1$	$t+2$	1	-	2	3	2	1	1	2

Fig. 2. Efficiency of Multi-Party Schnorr Schemes. All multi- and threshold signatures output a standard Schnorr signature except mBCJ. exp stands for the number of group exponentiations. The number of network rounds between participants is given in the round column. The number of group and field elements is denoted by $\mathbb{G}$ and $\mathbb{F}$, respectively, and is given as the total number of elements sent per signer.

2 Related Work

Bellare and Neven [8] introduced a multisignature scheme with three rounds of signing and verification matching that of a standard Schnorr signature (BN06). Maxwell et al. [34] expanded upon this scheme to allow for key aggregation (MuSig). Drijvers et al. [18] gave the first two-round scheme that is secure under the discrete logarithm assumption (and not susceptible to ROS attacks), but the resulting signature format is custom made (mBCJ). Nick et al. [37] presented an alternative two-round multisignature scheme that outputs a Schnorr signature. They rely on relatively expensive zero-knowledge proofs, which hurt the performance of the signer. Nick et al. [36] proposed a two-round multisignature scheme with efficient signing under the one-more discrete logarithm assumption (MuSig2). A variant of this two-round multisignature scheme was simultaneously proposed by Alper and Burdges [3]. Our work improves on these schemes in efficiency and also because our security reductions are more modular. Unlike prior schemes, we separate the part of the security reduction that depends on rewinding the adversary from the part of the reduction that analyzes the interactive nature of the multisignature scheme.

Proofs of security for multisignatures can alternatively be given in the algebraic group model (AGM) [36, 3]; however, AGM proofs in more complicated settings have their own delicacies and have been known to result in serious errors [23]. We limit the intricate linear algebra arguments inherent in algebraic group model

proofs to our analysis of the Schnorr knowledge of exponent assumption, for which the AGM proof can be kept simple.

Fuchsbauer et al. [22] demonstrated that the security of Schnorr signatures can be tightly reduced to the discrete logarithm assumption in the algebraic group model. However, they showed that the adversary cannot forge a signature under a public key given to them by the challenger. In the proof of the Schnorr knowledge of exponent assumption (Theorem 1), we show a stronger property: there exists an extractor that can output the secret key even when the adversary can forge under a public key of its choosing. In other words, we allow the adversary to produce new signatures, but when they do, they must also know the secret key.

Boldyreva [13] and Ristenpart and Yilek [40] showed that proofs of possession can be used to efficiently instantiate knowledge-of-secret-key assumptions for certain schemes in pairing-based groups. Boneh et al. [14] considered key aggregation in pairing-based groups. They also proposed a three-round multisignature scheme MSDL as well as a variant that included proofs of possession, called MSDL-pop. The modified scheme was claimed to have a proof of security similar to that of DG-CoSi [17] and was therefore omitted; however, in a follow-up work [18], the proof of DG-CoSi was determined to be flawed, leaving open the question of how to prove the security of MSDL-pop. We relabel MSDL-pop as SimpleMuSig and prove security, thus filling this gap in the literature.

Regarding two-round Schnorr-based threshold signature schemes, Gennaro et al. [27] proposed a protocol that is not secure in the concurrent setting due to ROS attacks. Komlo and Goldberg proposed FROST [31], which is concretely efficient and secure in the concurrent setting. We present an optimization to FROST that we call FROST2, which reduces the number of scalar multiplications required during signing from linear in the threshold to one single scalar multiplication. Further, the security proof presented by Komlo and Goldberg requires an interactive construction and cannot extend to the non-interactive variant without a heuristic assumption. Our techniques allow for a direct proof of security.

Concurrently to this work, Lindell [32] proposed a three-round Schnorr threshold signature scheme. To be secure in the programmable random oracle model under the discrete logarithm assumption only, the protocol requires the Fischlin transform [21], which is expensive due to the requirement that the prover brute-force a weak hash function. Our three-round threshold signature SimpleTSig can also be proven in the programmable random oracle model under the discrete logarithm assumption. To avoid Fischlin, we instantiate SimpleTSig with the PedPoP key generation algorithm, which uses the schnorr-koe assumption.

Bellare and Dai [6] recently proposed a new proving framework for multisignatures via a chain of sub-reductions and proved the security of existing three-round schemes [34, 8] as well as their own two-round scheme. However, their scheme is incompatible with standard Schnorr signature verification. This work proves the security of two- and three-round Schnorr multisignatures and threshold signatures.

3 Preliminaries

Let $\lambda \in \mathbb{N}$ denote the security parameter and 1^λ its unary representation. A function $\nu : \mathbb{N} \to \mathbb{R}$ is called *negligible* if for all $c > 0$, there exists k_0 such that $\nu(k) < \frac{1}{k^c}$ for all $k > k_0$. For a non-empty set S, let $x \leftarrow\!\!\$\ S$ denote sampling an element of S uniformly at random and assigning it to x.

Let PPT denote probabilistic polynomial time. Algorithms are randomized unless explicitly noted otherwise. Let $y \leftarrow A(x;\omega)$ denote running algorithm A on input x and randomness ω and assigning its output to y. Let $y \leftarrow\!\!\$\ A(x)$ denote $y \leftarrow A(x;\omega)$ for a uniformly random ω. The set of values that have non-zero probability of being output by A on input x is denoted by $[A(x)]$.

Code-based games are used in security definitions [10]. A game $\mathsf{Game}^{\mathsf{sec}}_{\mathcal{A}}(\lambda)$, played with respect to a security notion sec and adversary $\mathcal{A}$, has a MAIN procedure whose output is the output of the game.

Let GrGen be a deterministic polynomial-time algorithm that takes as input a security parameter 1^λ and outputs a group description $\mathcal{G} = (\mathbb{G}, p, g)$ consisting of a group $\mathbb{G}$ of order p, where p is a λ-bit prime, and a generator g of $\mathbb{G}$.

Definition 1 (Schnorr Signatures [41]). *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* H *be a hash function. The signer's secret key is a value* $x \leftarrow\!\!\$\ \mathbb{Z}_p$*, and its public key is* $X \leftarrow g^x$*. In order to sign a message* m*, the signer samples* $r \leftarrow\!\!\$\ \mathbb{Z}_p$ *and computes a nonce* $R \leftarrow g^r$*, hash* $\mathsf{H}(m, R)$*, and* $z = r + cx$*. The signature is the pair* (R, z)*, and it is valid if* $RX^c = g^z$*.*

Assumption 1 (Discrete Logarithm Assumption (DL)) *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*. The discrete logarithm assumption holds with respect to* $\mathcal{G}$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* ν *such that* $\Pr[\mathcal{G} \leftarrow \mathsf{GrGen}(1^\lambda); X \leftarrow\!\!\$\ \mathbb{G}; x \leftarrow\!\!\$\ \mathcal{A}(\mathcal{G}, X) : X = g^x] < \nu(\lambda)$.

Assumption 2 (One-More Discrete Logarithm Assumption (OMDL)) *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* $\mathcal{O}^{\mathsf{dl}}$ *be a discrete logarithm oracle that can be called at most* n *times. The one-more discrete logarithm assumption holds with respect to* $\mathcal{G}$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* ν *such that* $\Pr[\mathcal{G} \leftarrow \mathsf{GrGen}(1^\lambda); (X_0, \ldots, X_n) \leftarrow\!\!\$\ \mathbb{G}^{n+1}; (x_0, \ldots, x_n) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}^{\mathsf{dl}}}(\mathcal{G}, X_0, \ldots, X_n) : X_i = g^{x_i}\ \forall\ 0 \le i \le n] < \nu(\lambda)$.

Assumption 3 (Algebraic Group Model (AGM) [22]) *An adversary is algebraic if for every group element* $Z \in \mathbb{G} = \langle g \rangle$ *that it outputs, it is required to output a representation* $\boldsymbol{a} = (a_0, a_1, a_2, \ldots)$ *such that* $Z = g^{a_0} \prod {Y_i}^{a_i}$*, where* $Y_1, Y_2, \cdots \in \mathbb{G}$ *are group elements that the adversary has seen thus far.*

4 Proving the Security of Multisignatures

We begin by providing the definition of a multisignature scheme that employs proofs of possession as a component of key generation. We then introduce $\mathsf{SimpleMuSig}$, a three-round multisignature scheme with proofs of possession

(Fig. 3). This construction was proposed in [17] as MSDL-pop, but without a proof of security. We prove that SimpleMuSig is EUF-CMA secure in the programmable random oracle model under the discrete logarithm assumption and a new assumption linked to the proofs of possession: the Schnorr knowledge of exponent (schnorr-koe) assumption. We define and justify this assumption in Section 4.3. We then construct a variant of the two-round multisignature scheme MuSig2 [36] that employs proofs of possession in lieu of key aggregation, which we call SpeedyMuSig (Fig. 7). We prove the security of SpeedyMuSig under the one-more discrete logarithm assumption and the Schnorr knowledge of exponent assumption in the programmable random oracle model.

4.1 Definition of Security for Multisignatures

We build upon the definition of a multisignature scheme with proofs of possession given by Ristenpart and Yilek [40], assuming without loss of generality that there is a single honest signer whose index is 1.

Definition of Multisignatures. A multisignature scheme $\mathcal{M}$ with proofs of possession is a tuple of algorithms $\mathcal{M} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyVerify}, (\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}''), \mathsf{Combine}, \mathsf{Verify})$. The public parameters are generated by a trusted party $\mathsf{par} \leftarrow \mathsf{Setup}$ and given as input to all other algorithms. Each of the n signers generates a public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KeyGen}()$, where pk_i consists of a standard public key component X_i and a proof of possession π_i of X_i. Participants output their public keys and verify the public keys of others using KeyVerify. To collectively sign a message m, each of the signers calls the interactive signing protocol $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'')$ on its individual secret key sk_i, a set $\mathcal{PK}$ of public keys, and the message m. At the end of the signing protocol, the signers' individual signature shares are combined using the Combine algorithm to form the multisignature σ. Note that Combine may be performed by one of the signers or an external party. The multisignature σ on m is valid if $\mathsf{Verify}(\mathcal{PK}, m, \sigma) = 1$.

A multisignature scheme is *secure* if it is *correct* and *unforgeable*.

Correctness. Correctness requires that for all λ, for all n, and for all messages m, if $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KeyGen}()$ for $1 \leq i \leq n$ and all signers input $(\mathcal{PK} = \{X_1, \ldots, X_n\}, \mathsf{sk}_i, m)$ to the signing protocol $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'')$, then every signer will output a signature share that, when combined with all other shares, results in a signature σ satisfying $\mathsf{Verify}(\mathcal{PK}, m, \sigma) = 1$.

Unforgeability. EUF-CMA security is described by the following game. (See Fig. 11 in Appendix G for a formal definition.)

Setup. The challenger generates the public parameters $\mathsf{par} \leftarrow \mathsf{Setup}$ and a challenge key pair $(\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow\!\!\$\ \mathsf{KeyGen}()$, where $\mathsf{pk}_1 = (X_1, \pi_1)$. It runs the adversary $\mathcal{A}$ on input pk_1.

Signature Queries. $\mathcal{A}$ is allowed to make signature queries on any message m for any set of signer public keys $\mathcal{PK}$ with $X_1 \in \mathcal{PK}$, meaning that it has access to oracles $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}$ that will simulate the single honest signer interacting in a signing protocol with the other signers of $\mathcal{PK}$ to sign message m. Note that $\mathcal{A}$ may make any number of such queries concurrently.

Output. Finally, the adversary outputs a multisignature forgery σ^*, a message m^*, and a set of public keys $\mathcal{PK}^* = \{X_1^*, \ldots, X_n^*\}$. The adversary wins if $X_1^* = X_1$, $\mathcal{A}$ made no signing queries on m^*, and $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$.

4.2 Three-Round Multisignature SimpleMuSig

We now define a three-round multisignature scheme with proofs of possession for key generation, called SimpleMuSig (Fig. 3). The proofs of possession allow the aggregated public key $\tilde{X}$ to be computed simply as the product of the public keys $\mathcal{PK} = \{X_1, \ldots, X_n\}$ without being susceptible to rogue-key attacks. The public parameters par generated during setup are provided as input to all other algorithms and protocols.

SimpleMuSig Description.

Parameter Generation. On input the security parameter 1^λ, the setup algorithm runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$, selects hash functions $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$, and outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}})$.

Key Generation. Each signer generates a public/private key pair as follows. They first sample $x \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute $X \leftarrow g^x$. They then compute a proof of possession of X as a Schnorr signature on X as follows. They sample $\bar{r} \leftarrow\!\!\$\, \mathbb{Z}_p$ and compute $\bar{R} \leftarrow g^{\bar{r}}$. They then compute the hash $\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$ and $\bar{z} \leftarrow \bar{r} + \bar{c}x$. Their proof of possession is the signature $\pi \leftarrow (\bar{R}, \bar{z})$. The signer outputs their public/private key pair $(\mathsf{pk}, \mathsf{sk}) = ((X, \pi), x)$.

Key Verification. On input a public key $\mathsf{pk} = (X, \pi) = (X, (\bar{R}, \bar{z}))$, the verifier computes $\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$ and accepts if $\bar{R}X^{\bar{c}} = g^{\bar{z}}$, adding X to the set $\mathcal{LPK}$ of potential signers.

Signing Round 1 (Sign)**.** Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the public/private key pair of a specific signer. They sample $r_i \leftarrow\!\!\$\, \mathbb{Z}_p$, compute $R_i \leftarrow g^{r_i}$ and $\mathsf{cm}_i \leftarrow \mathsf{H}_{\mathsf{cm}}(R_i)$, and output their commitment cm_i.

Signing Round 2 (Sign')**.** On input a set $\mathcal{PK} = \{X_1, \ldots, X_n\}$ of public keys, the corresponding commitments $\{\mathsf{cm}_1, \ldots, \mathsf{cm}_n\}$, and the message m to be signed[5], the signer outputs their nonce R_i.

Signing Round 3 (Sign'')**.** On input a set $\mathcal{PK} = \{X_1, \ldots, X_n\}$ of public keys, the corresponding commitments and nonces $\{(R_1, \mathsf{cm}_1), \ldots, (R_n, \mathsf{cm}_n)\}$, and the message m, the signer first checks that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(R_j)$ for all $j \neq i$.

[5] While the input values are not explicitly used at this stage of signing, revealing nonces before these values are fixed leads to known insecurities [35].

$\mathsf{Setup}(1^\lambda)$

$\mathcal{LPK} \leftarrow \emptyset$ // registered public keys
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
select three hash functions
$\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$
$\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}})$
return par

$\mathsf{KeyGen}()$

$x \leftarrow\!\!\$\ \mathbb{Z}_p;\ X \leftarrow g^x$
$\bar{r} \leftarrow\!\!\$\ \mathbb{Z}_p;\ \bar{R} \leftarrow g^{\bar{r}}$
$\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$
$\bar{z} \leftarrow \bar{r} + \bar{c}x$
$\pi \leftarrow (\bar{R}, \bar{z})$ // PoP: Schnorr sig on X
$\mathsf{pk} \leftarrow (X, \pi);\ \mathsf{sk} \leftarrow x$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KeyVerify}(X, \pi)$

parse $(\bar{R}, \bar{z}) \leftarrow \pi$
$\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$
if $\bar{R}X^{\bar{c}} = g^{\bar{z}}$
$\mathcal{LPK} \leftarrow \mathcal{LPK} \cup \{X\}$
return 1
else return 0

$\mathsf{Sign}()$

// local signer has index 1
$r_1 \leftarrow\!\!\$\ \mathbb{Z}_p;\ R_1 \leftarrow g^{r_1}$
$\mathsf{cm}_1 \leftarrow \mathsf{H}_{\mathsf{cm}}(R_1)$
$\rho_1 \leftarrow \mathsf{cm}_1$
$st_1 \leftarrow r_1$
return (ρ_1, st_1)

$\mathsf{Sign}'(st_1, \mathsf{sk}_1, m, (X_2, \rho_2), \ldots, (X_n, \rho_n))$

parse $r_1 \leftarrow st_1$
$R_1 \leftarrow g^{r_1}$
$\rho_1' \leftarrow R_1;\ st_1' \leftarrow st_1$
return (ρ_1', st_1')

$\mathsf{Sign}''(st_1', \mathsf{sk}_1, m, \{(X_i, \rho_i, \rho_i')\}_{2 \le i \le n})$

parse $r_1 \leftarrow st_1';\ x_1 \leftarrow \mathsf{sk}_1$
$X_1 \leftarrow g^{x_1}$
parse $\mathsf{cm}_i \leftarrow \rho_i, R_i \leftarrow \rho_i', 2 \le i \le n$
if $\mathsf{cm}_i \neq \mathsf{H}_{\mathsf{cm}}(R_i)$ for some $2 \le i \le n$ return $\perp$
else $\tilde{X} \leftarrow \prod_{i=1}^{n} X_i;\ \tilde{R} \leftarrow \prod_{i=1}^{n} R_i$
$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$
$z_1 \leftarrow r_1 + cx_1$
$\rho_1'' \leftarrow z_1;\ st_1'' \leftarrow \tilde{R}$
return (ρ_1'', st_1'')

$\mathsf{Combine}(m, (X_1, \rho_1', \rho_1''), \ldots, (X_n, \rho_n', \rho_n''))$

parse $R_i \leftarrow \rho_i', z_i \leftarrow \rho_i'', 1 \le i \le n$
$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i;\ \tilde{R} \leftarrow \prod_{i=1}^{n} R_i;\ z \leftarrow \sum_{i=1}^{n} z_i$
$\sigma \leftarrow (\tilde{R}, z)$
return σ

$\mathsf{Verify}(\mathcal{PK}, m, \sigma)$

parse $\{X_1, \ldots, X_n\} \leftarrow \mathcal{PK};\ (\tilde{R}, z) \leftarrow \sigma$
$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i$
$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$
if $\tilde{R}\tilde{X}^c = g^z$ return 1
else return 0

Fig. 3. The three-round SimpleMuSig multisignature scheme with proofs of possession. The public parameters par are implicitly given as input to all algorithms.

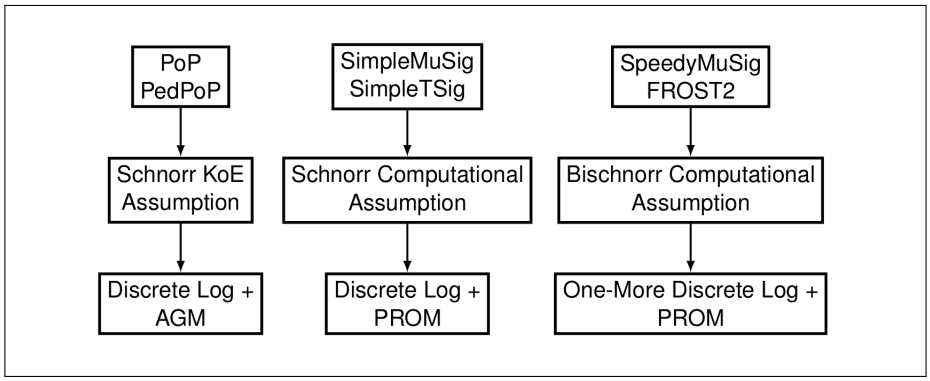

Fig. 4. Security models and intermediate assumptions underpinning the multisignature and threshold signature constructions in this work.

If for some j', $\mathsf{cm}_{j'} \neq \mathsf{H}_{\mathsf{cm}}(R_{j'})$, abort. Otherwise, the signer computes the aggregate key $\tilde{X} \leftarrow \prod_1^n X_j$, aggregate nonce $\tilde{R} \leftarrow \prod_1^n R_j$, hash $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, and $z_i \leftarrow r_i + cx_i$ and outputs z_i.

Combining Signatures. On input a set $\mathcal{PK} = \{X_1, \ldots, X_n\}$ of public keys and the corresponding signatures $\{(R_1, z_1), \ldots, (R_n, z_n)\}$ on the message m, the combiner computes $\tilde{X} \leftarrow \prod_1^n X_j, \tilde{R} \leftarrow \prod_1^n R_j, c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, and $z \leftarrow \sum_1^n z_j$ and outputs the signature $\sigma \leftarrow (\tilde{R}, z)$.

Verification. On input a set of public keys $\mathcal{PK} = \{X_1, \ldots, X_n\}$, a message m, and a signature $\sigma = (\tilde{R}, z)$, the verifier computes $\tilde{X} = \prod_1^n X_j$ and $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ and accepts if $\tilde{R}\tilde{X}^c = g^z$.

Correctness of SimpleMuSig is straightforward to verify. Note that verification of the multisignature σ is identical to verification of a standard, key-prefixed Schnorr signature with respect to the aggregate nonce $\tilde{R}$ and aggregate key $\tilde{X}$.

4.3 Proving the Security of SimpleMuSig

Here we introduce our framework for proving the concurrent security of multi-party signature schemes. Our framework captures the two parts of each scheme: key generation and signing. For key generation, we introduce and justify the Schnorr knowledge of exponent (schnorr-koe) assumption. For signing, we introduce two intermediate assumptions, the Schnorr computational (schnorr) assumption and the binonce Schnorr computational (bischnorr) assumption. (Discussion of bischnorr is deferred to Section 4.5, where it is first used.)

These assumptions allow us to separate the parts of the reduction that involve complex interactions between multiple parties from the parts of the reduction that require rewinding or forking the adversary. Specifically, the schnorr and bischnorr assumptions reduce to the discrete logarithm problem and one-more

discrete logarithm problem, respectively, using two iterations of the adversary. The reductions from our multi-party schemes to these assumptions are straight line. We present our modular framework in Figure 4.

We first apply our techniques to SimpleMuSig, demonstrating that it is EUF-CMA secure in the programmable random oracle model under the discrete logarithm assumption and the Schnorr knowledge of exponent assumption.

Schnorr Knowledge of Exponent Assumption. Here we formally introduce the Schnorr knowledge of exponent (schnorr-koe) assumption, which we show is true under the discrete logarithm assumption in the algebraic group model without any tightness loss. The schnorr-koe assumption allows us to prove the security of multi-party signatures in the setting where each participant is required to provide a proof of possession of their secret key during a key generation and registration phase. By formatting our desired security property directly as an assumption, we avoid the complexity of rewinding adversaries, which is required when proving security of Schnorr signatures in the random oracle model only, and which may result in a loss of tightness exponential in the number of parties that the adversary controls [44]. The schnorr-koe assumption implies that if an adversary can forge a Schnorr signature for some public key, then it must know the corresponding secret key. It is a non-falsifiable assumption. While new to the setting of multi-party signatures, schnorr-koe is reminiscent of prior knowledge of exponent assumptions [16, 9] employed to prove the security of Succinct NIZK arguments (SNARKs). We give more background on knowledge of exponent assumptions and their use in Appendix A.

