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. In this paper, we present new techniques for proving the security of multi- and threshold signature schemes under discrete logarithm assumptions in the random oracle model. The purpose is to provide a simple framework for analyzing the relatively complex interactions of these schemes in a concurrent model, thereby reducing the risk of attacks. We make use of proofs of possession and prove that a Schnorr signature suffices as a proof of possession in the algebraic group model without any tightness loss. We introduce and prove the security of a simple, three-round multisignature SimpleMuSig.

Using our new techniques, we prove the concurrent security of a variant of the MuSig2 multisignature scheme that includes proofs of possession as well as the FROST threshold signature scheme. These are currently the most efficient schemes in the literature for generating Schnorr signatures in a multiparty setting. Our variant of MuSig2, which we call SpeedyMuSig, has faster key aggregation due to the proofs of possession.

1 Introduction

Current methods for proving the security of multi- and threshold signature schemes can be overwhelmingly complex or difficult to audit. Even a seemingly intuitive analysis can contain subtle errors that render the proof completely invalid. Indeed, Drijver's et al. [13] demonstrated that a wide range of multisignature schemes cannot be proven secure under the one-more discrete logarithm assumption. Benhamouda et al. [7] 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.

The goal of this work is to provide an alternative method for proving the security of multisignature and threshold signature schemes under discrete logarithm assumptions in the random oracle model. 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. Threshold signature schemes define a t-out-of-n access structure of a private key that is shared by a set of n parties, at least t of which are required to cooperate in order to issue a valid signature. Each multi- and threshold signature scheme in this work produces a Schnorr signature [34], which is a Σ-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 [15]. Our proving methods involve only *straight-line* adversaries in the sense that the reduction only runs the adversary once in its attempt to break the appropriate security assumption. 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.

To achieve our results, we introduce three novel and relatively strong assumptions, which we justify in the algebraic group model and the random oracle model. Our first assumption is the *Schnorr knowledge of exponent assumption* (schnorr-koe), which 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. The schnorr-koe assumption allows us to use Schnorr signatures as "proofs of possession," wherein each signer must prove knowledge of its secret key upon registering the corresponding public key into the system. The proofs of possession cost no more to store and verify than a standard Schnorr signature. (512 bit proofs verify in 0.5 milliseconds [38].) While our schnorr-koe assumption is non-falsifiable, it allows us to obtain concretely efficient proofs of possession and prove tight security at the same time. It is similar in style to knowledge of exponent assumptions that are widely used in the SNARK literature [11].

Our second assumption is the *Schnorr computational assumption* (schnorr), which says that it is difficult for an adversary to forge a Schnorr signature. This assumption can be reduced to the discrete logarithm assumption in the random oracle model assuming that the adversary is run twice. We argue that these assumptions have 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. Armed with our new assumptions, we are able to prove the concurrent security of a simple, three-round multisignature

scheme SimpleMuSig without any explicit dependence on rewinding the adversary. There is still an implicit dependence, as the schnorr assumption is proven by rewinding the adversary once. Our simplification is that any advantage gained by a concurrent adversary is already modelled into the schnorr reduction to the discrete logarithm assumption and does not need to be reconsidered in any straight-line protocol that reduces to the schnorr assumption.

Having proven our simple scheme secure, we then look to the most efficient multi- and threshold signature schemes in the literature to see if we can apply our framework. In particular, we analyze a variant of the MuSig2 [29] multisignature scheme with proofs of possession as well as the FROST [25] threshold signature scheme, both of which only require two signing rounds. Here we find that our schnorr assumption is not strong enough. This is unsurprising because MuSig2 and FROST are secure under the one-more discrete logarithm assumption, which is a stronger assumption than the discrete logarithm assumption. We therefore introduce a third assumption, the binonce Schnorr computational assumption (bischnorr), and reduce its security to the one-more discrete logarithm assumption (omdl). Our proof of security encompasses nuanced attacks that prior security models did not consider, such as ROS-style attacks that emerge in the concurrent setting. Our security reduction is in the random oracle model and uses two iterations of the adversary. Equipped with our new assumption, we are able to prove the concurrent security of both a MuSig2 variant and FROST in a straightforward manner. To increase confidence in our arguments, we implement our bischnorr and FROST reductions 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, which we call 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 is formed 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 new techniques for proving the concurrent security of multi- and threshold signatures, where any dependence on rewinding is abstracted into the assumptions, making the security proofs more modular. As a result, our security reductions are less involved and easier to both write and audit.
- We introduce and prove the tight security of the Schnorr knowledge of exponent assumption (schnorr-koe) in the algebraic group model. This is of independent interest and useful for any application involving proofs of possession.
- We introduce and prove the security of the Schnorr computational assumption (schnorr) and the binonce Schnorr computational assumption (bischnorr). Our security reductions only run two iterations of the adversary.
- We introduce and prove secure a simple multisignature scheme with proofs of possession, called SimpleMuSig.
- We introduce and prove secure a variant of the 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 prove the security of the FROST threshold signature scheme using our new techniques. We introduce a modification to FROST that allows for improved efficiency during signing, reducing the number of exponentiations from at least t to one.
- We provide an open source python implementation of our reduction for bischnorr and our reduction for FROST.[4]

2 Related Work

Bellare and Neven [5] introduced a multisignature scheme with three rounds of signing and signature verification matching that of a standard Schnorr signature (BN06). Maxwell et al. [27] expanded upon this scheme to allow for key aggregation (MuSig). Drijvers et al. [13] 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. [30] 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. [29] 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 [2]. 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 adversaries from the part of the reduction that analyzes the interactive nature of the multisignature scheme.