For the definition, consider the game in Figure 5 associated to group $\mathbb{G}$, adversary $\mathcal{A}$, and an algorithm Ext, called an extractor. The adversary $\mathcal{A}$ is run with random coins ω. $\mathcal{A}$ has access to a signing oracle $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ that outputs a Schnorr signature under a randomly sampled key X on the message X. (The oracle samples a fresh public key with each invocation.) It can call its challenge oracle $\mathcal{O}^{\mathsf{chal}}$ with a triple $(X^*, \bar{R}^*, \bar{z}^*)$. If this is not a triple returned by $\mathcal{O}^{\mathsf{sch\text{-}pop}}$, yet verifies as a Schnorr signature under public key X^*, the extractor is asked to find the discrete logarithm x^* of X^*, and the adversary wins (the game sets win to true) if the extractor fails. The inputs to the extractor are the coins of the adversary, the description of the group $\mathbb{G}$, and the sets $Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}}$. The latter, for every query $(X, X, \bar{R})$ that $\mathcal{A}$ made to random oracle $\tilde{\mathsf{H}}_{\mathsf{reg}}$, stores the response of the oracle. Note that multiple queries to $\mathcal{O}^{\mathsf{chal}}$ are allowed, so that this captures the ability to perform multiple extractions.

Assumption 4 (The Schnorr Knowledge of Exponent Assumption) *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$ *in which the discrete logarithm assumption holds, and let* $\tilde{\mathsf{H}}_{\mathsf{reg}}$ *be a hash function. Let* $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda) = 1]$*, where* $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$ *is defined in Figure 5. The Schnorr knowledge of exponent assumption holds with respect to* $\mathcal{G}$ *and* $\tilde{\mathsf{H}}_{\mathsf{reg}}$ *if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT extractor* Ext *and a negligible function* ν *such that* $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda) < \nu(\lambda)$.

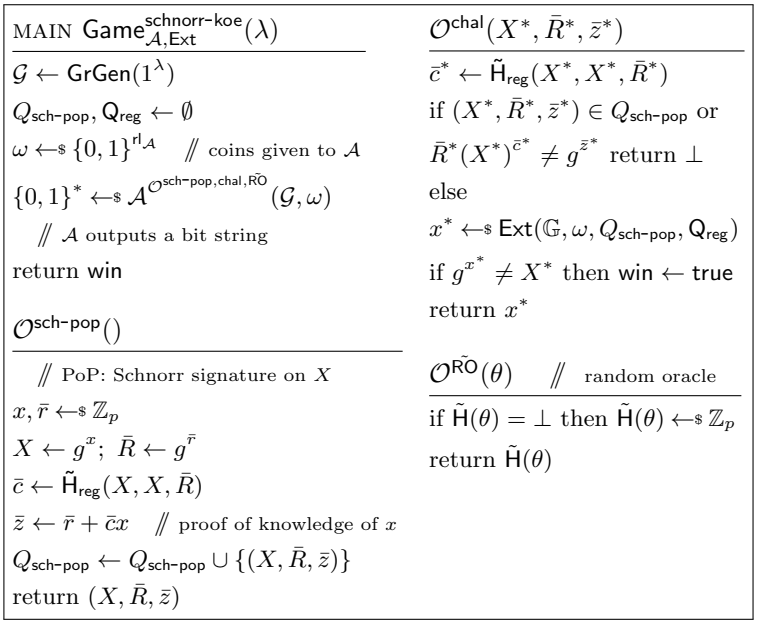

Fig. 5. Game used to define the Schnorr knowledge of exponent (schnorr-koe) assumption, where $\mathcal{G} = (\mathbb{G}, p, g)$ defines a cyclic group $\mathbb{G}$ of order p with generator g. PoP refers to "proof of possession," which in this setting means demonstrating knowledge of x. By $\mathsf{rl}_\mathcal{A}$ we denote the randomness length of $\mathcal{A}$. $\tilde{\mathsf{H}}$ is initialized to be an empty table. The hash functions $\tilde{\mathsf{H}}_{\mathsf{reg}}, \tilde{\mathsf{H}}_{\mathsf{cm}}, \tilde{\mathsf{H}}_{\mathsf{non}}, \tilde{\mathsf{H}}_{\mathsf{sig}}$ are computed as $\tilde{\mathsf{H}}(i, \cdot)$ for $i = 1, 2, 3, 4$.

Theorem 1 (dl $\Rightarrow$ schnorr-koe). *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* $\tilde{\mathsf{H}}_{\mathsf{reg}}$ *be a random oracle. The Schnorr knowledge of exponent assumption (Assumption 4) is implied by the discrete logarithm assumption in the algebraic group model with respect to* $\mathcal{G}$ *and* $\tilde{\mathsf{H}}_{\mathsf{reg}}$*.*

We present our proof of the schnorr-koe assumption in Appendix A.

Schnorr Computational Assumption. The Schnorr computational assumption is simply that (single-party) Schnorr signatures are unforgeable. An adversary with access to a Schnorr signing oracle $\mathcal{O}^{\mathsf{schnorr}}$ wins the Schnorr computational game if it can forge a Schnorr signature (m^*, R^*, z^*) under the challenge public key $\dot{X}$. For more discussion on this assumption, see Appendix B.

Assumption 5 (The Schnorr Computational Assumption) *Let* GrGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$ *in which the discrete logarithm assumption holds, and let* $\hat{\mathsf{H}}_{\mathsf{sig}}$ *be a hash function. Let* $\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) = 1]$*, where* $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda)$ *is defined in Figure 6. The Schnorr computational*

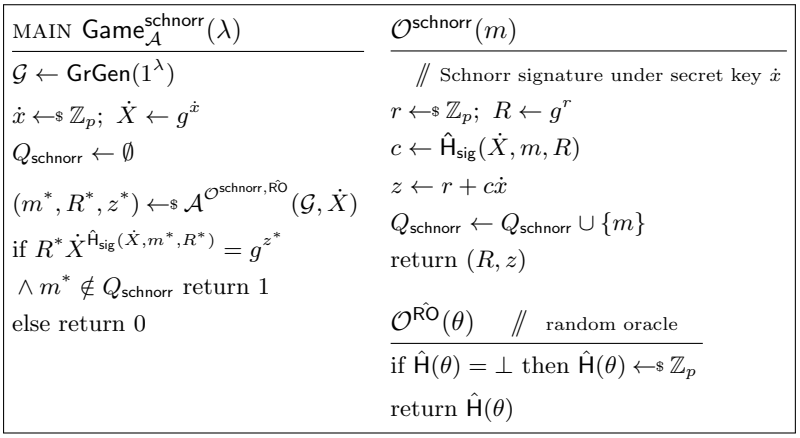

Fig. 6. Game used to define the Schnorr computational (schnorr) assumption, where $\mathcal{G} = (\mathbb{G}, p, g)$ defines a cyclic group $\mathbb{G}$ of order p with generator g. $\hat{\mathsf{H}}$ is initialized to be an empty table. The hash functions $\hat{\mathsf{H}}_{\mathsf{reg}}, \hat{\mathsf{H}}_{\mathsf{cm}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ are computed as $\hat{\mathsf{H}}(i, \cdot)$ for $i = 1, 2, 3$.

assumption holds with respect to $\mathcal{G}$ and $\hat{\mathsf{H}}_{\mathsf{sig}}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that $\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) < \nu(\lambda)$.

Theorem 2 (dl $\Rightarrow$ schnorr). *Let GrGen be a group generator that outputs $\mathcal{G} = (\mathbb{G}, p, g)$, and let $\hat{\mathsf{H}}_{\mathsf{sig}}$ be a random oracle. The Schnorr computational assumption (Assumption 5) is implied by the discrete logarithm assumption with respect to $\mathcal{G}$ and $\hat{\mathsf{H}}_{\mathsf{sig}}$.*

For completeness, we have included the proof in Appendix B, although we note that multiple versions of this reduction appear in the literature [39, 38, 42].

We are now ready to prove the following theorem.

Theorem 3 (SimpleMuSig). SimpleMuSig *is EUF-CMA secure under the discrete logarithm assumption and the Schnorr knowledge of exponent assumption in the programmable random oracle model.*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the EUF-CMA security of SimpleMuSig. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1, \mathsf{Ext}}(\lambda)$ and thence, from the schnorr-koe assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda)$ such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the schnorr-koe assumption or $\mathcal{B}_2$ breaks the schnorr assumption. Formally, we have

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1, \mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda) + \mathsf{negl}(\lambda)$$

where λ is the security parameter.

The Reduction $\mathcal{B}_1$: We first define the reduction $\mathcal{B}_1$ against $\mathsf{schnorr\text{-}koe}$. $\mathcal{B}_1$ is responsible for simulating oracle responses for key registration and queries to $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_1$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and random coins ω. It can query the random oracle $\mathcal{O}^{\tilde{\mathsf{RO}}}$ from $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$. It can also query $\mathcal{O}^{\mathsf{schnorr\text{-}koe}}$ to receive signatures under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and $\mathcal{O}^{\mathsf{chal}}$ on inputs $(X^*, \bar{R}^*, \bar{z}^*)$ to challenge the extractor Ext to output a discrete logarithm x^* for X^*.

Initialization. $\mathcal{B}_1$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\tilde{\mathsf{H}}_{\mathsf{reg}}$ (because it is part of $\mathcal{B}_1$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses.

Simulating Hash Queries. $\mathcal{B}_1$ handles $\mathcal{A}$'s hash queries throughout key registration as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_1$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_1$ queries $\bar{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on R, $\mathcal{B}_1$ queries $\mathsf{cm} \leftarrow \tilde{\mathsf{H}}_{\mathsf{cm}}(R)$ and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_1$ queries $\hat{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{sig}}(X, m, R)$ and returns $\hat{c}$.

Simulating Key Registration. $\mathcal{B}_1$ first queries $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ and receives $(\dot{X}, \bar{R}_1, \bar{z}_1)$. $\mathcal{B}_1$ runs $\mathcal{A}$ on input random coins ω and simulates key registration as follows. $\mathcal{B}_1$ embeds $\dot{X}$ as the public key X_1 of the honest party and adds X_1 to the list $\mathcal{LPK}$ of potential signers. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Register}}$ to register $\mathsf{pk}^* = (X^*, \pi^*)$ such that $\mathsf{KeyVerify}(X^*, \pi^*) = 1$, $\mathcal{B}_1$ adds X^* to $\mathcal{LPK}$ (if X^* isn't already included).

We now argue that: (1) $\mathcal{A}$ cannot distinguish between a real run of key registration and its interaction with $\mathcal{B}_1$; and (2) $\mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ outputs x^* such that $X^* = g^{x^*}$ whenever $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X^*, \bar{R}^*, \bar{z}^*)$.

(1) We will see shortly that $(\dot{X}, \bar{R}_1, \bar{z}_1)$ is computed as $\bar{R}_1 \leftarrow g^{\bar{z}_1}\dot{X}^{-\bar{c}_1}$ for random $\bar{c}_1, \bar{z}_1$, so performing validation of the honest party's $(X_1, \bar{R}_1, \bar{z}_1)$ holds and $\mathcal{B}_1$'s simulation of key registration is correct.

(2) Observe that $\mathsf{H}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*) = \tilde{\mathsf{H}}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*)$ unless $(X^*, \bar{R}^*) = (\dot{X}, \bar{R}_1)$. The latter happens only if $X^* = \dot{X}$, but in this case key registration outputs $\perp$. We thus have that $(X^*, \bar{R}^*, \bar{z}^*)$ is a verifying signature under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and either Ext succeeds, or $\mathcal{B}_1$ breaks the $\mathsf{schnorr\text{-}koe}$ assumption. Therefore, the probability of the event occurring where Ext fails to outputs x^* is bounded by $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

The Reduction $\mathcal{B}_2$: We next define the reduction $\mathcal{B}_2$ against $\mathsf{schnorr}$. $\mathcal{B}_2$ is responsible for simulating the honest party during signing and queries to $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_2$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}$. It can query $\mathcal{O}^{\mathsf{schnorr}}$ and $\mathcal{O}^{\tilde{\mathsf{RO}}}$ from $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda)$.

Initialization. $\mathcal{B}_2$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\hat{\mathsf{H}}_{\mathsf{sig}}$ (because it is part of $\mathcal{B}_2$'s challenge). Let $Q_{\mathsf{Sign}}, Q'_{\mathsf{Sign}}, Q''_{\mathsf{Sign}}$ be the set of $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}$ queries and responses in Signing Round 1, 2, 3, respectively.

DKG Extraction. $\mathcal{B}_2$ first simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}_2$ samples $\bar{c}_1, \bar{z}_1 \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $\bar{R}_1 \leftarrow g^{\bar{z}_1}\dot{X}^{-\bar{c}_1}$, and appends $(\dot{X}, \dot{X}, \bar{R}_1, \bar{c}_1)$ to $\mathsf{Q}_{\mathsf{reg}}$. Then, $\mathcal{B}_2$ runs $\mathcal{B}_1(\mathcal{G}; \omega)$ on random coins ω. $\mathcal{B}_2$ handles $\mathcal{B}_1$'s queries as follows. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{cm}}, \tilde{\mathsf{H}}_{\mathsf{sig}}$, $\mathcal{B}_2$ handles them the same way it handles $\mathcal{A}$'s $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries, described below. The first time $\mathcal{B}_1$ queries its $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ oracle, $\mathcal{B}_2$ returns $(\dot{X}, \bar{R}_1, \bar{z}_1)$. When $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_j^*, \bar{R}_j^*, \bar{z}_j^*)$, $\mathcal{B}_2$ runs $x_j^* \leftarrow \mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ to obtain x_j^* such that $X_j^* = g^{x_j^*}$ and aborts otherwise.

Simulating Hash Queries. $\mathcal{B}_2$ handles $\mathcal{A}$'s hash queries throughout the signing protocol as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. Note that $\mathcal{B}_1$ and $\mathcal{B}_2$ share the state of $\mathsf{Q}_{\mathsf{reg}}$.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on R, $\mathcal{B}_2$ checks whether $(R, \mathsf{cm}) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm. Else, $\mathcal{B}_2$ samples $\mathsf{cm} \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (R, cm) to Q_{cm}, and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}_2$ samples a random message $\hat{m}$, queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$, appends $(X, m, R, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\hat{c}$.

Simulating SimpleMuSig Signing. After $\mathcal{B}_1$ completes the simulation of key registration, $\mathcal{B}_2$ then simulates the honest party in the SimpleMuSig signing protocol.

Signing Round 1 (Sign). In the first round of signing, all parties who intend to participate send commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}_2$ samples a random message $\dot{m}$ and queries $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ to get a signature (R_1, z_1). $\mathcal{B}_2$ checks whether $(R_1, \mathsf{cm}_1) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm_1. Else, $\mathcal{B}_2$ samples $\mathsf{cm}_1 \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (R_1, cm_1) to Q_{cm}, and returns cm_1.

Signing Round 2 (Sign'). In the second round of signing, all parties corresponding to $\mathcal{PK} = \{X_1, \ldots, X_n\}$ take as input the message m to be signed and reveal nonces $R_1, \ldots, R_n$ such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(R_i)$. $\mathcal{B}_2$ looks up $\mathsf{cm}_2, \ldots, \mathsf{cm}_n$ for records $(R_i, \mathsf{cm}_i) \in \mathsf{Q}_{\mathsf{cm}}$. If there exists some j for which a record (R_j, cm_j) does not exist, then $\mathcal{B}_2$ aborts. If all records exist, then $\mathcal{B}_2$ computes $\tilde{X} \leftarrow \prod_{i=1}^n X_i$ and $\tilde{R} = \prod_{i=1}^n R_i$, queries $\dot{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \dot{m}, R_1)$ (not $\tilde{R}$), and appends $(\tilde{X}, m, \tilde{R}, \dot{m}, \dot{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. (However, if $\mathcal{A}$ has already queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$, then $\mathcal{B}_2$ aborts.) For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}_2$ returns R_1.

Signing Round 3 (Sign''). The third round of signing only proceeds if the second round terminated, i.e., all parties revealed their nonces in the second round. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}''}$, $\mathcal{B}_2$ returns z_1.

Output. When $\mathcal{A}$ returns $(\mathcal{PK}^*, m^*, \sigma^*)$ such that $\mathcal{PK}^* = \{X_1^*, \ldots, X_n^*\}, X_i^* \in \mathcal{LPK}\ \forall i, X_1^* = X_1, \sigma^* = (\tilde{R}^*, z^*)$, and $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$, $\mathcal{B}_2$ computes its output as follows. $\mathcal{B}_2$ looks up $x_2^*, \ldots, x_n^*$ (from extraction) such that $X_i^* = g^{x_i^*}$. $\mathcal{B}_2$

also looks up $\hat{m}^*$ from when $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and $\mathcal{B}_2$ had responded with $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}^*, \tilde{R}^*)$. $\mathcal{B}_2$ outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$.

To complete the proof, we must argue that: (1) $\mathcal{B}_2$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between the real EUF-CMA game and its interaction with $\mathcal{B}_2$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}_2$ succeeds.

(1) $\mathcal{B}_2$ aborts if Ext fails to return x_j^* such that $X_j^* = g^{x_j^*}$ for some j. This happens with maximum probability $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

$\mathcal{B}_2$ aborts in Signing Round 2 if $\mathcal{A}$ reveals R_j such that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(R_j)$ but $\mathcal{A}$ never queried H_{cm} on R_j. This requires $\mathcal{A}$ to have guessed cm_j ahead of time, which occurs with negligible probability $1/p$.

$\mathcal{B}_2$ also aborts in Signing Round 2 if $\mathcal{A}$ had previously queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$. In that case, $\mathcal{B}_2$ had returned $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}, \tilde{R})$ for some random message $\hat{m}$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_1 before $\mathcal{B}_2$ revealed it, which occurs with negligible probability $1/p$.

(2) As long as $\mathcal{B}_2$ does not abort, $\mathcal{B}_2$ is able to simulate the appropriate responses to $\mathcal{A}$'s oracle queries so that $\mathcal{A}$ cannot distinguish between the real EUF-CMA game and its interaction with $\mathcal{B}_2$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R}) \neq (X_1, X_1, \bar{R}_1)$, $\mathcal{B}_2$ queries its $\hat{\mathsf{H}}_{\mathsf{reg}}$ oracle on $(X, X, \bar{R})$, so $\mathcal{A}$ receives a random value. For $(X_1, X_1, \bar{R}_1)$, $\mathcal{B}_2$ programmed $\mathsf{H}_{\mathsf{reg}}$ to output its simulated $\bar{c}_1$ at the beginning of the game. The simulation of the proof of possession of X_1 is perfect because $\bar{c}_1$ is random and $\pi_1 = (\bar{R}_1, \bar{z}_1)$ verifies as $\bar{R}_1 X_1^{\bar{c}_1} = g^{\bar{z}_1}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}, R)$ on a random message $\hat{m}$. The random message prevents trivial collisions; for example, if $\mathcal{A}$ were to query $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R) and (X', m, R), where $X' \neq X$, $\mathcal{A}$ would receive the same value $c \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, m, R)$ for both and would know it was operating inside a reduction. Random messages ensure that the outputs are random, so $\mathcal{A}$'s view is correct.

When the three signing rounds have been completed, $\mathcal{A}$ may verify the signature share z_1 on m as follows. $\mathcal{A}$ checks if $R_1 X_1^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})} = g^{z_1}$, where $\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ was programmed by $\mathcal{B}_2$ as $\hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \dot{m}, R_1)$. When $\mathcal{B}_2$ queried $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ in Signing Round 1, the signature share z_1 was computed such that $R_1 X_1^{\hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \dot{m}, R_1)} = g^{z_1}$, so $\mathcal{B}_2$ simulates z_1 correctly.

(3) $\mathcal{A}$'s forgery satisfies $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$ and $X_1^* = X_1$, which implies:

$$
\begin{aligned}
&\tilde{R}^* (\tilde{X}^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} = g^{z^*} \\
&\tilde{R}^* (X_1^* \cdots X_n^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} = g^{z^*} \\
&\tilde{R}^* (X_1 g^{x_2 *} \cdots g^{x_n^*})^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} = g^{z^*} \\
&\tilde{R}^* X_1^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} = g^{z^* - \mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}
\end{aligned}
$$

$\mathcal{B}_2$ ran Ext to obtain $x_2^*, \ldots, x_n^*$. At some point, $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and received $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}^*, \tilde{R}^*)$. Thus, $\mathcal{A}$'s forgery satisfies

$$\tilde{R}^* X_1^{\hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}^*, \tilde{R}^*)} = g^{z^* - \hat{\mathsf{H}}_{\mathsf{sig}}(X_1, \hat{m}^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}$$

and $\mathcal{B}_2$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$ under $X_1 = \dot{X}$ is correct.

4.4 Two-Round Multisignature SpeedyMuSig

We now construct $\mathsf{SpeedyMuSig}$ (Fig. 7), a variant of the two-round multisignature scheme $\mathsf{MuSig2}$ [36] that includes proofs of possession. The proofs of possession allow the aggregated public key $\tilde{X}$ to be computed simply as the product of the public keys $\mathcal{PK} = \{X_1, \ldots, X_n\}$, rather than $\tilde{X} \leftarrow \prod_1^n X_j^{a_j}$, where $a_j \leftarrow \mathsf{H}_{\mathsf{agg}}(\mathcal{PK}, X_j)$, as in the original $\mathsf{MuSig2}$ scheme. The public parameters par generated during setup are provided as input to all other algorithms and protocols.

SpeedyMuSig Description.

Parameter Generation. On input the security parameter 1^λ, the setup algorithm runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$, selects hash functions $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$, and outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}})$.

Key Generation. Each signer generates a public/private key pair as follows. They first choose $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $X \leftarrow g^x$. They then compute a proof of possession of X as a Schnorr signature on X as follows. They choose $\bar{r} \leftarrow\!\!\$\ \mathbb{Z}_p$ and compute $\bar{R} \leftarrow g^{\bar{r}}$. They then compute the hash $\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$ and $\bar{z} \leftarrow \bar{r} + \bar{c}x$. Their proof of possession is the signature $\pi \leftarrow (\bar{R}, \bar{z})$. The signer outputs their public/private key pair $(\mathsf{pk}, \mathsf{sk}) = ((X, \pi), x)$.

Key Verification. On input a public key $\mathsf{pk} = (X, \pi) = (X, (\bar{R}, \bar{z}))$, the verifier computes $\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$ and accepts if $\bar{R} X^{\bar{c}} = g^{\bar{z}}$, adding X to the set $\mathcal{LPK}$ of potential signers.

Signing Round 1 (Sign)**.** Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the public/private key pair of a specific signer. They choose $r_i, s_i \leftarrow\!\!\$\ \mathbb{Z}_p$, compute $R_i, S_i \leftarrow g^{r_i}, g^{s_i}$, and output their two nonces (R_i, S_i).

Signing Round 2 (Sign')**.** On input a set $\mathcal{PK} = \{X_1, \ldots, X_n\}$ of public keys, the corresponding nonces $\{(R_1, S_1), \ldots, (R_n, S_n)\}$, and the message m to be signed, first check if $(R_i, S_i) = (R_j, S_j)$ for any $i \neq j$ and if so, abort. Else, the signer computes the aggregate key $\tilde{X} \leftarrow \prod_1^n X_j, a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(R_1, S_1), \ldots, (R_n, S_n)\})$, and the aggregate nonce $\tilde{R} \leftarrow \prod_1^n R_j S_j^a$. The signer computes the hash $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ and $z_i \leftarrow r_i + a s_i + c x_i$ and outputs z_i.

Combining Signatures. On input a set of public keys $\mathcal{PK} = \{X_1, \ldots, X_n\}$, the corresponding nonces $\{(R_1, S_1), \ldots, (R_n, S_n)\}$, and the message m, the combiner computes $\tilde{X} \leftarrow \prod_1^n X_j, a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(R_1, S_1), \ldots, (R_n, S_n)\}), \tilde{R} \leftarrow \prod_1^n R_j S_j^a$, and $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$. Finally, it computes $z \leftarrow \sum_1^n z_j$ and outputs the signature $\sigma \leftarrow (\tilde{R}, z)$.

$\mathsf{Setup}(1^\lambda)$

$\mathcal{LPK} \leftarrow \emptyset$ // registered public keys
$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$
select three hash functions
$\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$
$\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}})$
return par

$\mathsf{KeyGen}()$

$x \leftarrow\!\!\$\ \mathbb{Z}_p;\ X \leftarrow g^x$
$\bar{r} \leftarrow\!\!\$\ \mathbb{Z}_p;\ \bar{R} \leftarrow g^{\bar{r}}$
$\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$
$\bar{z} \leftarrow \bar{r} + \bar{c}x$
$\pi \leftarrow (\bar{R}, \bar{z})$ // PoP: Schnorr sig on X
$\mathsf{pk} \leftarrow (X, \pi);\ \mathsf{sk} \leftarrow x$
return $(\mathsf{pk}, \mathsf{sk})$

$\mathsf{KeyVerify}(X, \pi)$

parse $(\bar{R}, \bar{z}) \leftarrow \pi$
$\bar{c} \leftarrow \mathsf{H}_{\mathsf{reg}}(X, X, \bar{R})$
if $\bar{R}X^{\bar{c}} = g^{\bar{z}}$
$\mathcal{LPK} \leftarrow \mathcal{LPK} \cup \{X\}$
return 1
else return 0

$\mathsf{Sign}()$

// local signer has index 1
$r_1 \leftarrow\!\!\$\ \mathbb{Z}_p;\ R_1 \leftarrow g^{r_1}$
$s_1 \leftarrow\!\!\$\ \mathbb{Z}_p;\ S_1 \leftarrow g^{s_1}$
$\rho_1 \leftarrow (R_1, S_1)$
$st_1 \leftarrow (r_1, s_1)$
return (ρ_1, st_1)

$\mathsf{Sign}'(st_1, \mathsf{sk}_1, m, (X_2, \rho_2), \ldots, (X_n, \rho_n))$

// Sign' must be called at most once per st_1
parse $(r_1, s_1) \leftarrow st_1;\ x_1 \leftarrow \mathsf{sk}_1$
$R_1 \leftarrow g^{r_1};\ S_1 \leftarrow g^{s_1};\ X_1 \leftarrow g^{x_1}$
parse $(R_i, S_i) \leftarrow \rho_i, 2 \le i \le n$
if $(R_i, S_i) = (R_j, S_j)$ for $i \neq j$ return $\bot$
else $\tilde{X} \leftarrow \prod_{i=1}^{n} X_i$
$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, (R_1, S_1), \ldots, (R_n, S_n))$
$\tilde{R} \leftarrow \prod_{i=1}^{n} R_i S_i^a$
$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$
$z_1 \leftarrow r_1 + a s_1 + c x_1$
$\rho_1' \leftarrow z_1;\ st_1' \leftarrow \tilde{R}$
return (ρ_1', st_1')

$\mathsf{Combine}(m, (X_1, \rho_1, \rho_1'), \ldots, (X_n, \rho_n, \rho_n'))$

parse $(R_i, S_i) \leftarrow \rho_i, z_i \leftarrow \rho_i', 1 \le i \le n$
$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i$
$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, (R_1, S_1), \ldots, (R_n, S_n))$
$\tilde{R} \leftarrow \prod_{i=1}^{n} R_i S_i^a;\ z \leftarrow \sum_{i=1}^{n} z_i$
$\sigma \leftarrow (\tilde{R}, z)$
return σ

$\mathsf{Verify}(\mathcal{PK}, m, \sigma)$

parse $\{X_1, \ldots, X_n\} \leftarrow \mathcal{PK};\ (\tilde{R}, z) \leftarrow \sigma$
$\tilde{X} \leftarrow \prod_{i=1}^{n} X_i$
$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$
if $\tilde{R}\tilde{X}^c = g^z$ return 1
else return 0

Fig. 7. The two-round SpeedyMuSig multisignature scheme with proofs of possession. The public parameters par are implicitly given as input to all algorithms.

Verification. On input a set of public keys $\mathcal{PK} = \{X_1, \ldots, X_n\}$, a message m, and a signature $\sigma = (\tilde{R}, z)$, the verifier computes $\tilde{X} = \prod_1^n X_j$ and $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ and accepts if $\tilde{R}\tilde{X}^c = g^z$.

Correctness of SpeedyMuSig is straightforward to verify. Note that verification of the multisignature σ is identical to verification of a standard, key-prefixed Schnorr signature with respect to the aggregate nonce $\tilde{R}$ and aggregate key $\tilde{X}$.

4.5 Proving the Security of SpeedyMuSig

We now prove the EUF-CMA security of SpeedyMuSig in the programmable random oracle model under the one-more discrete logarithm assumption and the Schnorr knowledge of exponent assumption (Assumption 4). Since SpeedyMuSig consists of two rounds of signing, there is no need for a $\mathsf{Sign}''()$ algorithm, but for the purpose of aligning with our generic EUF-CMA definition, one may assume it returns no value. We apply our framework (Fig. 4), introduced in Section 4.2, by providing a straight-line reduction from SpeedyMuSig to the binonce Schnorr computational assumption.

The Binonce Schnorr Computational Assumption. We introduce the binonce Schnorr computational (bischnorr) assumption, which we prove under the one-more discrete logarithm assumption in the programmable random oracle model. It is inspired by the work of Nick et al. [36] and Komlo and Goldberg [31], whose constructions employ the approach of using two nonces to thwart a concurrent forgery attack [18]. We expand on this attack and why it does not affect our assumption in Appendix C.

The bischnorr assumption equips an adversary with two oracles, $\mathcal{O}^{\mathsf{binonce}}$ and $\mathcal{O}^{\mathsf{bisign}}$, and two hash functions, $\hat{\mathsf{H}}_{\mathsf{non}}$ and $\hat{\mathsf{H}}_{\mathsf{sig}}$, and asks it to forge a new Schnorr signature with respect to a challenge public key $\dot{X}$. The $\mathcal{O}^{\mathsf{binonce}}$ oracle takes no input and responds with two random nonces (R, S). The $\mathcal{O}^{\mathsf{bisign}}$ oracle takes as input a message m, an index k, and a set of nonces and scalars $\{(\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell)\}$. It checks that (R_k, S_k) is a $\mathcal{O}^{\mathsf{binonce}}$ response and that it has not been queried on (R_k, S_k) before. If these checks fail, it returns $\perp$. It then computes an aggregated randomized nonce $\tilde{R} = \prod_{i=1}^{\ell} R_i S_i^a$, where $a = \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell))$. $\mathcal{O}^{\mathsf{bisign}}$ then returns $z_k = r_k + a s_k + c\gamma_k x$, where $c = \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, m, \tilde{R})$. The adversary wins if it can output a verifying (m^*, R^*) that was not output by $\mathcal{O}^{\mathsf{bisign}}$.

The oracle $\mathcal{O}^{\mathsf{bisign}}$ can only be queried once for each pair of nonces (R, S) output by $\mathcal{O}^{\mathsf{binonce}}$. The index k denotes which (γ_k, R_k, S_k) out of the list $\{(\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell)\}$ is being queried; the remaining scalars and nonces appear only to inform $\mathcal{O}^{\mathsf{binonce}}$ what to include as input to $\hat{\mathsf{H}}_{\mathsf{non}}$. The scalar γ_k allows the response z_k to be given as $z_k = r_k + a s_k + c\gamma_k x$, as opposed to $r_k + a s_k + c x$. We will see that this is useful for threshold signatures, where γ_k will correspond to the Lagrange coefficient. Note that $(\gamma_1, \ldots, \gamma_\ell)$ (in addition to the nonces) must be included as input to $\hat{\mathsf{H}}_{\mathsf{non}}$ or else there is an attack.

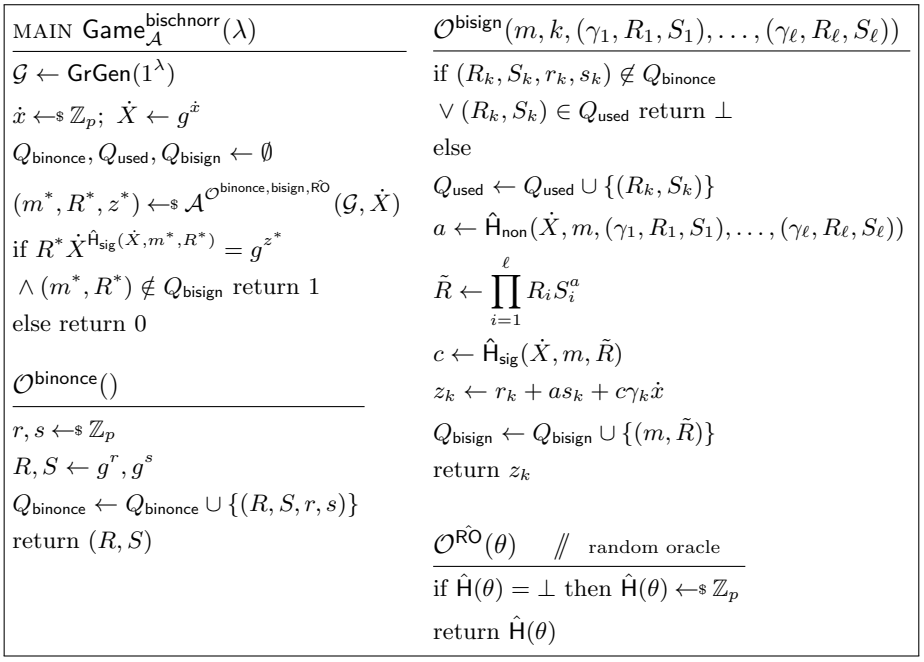

Fig. 8. Game used to define the binonce Schnorr computational (bischnorr) assumption, where $\mathcal{G} = (\mathbb{G}, p, g)$ defines a cyclic group $\mathbb{G}$ of order p with generator g. $\hat{\mathsf{H}}$ is initialized to be an empty table. The hash functions $\hat{\mathsf{H}}_{\mathsf{reg}}, \hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ are computed as $\hat{\mathsf{H}}(i, \cdot)$ for $i = 1, 2, 3$.

Assumption 6 (The Binonce Schnorr Computational Assumption) *Let* GrGen *be a group generator that outputs $\mathcal{G} = (\mathbb{G}, p, g)$ in which the discrete logarithm assumption holds, and let $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ be hash functions. Let $\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) = 1]$, where $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$ is defined in Figure 8. The binonce Schnorr computational assumption holds with respect to $\mathcal{G}$ and $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ if for all PPT adversaries $\mathcal{A}$, there exists a negligible function ν such that $\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) < \nu(\lambda)$.*

Theorem 4 (omdl $\Rightarrow$ bischnorr)**.** *Let* GrGen *be a group generator that outputs $\mathcal{G} = (\mathbb{G}, p, g)$, and let $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ be random oracles. The binonce Schnorr computational assumption (Assumption 6) is implied by the one-more discrete logarithm assumption with respect to $\mathcal{G}$ and $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$.*

We provide a proof of the bischnorr assumption in Appendix D.

We are now ready to prove the following theorem.

Theorem 5 (SpeedyMuSig). SpeedyMuSig *is EUF-CMA secure under the one-more discrete logarithm assumption and the Schnorr knowledge of exponent assumption in the programmable random oracle model.*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the EUF-CMA security of SpeedyMuSig. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$ (Fig. 5) and thence, from the schnorr-koe assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda)$ such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the schnorr-koe assumption or $\mathcal{B}_2$ breaks the bischnorr assumption. Formally, we have

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda) + \mathsf{negl}(\lambda)$$

where λ is the security parameter.

The Reduction $\mathcal{B}_1$: We first define the reduction $\mathcal{B}_1$ against schnorr-koe. $\mathcal{B}_1$ is responsible for simulating oracle responses for key registration and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_1$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and random coins ω. It can query the random oracle $\mathcal{O}^{\tilde{\mathsf{RO}}}$ from $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$. It can also query $\mathcal{O}^{\mathsf{schnorr\text{-}koe}}$ to receive signatures under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and $\mathcal{O}^{\mathsf{chal}}$ on inputs $(X^*, \bar{R}^*, \bar{z}^*)$ to challenge the extractor Ext to output a discrete logarithm x^* for X^*.

Initialization. $\mathcal{B}_1$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\tilde{\mathsf{H}}_{\mathsf{reg}}$ (because it is part of $\mathcal{B}_1$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses.

Simulating Hash Queries. $\mathcal{B}_1$ handles $\mathcal{A}$'s hash queries throughout key registration as follows.

$\mathsf{H}_{\mathsf{reg}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_1$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_1$ queries $\bar{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

$\mathsf{H}_{\mathsf{non}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}_1$ queries $\hat{a} \leftarrow \tilde{\mathsf{H}}_{\mathsf{non}}(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$ and returns $\hat{a}$.

$\mathsf{H}_{\mathsf{sig}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_1$ queries $\hat{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{sig}}(X, m, R)$ and returns $\hat{c}$.

Simulating Key Registration. $\mathcal{B}_1$ first queries $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ and receives $(\dot{X}, \bar{R}_1, \bar{z}_1)$. $\mathcal{B}_1$ runs $\mathcal{A}$ on input random coins ω and simulates key registration as follows. $\mathcal{B}_1$ embeds $\dot{X}$ as the public key X_1 of the honest party and adds X_1 to the list $\mathcal{LPK}$ of potential signers. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Register}}$ to register $\mathsf{pk}^* = (X^*, \pi^*)$ such that $\mathsf{KeyVerify}(X^*, \pi^*) = 1$, $\mathcal{B}_1$ adds X^* to $\mathcal{LPK}$ (if X^* isn't already included).

We now argue that: (1) $\mathcal{A}$ cannot distinguish between a real run of key registration and its interaction with $\mathcal{B}_1$; and (2) $\mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ outputs x^* such that $X^* = g^{x^*}$ whenever $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X^*, \bar{R}^*, \bar{z}^*)$.

(1) We will see shortly that $(\dot{X}, \bar{R}_1, \bar{z}_1)$ is computed as $\bar{R}_1 \leftarrow g^{\bar{z}_1}\dot{X}^{-\bar{c}_1}$ for random $\bar{c}_1, \bar{z}_1$, so performing validation of the honest party's $(X_1, \bar{R}_1, \bar{z}_1)$ holds and $\mathcal{B}_1$'s simulation of key registration is correct.

(2) Observe that $\mathsf{H}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*) = \tilde{\mathsf{H}}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*)$ unless $(X^*, \bar{R}^*) = (\dot{X}, \bar{R}_1)$. The latter happens only if $X^* = \dot{X}$, but in this case key registration outputs $\perp$. We thus have that $(X^*, \bar{R}^*, \bar{z}^*)$ is a verifying signature under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and either Ext succeeds, or $\mathcal{B}_1$ breaks the $\mathsf{schnorr\text{-}koe}$ assumption. Therefore, the probability of the event occurring where Ext fails to outputs x^* is bounded by $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

The Reduction $\mathcal{B}_2$: We next define the reduction $\mathcal{B}_2$ against $\mathsf{bischnorr}$. $\mathcal{B}_2$ is responsible for simulating the honest party during signing and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_2$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}$. It can query $\mathcal{O}^{\mathsf{binonce}}, \mathcal{O}^{\mathsf{bisign}}$, and $\mathcal{O}^{\hat{\mathsf{RO}}}$ from $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda)$.

Initialization. $\mathcal{B}_2$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\hat{\mathsf{H}}_{\mathsf{non}}$ or $\hat{\mathsf{H}}_{\mathsf{sig}}$ (because they are part of $\mathcal{B}_2$'s challenge). Let Q_{Sign} be the set of $\mathcal{O}^{\mathsf{Sign}}$ queries and responses in Signing Round 1, and let $Q_{\mathsf{Sign}'}$ be the set of $\mathcal{O}^{\mathsf{Sign}'}$ queries and responses in Signing Round 2.

DKG Extraction. $\mathcal{B}_2$ first simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}_2$ samples $\bar{c}_1, \bar{z}_1 \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $\bar{R}_1 \leftarrow g^{\bar{z}_1}\dot{X}^{-\bar{c}_1}$, and appends $(\dot{X}, \dot{X}, \bar{R}_1, \bar{c}_1)$ to $\mathsf{Q}_{\mathsf{reg}}$. Then, $\mathcal{B}_2$ runs $\mathcal{B}_1(\mathcal{G}; \omega)$ on random coins ω. $\mathcal{B}_2$ handles $\mathcal{B}_1$'s queries as follows. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{non}}, \tilde{\mathsf{H}}_{\mathsf{sig}}$, $\mathcal{B}_2$ handles them the same way it handles $\mathcal{A}$'s $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}}$ queries, described below. The first time $\mathcal{B}_1$ queries its $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ oracle, $\mathcal{B}_2$ returns $(\dot{X}, \bar{R}_1, \bar{z}_1)$. When $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X^*_j, \bar{R}^*_j, \bar{z}^*_j)$, $\mathcal{B}_2$ runs $x^*_j \leftarrow \mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ to obtain x^*_j such that $X^*_j = g^{x^*_j}$ and aborts otherwise.

Simulating Hash Queries. $\mathcal{B}_2$ handles $\mathcal{A}$'s hash queries throughout the signing protocol as follows.

$\mathsf{H}_{\mathsf{reg}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. Note that $\mathcal{B}_1$ and $\mathcal{B}_2$ share the state of $\mathsf{Q}_{\mathsf{reg}}$.

$\mathsf{H}_{\mathsf{non}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$, $\mathcal{B}_2$ checks whether $(X, m, R_1, S_1, \ldots, R_n, S_n, \hat{m}, \hat{a}) \in \mathsf{Q}_{\mathsf{non}}$ and, if so, returns $\hat{a}$. Else, $\mathcal{B}_2$ samples a random message $\hat{m}$ (to prevent trivial collisions), sets $\gamma_i = 1\ \forall i$[6], computes $\hat{a} \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$, and appends $(X, m, R_1, S_1, \ldots, R_n, S_n, \hat{m}, \hat{a})$ to $\mathsf{Q}_{\mathsf{non}}$. $\mathcal{B}_2$ then checks if there exists a record $(X, m, \prod_1^n R_i S_i^{\hat{a}}, \dot{m}, \dot{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, aborts. Else, $\mathcal{B}_2$ computes $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, \prod_1^n R_i S_i^{\hat{a}})$ and appends $(X, m, \prod_1^n R_i S_i^{\hat{a}}, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. Finally, $\mathcal{B}_2$ returns $\hat{a}$.

$\mathsf{H}_{\mathsf{sig}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}_2$ samples a random message $\hat{m}$, queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$, appends $(X, m, R, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\hat{c}$.

[6] Lagrange coefficients for multisignatures are 1.

Simulating SpeedyMuSig Signing. After $\mathcal{B}_1$ completes the simulation of key registration, $\mathcal{B}_2$ then simulates the honest party in the SpeedyMuSig signing protocol.

Signing Round 1 (Sign). In the first round of signing, all parties who intend to participate send two nonces $(R_1, S_1), \ldots, (R_n, S_n)$. $\mathcal{B}_2$ queries $\mathcal{O}^{\mathsf{binonce}}$ to get $(\dot{R}_1, \dot{S}_1)$. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}_2$ returns $(\dot{R}_1, \dot{S}_1)$.

Signing Round 2 (Sign'). In the second round of signing, all parties $\mathcal{PK} = \{X_1, \ldots, X_n\}$ take as input the message m to be signed. $\mathcal{B}_2$ checks if $(R_i, S_i) = (R_j, S_j)$ for any $i \neq j$ and if so, aborts. Else, $\mathcal{B}_2$ computes $\tilde{X} = \prod_1^n X_i$, checks if there exists a record $(\tilde{X}, m, \dot{R}_1, \dot{S}_1, \ldots, R_n, S_n, \hat{m}', \hat{a}') \in \mathsf{Q}_{\mathsf{non}}$ and, if so, sets $\gamma_i = 1\ \forall i$ and queries $\mathcal{O}^{\mathsf{bisign}}$ on $(\hat{m}', 1, (\gamma_1, \dot{R}_1, \dot{S}_1), \ldots, (\gamma_n, R_n, S_n))$ to get z_1. Else, $\mathcal{B}_2$ samples a random message $\hat{m}'$, programs $\mathsf{H}_{\mathsf{non}}$ and $\mathsf{H}_{\mathsf{sig}}$ as described above, and queries $\mathcal{O}^{\mathsf{bisign}}$ on $(\hat{m}', 1, (\gamma_1, \dot{R}_1, \dot{S}_1), \ldots, (\gamma_n, R_n, S_n))$ to get z_1. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}_2$ returns z_1.

Output. When $\mathcal{A}$ returns $(\mathcal{PK}^*, m^*, \sigma^*)$ such that $\mathcal{PK}^* = \{X_1^*, \ldots, X_n^*\}, X_i^* \in \mathcal{LPK}\ \forall i, X_1^* = X_1, \sigma^* = (\tilde{R}^*, z^*)$, and $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$, $\mathcal{B}_2$ computes its output as follows. It looks up $x_2^*, \ldots, x_n^*$ (from extraction) such that $X_i^* = g^{x_i^*}$. $\mathcal{B}_2$ also looks up $\hat{m}^*$ from when $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and $\mathcal{B}_2$ had responded with $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$. $\mathcal{B}_2$ outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$.

To complete the proof, we must argue that: (1) $\mathcal{B}_2$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between the real EUF-CMA game and its interaction with $\mathcal{B}_2$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}_2$ succeeds.

(1) $\mathcal{B}_2$ aborts if Ext fails to return x_j^* such that $X_j^* = g^{x_j^*}$ for some j. This happens with maximum probability $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1, \mathsf{Ext}}(\lambda)$.

$\mathcal{B}_2$ aborts if $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on $(X, m, \prod_1^n R_i S_i^{\hat{a}})$ without having first queried $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$. This requires $\mathcal{A}$ to have guessed $\hat{a}$ ahead of time, which occurs with negligible probability q_H/p.

(2) As long as $\mathcal{B}_2$ does not abort, $\mathcal{B}_2$ is able to simulate the appropriate responses to $\mathcal{A}$'s oracle queries so that $\mathcal{A}$ cannot distinguish between the real EUF-CMA game and its interaction with $\mathcal{B}_2$.

Indeed, as already observed, $\mathcal{B}_1$'s simulation of key generation is correct.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$, $\mathcal{B}_2$ queries $\hat{a} \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$ on a random message $\hat{m}$. The random message prevents trivial collisions; for example, if $\mathcal{A}$ were to query $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$ and $(X', m, R_1, S_1, \ldots, R_n, S_n)$ for $X' \neq X$, $\mathcal{A}$ would receive the same value $a \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$ for both and would know it was operating inside a reduction. Random messages ensure that the outputs are random, so $\mathcal{A}$'s view is correct. $\mathcal{B}_2$ also ensures that $\mathcal{A}$ receives $\mathsf{H}_{\mathsf{non}}$ values that are consistent with $\mathsf{H}_{\mathsf{sig}}$ queries.