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

Scheme	KeyGen			KeyVerify	Sign				Combine	Verify
	exp	$\mathbb{G}$	$\mathbb{F}$	exp	round	exp	$\mathbb{G}$	$\mathbb{F}$	exp	exp
BN06 [5]	1	1	0	-	3	1	1	1	0	$n+1$
mBCJ [13]	2	2	1	2	2	4	2	3	0	8
MuSig [27]	1	1	0	-	3	$n+1$	1	2	n	$n+1$
MuSig2 [29]	1	1	0	-	2	$n+3$	2	1	$n+1$	$n+2$
DWMS [2]	1	1	0	-	2	$3n+2$	2	1	$n+1$	$n+1$
This Work										
SimpleMuSig	2	2	1	2	3	1	1	2	0	2
SpeedyMuSig	2	2	1	2	2	3	2	1	1	2

Fig. 1. Table of Efficiency of Multisignature Schnorr Schemes. Here, exp stands for the number of group exponentiations. The number of rounds 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. In some protocols, there is the output of a hash function, which we have counted as a field element. The only protocol that outputs a signature that is not a standard Schnorr signature is mBCJ.

Multisignature proofs can also be given in the algebraic group model (AGM) [29, 2]; however, AGM proofs in more complicated settings have their own delicacies and have been known to result in serious errors [17]. We limit the intricate linear algebra arguments inherent in algebraic group model proofs to our analysis of the schnorr-koe assumption, for which the AGM proof can be kept simple.

Boldyreva [8] and Ristenpart and Yilek [33] 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. [9] 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 [12] and was therefore omitted; however, in a follow-up work [13], the proof of DG-CoSi was determined to be flawed, leaving open the question of how to prove security of MSDL-pop. We relabel MSDL-pop as SimpleMuSig and prove security, thus filling this gap in the literature.

Regarding Schnorr-based threshold signature schemes, Gennaro et al. [21] suggested a two-round protocol, which is not secure in the concurrent setting due to ROS attacks. Komlo and Goldberg proposed FROST [25], which is concretely efficient and secure in the concurrent setting. However, their security proof requires an interactive construction and cannot extend to the non-interactive variant without a heuristic assumption. Our techniques allow for proving the security of FROST directly.

Fuchsbauer et al. [16] 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 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.

Bellare and Dai [4] recently proposed a new proving framework for multisignatures via a chain of sub-reductions and proved the security of existing three-round schemes [27, 5] as well as their own two-round scheme. However, their scheme is incompatible with standard Schnorr signature verification. This work proves the security of existing two-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 $y \leftarrow A(x; r)$ denote running algorithm A on input x and randomness r and assigning its output to y. Let $y \leftarrow\!\$\, A(x)$ denote $y \leftarrow A(x; r)$ for a uniformly random r. 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 [6]. 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.

Definition 1 (Group Generator). *A group generator* $\mathsf{GroupGen}$ *is 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 2 (Schnorr Signatures). *Let* $\mathsf{GroupGen}$ *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* $\mathsf{GroupGen}$ *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* $\Pr[\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda); X \leftarrow\!\!\$\, \mathbb{G}; x \leftarrow\!\!\$\, \mathcal{A}(\mathcal{G}, X) : X = g^x] < \nu(\lambda)$*.*

Assumption 2 (One-More Discrete Logarithm Assumption (OMDL)) *Let* $\mathsf{GroupGen}$ *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* $\Pr[\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda); (X_0, \ldots, X_n) \leftarrow\!\!\$\, \mathbb{G}^{n+1}; (x_0, \ldots, x_n) \leftarrow\!\!\$\, \mathcal{A}^{\mathcal{O}^{\mathsf{dl}}}(\mathcal{G}, X_0, \ldots, X_n) : X_i = g^{x_i}\ \forall\ 0 \le i \le n] < \nu(\lambda)$*.*

Assumption 3 (Algebraic Group Model (AGM) [16]) *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 Schnorr Assumptions

In this section, we introduce and justify three new assumptions: the Schnorr knowledge of exponent assumption (schnorr-koe), the Schnorr computational assumption (schnorr), and the binonce Schnorr computational assumption (bischnorr). We reduce the security of these three assumptions to the discrete logarithm assumption (schnorr-koe and schnorr) and the one-more discrete logarithm assumption (bischnorr) in Theorems 1, 2, and 3, respectively. We then employ them to prove the security of a selection of multi- and threshold signature schemes.

4.1 Schnorr Knowledge of Exponent Assumption

We now introduce a new knowledge of exponent assumption that reduces to the discrete logarithm assumption in the algebraic group model. It greatly simplifies proofs for multiparty signature schemes, as we will see in Sections 5 and 6. Our Schnorr knowledge of exponent assumption states that if an adversary can forge a Schnorr signature for some public key, then it must know the corresponding secret key. This is a non-falsifiable assumption. We prove that the schnorr-koe assumption holds in the algebraic group model without any tightness loss (Appendix A). This reduction expands on a result by Fuchsbauer et al., which showed that the security of Schnorr signatures can be tightly reduced to the discrete logarithm assumption in the algebraic group model [16]. We improve on their result by allowing extraction even when the adversary chooses their own public key.

Our schnorr-koe assumption is reminiscent of the knowledge of exponent assumption first introduced by Damgard [11] and 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.

Our schnorr-koe assumption is as follows. An adversary $\mathcal{A}$ has access to a signing oracle $\mathcal{O}^{\mathsf{schnorr\text{-}pop}}$ that outputs a Schnorr signature under a randomly sampled key X on the message X. When $\mathcal{A}$ terminates, if it outputs a verifying Schnorr signature under X^* on the message X^* for X^* it has chosen itself, either $\mathcal{A}$'s signature is simply one it has already seen or $\mathcal{A}$ must know the discrete logarithm of X^*. In other words, for every PPT algorithm $\mathcal{A}$ given a group description $\mathbb{G} = \langle g \rangle$, if $\mathcal{A}$ outputs a Schnorr signature $(X, m, \sigma) = (X^*, X^*, \sigma^*)$ such that $\mathsf{Verify}(X^*, X^*, \sigma^*) = 1$, then there exists an extractor algorithm $\mathcal{E}$ that, given the transcript of $\mathcal{A}$, outputs x^* such that $X^* = g^{x^*}$.