After the signing rounds have been completed, $\mathcal{A}$ may verify the signature share z_1 on m as follows. $\mathcal{A}$ checks if

$$\dot{R}_1 \dot{S}_1^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \dot{R}_1, \dot{S}_1, \ldots, R_n, S_n)} \dot{X}^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_1^n R_i S_i^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \dot{R}_1, \dot{S}_1, \ldots, R_n, S_n)})} = g^{z_1}$$

When $\mathcal{B}_2$ queried $\mathcal{O}^{\mathsf{bisign}}$ on $(\hat{m}', 1, (\gamma_1, \dot{R}_1, \dot{S}_1), \ldots, (\gamma_n, R_n, S_n))$ in Signing Round 2, the signature share z_1 was computed such that

$$\dot{R}_1 \dot{S}_1^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}', (\gamma_1, \dot{R}_1, \dot{S}_1), \ldots, (\gamma_n, R_n, S_n))} \dot{X}^{\hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}', \prod_1^n R_i S_i^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}', (\gamma_1, \dot{R}_1, \dot{S}_1), \ldots, (\gamma_n, R_n, S_n))})} = g^{z_1}$$

$\mathcal{B}_2$ has programmed the hash values to be equal and therefore simulates z_1 correctly.

(3) $\mathcal{A}$'s forgery satisfies $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$ and $X_1^* = \dot{X}$, which implies:

$$\begin{aligned}
\tilde{R}^* (\tilde{X}^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* (X_1^* \cdots X_n^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* (X_1 g^{x_2 *} \cdots g^{x_n^*})^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* \dot{X}^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^* - \mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}
\end{aligned}$$

$\mathcal{B}_2$ has employed Ext to obtain $x_2^*, \ldots, x_n^*$. At some point, $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and received one of two values: (1) $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \prod_1^n R_i^* (S_i^*)^{\hat{a}^*})$ related to a query $\mathcal{A}$ made to $\mathsf{H}_{\mathsf{non}}$ on $(\tilde{X}^*, m^*, R_1^*, S_1^*, \ldots, R_n^*, S_n^*)$, where it received $\hat{a}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}^*, (1, R_1^*, S_1^*), \ldots, (1, R_n^*, S_n^*))$, or (2) $\dot{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \dot{m}^*, \tilde{R}^*)$ without having queried $\mathsf{H}_{\mathsf{non}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$. In either case, $\mathcal{B}_2$ has a record $(\tilde{X}^*, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ such that $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$. (Note that $\mathcal{B}_2$ can check which case occurred by looking for $\hat{m}^*$ in its $\mathsf{Q}_{\mathsf{non}}$ records.) Thus, $\mathcal{A}$'s forgery satisfies:

$$\tilde{R}^* \dot{X}^{\hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)} = g^{z^* - \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}$$

and $\mathcal{B}_2$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$ under $\dot{X}$ is correct.

5 Proving the Security of Threshold Signatures

In this section, we introduce and prove secure a three-round threshold signature scheme, called SimpleTSig (Fig. 9). We also propose and prove secure an optimization of the two-round threshold signature scheme FROST, which we call FROST2. Our optimization reduces the number of scalar multiplications required during signing from linear in the threshold to one. We prove the security of SimpleTSig and FROST2 together with distributed key generation (DKG). In particular, we prove the security of both schemes with the variant of the Pedersen DKG protocol [26] with proofs of possession originally proposed in combination with FROST [31]. We call this protocol PedPoP and provide a description in Figure 12.

Efficient distributed key generation. The Pedersen DKG can be viewed as n parallel instantiations of Feldman verifiable secret sharing (VSS) [19], which itself is derived from Shamir secret sharing [43] but additionally requires each participant to provide a vector commitment $\boldsymbol{C}$ to ensure their received share is

consistent with all other participants' shares. In addition, PedPoP requires each participant to provide a Schnorr proof of knowledge of the secret corresponding to the first term of their commitment. This is to ensure that unforgeability (but not liveness) holds even if more than half of the participants are dishonest.

We prove SimpleTSig+PedPoP secure under the discrete logarithm assumption and the Schnorr knowledge of exponent (schnorr-koe) assumption (Assumption 4) in the programmable random oracle model. We prove FROST2 + PedPoP secure under the one-more discrete logarithm assumption and the Schnorr knowledge of exponent assumption in the programmable random oracle model. The purpose of the schnorr-koe assumption is to ensure that the Pedersen DKG can be run in the honest minority setting, where we assume the existence of at least a single honest party and up to $t-1$ corrupt parties. The schnorr-koe assumption can be avoided if we assume an honest majority in the DKG. However, we prefer to allow more corruptions with the tradeoff of a stronger assumption.

5.1 Definition of Security for Threshold Signatures

We build upon prior definitions of threshold signature schemes in the literature [25, 28, 24], but define an additional algorithm for combining signatures that is separate from the signing rounds.

Definition of Threshold Signatures. A threshold signature scheme $\mathcal{T}$ is a tuple of algorithms $\mathcal{T} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{KeyVerify}, (\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}''), \mathsf{Combine}, \mathsf{Verify})$. The public parameters are generated by a trusted party $\mathsf{par} \leftarrow \mathsf{Setup}$ and given as input to all other algorithms. We assume the use of a distributed key generation protocol (DKG) for KeyGen, which outputs the signing group's public key $\tilde{X}$ and n secret keys, one held by each signer. To collectively sign a message m, at least t signers participate in an interactive signing protocol $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'')$. At the end of the signing protocol, the signers' individual signature shares are combined using the Combine algorithm to form the threshold signature σ. Note that Combine may be performed by one of the signers or an external party. The threshold signature σ on m is valid if $\mathsf{Verify}(\tilde{X}, m, \sigma) = 1$.

A threshold signature scheme is *secure* if it is *correct* and *unforgeable*.

Correctness. Correctness requires that for all λ, for all $t \leq \ell \leq n$, and for all messages m, if KeyGen outputs $\tilde{X}$ and ℓ signers input (sk_i, m) to the signing protocol $(\mathsf{Sign}, \mathsf{Sign}', \mathsf{Sign}'')$, then every signer will output a signature share that, when combined with all other shares, results in a signature σ satisfying $\mathsf{Verify}(\tilde{X}, m, \sigma) = 1$.

Unforgeability. EUF-CMA security is described by the following game. Assume without loss of generality that there are $t-1$ adversarial signers and at least one honest signer.

Setup. The challenger generates the parameters $\mathsf{par} \leftarrow \mathsf{Setup}$ and a challenge public key $\dot{X}$ used when running KeyGen with the adversary $\mathcal{A}$ to derive the joint public key $\tilde{X}$.

Signature Queries. $\mathcal{A}$ is allowed to make signature queries on any message m, meaning that it has access to oracles $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}$ that will simulate the honest signers interacting in a signing protocol to sign a message m with respect to $\tilde{X}$. Note that $\mathcal{A}$ may make any number of such queries concurrently.

Output. Finally, $\mathcal{A}$ outputs a threshold signature forgery σ^* and a message m^*. $\mathcal{A}$ wins if it made no signing queries on m^* and $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$.

5.2 Three-Round Threshold Signature SimpleTSig

We now introduce a three-round threshold signature scheme, called SimpleTSig (Fig. 9). We employ the PedPoP DKG protocol for key generation. The public parameters par generated during setup are provided as input to all other algorithms and protocols. We assume some external mechanism to choose the set of signers $\mathcal{S} \subseteq \{1, \ldots, n\}$, where $t \leq |\mathcal{S}| \leq n$ and $\mathcal{S}$ is ordered to ensure consistency.

We employ a model where the Lagrange coefficients are added to each participant's signature share *after* signing (by multiplying z_i by λ_i). While doing so is less efficient (as each nonce R_i must also be exponentiated by λ_i), the proof of security is simple, as there is a closer match to the assumptions defined in Section 4. While SimpleTSig is a less efficient construction than FROST, its proof of security relies on different assumptions and so is of theoretical interest.

Parameter Generation. On input the security parameter 1^λ, the setup algorithm runs $(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$, selects hash functions $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$, and outputs public parameters $\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}})$.

Key Generation. On input the number of signers n and the threshold t, all n signers cooperate to perform PedPoP.KeyGen (Fig. 12). At the end of the protocol, each signer holds a secret share $\bar{x}_i$, and the output is the group's public key $\tilde{X}$.

Signing Round 1 (Sign)**.** Each participant $P_i, i \in \mathcal{S}$, chooses $r_i \leftarrow\!\!\$\, \mathbb{Z}_p$, computes $R_i \leftarrow g^{r_i}$ and $\mathsf{cm}_i \leftarrow \mathsf{H}_{\mathsf{cm}}(R_i)$, and outputs their commitment cm_i.

Signing Round 2 (Sign')**.** On input commitments $\{\mathsf{cm}_j\}_{j \in \mathcal{S}}$ and the message m to be signed, participant P_i outputs their nonce R_i.

Signing Round 3 (Sign'')**.** On input commitments and nonces $\{(R_j, \mathsf{cm}_j)\}_{j \in \mathcal{S}}$ and the message m, participant P_i first checks that $\mathsf{cm}_j = \mathsf{H}_{\mathsf{cm}}(R_j)$ for all $j \in \mathcal{S}$. If for some j', $\mathsf{cm}_{j'} \neq \mathsf{H}_{\mathsf{cm}}(R_{j'})$, abort. Otherwise, P_i computes the aggregate nonce $\tilde{R} = \prod_{j \in \mathcal{S}} R_j{}^{\lambda_j}$, where λ_j is the j^{th} Lagrange coefficient. It then computes the hash $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ and $z_i \leftarrow r_i + cx_i$ and outputs z_i.

Combining Signatures. On input the joint public key $\tilde{X}$ and a set of signature shares $\{z_j\}_{j \in \mathcal{S}}$ on a message m, the combiner computes $\tilde{R} \leftarrow \prod_{j \in \mathcal{S}} R_j^{\lambda_j}$, $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$, and $z \leftarrow \sum_{j \in \mathcal{S}} \lambda_j z_j$ and outputs the signature $\sigma \leftarrow (\tilde{R}, z)$.

Verification. On input the joint public key $\tilde{X}$, a message m, and a signature $\sigma = (\tilde{R}, z)$, the verifier computes $c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$ and accepts if $\tilde{R}\tilde{X}^c = g^z$.

Correctness of SimpleTSig is straightforward to verify. Note that verification of the threshold signature σ is identical to verification of a standard, key-prefixed Schnorr signature with respect to the aggregate nonce $\tilde{R}$ and joint public key $\tilde{X}$.

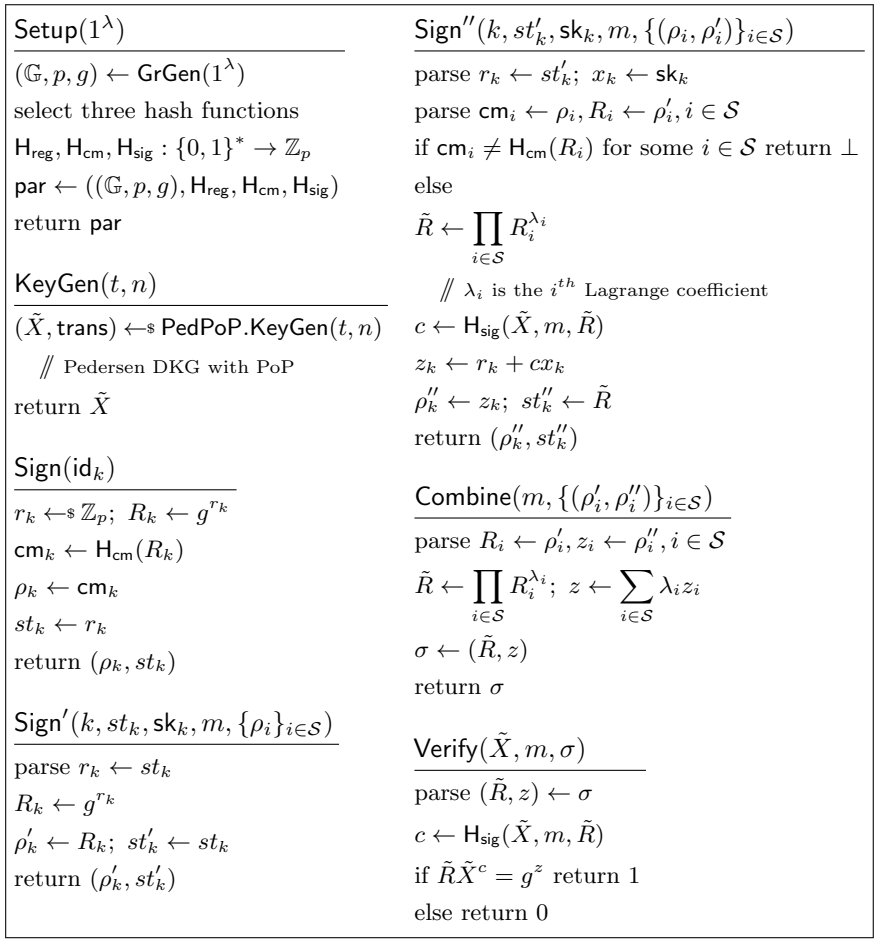

Fig. 9. The three-round SimpleTSig threshold signature scheme. The public parameters par are implicitly given as input to all algorithms and protocols. SimpleTSig assumes an external mechanism to choose the set $\mathcal{S} \subseteq \{1, \ldots, n\}$ of signers, where $t \leq |\mathcal{S}| \leq n$. $\mathcal{S}$ is required to be ordered to ensure consistency.

5.3 Proving the Security of SimpleTSig

We prove the security of SimpleTSig together with distributed key generation protocol PedPoP (Fig. 12) in Appendix E.

Theorem 6 (SimpleTSig + PedPoP)**.** SimpleTSig *is with distributed key generation protocol* PedPoP *is unforgeable under the discrete logarithm assumption and the Schnorr knowledge of exponent assumption (Assumption 4) in the programmable random oracle model.*

5.4 Optimized Two-Round Threshold Signature FROST2

A direct result of our new proving framework is that we are able to uncover a more efficient version of FROST, reducing the number of exponentiations required during signing from at least t to one. In particular, we show that security can be proven when using the same hash value for all signers. We call this optimization FROST2 (Fig. 10). We prove the security of FROST2 together with PedPoP distributed key generation.

First, the joint public key $\tilde{X}$ is generated using PedPoP (Fig. 12). At the end of key generation, there exists a degree $t-1$ polynomial $f(Z)$ such that $f(0) = \tilde{x}$, where $\tilde{X} = g^{\tilde{x}}$, and each party id_i knows $f(\mathsf{id}_i)$. The first round of signing can be run in advance of knowing the participants or the message (or even $\tilde{X}$); signers simply generate two random nonces $R_i = g^{r_i}$ and $S_i = g^{s_i}$.

In the second round of signing, signers hash $\tilde{X}$, the message, the participant identifiers, and the nonces of all of the parties that are expected to sign:

$$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}}) \tag{1}$$

where $t \leq |\mathcal{S}| \leq n$ and $\mathcal{S}$ is ordered to ensure consistency. They then compute the aggregate nonce and hash it together with $\tilde{X}$ and m:

$$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_{i \in \mathcal{S}} R_i S_i^a) \tag{2}$$

They also compute the Lagrange coefficients $\{\lambda_i\}_{i \in \mathcal{S}}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i \in \mathcal{S}}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i \in \mathcal{S}}$. Finally, the i^{th} signer returns $z_i = r_i + as_i + c\lambda_i f(\mathsf{id}_i)$.

A combine algorithm computes the value a and the aggregate nonce $\tilde{R} = \prod_{i \in \mathcal{S}} R_i S_i^a$ the same as the signers in the second round of signing. It then computes $z = \sum_{i \in \mathcal{S}} z_i$ and returns the signature $(\tilde{R}, z)$.

The difference between FROST and FROST2 is that, for FROST, Equations 1 and 2 are computed as

$$a_j \leftarrow \mathsf{H}_{\mathsf{non}}(j, \tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$$
$$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_{i \in \mathcal{S}} R_i S_i^{a_i})$$

which requires an additional exponentiation for each signer.

Additionally, in FROST2 signers must check that $(R_i, S_i) \neq (R_j, S_j)$ for $i \neq j \in S$. The check to prevent identical nonces is an artifact of our proof of security for FROST2. Note that using a generalized forking lemma instead of the local forking lemma obviates the need for this check, as shown in [11].

$\mathsf{Setup}(1^\lambda)$

$(\mathbb{G}, p, g) \leftarrow \mathsf{GrGen}(1^\lambda)$

select three hash functions

$\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}} : \{0,1\}^* \to \mathbb{Z}_p$

$\mathsf{par} \leftarrow ((\mathbb{G}, p, g), \mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}})$

return par

$\mathsf{KeyGen}(t, n)$

$(\tilde{X}, \mathsf{trans}) \leftarrow\!\!\$\ \mathsf{PedPoP.KeyGen}(t, n)$

// Pedersen DKG with PoP

return $\tilde{X}$

$\mathsf{Sign}(\mathsf{id}_k)$

$r_k \leftarrow\!\!\$\ \mathbb{Z}_p; R_k \leftarrow g^{r_k}$

$s_k \leftarrow\!\!\$\ \mathbb{Z}_p; S_k \leftarrow g^{s_k}$

$\rho_k \leftarrow (\mathsf{id}_k, R_k, S_k)$

$st_k \leftarrow (r_k, s_k)$

return (ρ_k, st_k)

$\mathsf{Sign}'(k, st_k, \mathsf{sk}_k, m, \{\rho_i\}_{i \in \mathcal{S}})$

// Sign' must be called at most once per st_k

// $\mathcal{S} \subseteq \{1, \ldots, n\}$ is the *ordered* signing set

parse $(r_k, s_k) \leftarrow st_k$; $x_k \leftarrow \mathsf{sk}_k$

$R_k \leftarrow g^{r_k}$; $S_k \leftarrow g^{s_k}$

parse $(\mathsf{id}_i, R_i, S_i) \leftarrow \rho_i, i \in \mathcal{S}$

if $(R_i, S_i) = (R_j, S_j)$ for $i \neq j$ return $\bot$

else

$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$

// $a_j \leftarrow \mathsf{H}_{\mathsf{non}}(j, \tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$

$\tilde{R} \leftarrow \prod_{i \in \mathcal{S}} R_i S_i^a$ // $\tilde{R} \leftarrow \Pi_{i \in \mathcal{S}} R_i S_i^{a_i}$

$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$

$z_k \leftarrow r_k + a s_k + c \lambda_k x_k$

// λ_k is the k^{th} Lagrange coefficient

$st'_k \leftarrow \tilde{R}$; $\rho'_k \leftarrow z_k$

return (ρ'_k, st'_k)

$\mathsf{Combine}(m, \{(\rho_i, \rho'_i)\}_{i \in \mathcal{S}})$

parse $(\mathsf{id}_i, R_i, S_i) \leftarrow \rho_i, z_i \leftarrow \rho'_i, i \in \mathcal{S}$

$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$

$\tilde{R} \leftarrow \prod_{i \in \mathcal{S}} R_i S_i^a$; $z \leftarrow \sum_{i \in \mathcal{S}} z_i$

$\sigma \leftarrow (\tilde{R}, z)$;

return σ

$\mathsf{Verify}(\tilde{X}, m, \sigma)$

parse $(\tilde{R}, z) \leftarrow \sigma$

$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$

if $\tilde{R}\tilde{X}^c = g^z$ return 1

else return 0

Fig. 10. The two-round FROST2 threshold signature scheme. The public parameters par are implicitly given as input to all algorithms. FROST2 assumes an external mechanism to choose the set $\mathcal{S} \subseteq \{1, \ldots, n\}$ of signers, where $t \leq |\mathcal{S}| \leq n$. $\mathcal{S}$ is required to be ordered to ensure consistency. We highlight the differences between FROST2 (solid boxes) and FROST (dashed boxes).

5.5 Proving the Security of FROST2

We demonstrate the strength of our framework (Section 4.3) by proving the unforgeability of FROST2 together with distributed key generation protocol PedPoP (Fig. 12).

Theorem 7 (FROST2 + PedPoP). FROST2 *with distributed key generation protocol* PedPoP *is unforgeable under the one-more discrete logarithm assumption and the Schnorr knowledge of exponent assumption (Assumption 4) in the programmable random oracle model.*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the unforgeability of FROST2. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$ (Fig. 5) and thence, from the schnorr-koe assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda)$ (Fig. 8) such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the schnorr-koe assumption or $\mathcal{B}_2$ breaks the bischnorr assumption. (Recall that bischnorr is implied by the OMDL assumption (Theorem 4)). Formally, we have

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda) + \mathsf{negl}(\lambda)$$

where λ is the security parameter.

The Reduction $\mathcal{B}_1$: We first define the reduction $\mathcal{B}_1$ against schnorr-koe. Let $\mathsf{cor} = \{\mathsf{id}_j\}$ be the set of corrupt parties, and let $\mathsf{hon} = \{\mathsf{id}_k\}$ be the set of honest parties. Assume without loss of generality that $|\mathsf{cor}| = t-1$ and $|\mathsf{hon}| = n-(t-1)$. We will show that when PedPoP outputs public key share $\tilde{X}_k = g^{\bar{x}_k}$ for each honest party $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}_1$ returns (α_k, β_k) such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$.

$\mathcal{B}_1$ is responsible for simulating honest parties in PedPoP (Fig. 12) and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_1$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and random coins ω. It can query the random oracle $\mathcal{O}^{\mathsf{RO}}$ from $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$. It can also query $\mathcal{O}^{\mathsf{schnorr\text{-}koe}}$ to receive signatures under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and $\mathcal{O}^{\mathsf{chal}}$ on inputs $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ to challenge the extractor Ext to output a discrete logarithm $a_{j,0}$ for $X_{j,0}$.

Initialization. $\mathcal{B}_1$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\tilde{\mathsf{H}}_{\mathsf{reg}}$ (because it is part of $\mathcal{B}_1$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses. $\mathcal{B}_1$ computes α_k for each honest party $\mathsf{id}_k \in \mathsf{hon}$ as follows. First, $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_k(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in \mathsf{cor}}\}$ relating to the set $\mathsf{id}_k \cup \mathsf{cor}$. Then, $\mathcal{B}_1$ sets $\alpha_k \leftarrow L'_k(0)^{-1}$. It will later become clear why α_k is computed this way.

Simulating Hash Queries. $\mathcal{B}_1$ handles $\mathcal{A}$'s hash queries throughout the DKG protocol as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_1$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_1$ queries $\bar{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

$\underline{\mathsf{H}_{\mathsf{non}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$, $\mathcal{B}_1$ queries $\hat{a} \leftarrow \tilde{\mathsf{H}}_{\mathsf{non}}(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$ and returns $\hat{a}$.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_1$ queries $\hat{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{sig}}(X, m, R)$ and returns $\hat{c}$.

Simulating the DKG. $\mathcal{B}_1$ runs $\mathcal{A}$ on input random coins ω and simulates PedPoP as follows. $\mathcal{B}_1$ first queries $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ and receives $(\dot{X}, \bar{R}_\tau, \bar{z}_\tau)$. $\mathcal{B}_1$ embeds $\dot{X}$ as the public key of the honest party that the adversary queries first. Let this first honest party be id_τ. $\mathcal{B}_1$ simulates the public view of id_τ but follows the PedPoP protocol for all other honest parties $\{\mathsf{id}_k\}_{k \neq \tau}$ as prescribed. Note that $\mathcal{A}$ can choose the order in which it interacts with honest parties, so $\mathcal{B}_1$ must be able to simulate any of them.

Honest Party id_τ. $\mathcal{B}_1$ is required to output

$$(\bar{R}_\tau, \bar{z}_\tau), \boldsymbol{C}_\tau = (A_{\tau,0} = X_{\tau,0}, A_{\tau,1}, ..., A_{\tau,t-1})$$

that are indistinguishable from valid outputs as well as $t-1$ shares $f_\tau(\mathsf{id}_j) = \bar{x}_{\tau,j}$, one to be sent to each corrupt party $\mathsf{id}_j \in \mathsf{cor}$. Here, $(\bar{R}_\tau, \bar{z}_\tau)$ is a Schnorr signature proving knowledge of the discrete logarithm of $X_{\tau,0}$, and $\boldsymbol{C}_\tau$ is a commitment to the coefficients that represent f_τ. $\mathcal{B}_1$ simulates honest party id_τ as follows.

1. $\mathcal{B}_1$ sets the public key $X_{\tau,0} \leftarrow \dot{X}$.
2. $\mathcal{B}_1$ simulates a verifiable Shamir secret sharing of the discrete logarithm of $\dot{X}$ by performing the following steps.
   (a) $\mathcal{B}_1$ samples $t-1$ random values $\bar{x}_{\tau,j} \leftarrow\!\!\$\ \mathbb{Z}_p$ for $\mathsf{id}_j \in \mathsf{cor}$.
   (b) Let f_τ be the polynomial whose constant term is the challenge $f_\tau(0) = \dot{x}$ and for which $f_\tau(\mathsf{id}_j) = \bar{x}_{\tau,j}$ for all $\mathsf{id}_j \in \mathsf{cor}$. $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_0(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in \mathsf{cor}}\}$ relating to the set $0 \cup \mathsf{cor}$.
   (c) For $1 \leq i \leq t-1$, $\mathcal{B}_1$ computes

$$A_{\tau,i} = \dot{X}^{L'_{0,i}} \prod_{\mathsf{id}_j \in \mathsf{cor}} g^{\bar{x}_{\tau,j} L'_{j,i}} \tag{3}$$

   where $L'_{j,i}$ is the i^{th} coefficient of $L'_j(Z) = L'_{j,0} + L'_{j,1} Z + \cdots + L'_{j,t-1} Z^{t-1}$.
   (d) $\mathcal{B}_1$ outputs $(\bar{R}_\tau, \bar{z}_\tau), \boldsymbol{C}_\tau = (A_{\tau,0} = X_{\tau,0}, A_{\tau,1}, ..., A_{\tau,t-1})$ for the broadcast round, and then sends shares $\bar{x}_{\tau,j}$ for each $\mathsf{id}_j \in \mathsf{cor}$.
3. $\mathcal{B}_1$ simulates private shares $f_\tau(\mathsf{id}_k) = \bar{x}_{\tau,k}$ for honest parties $\mathsf{id}_k \in \mathsf{hon}$ by computing α'_k, β'_k such that $g^{\bar{x}_{\tau,k}} = \dot{X}^{\alpha'_k} g^{\beta'_k}$. First, $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_k(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in \mathsf{cor}}\}$ relating to the set $\mathsf{id}_k \cup \mathsf{cor}$. Then, implicitly,

$$f_\tau(0) = \dot{x} = \bar{x}_{\tau,k} L'_k(0) + \sum_{\mathsf{id}_j \in \mathsf{cor}} \bar{x}_{\tau,j} L'_j(0)$$

   Solving for $\bar{x}_{\tau,k}$, $\mathcal{B}_1$ sets $\alpha'_k = L'_k(0)^{-1}$ and $\beta'_k = -\alpha'_k \sum_{\mathsf{id}_j \in \mathsf{cor}} \bar{x}_{\tau,j} L'_j(0)$.

All Other Honest Parties. For all other honest parties $\mathsf{id}_k \in \mathsf{hon}, k \neq \tau$, $\mathcal{B}_1$ follows the protocol: $\mathcal{B}_1$ samples $f_k(Z) = a_{k,0} + a_{k,1} Z + ... + a_{k,t-1} Z^{t-1} \leftarrow\!\!\$\ \mathbb{Z}_p[Z]$ and sets $A_{k,i} \leftarrow g^{a_{k,i}}$ for all $i = 0, \ldots, t-1$. $\mathcal{B}_1$ provides a proof of possession

$(\bar{R}_k, \bar{z}_k)$ of the public key $X_{k,0} = A_{k,0}$ and computes the private shares $\bar{x}_{k,i} = f_k(\mathsf{id}_i)$.

Adversarial Contributions. When $\mathcal{A}$ returns a contribution

$$(\bar{R}_j, \bar{z}_j), \boldsymbol{C}_j = (A_{j,0} = X_{j,0}, A_{j,1}, ..., A_{j,t-1})$$

if $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ verifies (i.e., $\bar{R}_j X_{j,0}^{\tilde{\mathsf{H}}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j)} = g^{\bar{z}_j}$) and $X_{j,0} \neq \dot{X}$, then $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_{j,0}, \bar{R}_j, \bar{z}_j)$ from $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

Complaints. If $\mathcal{A}$ broadcasts a complaint, $\mathcal{B}_1$ reveals the relevant $\bar{x}_{k,j}$. If $\mathcal{A}$ does not send verifying $\bar{x}_{j,k}$ to party $\mathsf{id}_k \in \mathsf{hon}$, then $\mathcal{B}_1$ broadcasts a complaint. If $\bar{x}_{j,k}$ fails to satisfy the equation, or should $\mathcal{A}$ not broadcast a share at all, then id_j is disqualified.

DKG Termination. When PedPoP terminates, the output is the joint public key $\tilde{X} = \prod_{i=0}^{n} X_{i,0}$. $\mathcal{B}_1$ simulates private shares $\bar{x}_k$ for honest parties $\mathsf{id}_k \in \mathsf{hon}$ by computing α_k, β_k such that $\tilde{X}_k = g^{\bar{x}_k} = \dot{X}^{\alpha_k} g^{\beta_k}$. Implicitly, $\bar{x}_k = \bar{x}_{\tau,k} + \sum_{i=1, i\neq\tau}^{n} \bar{x}_{i,k}$ and $\bar{x}_{\tau,k} = \dot{x}\alpha'_k + \beta'_k$ from Step 2 above, so $\alpha_k = \alpha'_k$ and $\beta_k = \beta'_k + \sum_{i=1, i\neq\tau}^{n} \bar{x}_{i,k}$. $\mathcal{B}_1$ returns $\{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in \mathsf{hon}}$.

We now argue that: (1) $\mathcal{A}$ cannot distinguish between a real run of the DKG protocol and its interaction with $\mathcal{B}_1$; and (2) $\mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ outputs $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ whenever $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_{j,0}, \bar{R}_j, \bar{z}_j)$.

(1) Observe that $\mathcal{B}_1$'s simulation of PedPoP is perfect, as performing validation of each player's share (Step 4 in Fig. 12) holds, and by Equation 3, interpolation in the exponent correctly evaluates to the challenge $\dot{X}$.

(2) Observe that $\mathsf{H}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j) = \tilde{\mathsf{H}}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j)$ unless $(X_{j,0}, \bar{R}_j) = (\dot{X}, \bar{R}_\tau)$. The latter happens only if $X_{j,0} = X_{\tau,0}$, but in this case PedPoP will not terminate. We thus have that $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ is a verifying signature under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and either Ext succeeds, or $\mathcal{B}_1$ breaks the schnorr-koe assumption. Therefore, the probability of the event occurring where Ext fails to outputs $a_{j,0}$ is bounded by $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

The Reduction $\mathcal{B}_2$: We next define the reduction $\mathcal{B}_2$ against bischnorr. We will show that when PedPoP outputs the joint public key $\tilde{X}$, $\mathcal{B}_2$ returns y such that $\tilde{X} = \dot{X} g^y$. Together with the (α_k, β_k) returned by $\mathcal{B}_1$ such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$, this representation allows $\mathcal{B}_2$ to simulate FROST2 signing under each public key share $\tilde{X}_k$. $\mathcal{B}_2$ is responsible for simulating honest parties during signing and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_2$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}$. It can query $\mathcal{O}^{\mathsf{binonce}}, \mathcal{O}^{\mathsf{bisign}}$ and $\mathcal{O}^{\mathsf{RO}}$ from $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{B}_2}(\lambda)$.

Initialization. $\mathcal{B}_2$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\hat{\mathsf{H}}_{\mathsf{non}}$ or $\hat{\mathsf{H}}_{\mathsf{sig}}$ (because they are part of $\mathcal{B}_2$'s challenge). Let Q_{Sign} be the set of $\mathcal{O}^{\mathsf{Sign}}$ queries and responses in Signing Round 1, and let $Q_{\mathsf{Sign}'}$ be the set of $\mathcal{O}^{\mathsf{Sign}'}$ queries and responses in Signing Round 2.

DKG Extraction. $\mathcal{B}_2$ first simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}_2$ samples $\bar{c}_\tau, \bar{z}_\tau \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $\bar{R}_\tau \leftarrow g^{\bar{z}_\tau}\dot{X}^{-\bar{c}_\tau}$, and appends $(\dot{X}, \dot{X}, \bar{R}_\tau, \bar{c}_\tau)$ to $\mathsf{Q}_{\mathsf{reg}}$. Then, $\mathcal{B}_2$ runs

$$\{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in \mathsf{hon}} \leftarrow\!\!\$\ \mathcal{B}_1(\mathcal{G}; \omega)$$

on random coins ω. $\mathcal{B}_2$ handles $\mathcal{B}_1$'s queries as follows. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{non}}, \tilde{\mathsf{H}}_{\mathsf{sig}}$, $\mathcal{B}_2$ handles them the same way it handles $\mathcal{A}$'s $\mathsf{H}_{\mathsf{non}}, \mathsf{H}_{\mathsf{sig}}$ queries, described below. The first time $\mathcal{B}_1$ queries its $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ oracle, $\mathcal{B}_2$ returns $(\dot{X}, \bar{R}_\tau, \bar{z}_\tau)$. When $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_{j,0}, \bar{R}_j, \bar{z}_j)$, $\mathcal{B}_2$ runs $a_{j,0} \leftarrow \mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ to obtain $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ and aborts otherwise. Then $y = \sum_{i=1, i\neq\tau}^{n} a_{i,0}$ such that $\dot{X} = \dot{X}g^y$.

Simulating Hash Queries. $\mathcal{B}_2$ handles $\mathcal{A}$'s hash queries throughout the signing protocol as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. Note that $\mathcal{B}_1$ and $\mathcal{B}_2$ share the state of $\mathsf{Q}_{\mathsf{reg}}$.

$\underline{\mathsf{H}_{\mathsf{non}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}_2$ checks whether $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}}, \hat{m}, \hat{a}) \in \mathsf{Q}_{\mathsf{non}}$ and, if so, returns $\hat{a}$. Else, $\mathcal{B}_2$ checks whether there exists some $k' \in \mathcal{S}$ such that $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$. If not, $\mathcal{B}_2$ samples a random message $\hat{m}$ and a random value $\hat{a}$, appends $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}}, \hat{m}, \hat{a})$ to $\mathsf{Q}_{\mathsf{non}}$, and returns $\hat{a}$.

If there does exist some $k' \in \mathcal{S}$ such that $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$, $\mathcal{B}_2$ computes the Lagrange coefficients $\{\lambda_i\}_{i\in\mathcal{S}}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i\in\mathcal{S}}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i\in\mathcal{S}}$. $\mathcal{B}_2$ sets $\gamma_k = \lambda_k\alpha_k$ for all $\mathsf{id}_k \in \mathsf{hon}$ and $\gamma_j = \lambda_j$ for all $\mathsf{id}_j \in \mathsf{cor}$ in the set $\mathcal{S}$. $\mathcal{B}_2$ then samples a random message $\hat{m}$, queries $\hat{a} \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}, \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})$, and appends $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}}, \hat{m}, \hat{a})$ to $\mathsf{Q}_{\mathsf{non}}$. $\mathcal{B}_2$ computes $\hat{R} = \prod_{i\in\mathcal{S}} R_i S_i^{\hat{a}}$ and checks if there exists a record $(X, m, \hat{R}, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$. If so, $\mathcal{B}_2$ aborts. Else, $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, \hat{R})$ and appends $(\dot{X}, m, \hat{R}, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. Finally, $\mathcal{B}_2$ returns $\hat{a}$.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}_2$ samples a random message $\hat{m}$, queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$, appends $(X, m, R, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\hat{c}$.

Simulating FROST2 Signing. After $\mathcal{B}_1$ completes the simulation of PedPoP, $\mathcal{B}_2$ then simulates honest parties in the FROST2 signing protocol.

Signing Round 1 (Sign)**.** When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}}$ on $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}_2$ queries $\mathcal{O}^{\mathsf{binonce}}$ to get (R_k, S_k), appends $(\mathsf{id}_k, R_k, S_k)$ to Q_{Sign}, and returns (R_k, S_k).

Signing Round 2 (Sign′)**.** When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}'}$ on $(k', m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}_2$ first checks whether $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$ and, if not, returns $\perp$. Then, $\mathcal{B}_2$ checks whether $(R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}'}$ and, if so, returns $\perp$.

If all checks pass, $\mathcal{B}_2$ internally queries $\hat{\mathsf{H}}_{\mathsf{non}}$ on $(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$ to get $\hat{a}'$ and looks up $\hat{m}'$ such that $(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}}), \hat{m}', \hat{a}') \in \mathsf{Q}_{\mathsf{non}}$. $\mathcal{B}_2$ computes $\hat{R}' = \prod_{i\in\mathcal{S}} R_i S_i^{\hat{a}'}$ and internally queries $\hat{\mathsf{H}}_{\mathsf{sig}}$ on $(\tilde{X}, m, \hat{R}')$ to get $\hat{c}'$.

Next, $\mathcal{B}_2$ computes the Lagrange coefficients $\{\lambda_i\}_{i\in\mathcal{S}}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i\in\mathcal{S}}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i\in\mathcal{S}}$. $\mathcal{B}_2$ sets $\gamma_k = \lambda_k\alpha_k$ for all $\mathsf{id}_k \in \mathsf{hon}$ and $\gamma_j = \lambda_j$ for all $\mathsf{id}_j \in \mathsf{cor}$ in the set $\mathcal{S}$. Then, $\mathcal{B}_2$ queries $\mathcal{O}^{\mathsf{bisign}}$ on $(k', \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})$ to get $z_{k'}$. Finally, $\mathcal{B}_2$ computes

$$\tilde{z}_{k'} = z_{k'} + \hat{c}'\lambda_{k'}\beta_{k'} \tag{4}$$

For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}_2$ returns $\tilde{z}_{k'}$.

Output. When $\mathcal{A}$ returns $(\tilde{X}, m^*, \sigma^*)$ such that $\sigma^* = (\tilde{R}^*, z^*)$ and $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, $\mathcal{B}_2$ computes its output as follows. $\mathcal{B}_2$ looks up $\hat{m}^*$ such that $(\tilde{X}, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ and outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^* y)$.

To complete the proof, we must argue that: (1) $\mathcal{B}_2$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}_2$ succeeds.

(1) $\mathcal{B}_2$ aborts if Ext fails to return $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ for some $\mathsf{id}_j \in \mathsf{cor}$. This happens with maximum probability $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

$\mathcal{B}_2$ aborts if $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \prod_{i\in\mathcal{S}} R_i S_i^{\hat{a}})$ before having first queried $\mathsf{H}_{\mathsf{non}}$ on $(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$. This requires $\mathcal{A}$ to have guessed $\hat{a}$ ahead of time, which occurs with negligible probability q_H/p.

(2) As long as $\mathcal{B}_2$ does not abort, $\mathcal{B}_2$ is able to simulate the appropriate responses to $\mathcal{A}$'s oracle queries so that $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$.

Indeed, as already observed, $\mathcal{B}_1$'s simulation of PedPoP is perfect.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$ on a random message $\hat{m}$. The random message prevents trivial collisions; for example, if $\mathcal{A}$ were to query $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R) and (X', m, R), where $X' \neq X$, $\mathcal{A}$ would receive the same value $c \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, m, R)$ for both and would know it was operating inside a reduction. Random messages ensure that the outputs are random, so $\mathcal{A}$'s view is correct. $\mathcal{B}_2$ also ensures that $\mathcal{A}$ receives $\mathsf{H}_{\mathsf{non}}$ values that are consistent with $\mathsf{H}_{\mathsf{sig}}$ queries.

After the signing rounds have been completed, $\mathcal{A}$ may verify the signature share $\tilde{z}_{k'}$ on m as follows. $\mathcal{A}$ checks if

$$R_{k'} S_{k'}^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})} \tilde{X}_{k'}^{\lambda_{k'}\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_{i\in\mathcal{S}} R_i S_i^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})})} = g^{\tilde{z}_{k'}} \tag{5}$$

When $\mathcal{B}_2$ queried $\mathcal{O}^{\mathsf{bisign}}$ on $(k', \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})$ in Signing Round 2, the signature share $z_{k'}$ was computed such that

$$R_{k'} S_{k'}^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})} \dot{X}^{\gamma_{k'}\hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}', \prod_{i\in\mathcal{S}} R_i S_i^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})})} = g^{z_{k'}}$$

$\mathcal{B}$ computed the signature share $\tilde{z}_{k'}$ (Equation 4) as

$$\begin{aligned}\tilde{z}_{k'} = z_{k'} + \hat{c}'\lambda_{k'}\beta_{k'} &= r_{k'} + as_{k'} + \hat{c}'\gamma_{k'}\dot{x} + \hat{c}'\lambda_{k'}\beta_{k'} \\ &= r_{k'} + as_{k'} + \hat{c}'\lambda_{k'}(\alpha_{k'}\dot{x} + \beta_{k'})\end{aligned}$$

where $\hat{c}' = \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}', \prod_{i\in\mathcal{S}} R_i S_i^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X},\hat{m}',\{(\gamma_i,R_i,S_i)\}_{i\in\mathcal{S}})})$. Thus, $\tilde{z}_{k'}$ satisfies

$$R_{k'} S_{k'}^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X},\hat{m}',\{(\gamma_i,R_i,S_i)\}_{i\in\mathcal{S}})} \tilde{X}_{k'}^{\lambda_{k'}\hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X},\hat{m}',\prod_{i\in\mathcal{S}} R_i S_i^{\hat{\mathsf{H}}_{\mathsf{non}}(\dot{X},\hat{m}',\{(\gamma_i,R_i,S_i)\}_{i\in\mathcal{S}})})} = g^{\tilde{z}_{k'}} \tag{6}$$

$\mathcal{B}_2$ has programmed the hash values in Equations 5 and 6 to be equal and therefore simulates $\tilde{z}_{k'}$ correctly.

(3) $\mathcal{A}$'s forgery satisfies $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, which implies:

$$\begin{aligned}\tilde{R}^*(\tilde{X})^{\mathsf{H}_{\mathsf{sig}}(\tilde{X},m^*,\tilde{R}^*)} &= g^{z^*} \\ \tilde{R}^*(\dot{X}g^y)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X},m^*,\tilde{R}^*)} &= g^{z^*} \\ \tilde{R}^*\dot{X}^{\mathsf{H}_{\mathsf{sig}}(\tilde{X},m^*,\tilde{R}^*)} &= g^{z^* - \mathsf{H}_{\mathsf{sig}}(\tilde{X},m^*,\tilde{R}^*)y}\end{aligned}$$

At some point, $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m^*, \tilde{R}^*)$ and received one of two values: (1) $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \prod_{i\in\mathcal{S}^*} R_i^*(S_i^*)^{\hat{a}^*})$ related to a query $\mathcal{A}$ made to $\mathsf{H}_{\mathsf{non}}$ on $(m^*, \{(\mathsf{id}_i^*, R_i^*, S_i^*)\}_{i\in\mathcal{S}^*})$, where it received $\hat{a}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(\dot{X}, \hat{m}^*, (\gamma_i^*, R_i^*, S_i^*)_{i\in\mathcal{S}^*})$, or (2) $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$ without having queried $\mathsf{H}_{\mathsf{non}}$ first. In either case, $\mathcal{B}_2$ has a record $(\tilde{X}, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ such that $\hat{c}^* \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$. (Note that $\mathcal{B}_2$ can check which case occurred by looking for $\hat{m}^*$ in its $\mathsf{Q}_{\mathsf{non}}$ records.) Thus, $\mathcal{A}$'s forgery satisfies

$$\tilde{R}^*\dot{X}^{\hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X},\hat{m}^*,\tilde{R}^*)} = g^{z^* - \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X},\hat{m}^*,\tilde{R}^*)y}$$

and $\mathcal{B}_2$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^* y)$ under $\dot{X}$ is correct.

6 Conclusion

We present new techniques for proving the security of multi- and threshold Schnorr signature schemes. We demonstrate the strength of this methodology by proving the security of a range of multi-party schemes, including a three-round threshold signature SimpleTSig, a variant of the two-round multisignature MuSig2 [36], and an optimization to the FROST [31] threshold signature scheme that we call FROST2. A goal of this work is to provide a simple framework for proving the security of other multi-party Schnorr signature schemes in the future, perhaps even blinded or unlinkable variants.

Acknowledgements. Elizabeth Crites was supported by Input Output through their funding of the Edinburgh Blockchain Technology Lab.

References

[1] M. Abdalla, F. Benhamouda, and P. MacKenzie. "Security of the J-PAKE password-authenticated key exchange protocol". In: *2015 IEEE Symposium on Security and Privacy*. IEEE. 2015, pp. 571–587.

[2] B. Abdolmaleki, K. Baghery, H. Lipmaa, and M. Zajac. "A Subversion-Resistant SNARK". In: *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017*. Ed. by T. Takagi and T. Peyrin. Vol. 10626. LNCS. Springer, 2017, pp. 3–33.

[3] H. K. Alper and J. Burdges. "Two-Round Trip Schnorr Multi-signatures via Delinearized Witnesses". In: *CRYPTO 2021, Virtual Event, August 16-20, 2021*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Springer, 2021, pp. 157–188.

[4] A. Bagherzandi, J. H. Cheon, and S. Jarecki. "Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma". In: *CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*. Ed. by P. Ning, P. F. Syverson, and S. Jha. ACM, 2008, pp. 449–458.

[5] M. Bellare, E. Crites, C. Komlo, M. Maller, S. Tessaro, and C. Zhu. *Better than advertised security for non-interactive threshold signatures*. CRYPTO 2022. To appear. 2022.

[6] M. Bellare and W. Dai. "Chain Reductions for Multi-Signatures". In: *IACR Cryptol. ePrint Arch.* (2021), p. 404. URL: `https://eprint.iacr.org/2021/404`.

[7] M. Bellare, W. Dai, and L. Li. "The Local Forking Lemma and Its Application to Deterministic Encryption". In: *ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019*. Ed. by S. D. Galbraith and S. Moriai. Vol. 11923. LNCS. Springer, 2019, pp. 607–636.

[8] M. Bellare and G. Neven. "Multi-signatures in the plain public-Key model and a general forking lemma". In: *CCS 2006, Alexandria, VA, USA, October 30 - November 3, 2006*. Ed. by A. Juels, R. N. Wright, and S. D. C. di Vimercati. ACM, 2006, pp. 390–399.