Assumption 4 (The Schnorr Knowledge of Exponent Assumption) *Let* $\mathsf{GroupGen}$ *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$ *in which the discrete logarithm assumption holds, and let* H *be a hash function. Let* $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}(\lambda) = 1]$*, where* $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}(\lambda)$ *is defined in Figure 2. The Schnorr knowledge of exponent assumption holds with respect to* $\mathcal{G}$ *and* H *if for all PPT adversaries* $\mathcal{A}$*, there exists a PPT extractor* $\mathcal{E}$ *and a negligible function* ν *such that* $\mathsf{Adv}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}(\lambda) < \nu(\lambda)$*.*

Theorem 1 (dl ⇒ schnorr-koe). *Let* $\mathsf{GroupGen}$ *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* $\mathsf{H}_{\mathsf{pop}}$ *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* $\mathsf{H}_{\mathsf{pop}}$*.*

MAIN $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}(\lambda)$	$\mathcal{O}^{\mathsf{schnorr\text{-}pop}}()$
$\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$	// PoP: Schnorr signature on X
$Q \leftarrow \emptyset$	$x, r \leftarrow\$ \mathbb{F}$
$(X^*, R^*, z^*) \leftarrow\$ \mathcal{A}^{\mathcal{O}^{\mathsf{schnorr\text{-}pop}}}(\mathcal{G})$	$X \leftarrow g^x$
$\alpha \leftarrow \mathcal{E}(\mathsf{trans}_{\mathcal{A}})$	$R \leftarrow g^r$
// $\mathsf{trans}_{\mathcal{A}}$ is the transcript of $\mathcal{A}$	$c \leftarrow \mathsf{H}_{\mathsf{pop}}(X, X, R)$
if $R^*(X^*)^{\mathsf{H}_{\mathsf{pop}}(X^*,X^*,R^*)} = g^{z^*}$	$z \leftarrow r + cx$
$\wedge\, (X^*, R^*, z^*) \notin Q$	// proof of knowledge of x
$\wedge\, g^\alpha \neq X^*$ return 1	$Q \leftarrow Q \cup \{(X, R, z)\}$
else return 0	return (X, R, z)

Fig. 2. The Schnorr knowledge of exponent game. PoP refers to "proof of possession," which in this setting means demonstrating knowledge of x.

MAIN $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda)$	$\mathcal{O}^{\mathsf{schnorr}}(m)$
$\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$	$r \leftarrow\$ \mathbb{F};\ R \leftarrow g^r$
$x \leftarrow\$ \mathbb{F};\ X \leftarrow g^x$	$c \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X, m, R)$
$Q \leftarrow \emptyset$	$z \leftarrow r + cx$
$(m^*, R^*, z^*) \leftarrow\$ \mathcal{A}^{\mathcal{O}^{\mathsf{schnorr}}}(\mathcal{G}, X)$	$Q \leftarrow Q \cup \{m\}$
if $R^* X^{\mathsf{H}_{\mathsf{schnorr}}(X,m^*,R^*)} = g^{z^*}$	return (R, z)
$\wedge\, m^* \notin Q$ return 1	
else return 0	

Fig. 3. The Schnorr computational game.

4.2 Schnorr Computational Assumption

Our second assumption is that Schnorr signatures are unforgeable. An adversary with access to a Schnorr signing oracle wins the Schnorr computational game if it can forge a Schnorr signature. Security follows from the discrete logarithm assumption in the random oracle model and requires rewinding the adversary once. For completeness, we have included the proof in Appendix B, although we note that multiple versions of this reduction appear in the literature [32, 31, 35]. Mostly, it allows for comparison to our reduction in Section 4.3 of the bischnorr assumption to the omdl assumption, which is new to this work. The schnorr assumption allows us to abstract away details involving rewinding adversaries so that our reduction for SimpleMuSig can focus solely on a straight-line adversary.

Assumption 5 (The Schnorr Computational Assumption) *Let* GroupGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$ *in which the discrete logarithm assumption holds, and let* $\mathsf{H}_{\mathsf{schnorr}}$ *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 3. The Schnorr computational assumption holds with respect to* $\mathcal{G}$ *and* $\mathsf{H}_{\mathsf{schnorr}}$ *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* GroupGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* $\mathsf{H}_{\mathsf{schnorr}}$ *be a random oracle. The Schnorr computational assumption (Assumption 5) is implied by the discrete logarithm assumption with respect to* $\mathcal{G}$ *and* $\mathsf{H}_{\mathsf{schnorr}}$*.*

The Schnorr signature scheme was proven to reduce to the hardness of the discrete logarithm problem with a tightness loss of $\mathcal{O}(q_H)$ by Pointcheval and Stern [32], where q_H is the number of random oracle queries the forger makes. The reason for the tightness loss q_H is that the probability that the adversary outputs two distinct forgeries with the same random oracle query $\mathsf{H}_{\mathsf{schnorr}}(X^*, m^*, R^*)$ could be as as low as $1/q_H$. 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 [16]. While the schnorr assumption could be proven in the AGM, we find the proof to be cleaner in the ROM. 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.

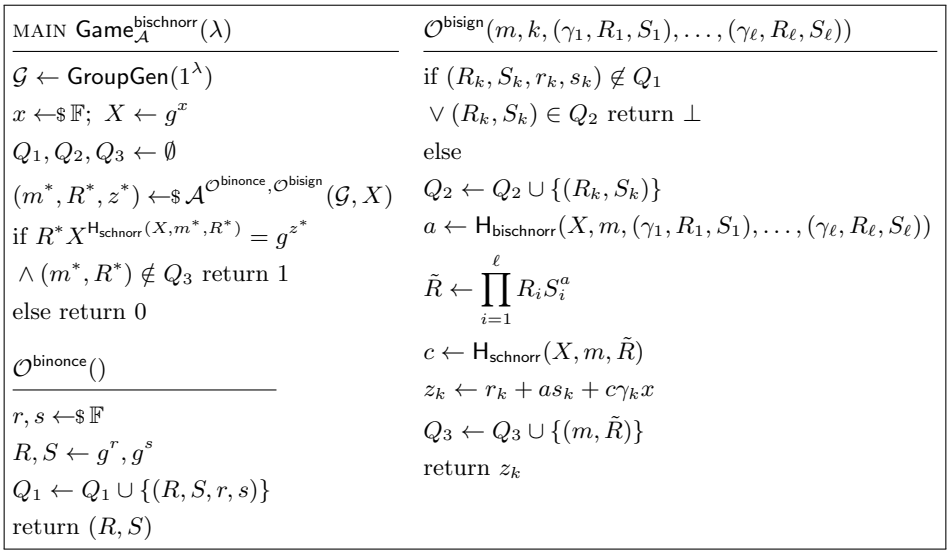

Fig. 4. The binonce Schnorr computational game.

4.3 The Binonce Schnorr Computational Assumption

Our third assumption is computational and proven secure under the one-more discrete logarithm assumption in the random oracle model. We do not know how to reduce the security to the discrete logarithm assumption. Our binonce Schnorr computational assumption (bischnorr) is inspired by the work of Nick et al. [29] and Komlo and Goldberg [25], whose constructions employ the approach of using two nonces to thwart a concurrent forgery attack [13]. We expand on this attack and why it does not affect the bischnorr assumption in Appendix C.

The bischnorr assumption is our most complex, but seemingly the simplest that allows us to prove the security of FROST. The assumption equips an adversary with two oracles, $\mathcal{O}^{\mathsf{binonce}}$ and $\mathcal{O}^{\mathsf{bisign}}$, and two hash functions, $\mathsf{H}_{\mathsf{bischnorr}}$ and $\mathsf{H}_{\mathsf{schnorr}}$, and asks it to forge a new Schnorr signature with respect to a challenge public key 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. It returns an error if not. It then computes an aggregated randomized nonce $\tilde{R} = \prod_{i=1}^{\ell} R_i S_i^a$, where $a = \mathsf{H}_{\mathsf{bischnorr}}(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell))$. $\mathcal{O}^{\mathsf{bisign}}$ then returns z such that $(\tilde{R}, z)$ is a valid Schnorr signature with respect to $\mathsf{H}_{\mathsf{schnorr}}$. The adversary wins if it can output a verifying (m^*, R^*, z^*) 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 $\mathsf{H}_{\mathsf{bischnorr}}$. The scalar γ_k allows the response z to be given as $z = r_k + a s_k + c \gamma_k x$, 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 $(\gamma_1, \ldots, \gamma_k)$ (in addition to the nonces) must be included as input to $\mathsf{H}_{\mathsf{bischnorr}}$ or else there is an attack.

Assumption 6 (The Binonce Schnorr Computational Assumption) *Let* GroupGen *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$ *in which the discrete logarithm assumption holds, and let* H *be a hash function. Let* $\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) = 1]$*, where* $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda)$ *is defined in Figure 4. The binonce Schnorr computational assumption holds with respect to* $\mathcal{G}$ *and* H *if for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* ν *such that* $\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) < \nu(\lambda)$.

Using two nonces does avoid one specific ROS attack (Appendix C), but to be certain that there are not others, we need a watertight security reduction. Nick et al. [29] provide two reductions to the one-more discrete logarithm assumption: one in the random oracle model and one in the algebraic group model. We use their intuition and techniques heavily in reducing our binonce assumption to the omdl assumption in Theorem 3. We do not require a double forking lemma, as in Nick et al., and only rewind the adversary once. Our tightness loss is q_H rather than q_H^2, where q_H the number of hash queries that the adversary can make.

Our proof resembles the reduction for Schnorr signatures to the discrete logarithm assumption in the random oracle model, but here the adversary is allowed to adaptively influence the messages, scalars, and nonces. We implement our security reduction in python and check that it succeeds against an adversary that makes a wide variety of queries.

Theorem 3 ($\mathsf{omdl} \Rightarrow \mathsf{bischnorr}$). *Let* $\mathsf{GroupGen}$ *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$*, and let* $\mathsf{H}_{\mathsf{bischnorr}}, \mathsf{H}_{\mathsf{schnorr}}$ *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* $\mathsf{H}_{\mathsf{bischnorr}}, \mathsf{H}_{\mathsf{schnorr}}$*.*

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

$$\mathsf{Adv}^{\mathsf{bischnorr}}_{\mathcal{A}}(\lambda) \leq q_H \mathsf{Adv}^{\mathsf{omdl}}_{\mathcal{B}}(\lambda)$$

The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times in total. On the second iteration, $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{schnorr}}$ to output a different random value on a single point so that it can extract a discrete logarithm from $\mathcal{A}$. Over the two iterations, $\mathcal{B}$ makes no more than n queries to a 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 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 includes the same nonce R^*. This happens with probability at least equal to $1/q_H$, resulting in a tightness loss of q_H.

$\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{binonce}}$ and $\mathcal{O}^{\mathsf{bisign}}$ as well as $\mathsf{H}_{\mathsf{bischnorr}}$ and $\mathsf{H}_{\mathsf{schnorr}}$. Let $Q_{\mathsf{bischnorr}}, Q_{\mathsf{schnorr}}$ be the set of $\mathsf{H}_{\mathsf{bischnorr}}, \mathsf{H}_{\mathsf{schnorr}}$ queries and responses. $\mathcal{B}$ initializes them to the empty set and maintains them across both iterations of the adversary. Let Q_1, Q_2, Q_3 be the set of $\mathcal{O}^{\mathsf{binonce}}, \mathcal{O}^{\mathsf{bisign}}$ queries and responses 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 $\bar{Q}_2$ of Q_2 to ensure it responds to the same $\mathcal{O}^{\mathsf{bisign}}$ query with the same answer across the two iterations. It then resets Q_1, Q_2, Q_3 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 $0 \leq i \leq n$.