[9] M. Bellare and A. Palacio. "The Knowledge-of-Exponent Assumptions and 3-Round Zero-Knowledge Protocols". In: *CRYPTO 2004, Santa Barbara, California, USA, August 15-19, 2004*. Ed. by M. K. Franklin. Vol. 3152. LNCS. Springer, 2004, pp. 273–289.

[10] M. Bellare and P. Rogaway. "The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs". In: *EUROCRYPT 2006, St. Petersburg, Russia, May 28 - June 1, 2006*. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, 2006, pp. 409–426.

[11] M. Bellare, S. Tessaro, and C. Zhu. "Stronger Security for Non-Interactive Threshold Signatures". In: *IACR Cryptol. ePrint Arch.* (2022), p. 833. URL: `https://eprint.iacr.org/2022/833`.

[12] F. Benhamouda, T. Lepoint, J. Loss, M. Orrù, and M. Raykova. "On the (in)security of ROS". In: *EUROCRYPT 2021, Zagreb, Croatia, October 17-21, 2021*. Ed. by A. Canteaut and F. Standaert. Vol. 12696. LNCS. Springer, 2021, pp. 33–53.

[13] A. Boldyreva. "Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme". In: *PKC 2003, Miami, FL, USA, January 6-8, 2003*. Ed. by Y. Desmedt. Vol. 2567. LNCS. Springer, 2003, pp. 31–46.

[14] D. Boneh, M. Drijvers, and G. Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6,*

2018. Ed. by T. Peyrin and S. D. Galbraith. Vol. 11273. LNCS. Springer, 2018, pp. 435–464.

[15] D. Boneh and V. Shoup. *A Graduate Course in Applied Cryptography*. 2020. URL: http://toc.cryptobook.us/book.pdf.

[16] I. Damgård. "Towards Practical Public Key Systems Secure Against Chosen Ciphertext Attacks". In: *CRYPTO '91, Santa Barbara, California, USA, August 11-15, 1991*. Ed. by J. Feigenbaum. Vol. 576. LNCS. Springer, 1991, pp. 445–456.

[17] M. Drijvers, K. Edalatnejad, B. Ford, and G. Neven. "Okamoto Beats Schnorr: On the Provable Security of Multi-Signatures". In: *IACR Cryptol. ePrint Arch.* (2018), p. 417. URL: https://eprint.iacr.org/2018/417.

[18] M. Drijvers et al. "On the Security of Two-Round Multi-Signatures". In: *SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1084–1101.

[19] P. Feldman. "A Practical Scheme for Non-interactive Verifiable Secret Sharing". In: *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, October 27-29, 1987*. IEEE, 1987, pp. 427–437.

[20] A. Fiat and A. Shamir. "How to Prove Yourself: Practical Solutions to Identification and Signature Problems". In: *CRYPTO 1986, Santa Barbara, California, USA, 1986*. Ed. by A. M. Odlyzko. Vol. 263. LNCS. Springer, 1986, pp. 186–194.

[21] M. Fischlin. "Communication-Efficient Non-interactive Proofs of Knowledge with Online Extractors". In: *CRYPTO 2005, Santa Barbara, California, USA, August 14-18, 2005*. Ed. by V. Shoup. Vol. 3621. LNCS. Springer, 2005, pp. 152–168.

[22] G. Fuchsbauer, E. Kiltz, and J. Loss. "The Algebraic Group Model and its Applications". In: *CRYPTO 2018, Santa Barbara, CA, USA, August 19-23, 2018*. Ed. by H. Shacham and A. Boldyreva. Vol. 10992. LNCS. Springer, 2018, pp. 33–62.

[23] A. Gabizon. "On the security of the BCTV Pinocchio zk-SNARK variant". In: *IACR Cryptol. ePrint Arch.* (2019), p. 119. URL: https://eprint.iacr.org/2019/119.

[24] R. Gennaro and S. Goldfeder. "Fast Multiparty Threshold ECDSA with Fast Trustless Setup". In: *CCS 2018, Toronto, ON, Canada, October 15-19, 2018*. Ed. by D. Lie, M. Mannan, M. Backes, and X. Wang. ACM, 2018, pp. 1179–1194.

[25] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Robust Threshold DSS Signatures". In: *Inf. Comput.* 164.1 (2001), pp. 54–84.

[26] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Applications of Pedersen's Distributed Key Generation Protocol". In: *CT-RSA 2003, San Francisco, CA, USA, April 13-17, 2003*. Ed. by M. Joye. Vol. 2612. LNCS. Springer, 2003, pp. 373–390.

[27] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. "Secure Distributed Key Generation for Discrete-Log Based Cryptosystems". In: *J. Cryptol.* 20.1 (2007), pp. 51–83.

[28] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk. "Robust and Efficient Sharing of RSA Functions". In: *J. Cryptol.* 20.3 (2007), p. 393.

[29] C. Gentry and D. Wichs. "Separating Succinct Non-Interactive Arguments From All Falsifiable Assumptions". In: *STOC 2011, San Jose, CA, USA, 6-8 June 2011*. Ed. by L. Fortnow and S. P. Vadhan. ACM, 2011, pp. 99–108.

[30] J. Groth. "Short Pairing-Based Non-interactive Zero-Knowledge Arguments". In: *ASIACRYPT 2010, Singapore, December 5-9, 2010*. Ed. by M. Abe. Vol. 6477. LNCS. Springer, 2010, pp. 321–340.

[31] C. Komlo and I. Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *SAC 2020, Halifax, NS, Canada (Virtual Event), October*

21-23, 2020. Ed. by O. Dunkelman, M. J. J. Jr., and C. O'Flynn. Vol. 12804. LNCS. Springer, 2020, pp. 34–65.

[32] Y. Lindell. "Simple Three-Round Multiparty Schnorr Signing with Full Simulatability". In: *IACR Cryptol. ePrint Arch.* (2022), p. 374. URL: https://eprint.iacr.org/2022/374.

[33] C. Ma, J. Weng, Y. Li, and R. H. Deng. "Efficient discrete logarithm based multi-signature scheme in the plain public key model". In: *Des. Codes Cryptogr.* 54.2 (2010), pp. 121–133.

[34] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. "Simple Schnorr multi-signatures with applications to Bitcoin". In: *Des. Codes Cryptogr.* 87.9 (2019), pp. 2139–2164.

[35] J. Nick. *Insecure Shortcuts in MuSig*. 2019. URL: https://medium.com/blockstream/insecure-shortcuts-in-musig-2ad0d38a97da.

[36] J. Nick, T. Ruffing, and Y. Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *CRYPTO 2021, Virtual Event, August 16-20, 2021*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Springer, 2021, pp. 189–221.

[37] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille. "MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces". In: *CCS 2020, Virtual Event, USA, November 9-13, 2020*. Ed. by J. Ligatti, X. Ou, J. Katz, and G. Vigna. ACM, 2020, pp. 1717–1731.

[38] D. Pointcheval and J. Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *J. Cryptol.* 13.3 (2000), pp. 361–396.

[39] D. Pointcheval and J. Stern. "Security Proofs for Signature Schemes". In: *EUROCRYPT 1996, Saragossa, Spain, May 12-16, 1996*. Ed. by U. M. Maurer. Vol. 1070. LNCS. Springer, 1996, pp. 387–398.

[40] T. Ristenpart and S. Yilek. "The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks". In: *EUROCRYPT 2007, Barcelona, Spain, May 20-24, 2007*. Ed. by M. Naor. Vol. 4515. LNCS. Springer, 2007, pp. 228–245.

[41] C. Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptol.* 4.3 (1991), pp. 161–174.

[42] Y. Seurin. "On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model". In: *EUROCRYPT 2012, Cambridge, UK, April 15-19, 2012*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, 2012, pp. 554–571.

[43] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.

[44] V. Shoup and R. Gennaro. "Securing Threshold Cryptosystems against Chosen Ciphertext Attack". In: *EUROCRYPT '98, Finland, May 31 - June 4, 1998*. Ed. by K. Nyberg. Vol. 1403. LNCS. Springer, 1998, pp. 1–16.

[45] E. Syta et al. "Scalable Bias-Resistant Distributed Randomness". In: *SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE, 2017, pp. 444–460.

[46] M. Zochowski. *Benchmarking Hash and Signature Algorithms*. 2019. URL: https://medium.com/logos-network/benchmarking-hash-and-signature-algorithms-6079735ce05.

A Proof of the Schnorr Knowledge of Exponent Assumption

Here we frame our Schnorr knowledge of exponent (schnorr-koe) assumption (Fig. 5) in the context of prior knowledge of exponent assumptions.

Signature schemes are not the only branch of cryptography where standard model security proofs are evasive. For example, there exist CCA encryption schemes that are provably secure in the random oracle model but not in the standard model [15]. In arguing the security of a practical CCA encryption scheme using assumptions that more closely resemble real restrictions on an adversary, Damgard [16] introduced the knowledge of exponent (KoE) assumption. KoE says that for every algorithm $\mathcal{A}$ given a generator g and random $X = g^x$, if $\mathcal{A}$ outputs (A, B) such that $B = A^x$, then there exists an extractor algorithm Ext that, given the same input, outputs a such that $(A, B) = (g^a, X^a)$. Informally, this means that, given a pair (g, g^x), the only way to produce such a pair (A, B) is by exponentiating the original pair (g, g^x) by the exponent a, thereby implying knowledge of a. This assumption is non-falsifiable, i.e., one cannot show the non-existence of an extractor, and provides one of the few alternatives to the random oracle model should a standard model proof be impossible.

Knowledge of exponent assumptions have subsequently been generalized, with Abdolmaleki et al. [2] extending them to bilinear groups and Groth [30] suggesting a "q-type" variation. They are used extensively in the security proofs of Succinct NIZK Arguments (SNARKs). This is often justified by arguing that SNARKs do not exist in the standard model due to an impossibility result of Gentry and Wich's [29]. One extreme variation is the algebraic group model (AGM) [22], which suggests that all natural KoE assumptions hold but that security proofs should still reduce to computational assumptions.

We now prove Theorem 1, which states that our schnorr-koe assumption is implied by the discrete logarithm assumption in the algebraic group model.

Proof. ($\mathsf{dl} \Rightarrow$ schnorr-koe) Let $\mathcal{A}$ be an algebraic adversary playing $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$ (Fig. 5). $\mathcal{A}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and random coins ω. Let $Q_{\mathsf{sch\text{-}pop}} = \{(X_1, \bar{R}_1, \bar{z}_1), \ldots, (X_{q_S}, \bar{R}_{q_S}, \bar{z}_{q_S})\}$ denote the responses made by the signing oracle $\mathcal{O}^{\mathsf{sch\text{-}pop}}$. Whenever $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{chal}}$ on verifying $(X^*, \bar{R}^*, \bar{z}^*) \notin Q_{\mathsf{sch\text{-}pop}}$, it also outputs an algebraic representation of X^* and $\bar{R}^*$. That is, it outputs a representation $(\alpha_0, \alpha_1, \beta_1, \ldots, \alpha_{q_S}, \beta_{q_S})$ such that

$$X^* = g^{\alpha_0} \prod_{j \in [q_S]} X_j^{\alpha_j} \bar{R}_j^{\beta_j}$$

Let Ext be the extractor in $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$ that returns x^*, where $x^* = \alpha_0 + \sum_j \beta_j \bar{z}_j$. We argue that there exists a reduction $\mathcal{B}$ playing the discrete logarithm game $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}}(\lambda)$ such that whenever Ext does not succeed, i.e., $X^* \neq g^{x^*}$, then $\mathcal{B}$ returns the solution to a discrete logarithm challenge. $\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{sch\text{-}pop}}, \mathcal{O}^{\mathsf{chal}}, \mathcal{O}^{\tilde{\mathsf{RO}}}$, and $\tilde{\mathsf{H}}_{\mathsf{reg}}$. $\mathcal{B}$ may program $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and has access to its own oracle $\mathcal{O}^{\mathsf{RO}}$. $\mathcal{B}$ sees $\mathcal{A}$'s algebraic

representations (as the adversary is working within the AGM) but does not rewind the adversary.

The reduction $\mathcal{B}$. $\mathcal{B}$ takes as input the discrete logarithm challenge $\dot{X}$ and aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$. $\mathcal{B}$ initializes $Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}}$ to the empty set, chooses random coins $\omega \leftarrow\!\!\$\ \{0,1\}^{\mathsf{rl}_{\mathcal{A}}}$ and runs $\mathcal{A}(\mathcal{G}, \omega)$. We describe how $\mathcal{B}$ responds to oracle queries below, such that if $\mathcal{O}^{\mathsf{chal}}$ sets win to true at any point in the execution, then $\mathcal{B}$ returns $\dot{x}$. If $\mathcal{A}$ terminates and $\mathsf{win} \neq \mathsf{true}$, $\mathcal{A}$ fails to win $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$ and $\mathcal{B}$ aborts.

Simulating Hash Queries. When $\mathcal{A}$ queries $\tilde{\mathsf{H}}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}$ samples $\bar{c} \leftarrow\!\!\$\ \mathbb{Z}_p$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

When $\mathcal{A}$ queries $\mathcal{O}^{\tilde{\mathsf{RO}}}(\theta)$, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{RO}}(\theta)$ and forwards the response to $\mathcal{A}$.

$\mathcal{O}^{\mathsf{sch\text{-}pop}}$ Queries. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ for the j^{th} time, $\mathcal{B}$ samples $a_j, \bar{c}_j, \bar{z}_j \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets $X_j \leftarrow \dot{X}^{\bar{a}_j}$ and $\bar{R}_j \leftarrow g^{\bar{z}_j} X_j^{-\bar{c}_j}$. $\mathcal{B}$ appends $(X_j, X_j, \bar{R}_j, \bar{c}_j)$ to $\mathsf{Q}_{\mathsf{reg}}$, $(X_j, \bar{R}_j, \bar{z}_j)$ to $Q_{\mathsf{sch\text{-}pop}}$, and returns $(X_j, \bar{R}_j, \bar{z}_j)$.

$\mathcal{O}^{\mathsf{chal}}$ Queries. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{chal}}$ on input $(X^*, \bar{R}^*, \bar{z}^*)$, $\mathcal{B}$ queries $\bar{c}^* \leftarrow \tilde{\mathsf{H}}_{\mathsf{reg}}(X^*, X^*, \bar{R}^*)$ and checks that $(X^*, \bar{R}^*, \bar{z}^*) \notin Q_{\mathsf{sch\text{-}pop}}$ and $\bar{R}^*(X^*)^{\bar{c}^*} = g^{\bar{z}^*}$. If these checks fail, $\mathcal{B}$ returns $\perp$. Else, $\mathcal{B}$ runs Ext to obtain x^* from the entries in $Q_{\mathsf{sch\text{-}pop}}$ and $\mathsf{Q}_{\mathsf{reg}}$. Namely, the extractor uses the algebraic representation $(\alpha_0, \alpha_1, \beta_1, \ldots, \alpha_{q_S}, \beta_{q_S})$ such that

$$X^* = g^{\alpha_0} \prod_{j \in [q_S]} X_j^{\alpha_j} \bar{R}_j^{\beta_j}$$

and sets $x^* = \alpha_0 + \sum_j \beta_j \bar{z}_j$. If $g^{x^*} = X^*$, then $\mathcal{B}$ outputs x^*.

If $g^{x^*} \neq X^*$, then $\mathcal{B}$ looks up the algebraic representation $(\gamma_0, \gamma_1, \delta_1, \ldots, \gamma_{q_S}, \delta_{q_S})$ (which $\mathcal{A}$ provides when querying $\mathcal{O}^{\mathsf{chal}}$ on $\bar{R}^*$) such that

$$\bar{R}^* = g^{\gamma_0} \prod_{j \in [q_S]} X_j^{\gamma_j} \bar{R}_j^{\delta_j}$$

Then $\mathcal{B}$ computes

$$f(W) = (\alpha_0 + \sum_{j=1}^{q_S} \beta_j \bar{z}_j) + \sum_{j=1}^{q_S} a_j(\alpha_j - \bar{c}_j \beta_j) W = f_0 + f_1 W$$
$$h(W) = (\gamma_0 + \sum_{j=1}^{q_S} \delta_j \bar{z}_j) + \sum_{j=1}^{q_S} a_j(\gamma_j - \bar{c}_j \delta_j) W = h_0 + h_1 W$$

where W is some indeterminate value, $X^* = g^{f(\dot{x})}$, and $\bar{R}^* = g^{h(\dot{x})}$. Now if $h_1 + \bar{c}^* f_1 = 0$, then $\mathcal{B}$ aborts. Else, $\mathcal{B}$ terminates with

$$\dot{x} = \frac{\bar{z}^* - h_0 - \bar{c}^* f_0}{h_1 + \bar{c}^* f_1}$$

Advantage of $\mathcal{B}$. First, observe that $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda)$ except in the event that $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ randomly outputs some $(X, \bar{R}, \bar{z})$ such that $\tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$ has already been programmed. This happens with maximum probability $\frac{q_S q_H}{p}$, where q_S is the maximum number of signing queries and q_H is the maximum number of hash queries. If $\mathcal{A}$ wins, then there exists some $\mathcal{O}^{\mathsf{chal}}$ query $(X^*, \bar{R}^*, \bar{z}^*)$ such that

$$\bar{R}^*(X^*)^{\bar{c}^*} = g^{\bar{z}^*} \text{ and } X^* \neq g^{\alpha_0 + \sum_j \beta_j \bar{z}_j}$$

Substituting in the algebraic representation, this implies

$$\begin{aligned}
g^{\bar{z}^*} &= g^{\gamma_0} \prod_j X_j^{\gamma_j} \bar{R}_j^{\delta_j} \left(g^{\alpha_0} \prod_j X_j^{\alpha_j} \bar{R}_j^{\beta_j} \right)^{\bar{c}^*} \\
g^{\bar{z}^*} &= g^{\gamma_0 + \bar{c}^* \alpha_0} \prod_j X_j^{\gamma_j + \bar{c}^* \alpha_j} \bar{R}_j^{\delta_j + \bar{c}^* \beta_j} \\
\Rightarrow \bar{z}^* &= \gamma_0 + \bar{c}^* \alpha_0 + \sum_j \dot{x} a_j (\gamma_j + \bar{c}^* \alpha_j) + (\bar{z}_j - \bar{c}_j a_j \dot{x})(\delta_j + \bar{c}^* \beta_j) \\
\Rightarrow \dot{x} &= \frac{-\bar{z}^* + \gamma_0 + \bar{c}^* \alpha_0 + \sum_j \bar{z}_j (\delta_j + \bar{c}^* \beta_j)}{\sum_j \bar{c}_j a_j (\delta_j + \bar{c}^* \beta_j) - a_j (\gamma_j + \bar{c}^* \alpha_j)} \\
\Rightarrow \dot{x} &= \frac{\bar{z}^* - h_0 - \bar{c}^* f_0}{h_1 + \bar{c}^* f_1}
\end{aligned}$$

Thus, $\mathcal{B}$ returns a correct discrete logarithm provided that $h_1 + \bar{c}^* f_1 \neq 0$. The probability that $h_1 + \bar{c}^* f_1 = 0$ for h_1, f_1 not both 0 and for $(X^*, \bar{R}^*, \bar{z}^*) \notin Q_{\mathsf{sch\text{-}pop}}$ is $\frac{q_S q_H}{p}$.

If $h_1 = f_1 = 0$, then $f(\dot{x}) = f_0 = (\alpha_0 + \sum_{j=1}^{q_S} \beta_j \bar{z}_j)$. Recall that

$$\begin{aligned}
X^* &= g^{\alpha_0} \prod_j X_j^{\alpha_j} \bar{R}_j^{\beta_j} \\
&= g^{\alpha_0 + \sum_j \dot{x} a_j \alpha_j + \beta_j (\bar{z}_j - \bar{c}_j a_j \dot{x})} \\
&= g^{\alpha_0 + \sum_j \beta_j \bar{z}_j + \dot{x}(\sum_j \alpha_j - \beta_j \bar{c}_j a_j)} \\
&= g^{f(\dot{x})}
\end{aligned}$$

However, this implies that $x^* = \alpha_0 + \sum_{j=1}^{q_S} \beta_j \bar{z}_j$ output by the extractor is correct, which is a contradiction.

Thus,

$$\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathsf{Ext}}(\lambda) \leq \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\lambda) + \frac{2 q_S q_H}{p}$$

B Proof of the Schnorr Computational Assumption

The Schnorr signature scheme was proven to reduce to the hardness of the discrete logarithm problem by Pointcheval and Stern [39]. There is a tightness loss because the probability that the adversary outputs two distinct forgeries with the same random oracle query $\hat{\mathsf{H}}_{\mathsf{sig}}(X, m^*, R^*)$, which is needed to extract the discrete logarithm solution, could be as low as $1/q_H$, where q_H is the number of random oracle queries the adversary makes. This is still non-negligible because the adversary can only make a polynomial number of queries. Currently, the only tight reduction is in the algebraic group model. While our Schnorr computational ($\mathsf{schnorr}$) assumption (Fig. 6) could be proven in the AGM, we find the proof to be cleaner in the random oracle model. Note that even when the adversary is allowed to interact with the simulator concurrently, the tightness loss for the security reduction remains q_H, since the simulator can always correctly program the random oracle to return a valid signing response.

We now prove Theorem 2, which states that our $\mathsf{schnorr}$ assumption is implied by the discrete logarithm assumption in the programmable random oracle model.

Proof. ($\mathsf{dl} \Rightarrow \mathsf{schnorr}$) Let $\mathcal{A}$ be a PPT adversary playing $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda)$ (Fig. 6) that makes up to q_H queries to $\hat{\mathsf{H}}_{\mathsf{sig}}$. We describe a PPT reduction $\mathcal{B}$ playing $\mathsf{Game}^{\mathsf{dl}}_{\mathcal{B}}(\lambda)$ that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) \leq \sqrt{q_H \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\lambda) + \frac{q_H^2}{p}}$$

The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times. Suppose $\mathcal{A}$ makes no more than q_H queries to $\hat{\mathsf{H}}_{\mathsf{sig}}$ in total over the two iterations. On the second iteration, $\mathcal{B}$ programs $\hat{\mathsf{H}}_{\mathsf{sig}}$ to output a different random value on a single point so that it can extract a discrete logarithm solution from $\mathcal{A}$'s outputs. $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda)$. However, $\mathcal{B}$ can only extract a discrete logarithm if $\mathcal{A}$'s output (m^*, R^*, z^*) at the end of each iteration verifies and includes the same nonce R^*. By the local forking lemma [7], this happens with probability $\frac{1}{q_H}(\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda))^2$.

$\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{schnorr}}, \mathcal{O}^{\hat{\mathsf{RO}}}$, and $\hat{\mathsf{H}}_{\mathsf{sig}}$. Let Q_{schnorr} be the set of $\mathcal{O}^{\mathsf{schnorr}}$ queries, and let $\mathsf{Q}_{\mathsf{sig}}$ be the set of $\hat{\mathsf{H}}_{\mathsf{sig}}$ queries and responses as in $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda)$. $\mathcal{B}$ may program $\hat{\mathsf{H}}_{\mathsf{sig}}$.

DL Input. $\mathcal{B}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and a discrete logarithm challenge $\dot{X}$. $\mathcal{B}$ aims to output $\dot{x}$ such that $\dot{X} = g^{\dot{x}}$.