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

$\underline{\mathsf{H}_{\mathsf{bischnorr}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{bischnorr}}$ on $(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell))$ (ℓ can vary), $\mathcal{B}$ checks whether $(X, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_\ell, R_\ell, S_\ell), a) \in Q_{\mathsf{bischnorr}}$ 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 $Q_{\mathsf{bischnorr}}$, and returns a.

$\underline{\mathsf{H}_{\mathsf{schnorr}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{schnorr}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in Q_{\mathsf{schnorr}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\$\, \mathbb{Z}_p$, appends (X, m, R, c) to Q_{schnorr}, 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 $\tilde{Q}_1$ and returns $(R_i, S_i) = (X_{2i-1}, X_{2i})$. Note that $\mathcal{B}$ cannot keep track of the set $Q_1 = \{(R, S, r, s)\}$ as in the real $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}$ 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 $(X_{2i-1}, X_{2i}, i) \in \tilde{Q}_1$ for some i and, if so, that $((R_{k_j}, S_{k_j}), (i, \cdot, \cdot, \cdot), \cdot) \notin Q_2$ as in the real $\mathsf{Game}^{\mathsf{bischnorr}}_{\mathcal{A}}$. 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 $((R_{k_t}, S_{k_t}), (i, z_t, a_t, c_t\gamma_{k_t}), \mathsf{query}_t) \in \bar{Q}_2$ 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_2 and returns z_t.

If not, $\mathcal{B}$ computes $a_j \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(X_0, m_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{\ell_j}))$, $\tilde{R}_j \leftarrow \prod_{i=1}^{\ell} R_{i_j} S_{i_j}^{a_j}$, and $c_j \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_0, m_j, \tilde{R}_j)$. 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 $Q_{\mathsf{bischnorr}}$, (query_j, c_j) to Q_{schnorr}, $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j)$ to Q_2, (query_j, z_j) to $\tilde{Q}_2$, and $(m_j, \tilde{R}_j)$ to Q_3. 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 coins. It then runs $\mathcal{A}(X_0; \mathsf{coins})$ 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 $c^* = \mathsf{H}_{\mathsf{schnorr}}(X_0, m^*, R^*)$ and $(m^*, R^*) \notin Q_3$. 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 $\mathsf{H}_{\mathsf{schnorr}}$ to output c' on input (X_0, m^*, R^*). $\mathcal{B}$ copies $\bar{Q}_2 = Q_2$ to have a record of these queries, and then resets $\tilde{Q}_1, Q_2, Q_3$ to the empty set. The sets $\tilde{Q}_2$, Q_{schnorr}, $Q_{\mathsf{bischnorr}}$ are also kept for the second iteration of the adversary. $\mathcal{B}$ then runs $\mathcal{A}(X_0; \mathsf{coins})$ again on the same random coins.

After the second iteration, suppose $\mathcal{A}$ terminates with (m', R', z'). If $(m', R') = (m^*, R^*)$, then $\mathcal{B}$ can extract $x_0 = \frac{z^* - z'}{c^* - c'}$ such that $X_0 = g^{x_0}$. If $(m', R') \neq (m^*, R^*)$, then $\mathcal{B}$ must abort. The probability that $\mathcal{B}$ does not abort is strictly greater than $1/q_H$ (which is polynomial) because: (1) $\mathcal{A}$ makes no more than q_H queries to $\mathsf{H}_{\mathsf{bischnorr}}, \mathsf{H}_{\mathsf{schnorr}}$ in total; and (2) the

statistical probability of $\mathcal{A}$ succeeding if it did not query $\mathsf{H}_{\mathsf{schnorr}}$ on (X_0, m^*, R^*) and (X_0, m', R') is less than $\frac{2}{p} < \frac{1}{q_H}$. Thus, the probability that $\mathcal{B}$ extracts x_0 is at least $1/q_H$.

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 $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_2$ 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 $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_2$ 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 $((R_{k_j}, S_{k_j}), (i, z_j, a_j, c_j\gamma_{k_j}), \mathsf{query}_j) \in \bar{Q}_2$ and $((R_{k_{j'}}, S_{k_{j'}}), (i, z_{j'}, a_{j'}, c_{j'}\gamma_{k_{j'}}), \mathsf{query}_{j'}) \in Q_2$. Then $\mathcal{B}$ sets

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

Suppose $a_j = a_{j'}$. That is,

$$\mathsf{H}_{\mathsf{bischnorr}}(X_0, m_j, (\gamma_{1_j}, R_{1_j}, S_{1_j}), \ldots, (\gamma_{\ell_j}, R_{\ell_j}, S_{\ell_j})) = \mathsf{H}_{\mathsf{bischnorr}}(X_0, m_{j'}, (\gamma_{1_{j'}}, R_{1_{j'}}, S_{1_{j'}}), \ldots, (\gamma_{\ell_{j'}}, R_{\ell_{j'}}, S_{\ell_{j'}}))$$

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 $\mathsf{H}_{\mathsf{bischnorr}}$ 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. 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.

5 Proving the Security of Multisignatures

In this section, we first discuss the definition of a multisignature scheme. We then construct and prove the security of a simple, three-round multisignature scheme with proofs of possession under the Schnorr computational assumption and the Schnorr knowledge of exponent assumption. Finally, we construct and prove the security of a variant of the two-round MuSig2 multisignature scheme with proofs of possession under the binonce Schnorr computational assumption and the Schnorr knowledge of exponent assumption.

5.1 Definition of Security for Multisignatures

We build upon the definition of a multisignature scheme with proofs of possession given by Ristenpart and Yilek [33], 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 $\mathsf{KeyVerify}$. 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 $\mathsf{Combine}$ algorithm to form the multisignature σ. Note that $\mathsf{Combine}$ 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 should be 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 $1 \le i \le n$ 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. 6 in Appendix D 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 $\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}''}$ 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 $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$.

5.2 Three-Round Proof-of-Possession Multisignature SimpleMuSig

In this section, we define a three-round multisignature scheme $\mathsf{SimpleMuSig}$ (Fig. 7) with proofs of possession and prove its EUF-CMA security under the Schnorr computational assumption and the Schnorr knowledge of exponent assumption (Theorem 4). Any complexity around rewinding the adversary has already been abstracted away into the assumptions, allowing us to focus on the primary concerns of the security proof, namely (1) whether the reduction aborts; (2) whether the adversary can distinguish between its interaction with the reduction and the real EUF-CMA game; and (3) whether the reduction succeeds whenever the adversary succeeds.

In all of our proofs, we assume the adversary only makes well-formed queries. Any ill-formed query could be answered with random values by the reduction. We also assume the adversary completes every signing session and that the reduction does not program a random oracle on a point that has already been queried by the adversary.

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

[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 [28].

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 $j \neq i$. 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 $\tilde{X}$.

Theorem 4 (SimpleMuSig)**.** SimpleMuSig *is EUF-CMA secure under the Schnorr computational assumption (Assumption 5) and the Schnorr knowledge of exponent assumption (Assumption 4).*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the EUF-CMA security of SimpleMuSig. We construct a PPT reduction $\mathcal{B}$ playing $\mathsf{Game}_{\mathcal{B}}^{\mathsf{schnorr}}(\lambda)$ such that whenever $\mathcal{A}$ outputs a valid forgery, $\mathcal{B}$ breaks the Schnorr computational assumption.

The reduction $\mathcal{B}$ is responsible for simulating oracle responses for key registration, the three rounds of signing, and queries to $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}$ may program $\mathsf{H}_{\mathsf{reg}}$, H_{cm}, and $\mathsf{H}_{\mathsf{sig}}$, but not $\mathsf{H}_{\mathsf{pop}}$ or $\mathsf{H}_{\mathsf{schnorr}}$ (because it is part of $\mathcal{B}$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses, and similarly for Q_{cm} and $\mathsf{Q}_{\mathsf{sig}}$. $\mathcal{B}$ initializes $\mathsf{Q}_{\mathsf{reg}}, \mathsf{Q}_{\mathsf{cm}}, \mathsf{Q}_{\mathsf{sig}}$ to the empty set.

$\mathcal{B}$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key X_1. $\mathcal{B}$ simulates a proof of possession of X_1 as follows. It chooses $\bar{c}_1, \bar{z}_1 \leftarrow\!\!\$\, \mathbb{Z}_p$ and sets $\bar{R}_1 \leftarrow g^{\bar{z}_1} X_1^{-\bar{c}_1}$. It then sets $\pi_1 \leftarrow (\bar{R}_1, \bar{z}_1)$ and $\mathsf{pk}_1 \leftarrow (X_1, \pi_1)$. $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{reg}}$ to return $\bar{c}_1$ on input $(X_1, X_1, \bar{R}_1)$ and appends $(X_1, X_1, \bar{R}_1, \bar{c}_1)$ to $\mathsf{Q}_{\mathsf{reg}}$. It adds X_1 to the list $\mathcal{LPK}$ of potential signers and runs $\mathcal{A}$ on input pk_1.

Hash Queries. When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ 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, $\mathcal{B}$ queries $\bar{c} \leftarrow \mathsf{H}_{\mathsf{pop}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

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

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}$ samples a random message $\hat{m}$ (to prevent trivial collisions), queries $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \hat{m}, R)$, appends $(X, m, R, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\hat{c}$.

Key Registration. During key registration, all parties send their public keys. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Register}}$ to register $\mathsf{pk} = (X, \pi)$ such that $\mathsf{KeyVerify}(X, \pi) = 1$, $\mathcal{B}$ adds X to the list $\mathcal{LPK}$ of potential signers (if X isn't already included) and runs the extractor $\mathcal{E}$ to obtain x such that $X = g^x$.

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 $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}$ samples a random message $\dot{m}$ and queries $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ to get a signature (R_1, z_1). $\mathcal{B}$ checks whether $(R_1, \mathsf{cm}_1) \in \mathsf{Q}_{\mathsf{cm}}$ and, if so, returns cm_1. Else, $\mathcal{B}$ 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 $\mathsf{cm}_i = \mathsf{H}_{\mathsf{cm}}(R_i)$. $\mathcal{B}$ 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 $\mathcal{B}$ aborts. If all records exist, then $\mathcal{B}$ computes $\tilde{X} \leftarrow \prod_{i=1}^n X_i$ and $\tilde{R} = \prod_{i=1}^n R_i$, queries $\dot{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \dot{m}, R_1)$ (note that it's 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 $\mathcal{B}$ aborts.) For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}$ 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 $\mathcal{O}^{\mathsf{Sign}''}$, $\mathcal{B}$ 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 $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$, $\mathcal{B}$ computes its output as follows. $\mathcal{B}$ looks up $x_2^*, \ldots, x_n^*$ from registration such that $X_i^* = g^{x_i^*}$. $\mathcal{B}$ also looks up $\hat{m}^*$ from when $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and $\mathcal{B}$ had responded with $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \hat{m}^*, \tilde{R}^*)$. $\mathcal{B}$ outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$.

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

(1) $\mathcal{B}$ 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. The probability that $\mathcal{A}$ guessed cm_j correctly is $1/p$, which is negligible.

$\mathcal{B}$ 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, $\mathcal{B}$ had returned $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \hat{m}, \tilde{R})$ for some random message $\hat{m}$, so the reduction fails. However, this implies that $\mathcal{A}$ guessed R_1 before $\mathcal{B}$ revealed it, which occurs with negligible probability.