Simulating Hash Queries. When $\mathcal{A}$ queries $\hat{\mathsf{H}}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (X, m, R, c) to $\mathsf{Q}_{\mathsf{sig}}$, and returns c.

Simulating Oracle Queries. For $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{schnorr}}$ on input m_i, $\mathcal{B}$ samples $c_i, z_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets $R_i \leftarrow g^{z_i}\dot{X}^{-c_i}$. $\mathcal{B}$ appends $(\dot{X}, m_i, R_i, c_i)$ to $\mathsf{Q}_{\mathsf{sig}}$, m_i to Q_{schnorr}, and returns (R_i, z_i).

Extracting the Discrete Logarithm of $\dot{X}$ from the Adversary. $\mathcal{B}$ initializes $\mathsf{Q}_{\mathsf{sig}}$ and Q_{schnorr} to the empty set. $\mathcal{B}$ then runs $\mathcal{A}(\dot{X};\omega)$ on the challenge $\dot{X}$ and random coins ω.

Suppose $\mathcal{A}$ terminates with (m^*, R^*, z^*). If $\mathcal{A}$ succeeds, then $R^*\dot{X}^{c^*} = g^{z^*}$, where $c^* = \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, m^*, R^*)$. Here, z^* does not suffice for $\mathcal{B}$ to extract the discrete logarithm of $\dot{X}$ because it does not necessarily know the discrete logarithm of R^*. Thus, $\mathcal{B}$ chooses $c' \leftarrow\!\!\$\ \mathbb{Z}_p$ and programs $\hat{\mathsf{H}}_{\mathsf{sig}}$ such that $c' = \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, m^*, R^*)$. $\mathcal{B}$ then runs $\mathcal{A}(\dot{X};\omega)$ again on the same random coins ω.

After the second iteration, suppose $\mathcal{A}$ terminates with (m', R', z'). If $(m', R') = (m^*, R^*)$ but $z' \neq z^*$, then $\mathcal{B}$ can extract $\dot{x} = \frac{z^*-z'}{c^*-c'}$ such that $\dot{X} = g^{\dot{x}}$. If $(m', R') \neq (m^*, R^*)$ or $\mathcal{A}$'s output does not verify, then $\mathcal{B}$ must abort. By the local forking lemma [7], this occurs with probability less than $\frac{1}{q_H}(\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda))^2$. If $\mathcal{A}$ succeeds having not queried $\hat{\mathsf{H}}_{\mathsf{sig}}$ on $(\dot{X}, m^*, R^*)$ or $(\dot{X}, m', R')$, then $\mathcal{B}$ aborts. This occurs with probability less than $\frac{q_H}{p}$. Thus,

$$\frac{1}{q_H}(\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda))^2 \leq \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\lambda) + \frac{q_H}{p}$$

C Background on the Two-Nonce Fix

In Section 4.5, we introduce the binonce Schnorr computational (bischnorr) assumption, which we use to prove the security of SpeedyMuSig and FROST2. We now expand on the use of two nonces in SpeedyMuSig and FROST/FROST2.

Why two nonces are necessary for two-round schemes. There is a danger when reducing two-round multi-party Schnorr signatures in the concurrent setting to the discrete logarithm assumption that if the adversary can learn the nonce *before* the reduction does, the reduction cannot always correctly program the random oracle (unlike in the single-party setting). Specifically, when the reduction publishes its nonce R_1 (simulating the honest signer), it must *guess* when programming the random oracle whether or not the adversary will query this nonce in the second round (and so is only guaranteed to succeed with likelihood $1/q_H$). Consequently, if κ is the number of signing requests the adversary can open at once, the reduction might only be able to simulate responses with probability $\mathcal{O}({q_H}^{\kappa})$. A failure of the simulator in any "open" signing session requires re-setting *all* open sessions. Therefore, an adversary that is allowed to open an unlimited number of signing queries in parallel leads to an exponential tightness loss in the security reduction when proving security purely in the random oracle model.

Two avoid this concurrency failure, many schemes in the literature instead reduce security to the OMDL assumption, so that the reduction can adaptively request information about the discrete logarithm and answer the oracle queries. However, as observed by Drijvers et al. [18], many previous security proofs [4, 33, 45, 34] did not correctly count the number of OMDL queries made by the reduction, which in fact might exceed the number of OMDL challenges. They did not observe that the reduction might make up to twice as many queries *with*

respect to the same challenges should the messages that the adversary queries in its second iteration be different. This invalidated the reductions, and Drijvers et al. showed that a variety of schemes in the literature cannot be proven secure under OMDL.

Confirming this danger for concrete protocols, Benhamouda et al. [12] designed a concurrent ROS attack against many schemes with broken security proofs. This ROS attack relies on the fact that the adversary can choose their nonce R^* as a linear combination of X_1 and the simulated nonces such that any dependence on X_1 will ultimately cancel out. The idea in recent protocols to thwart this attack is to essentially enforce that the honest signers only respond in the second round with a nonce $R_i S_i^{a_i}$, where a_i is the output of a hash function whose inputs include all nonces for all signing parties as well as the message being signed. As such, the forger cannot determine a_i at the point of choosing their own nonce. Preventing the forger from cancelling a_i requires two nonces; otherwise, the forger could simply incorporate a_i^{-1} into the linear verifier's equation. We implemented a basic version of the attack in python against an early version of MuSig on the BN254 curve. Our script generates a forgery in an average of 4.58 seconds over 100 trials on an Intel Core i5 processor with 2.3 GHz.[7]

Two nonces mean the number of OMDL challenges given to the reduction is twice as large. Thus, the reduction succeeds when making twice as many OMDL queries over the two iterations of the adversary.

D Proof of the Binonce Schnorr Computational Assumption

We now prove Theorem 4, which states that our bischnorr assumption is implied by the one-more discrete logarithm assumption in the programmable random oracle model.

Proof. (omdl $\Rightarrow$ bischnorr) Let $\mathcal{A}$ be a PPT adversary playing $\mathsf{Game}_{\mathcal{A}}^{\mathsf{bischnorr}}(\lambda)$ (Fig. 8) that makes up to q_H queries to $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ in total. We describe a PPT reduction $\mathcal{B}$ playing $\mathsf{Game}_{\mathcal{B}}^{\mathsf{omdl}}(\lambda)$ that uses $\mathcal{A}$ as a subroutine such that

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{bischnorr}}(\lambda) \leq \sqrt{q_H \mathsf{Adv}_{\mathcal{B}}^{\mathsf{omdl}}(\lambda) + \frac{q_H^2}{p}}$$

The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times. Suppose $\mathcal{A}$ makes no more than q_H queries to $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ in total over the two iterations. On the second iteration, $\mathcal{B}$ programs $\hat{\mathsf{H}}_{\mathsf{sig}}$ to output a different random value on a single point so that it can extract a discrete logarithm solution from $\mathcal{A}$'s outputs. Over the two iterations, $\mathcal{B}$ makes no more than n queries to its discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$ and aims to output $n+1$ discrete logarithms that constitute a valid solution to the OMDL challenge. If $\mathcal{B}$ makes fewer than n queries while responding to $\mathcal{A}$'s oracle queries, then it

[7] `https://github.com/mmaller/multi_and_threshold_signature_reductions`

makes the additional queries necessary to extract a OMDL solution. $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$. However, $\mathcal{B}$ can only extract a discrete logarithm if $\mathcal{A}$'s output (m^*, R^*, z^*) at the end of each iteration verifies and includes the same nonce R^*. By the local forking lemma [7], this occurs with probability $\frac{1}{q_H}(\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda))^2$.

$\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{binonce}}, \mathcal{O}^{\mathsf{bisign}}$, $\mathcal{O}^{\mathsf{RO}}, \hat{\mathsf{H}}_{\mathsf{non}}$, and $\hat{\mathsf{H}}_{\mathsf{sig}}$. Let $\mathsf{Q}_{\mathsf{non}}, \mathsf{Q}_{\mathsf{sig}}$ be the set of $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$ queries and responses. $\mathcal{B}$ initializes them to the empty set and maintains them across both iterations of the adversary. $\mathcal{B}$ may program $\hat{\mathsf{H}}_{\mathsf{non}}, \hat{\mathsf{H}}_{\mathsf{sig}}$. Let $Q_{\mathsf{binonce}}, Q_{\mathsf{bisign}}$ be the set of $\mathcal{O}^{\mathsf{binonce}}, \mathcal{O}^{\mathsf{bisign}}$ queries and Q_{used} the set of used nonce pairs as in $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$. $\mathcal{B}$ initializes them to the empty set. At the beginning of the second iteration, $\mathcal{B}$ makes a copy $\bar{Q}_{\mathsf{used}}$ of Q_{used} to ensure it responds to the same $\mathcal{O}^{\mathsf{bisign}}$ query with the same answer across the two iterations. It then resets $Q_{\mathsf{binonce}}, Q_{\mathsf{used}}, Q_{\mathsf{bisign}}$ to the empty set.

OMDL Input. $\mathcal{B}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$ and a OMDL challenge of $n+1$ values $(X_0, \ldots, X_n)$, where $n/2$ is greater than the number q_B of $\mathcal{O}^{\mathsf{bisign}}$ queries that $\mathcal{A}$ may make in an iteration. $\mathcal{B}$ has access to a discrete logarithm oracle $\mathcal{O}^{\mathsf{dl}}$, which it may query up to n times. $\mathcal{B}$ aims to output $(x_0, \ldots, x_n)$ such that $X_i = g^{x_i}$ for all $0 \leq i \leq n$.

Simulating Hash Queries. $\mathcal{B}$ responds to $\mathcal{A}$'s hash queries as follows.

$\underline{\hat{\mathsf{H}}_{\mathsf{non}}}$: When $\mathcal{A}$ queries $\hat{\mathsf{H}}_{\mathsf{non}}$ on $(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell))$ (ℓ can vary), $\mathcal{B}$ checks whether $(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell), a) \in \mathsf{Q}_{\mathsf{non}}$ and, if so, returns a. Else, $\mathcal{B}$ samples $a \leftarrow\!\!\$\ \mathbb{Z}_p$, appends $(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell), a)$ to $\mathsf{Q}_{\mathsf{non}}$, and returns a.

$\underline{\hat{\mathsf{H}}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\hat{\mathsf{H}}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (X, m, R, c) to $\mathsf{Q}_{\mathsf{sig}}$, and returns c.

$\mathcal{O}^{\mathsf{binonce}}$ Queries. For $\mathcal{A}$'s i^{th} query to $\mathcal{O}^{\mathsf{binonce}}$, $\mathcal{B}$ adds (X_{2i-1}, X_{2i}, i) to a set $\tilde{Q}_{\mathsf{binonce}}$ and returns $(R_i, S_i) = (X_{2i-1}, X_{2i})$. Note that $\mathcal{B}$ cannot keep track of the set $Q_{\mathsf{binonce}} = \{(R, S, r, s)\}$ as in the real $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$ since it does not know the discrete logarithms of (X_{2i-1}, X_{2i}); however, the OMDL challenge components (X_{2i-1}, X_{2i}) are randomly distributed, so $\mathcal{B}$'s simulation is perfect.

$\mathcal{O}^{\mathsf{bisign}}$ Queries. For $\mathcal{A}$'s j^{th} query to $\mathcal{O}^{\mathsf{bisign}}$ on $\mathsf{query}_j = (m_j, k_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{\ell_j}))$, $\mathcal{B}$ checks if (R_{k_j}, S_{k_j}) corresponds to $(X_{2i-1}, X_{2i}, i) \in \tilde{Q}_{\mathsf{binonce}}$ for some i and, if so, that $((R_{k_j}, S_{k_j}), (i, \cdot, \cdot, \cdot), \cdot) \notin Q_{\mathsf{used}}$ as in the real $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$. If these checks hold, then the query is valid and $\mathcal{B}$ will respond. $\mathcal{B}$ checks whether (R_{k_j}, S_{k_j}) corresponds to the query $((R_{k_t}, S_{k_t}), (i, z_t, a_t, c_t\gamma_{k_t}), \mathsf{query}_t) \in \bar{Q}_{\mathsf{used}}$ for some t. If so, $\mathcal{B}$ adds $((R_{k_j}, S_{k_j}), (i, z_t, a_t, c_t\gamma_{k_t}), \mathsf{query}_t)$ to Q_{used} and returns z_t.

If not, $\mathcal{B}$ computes $a_j \leftarrow \hat{\mathsf{H}}_{\mathsf{non}}(X_0, m_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{\ell_j}))$, $\tilde{R}_j \leftarrow \prod_{i=1}^{\ell} R_{i_j} S_{i_j}^{a_j}$, and $c_j \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(X_0, m_j, \tilde{R}_j)$. Now, $\mathcal{B}$ must return $z_j = x_{2i-1} + a_j x_{2i} + c_j\gamma_j x_0$ without knowledge of x_0, x_{2i-1}, x_{2i}. To accomplish this, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ on

$R_{k_j} S_{k_j}^{a_j} X_0^{c_j \gamma_{k_j}}$ to get z_j such that $g^{z_j} = R_{k_j} S_{k_j}^{a_j} X_0^{c_j \gamma_{k_j}}$. $\mathcal{B}$ appends (query_j, a_j) to $\mathsf{Q}_{\mathsf{non}}$, (query_j, c_j) to $\mathsf{Q}_{\mathsf{sig}}$, $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j \gamma_{k_j}), \mathsf{query}_j)$ to Q_{used}, (query_j, z_j) to $\tilde{Q}_{\mathsf{used}}$, and $(m_j, \tilde{R}_j)$ to Q_{bisign}. Finally, $\mathcal{B}$ returns z_j to $\mathcal{A}$.

Extracting the Discrete Logarithm of X_0 from the Adversary. The reduction $\mathcal{B}$ first selects some random coins ω. It then runs $\mathcal{A}(X_0; \omega)$ and responds to $\mathcal{A}$'s oracle queries as above.

Suppose $\mathcal{A}$ terminates with (m^*, R^*, z^*). If $\mathcal{A}$ succeeds, then $R^* X_0^{c^*} = g^{z^*}$, where $c^* = \hat{\mathsf{H}}_{\mathsf{sig}}(X_0, m^*, R^*)$ and $(m^*, R^*) \notin Q_{\mathsf{bisign}}$. Here, z^* does not suffice for $\mathcal{B}$ to extract the discrete logarithm of X_0 because it does not necessarily know the discrete logarithm of R^*. Thus, $\mathcal{B}$ chooses $c' \leftarrow_{\$} \mathbb{Z}_p$ and programs $\hat{\mathsf{H}}_{\mathsf{sig}}$ to output c' on input (X_0, m^*, R^*). $\mathcal{B}$ copies $\bar{Q}_{\mathsf{used}} = Q_{\mathsf{used}}$ to have a record of these queries, and then resets $\tilde{Q}_{\mathsf{binonce}}, Q_{\mathsf{used}}, Q_{\mathsf{bisign}}$ to the empty set. The sets $\tilde{Q}_{\mathsf{used}}$, $\mathsf{Q}_{\mathsf{sig}}$, $\mathsf{Q}_{\mathsf{non}}$ are also kept for the second iteration of the adversary. $\mathcal{B}$ then runs $\mathcal{A}(X_0; \omega)$ again on the same random coins.

After the second iteration, suppose $\mathcal{A}$ terminates with (m', R', z'). If $(m', R') = (m^*, R^*)$ and $\mathcal{A}$'s outputs both verify, then $\mathcal{B}$ can extract $x_0 = \frac{z^* - z'}{c^* - c'}$ such that $X_0 = g^{x_0}$. If $(m', R') \neq (m^*, R^*)$ or $\mathcal{A}$'s output does not verify, then $\mathcal{B}$ must abort. By the local forking lemma [7], this happens with probability less than $\frac{1}{q_H}(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{bischnorr}}(\lambda))^2$. If $\mathcal{A}$ succeeds having not queried $\hat{\mathsf{H}}_{\mathsf{sig}}$ on (X, m^*, R^*) or (X, m', R'), then $\mathcal{B}$ aborts. This occurs with probability less than $\frac{q_H}{p}$. Thus,

$$\frac{1}{q_H}(\mathsf{Adv}_{\mathcal{A}}^{\mathsf{bischnorr}}(\lambda))^2 \leq \Pr[\mathcal{B} \text{ extracts } x_0] + \frac{q_H}{p}$$

If $\mathcal{B}$ extracts x_0, then we use this to extract a full OMDL solution as follows.

Extracting a OMDL Solution. The reduction $\mathcal{B}$ must now extract the remaining $x_1, \ldots, x_n$ such that $X_i = g^{x_i}$. For each X_i, the method for extracting x_i will be one of four cases, depending on how $\mathcal{A}$ queried $\mathcal{O}^{\mathsf{binonce}}$ and $\mathcal{O}^{\mathsf{bisign}}$ over the two iterations.

Case 1 ((X_{2i-1}, X_{2i}) has not appeared in an $\mathcal{O}^{\mathsf{bisign}}$ query over the two iterations). In this case, X_{2i-1} and X_{2i} have not yet been queried by $\mathcal{B}$. Thus, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{dl}}$ directly to obtain x_{2i-1} and x_{2i}. Two queries are made for each i in this case.

Case 2 ((X_{2i-1}, X_{2i}) has appeared in an $\mathcal{O}^{\mathsf{bisign}}$ query in a single iteration). A single query has been made by $\mathcal{B}$ to $\mathcal{O}^{\mathsf{dl}}$ containing (X_{2i-1}, X_{2i}). If it occurred in the first iteration, then $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j \gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_{\mathsf{used}}$ is such that $z_j = r_{k_j} + a_j s_{k_j} + c_j \gamma_{k_j} x_0$. To obtain a second value, $\mathcal{B}$ queries X_{2i} to $\mathcal{O}^{\mathsf{dl}}$ and thus learns x_{2i}. Then $\mathcal{B}$ sets $x_{2i-1} = z_j - a_j x_{2i} - c_j \gamma_{k_j} x_0$. The case where the query occurred in the second iteration is similar. In total, two queries are made for each i in this case.

Now consider when (X_{2i-1}, X_{2i}) appears in an $\mathcal{O}^{\mathsf{bisign}}$ query in both iterations. Let query_j be that query from the first iteration and $\mathsf{query}_{j'}$ from the second

$$\mathsf{query}_j = (m_j, k_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{1,\ell_j}))$$
$$\mathsf{query}_{j'} = (m_{j'}, k_{j'}, (\gamma_{1_{j'}}, R_{1_{j'}}, S_{1_{j'}}), \ldots, (\gamma_{\ell_{j'}}, R_{\ell_{j'}}, S_{\ell_{j'}}))$$

such that $(X_{2i-1}, X_{2i}) = (R_{k_j}, S_{k_j}) = (R_{k_{j'}}, S_{k_{j'}})$.

Case 3 (The query containing (X_{2i-1}, X_{2i}) is the same in both iterations). In this case, $\mathsf{query}_j = \mathsf{query}_{j'}$. A single query has been made by $\mathcal{B}$ to $\mathcal{O}^{\mathsf{dl}}$ containing (X_{2i-1}, X_{2i}), and $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_{\mathsf{used}}$ is such that $z_j = r_{k_j} + a_j s_{k_j} + c_j\gamma_{k_j} x_0$. To obtain a second value, $\mathcal{B}$ queries X_{2i} to $\mathcal{O}^{\mathsf{dl}}$ and thus learns x_{2i}. Then $\mathcal{B}$ sets $x_{2i-1} = z_j - a_j x_{2i} - c_j\gamma_{k_j} x_0$. In total, two queries are made for each i in this case.

Case 4 (There exist two distinct queries containing (X_{2i-1}, X_{2i}) over the two iterations). In this case, $\mathsf{query}_j \neq \mathsf{query}_{j'}$ and $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_{\mathsf{used}}$ and $((R_{k_{j'}}, S_{k_{j'}}), (i, z_{j'}, a_{j'}, c_{j'}\gamma_{k_{j'}}), \mathsf{query}_{j'}) \in Q_{\mathsf{used}}$. Then $\mathcal{B}$ sets

$$x_{2i} = \frac{z_{j'} - z_j + (c_j\gamma_{k_j} - c_{j'}\gamma_{k_{j'}})x_0}{a_{j'} - a_j} \qquad x_{2i-1} = z_j - a_j x_{2i} - c_j\gamma_{k_j} x_0 \tag{7}$$

Suppose $a_j = a_{j'}$. That is, $\hat{\mathsf{H}}_{\mathsf{non}}(X_0, m_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{\ell_j})) = \hat{\mathsf{H}}_{\mathsf{non}}(X_0, m_{j'}, (\gamma_{1_{j'}}, R_{1_{j'}}, S_{1_{j'}}), \ldots, (\gamma_{\ell_{j'}}, R_{\ell_{j'}}, S_{\ell_{j'}}))$. With probability greater than $1 - q_H/p$, the inputs are equal. This implies $m_j = m_{j'}$ and $(\gamma_{i_j}, R_{i_j}, S_{i_j}) = (\gamma_{i_{j'}}, R_{i_{j'}}, S_{i_{j'}})$ for all $1 \leq i \leq \ell$. The only values in query_j and $\mathsf{query}_{j'}$ not hashed in $\hat{\mathsf{H}}_{\mathsf{non}}$ are k_j and $k_{j'}$. For $\mathsf{query}_j \neq \mathsf{query}_{j'}$, we must have $k_j \neq k_{j'}$. But it is possible that $(X_{2i-1}, X_{2i}) = (R_{k_j}, S_{k_j}) = (R_{k_{j'}}, S_{k_{j'}})$ in *both* queries. There must be an explicit check for repeated nonce pairs within a query to eliminate this Bad Case[8]. If Bad Case does not occur, then the probability that $a_{j'} = a_j$ for $\mathsf{query}_{j'} \neq \mathsf{query}_j$ is less than q_H/p. Two queries are made for each i in this case.

Thus, $\mathcal{B}$ has extracted x_i for all X_i using exactly n queries and returns $(x_0, x_1, \ldots, x_n)$ to win the $\mathsf{Game}^{\mathsf{omdl}}_{\mathcal{B}}(\lambda)$ game.

E Proof of Security for SimpleTSig

We now prove Theorem 6, which states that SimpleTSig (Fig. 9) with distributed key generation protocol PedPoP (Fig. 12) is unforgeable under the discrete logarithm assumption and the Schnorr knowledge of exponent assumption (Assumption 4) in the programmable random oracle model.

[8] We strongly suspect that using a generalized forking lemma instead of the local forking lemma obviates the need for this check, due to [11].

Proof. (of Theorem 6) Let $\mathcal{A}$ be a PPT adversary attempting to break the unforgeability of $\mathsf{SimpleTSig}$. We construct a PPT adversary $\mathcal{B}_1$ playing game $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$ (Fig. 5) and thence, from the $\mathsf{schnorr\text{-}koe}$ assumption, obtain an extractor Ext for it. We construct a PPT adversary $\mathcal{B}_2$ playing game $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda)$ (Fig. 6) such that whenever $\mathcal{A}$ outputs a valid forgery, either $\mathcal{B}_1$ breaks the $\mathsf{schnorr\text{-}koe}$ assumption or $\mathcal{B}_2$ breaks the $\mathsf{schnorr}$ assumption. (Recall that $\mathsf{schnorr}$ is implied by the DL assumption (Theorem 2). Formally, we have

$$\mathsf{Adv}^{\mathsf{EUF\text{-}CMA}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda) + \mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda) + \mathsf{negl}(\lambda)$$

where λ is the security parameter.