(2) As long as $\mathcal{B}$ does not abort, $\mathcal{B}$ 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 $\mathcal{B}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ on $(X, X, \bar{R}) \neq (X_1, X_1, \bar{R}_1)$, $\mathcal{B}$ queries its $\mathsf{H}_{\mathsf{pop}}$ oracle on $(X, X, \bar{R})$, so $\mathcal{A}$ receives a random value. For $(X_1, X_1, \bar{R}_1)$, $\mathcal{B}$ programmed $\mathsf{H}_{\mathsf{reg}}$ to output its simulated $\bar{c}_1$ at the beginning of the game. $\mathcal{B}$'s 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), $\mathcal{B}$ queries $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \hat{m}, R)$ 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 $c \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, m, R)$ 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 $\mathcal{B}$ as $\mathsf{H}_{\mathsf{schnorr}}(X_1, \dot{m}, R_1)$. When $\mathcal{B}$ queried $\mathcal{O}^{\mathsf{schnorr}}$ on $\dot{m}$ in Signing Round 1, the signature share z_1 was computed such that $R_1 X_1^{\mathsf{H}_{\mathsf{schnorr}}(X_1, \dot{m}, R_1)} = g^{z_1}$, so $\mathcal{B}$ simulates z_1 correctly.

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

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

At some point, for all $2 \leq i \leq n$, $\mathcal{A}$ queried $\mathcal{O}^{\mathsf{Register}}$ to register (X_i^*, π_i^*) such that $\mathsf{KeyVerify}(X_i^*, \pi_i^*) = 1$. This implies that $\bar{R}_i^* (X_i^*)^{\mathsf{H}_{\mathsf{pop}}(X_i^*, X_i^*, \bar{R}_i^*)} = g^{\bar{z}_i^*}$, which is a valid output in the Schnorr knowledge of exponent game (provided that $X_i^* \neq X_1$). Thus, the Schnorr knowledge of exponent assumption guarantees that $\mathcal{E}$ succeeds at outputting x_i^* such that $X_i^* = g^{x_i^*}$ with overwhelming probability. This allows $\mathcal{B}$ 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 $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(X_1, \hat{m}^*, \tilde{R}^*)$, so $\mathcal{A}$'s forgery satisfies:

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

Thus, $\mathcal{B}$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$ under X_1 is correct. If $\mathcal{A}$ outputs a valid forgery for $\mathsf{SimpleMuSig}$ with non-negligible probability, then $\mathcal{B}$ breaks the Schnorr computational assumption.

5.3 Two-Round Proof-of-Possession Multisignature SpeedyMuSig

We now construct a variant of the two-round multisignature scheme $\mathsf{MuSig2}$ [29] that includes proofs of possession, which we call $\mathsf{SpeedyMuSig}$ (Fig. 8). Similar to $\mathsf{SimpleMuSig}$, 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. We show that $\mathsf{SpeedyMuSig}$ is EUF-CMA secure under the binonce Schnorr computational assumption and the Schnorr knowledge of exponent assumption (Theorem 5). Since $\mathsf{SpeedyMuSig}$ consists of two rounds of signing, there is no need for a $\mathsf{Sign}''()$ algorithm, but for the purpose of aligning with our generic definition of EUF-CMA security (Fig. 6), one may assume it returns no value. 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 $(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$, 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.[6] 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 $z_i \leftarrow r_i + as_i + cx_i$ 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 $\sigma \leftarrow (\tilde{R}, z)$.

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 $\tilde{X}$.

Theorem 5 (SpeedyMuSig). SpeedyMuSig *is EUF-CMA secure under the binonce Schnorr computational assumption (Assumption 6) and the Schnorr knowledge of exponent assumption (Assumption 4).*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the EUF-CMA security of SpeedyMuSig. We construct a PPT reduction $\mathcal{B}$ playing $\mathsf{Game}_{\mathcal{B}}^{\mathsf{bischnorr}}(\lambda)$ such that whenever $\mathcal{A}$ outputs a valid forgery, $\mathcal{B}$ breaks the binonce Schnorr computational assumption.

The reduction $\mathcal{B}$ is responsible for simulating oracle responses for key registration, the two rounds of signing, and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}$ may program $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\mathsf{H}_{\mathsf{pop}}, \mathsf{H}_{\mathsf{schnorr}}$, or $\mathsf{H}_{\mathsf{bischnorr}}$ (because they are part of $\mathcal{B}$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses, and similarly for $\mathsf{Q}_{\mathsf{non}}$ and $\mathsf{Q}_{\mathsf{sig}}$. $\mathcal{B}$ initializes $\mathsf{Q}_{\mathsf{reg}}, \mathsf{Q}_{\mathsf{non}}, \mathsf{Q}_{\mathsf{sig}}$ to the empty set.

$\mathcal{B}$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}_1$. $\mathcal{B}$ simulates a proof of possession of $\dot{X}_1$ as follows. It chooses $\bar{c}_1, \bar{z}_1 \leftarrow\!\!\$\, \mathbb{Z}_p$ and sets $\bar{R}_1 \leftarrow g^{\bar{z}_1}\dot{X}_1^{-\bar{c}_1}$. $\mathcal{B}$ sets $\pi_1 \leftarrow (\bar{R}_1, \bar{z}_1)$ and $\mathsf{pk}_1 \leftarrow (\dot{X}_1, \pi_1)$. $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{reg}}$ to return $\bar{c}_1$ on input $(\dot{X}_1, \dot{X}_1, \bar{R}_1)$ and appends $(\dot{X}_1, \bar{R}_1, \bar{c}_1)$ to $\mathsf{Q}_{\mathsf{reg}}$. $\mathcal{B}$ then runs $\mathcal{A}$ on input pk_1.