The Reduction $\mathcal{B}_1$: We first define the reduction $\mathcal{B}_1$ against $\mathsf{schnorr\text{-}koe}$. Let $\mathsf{cor} = \{\mathsf{id}_j\}$ be the set of corrupt parties, and let $\mathsf{hon} = \{\mathsf{id}_k\}$ be the set of honest parties. Assume without loss of generality that $|\mathsf{cor}| = t-1$ and $|\mathsf{hon}| = n-(t-1)$. We will show that when PedPoP outputs public key share $\tilde{X}_k = g^{\bar{x}_k}$ for each honest party $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}_1$ returns (α_k, β_k) such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$.

$\mathcal{B}_1$ is responsible for simulating honest parties in PedPoP (Fig. 12) and queries to $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_1$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and random coins ω. It can query the random oracle $\mathcal{O}^{\mathsf{RO}}$ from $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$. It can also query $\mathcal{O}^{\mathsf{schnorr\text{-}koe}}$ to receive signatures under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and $\mathcal{O}^{\mathsf{chal}}$ on inputs $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ to challenge the extractor Ext to output a discrete logarithm $a_{j,0}$ for $X_{j,0}$.

Initialization. $\mathcal{B}_1$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\tilde{\mathsf{H}}_{\mathsf{reg}}$ (because it is part of $\mathcal{B}_1$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses. $\mathcal{B}_1$ computes α_k for each honest party $\mathsf{id}_k \in \mathsf{hon}$ as follows. First, $\mathcal{B}_1$ computes the t Lagrange polynomials $\{L'_k(Z), \{L'_j(Z)\}_{\mathsf{id}_j \in \mathsf{cor}}\}$ relating to the set $\mathsf{id}_k \cup \mathsf{cor}$. Then, $\mathcal{B}_1$ sets $\alpha_k \leftarrow L'_k(0)^{-1}$. It will later become clear why α_k is computed this way.

Simulating Hash Queries. $\mathcal{B}_1$ handles $\mathcal{A}$'s hash queries throughout the DKG protocol as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_1$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_1$ queries $\bar{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on R, $\mathcal{B}_1$ queries $\mathsf{cm} \leftarrow \tilde{\mathsf{H}}_{\mathsf{cm}}(R)$ and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_1$ queries $\hat{c} \leftarrow \tilde{\mathsf{H}}_{\mathsf{sig}}(X, m, R)$ and returns $\hat{c}$.

Simulating the DKG. $\mathcal{B}_1$ runs $\mathcal{A}$ on input random coins ω and simulates PedPoP as follows. $\mathcal{B}_1$ first queries $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ and receives $(\dot{X}, \bar{R}_\tau, \bar{z}_\tau)$. $\mathcal{B}_1$ embeds $\dot{X}$ as the public key of the honest party that the adversary queries first. Let this first honest party be id_τ. $\mathcal{B}_1$ simulates the public view of id_τ but follows the PedPoP protocol for all other honest parties $\{\mathsf{id}_k\}_{k \neq \tau}$ as prescribed. Note that $\mathcal{A}$ can choose the order in which it interacts with honest parties, so $\mathcal{B}_1$ must be able to simulate any of them. The simulation of PedPoP is the same as in the

proof of $\mathsf{FROST2}+\mathsf{PedPoP}$ (Theorem 7). When the DKG terminates, $\mathcal{B}_1$ returns $\{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in \mathsf{hon}}$.

We now argue that: (1) $\mathcal{A}$ cannot distinguish between a real run of the DKG protocol and its interaction with $\mathcal{B}_1$; and (2) $\mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ outputs $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ whenever $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_{j,0}, \bar{R}_j, \bar{z}_j)$.

(1) $\mathcal{B}_1$'s simulation of PedPoP is perfect, as in the proof of $\mathsf{FROST2}+\mathsf{PedPoP}$ (Theorem 7).

(2) Observe that $\mathsf{H}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j) = \tilde{\mathsf{H}}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j)$ unless $(X_{j,0}, \bar{R}_j) = (\dot{X}, \bar{R}_\tau)$. The latter happens only if $X_{j,0} = X_{\tau,0}$, but in this case PedPoP will not terminate. We thus have that $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ is a verifying signature under $\tilde{\mathsf{H}}_{\mathsf{reg}}$ and either Ext succeeds, or $\mathcal{B}_1$ breaks the $\mathsf{schnorr\text{-}koe}$ assumption. Therefore, the probability of the event occurring where Ext fails to outputs $a_{j,0}$ is bounded by $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1,\mathsf{Ext}}(\lambda)$.

The Reduction $\mathcal{B}_2$: We next define the reduction $\mathcal{B}_2$ against $\mathsf{schnorr}$. We will show that when PedPoP outputs the joint public key $\tilde{X}$, $\mathcal{B}_2$ returns y such that $\tilde{X} = \dot{X} g^y$. Together with the (α_k, β_k) returned by $\mathcal{B}_1$ such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$, this representation allows $\mathcal{B}_2$ to simulate $\mathsf{SimpleTSig}$ signing under each public key share $\tilde{X}_k$. $\mathcal{B}_2$ is responsible for simulating honest parties during signing and queries to $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}_2$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}$. It can query $\mathcal{O}^{\mathsf{schnorr}}$ and $\mathcal{O}^{\hat{\mathsf{RO}}}$ from $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{B}_2}(\lambda)$.

Initialization. $\mathcal{B}_2$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{cm}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\hat{\mathsf{H}}_{\mathsf{sig}}$ (because it is part of $\mathcal{B}_2$'s challenge). Let $Q_{\mathsf{Sign}}, Q_{\mathsf{Sign}'}, Q_{\mathsf{Sign}''}$ be the set of $\mathcal{O}^{\mathsf{Sign}}, \mathcal{O}^{\mathsf{Sign}'}, \mathcal{O}^{\mathsf{Sign}''}$ queries and responses in Signing Round 1, 2, 3, respectively.

DKG Extraction. $\mathcal{B}_2$ first simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}_2$ samples $\bar{c}_\tau, \bar{z}_\tau \leftarrow\!\!\$\ \mathbb{Z}_p$, computes $\bar{R}_\tau \leftarrow g^{\bar{z}_\tau} \dot{X}^{-\bar{c}_\tau}$, and appends $(\dot{X}, \dot{X}, \bar{R}_\tau, \bar{c}_\tau)$ to $\mathsf{Q}_{\mathsf{reg}}$. Then, $\mathcal{B}_2$ runs

$$\{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in \mathsf{hon}} \leftarrow\!\!\$\ \mathcal{B}_1(\mathcal{G}; \omega)$$

on random coins ω. $\mathcal{B}_2$ handles $\mathcal{B}_1$'s queries as follows. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. When $\mathcal{B}_1$ queries $\tilde{\mathsf{H}}_{\mathsf{cm}}, \tilde{\mathsf{H}}_{\mathsf{sig}}$, $\mathcal{B}_2$ handles them the same way it handles $\mathcal{A}$'s $\mathsf{H}_{\mathsf{cm}}, \mathsf{H}_{\mathsf{sig}}$ queries, described below. The first time $\mathcal{B}_1$ queries its $\mathcal{O}^{\mathsf{sch\text{-}pop}}$ oracle, $\mathcal{B}_2$ returns $(\dot{X}, \bar{R}_\tau, \bar{z}_\tau)$. When $\mathcal{B}_1$ queries $\mathcal{O}^{\mathsf{chal}}(X_{j,0}, \bar{R}_j, \bar{z}_j)$, $\mathcal{B}_2$ runs $a_{j,0} \leftarrow \mathsf{Ext}(\mathcal{G}, \omega, Q_{\mathsf{sch\text{-}pop}}, \mathsf{Q}_{\mathsf{reg}})$ to obtain $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ and aborts otherwise. Then $y = \sum_{i=1, i\neq\tau}^{n} a_{i,0}$ such that $\tilde{X} = \dot{X} g^y$.

Simulating Hash Queries. $\mathcal{B}_2$ handles $\mathcal{A}$'s hash queries throughout the signing protocol as follows.

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R})$, $\mathcal{B}_2$ checks whether $(X, X, \bar{R}, \bar{c}) \in \mathsf{Q}_{\mathsf{reg}}$ and, if so, returns $\bar{c}$. Else, $\mathcal{B}_2$ queries $\bar{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{reg}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$. Note that $\mathcal{B}_1$ and $\mathcal{B}_2$ share the state of $\mathsf{Q}_{\mathsf{reg}}$.

$\underline{\mathsf{H}_{\mathsf{cm}}}$: When $\mathcal{A}$ queries H_{cm} on R, $\mathcal{B}_2$ checks whether $(R, \mathsf{cm}) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm. Else, $\mathcal{B}_2$ samples $\mathsf{cm} \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (R, cm) to Q_{cm}, and returns cm.

$\underline{\mathsf{H}_{\mathsf{sig}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \underline{\mathsf{Q}_{\mathsf{sig}}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}_2$ samples a random message $\hat{m}$, queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$, appends $(X, m, R, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\hat{c}$.

Simulating SimpleTSig Signing. After $\mathcal{B}_1$ completes the simulation of PedPoP, $\mathcal{B}_2$ then simulates honest parties in the SimpleTSig signing protocol.

Signing Round 1 (Sign). In the first round of signing, all parties who intend to participate send commitments $\{\mathsf{cm}_i\}_{i \in \mathcal{S}}$. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}}$ on $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}_2$ samples a random message $\dot{m}$ and queries $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ to get a signature $(\dot{R}_k, \dot{z}_k)$. Then $\mathcal{B}_2$ sets $R_k = \dot{R}_k^{\alpha_k}$, checks whether $(R_k, \mathsf{cm}_k) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm_k. Else, $\mathcal{B}_2$ samples $\mathsf{cm}_k \leftarrow\!\!\$\ \mathbb{Z}_p$, appends (R_k, cm_k) to Q_{cm}, and returns cm_k.

Signing Round 2 (Sign'). In the second round of signing, all parties take as input the message m to be signed and reveal nonces $\{R_i\}_{i \in \mathcal{S}}$ such that $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(R_i)$. $\mathcal{B}_2$ looks up $\{\mathsf{cm}_i\}_{i \in \mathcal{S} \setminus \{k\}}$ for records $(R_i, \mathsf{cm}_i) \in \mathsf{Q}_{\mathsf{cm}}$. If there exists some i' for which a record $(R_{i'}, \mathsf{cm}_{i'})$ does not exist, then $\mathcal{B}_2$ aborts. If all records exist, then $\mathcal{B}_2$ computes the Lagrange coefficients $\{\lambda_i\}_{i \in \mathcal{S}}$, where $\lambda_i = L_i(0)$ and $\{L_i(Z)\}_{i \in \mathcal{S}}$ are the Lagrange polynomials relating to the set $\{\mathsf{id}_i\}_{i \in \mathcal{S}}$. $\mathcal{B}_2$ then computes $\tilde{R} = \prod_{i \in \mathcal{S}} R_i^{\lambda_i}$ and queries $\dot{c} \leftarrow \dot{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \dot{m}, \dot{R}_k)$ (not $\tilde{R}$) and appends $(\tilde{X}, m, \tilde{R}, \dot{m}, \dot{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. (However, if $\mathcal{A}$ has already queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$, then $\mathcal{B}_2$ aborts.) For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}_2$ returns R_k.

Signing Round 3 (Sign''). The third round of signing only proceeds if the second round terminated, i.e., all parties revealed their nonces in the second round. $\mathcal{B}_2$ computes

$$\tilde{z}_k = \alpha_k \dot{z}_k + \dot{c}\beta_k \tag{8}$$

For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}''}$, $\mathcal{B}_2$ returns $\tilde{z}_k$.

Output. When $\mathcal{A}$ returns $(\tilde{X}, m^*, \sigma^*)$ such that $\sigma^* = (\tilde{R}^*, z^*)$ and $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, $\mathcal{B}_2$ computes its output as follows. $\mathcal{B}_2$ looks up $\hat{m}^*$ such that $(\tilde{X}, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ and outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^* y)$.

To complete the proof, we must argue that: (1) $\mathcal{B}_2$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}_2$ succeeds.

(1) $\mathcal{B}_2$ aborts if Ext fails to return $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$ for some $\mathsf{id}_j \in \mathsf{cor}$. This happens with maximum probability $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{B}_1, \mathsf{Ext}}(\lambda)$.

$\mathcal{B}_2$ aborts in Signing Round 2 if $\mathcal{A}$ reveals $R_{i'}$ such that $\mathsf{cm}_{i'} = \mathsf{H}_{\mathsf{cm}}(R_{i'})$ but $\mathcal{A}$ never queried H_{cm} on $R_{i'}$. This requires $\mathcal{A}$ to have guessed $\mathsf{cm}_{i'}$ ahead of time, which occurs with negligible probability $1/p$.

$\mathcal{B}_2$ also aborts in Signing Round 2 if $\mathcal{A}$ had previously queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m, \tilde{R})$. In that case, $\mathcal{B}_2$ had returned $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, \tilde{R})$ for some random message $\hat{m}$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_k before $\mathcal{B}_2$ revealed it, which occurs with negligible probability $1/p$.

(2) As long as $\mathcal{B}_2$ does not abort, $\mathcal{B}_2$ is able to simulate the appropriate responses to $\mathcal{A}$'s oracle queries so that $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}_2$.

Indeed, $\mathcal{B}_1$'s simulation of PedPoP is perfect, as in the proof of $\mathsf{FROST2} + \mathsf{PedPoP}$ (Theorem 7). (Performing validation of each player's share (Step 4 in Fig. 12) holds, and interpolation in the exponent correctly evaluates to the challenge $\dot{X}$.)

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}_2$ queries $\hat{c} \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}, R)$ on a random message $\hat{m}$. The random message prevents trivial collisions; for example, if $\mathcal{A}$ were to query $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R) and (X', m, R), where $X' \neq X$, $\mathcal{A}$ would receive the same value $c \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, m, R)$ for both and would know it was operating inside a reduction. Random messages ensure that the outputs are random, so $\mathcal{A}$'s view is correct.

After the signing rounds have been completed, $\mathcal{A}$ may verify the signature share $\tilde{z}_k$ on m as follows. $\mathcal{A}$ checks if

$$R_k \tilde{X}_k^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_{i \in \mathcal{S}} R_i^{\lambda_i})} = g^{\tilde{z}_k} \tag{9}$$

When $\mathcal{B}_2$ queried $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ in Signing Round 2, the signature share $\dot{z}_k$ was computed such that

$$\dot{R}_k \dot{X}^{\tilde{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \dot{m}, \dot{R}_k)} = g^{\dot{z}_k}$$

Recall that $R_k = \dot{R}_k^{\alpha_k} = g^{\alpha_k \dot{r}_k}$ and $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$. $\mathcal{B}_2$ computed the signature share $\tilde{z}_k$ (Equation 8) as $\tilde{z}_k = \alpha_k \dot{z}_k + \dot{c}\beta_k$, where $\dot{c} = \tilde{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \dot{m}, \dot{R}_k)$. Thus,

$$g^{\tilde{z}_k} = g^{\alpha_k \dot{z}_k + \dot{c}\beta_k} = g^{\alpha_k(\dot{r}_k + \dot{c}\dot{x}) + \dot{c}\beta_k} = g^{\alpha_k \dot{r}_k + (\alpha_k \dot{x} + \beta_k)\dot{c}} = R_k \tilde{X}_k^{\dot{c}} \tag{10}$$

$\mathcal{B}_2$ has programmed the hash values in Equations 9 and 10 to be equal and therefore simulates $\tilde{z}_k$ correctly.

(3) $\mathcal{A}$'s forgery satisfies $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, which implies:

$$\begin{aligned}
\tilde{R}^* (\tilde{X})^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* (\dot{X} g^y)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* \dot{X}^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*)} &= g^{z^* - \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m^*, \tilde{R}^*) y}
\end{aligned}$$

At some point, $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}, m^*, \tilde{R}^*)$ and received $\hat{c}^* \leftarrow \tilde{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$. Thus, $\mathcal{A}$'s forgery satisfies

$$\tilde{R}^* \dot{X}^{\tilde{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*)} = g^{z^* - \tilde{\mathsf{H}}_{\mathsf{sig}}(\dot{X}, \hat{m}^*, \tilde{R}^*) y}$$

and $\mathcal{B}$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^* y)$ under $\dot{X}$ is correct.

F Changelog

12-10-2021: First version of the paper.

3-08-2022: Incorporated the following improvements:

1. New contributions: We provide and prove secure SimpleTSig, a three-round threshold signature scheme that depends only on the discrete logarithm assumption and proofs of possession.
2. Corrected tightness bounds: The reductions for Theorems 2 and 4 now depend explicitly on the local forking lemma [7].
3. More precise proof of possession analysis: The security proofs for Theorems 3, 5, and 7 now explicitly state which reduction runs the schnorr-koe extractor, and the schnorr-koe assumption has been updated accordingly.

G Figures

Here we provide figures for the multisignature EUF-CMA security game as well as the PedPoP distributed key generation protocol.

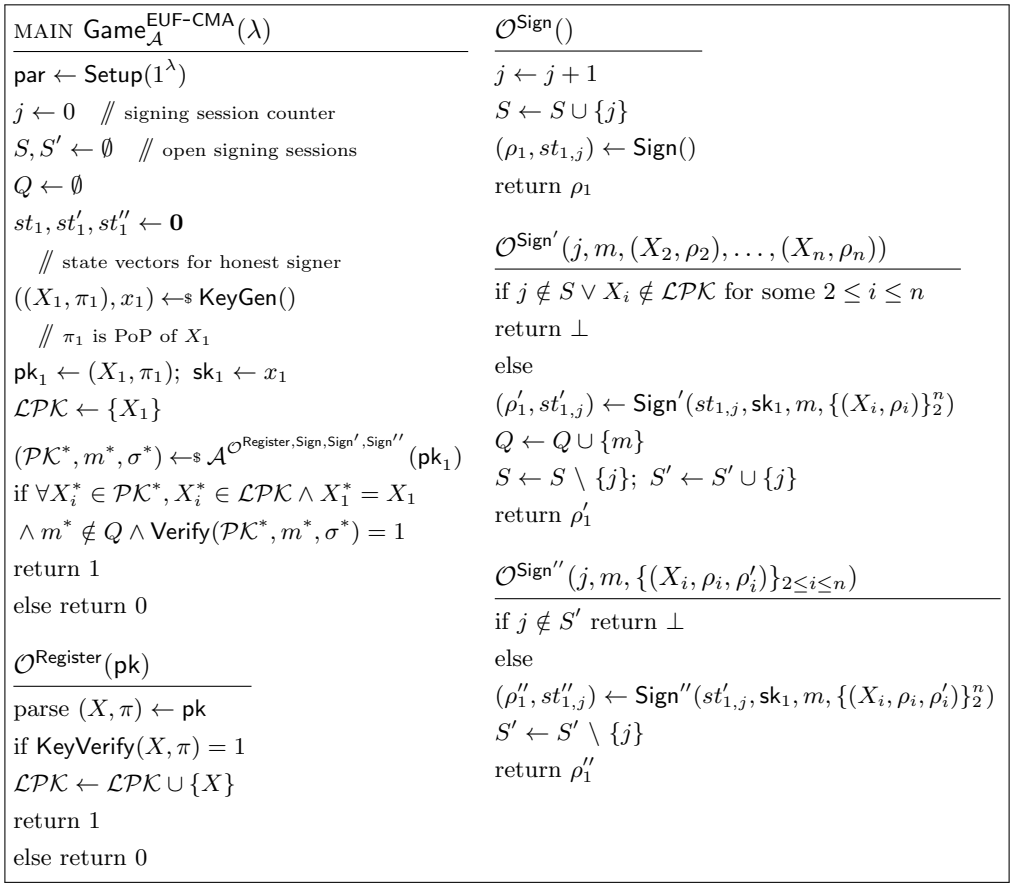

Fig. 11. The EUF-CMA security game for a multisignature scheme with proofs of possession. The public parameters par are implicitly given as input to all algorithms, and ρ represents messages defined within the construction. Note that the winning condition cannot be $(\mathcal{PK}^*, m^*) \notin Q$ because $\mathcal{A}$ can find $\mathcal{PK}^* \neq \mathcal{PK}$ such that $\tilde{X}^* = \tilde{X}$.

$\mathsf{PedPoP.KeyGen}(t, n)$

1. Each party P_i chooses a random polynomial $f_i(Z)$ over $\mathbb{Z}_p$ of degree $t-1$

$$f_i(Z) = a_{i,0} + a_{i,1}Z + \cdots + a_{i,t-1}Z^{t-1}$$

and computes $A_{i,k} = g^{a_{i,k}}$ for $k = 0, \ldots, t-1$. Denote $x_i = a_{i,0}$ and $X_{i,0} = A_{i,0}$. Each P_i computes a proof of possession of $X_{i,0}$ as a Schnorr signature on $X_{i,0}$ as follows. They sample $\bar{r}_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $\bar{R}_i \leftarrow g^{\bar{r}_i}$. They compute $\bar{c}_i \leftarrow \mathsf{H}_{\mathsf{reg}}(X_{i,0}, X_{i,0}, \bar{R}_i)$ and set $\bar{z}_i \leftarrow \bar{r}_i + \bar{c}_i x_i$. They then derive a commitment $\boldsymbol{C}_i = (A_{i,0}, \ldots, A_{i,t-1})$ and broadcast $((\bar{R}_i, \bar{z}_i), \boldsymbol{C}_i)$.
2. After receiving commitments from all other parties, each participant verifies the Schnorr signatures by computing $\bar{c}_j \leftarrow \mathsf{H}_{\mathsf{reg}}(A_{j,0}, A_{j,0}, \bar{R}_j)$ and checking that

$$\bar{R}_j {A_{j,0}}^{\bar{c}_j} = g^{\bar{z}_j} \text{ for } j = 1, \ldots, n$$

If any checks fail, they disqualify the corresponding participant; otherwise, they continue to the next step.
3. Each P_i computes secret shares $\bar{x}_{i,j} = f_i(\mathsf{id}_j)$ for $j = 1, \ldots, n$, where id_j is the participant identifier, and sends $\bar{x}_{i,j}$ secretly to party P_j.
4. Each party P_j verifies the shares they received from the other parties by checking that

$$g^{\bar{x}_{i,j}} = \prod_{k=0}^{t-1} A_{i,k}^{\mathsf{id}_j^k}$$

If the check fails for an index i, then P_j broadcasts a complaint against P_i.
5. For each of the complaining parties P_j against P_i, P_i broadcasts the share $\bar{x}_{i,j}$. If any of the revealed shares fails to satisfy the equation, or should P_i not broadcast anything for a complaining player, then P_i is disqualified. The share of a disqualified party P_i is set to 0.
6. The secret share for each P_j is $\bar{x}_j = \sum_{i=1}^{n} \bar{x}_{i,j}$.
7. If $X_{i,0} = X_{j,0}$ for any $i \neq j$, then abort. Else, the output is the joint public key $\tilde{X} = \prod_{i=1}^{n} X_{i,0}$.

Fig. 12. PedPoP: Pedersen's distributed key generation protocol with proofs of possession.