Hash Queries. When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ 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, $\mathcal{B}$ queries $\bar{c} \leftarrow \mathsf{H}_{\mathsf{pop}}(X, X, \bar{R})$, appends $(X, X, \bar{R}, \bar{c})$ to $\mathsf{Q}_{\mathsf{reg}}$, and returns $\bar{c}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$, $\mathcal{B}$ 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, $\mathcal{B}$ samples a random message $\hat{m}$ (to prevent trivial collisions), sets $\gamma_i = 1\ \forall i$[7], computes $\hat{a} \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}_1, \hat{m}, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$, and appends $(X, m, R_1, S_1, \ldots, R_n, S_n, \hat{m}, \hat{a})$ to $\mathsf{Q}_{\mathsf{non}}$. $\mathcal{B}$ 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, $\mathcal{B}$ computes $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}, \prod_1^n R_i S_i^{\hat{a}})$ and appends $(X, m, \prod_1^n R_i S_i^{\hat{a}}, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. Finally, $\mathcal{B}$ returns $\hat{a}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, \hat{m}, \hat{c}) \in \mathsf{Q}_{\mathsf{sig}}$ and, if so, returns $\hat{c}$. Else, $\mathcal{B}$ samples a random message $\dot{m}$, computes $\dot{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \dot{m}, R)$, appends $(X, m, R, \dot{m}, \dot{c})$ to $\mathsf{Q}_{\mathsf{sig}}$, and returns $\dot{c}$.

Key Registration. During key registration, all parties send their public keys. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Register}}$ to register $\mathsf{pk} = (X, \pi)$ such that $\mathsf{KeyVerify}(X, \pi) = 1$, $\mathcal{B}$ adds X to the list $\mathcal{LPK}$ of potential signers (if X isn't already included) and runs the extractor $\mathcal{E}$ to obtain x such that $X = g^x$.

Signing Round 1 (Sign). In the first round of signing, all parties who intend to participate send two nonces $(R_1, S_1), \ldots, (R_n, S_n)$. $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{binonce}}$ to get $(\dot{R}_1, \dot{S}_1)$. For $\mathcal{A}$'s query to $\mathcal{O}^{\mathsf{Sign}}$, $\mathcal{B}$ 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. $\mathcal{B}$ checks if $(R_i, S_i) = (R_j, S_j)$ for any $i \neq j$ and if so, aborts. Else, $\mathcal{B}$ 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, $\mathcal{B}$ 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 $\mathcal{O}^{\mathsf{Sign}'}$, $\mathcal{B}$ 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 $\mathsf{Verify}(\mathcal{PK}^*, m^*, \sigma^*) = 1$, $\mathcal{B}$ computes its output as follows. It looks up $x_2^*, \ldots, x_n^*$ from registration such that $X_i^* = g^{x_i^*}$.

[6] The proof of Theorem 3 demonstrates why this is required.

[7] Lagrange coefficients for multisignatures are 1.

$\mathcal{B}$ also looks up $\hat{m}^*$ from when $\mathcal{A}$ queried $\mathsf{H}_{\mathsf{sig}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$ and $\mathcal{B}$ had responded with $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}^*, \tilde{R}^*)$. $\mathcal{B}$ outputs $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$.

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

(1) $\mathcal{B}$ 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.

(2) As long as $\mathcal{B}$ does not abort, $\mathcal{B}$ 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 $\mathcal{B}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{non}}$ on $(X, m, R_1, S_1, \ldots, R_n, S_n)$, $\mathcal{B}$ queries $\hat{a} \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}_1, \hat{m}, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$ 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 $a \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}_1, m, (\gamma_1, R_1, S_1), \ldots, (\gamma_n, R_n, S_n))$ 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. $\mathcal{B}$ 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

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

When $\mathcal{B}$ 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

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

$\mathcal{B}$ 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 $X_1^* = \dot{X}_1$, which implies:

$$\begin{aligned}
\tilde{R}^* (\tilde{X}^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* (X_1^* \cdots X_n^*)^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* (\dot{X}_1 g^{x_2 *} \cdots g^{x_n^*})^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^*} \\
\tilde{R}^* \dot{X}_1^{\mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)} &= g^{z^* - \mathsf{H}_{\mathsf{sig}}(\tilde{X}^*, m^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}
\end{aligned}$$

At some point, for all $2 \leq i \leq n$, $\mathcal{A}$ queried $\mathcal{O}^{\mathsf{Register}}$ to register (X_i^*, π_i^*) such that $\mathsf{KeyVerify}(X_i^*, \pi_i^*) = 1$. This implies that $\bar{R}_i^* (X_i^*)^{\mathsf{H}_{\mathsf{pop}}(X_i^*, X_i^*, \bar{R}_i^*)} = g^{\bar{z}_i^*}$, which is a valid output in the Schnorr knowledge of exponent game (provided that $X_i^* \neq \dot{X}_1$). Thus, the Schnorr knowledge of exponent assumption guarantees that $\mathcal{E}$ succeeds at outputting x_i^* such that $X_i^* = g^{x_i^*}$ with overwhelming probability. This allows $\mathcal{B}$ 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) $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}^*, \prod_1^n R_i^* (S_i^*)^{\hat{a}^*})$ 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 $\hat{a}^* \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}_1, \hat{m}^*, (1, R_1^*, S_1^*), \ldots, (1, R_n^*, S_n^*))$, or (2) $\dot{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \dot{m}^*, \tilde{R}^*)$ without having queried $\mathsf{H}_{\mathsf{non}}$ on $(\tilde{X}^*, m^*, \tilde{R}^*)$. In either case, $\mathcal{B}$ has a record $(\tilde{X}^*, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ such that $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}^*, \tilde{R}^*)$. (Note that $\mathcal{B}$ can check which case occurred by looking for $\hat{m}^*$ in its $\mathsf{Q}_{\mathsf{non}}$ records.) Thus, $\mathcal{A}$'s forgery satisfies:

$$\tilde{R}^* \dot{X}_1^{\mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}^*, \tilde{R}^*)} = g^{z^* - \mathsf{H}_{\mathsf{schnorr}}(\dot{X}_1, \hat{m}^*, \tilde{R}^*)(x_2^* + \cdots + x_n^*)}$$

and $\mathcal{B}$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^*(x_2^* + \cdots + x_n^*))$ under $\dot{X}_1$ is correct. If $\mathcal{A}$ outputs a valid forgery for SpeedyMuSig with non-negligible probability, then $\mathcal{B}$ breaks the binonce Schnorr computational assumption.

6 Proving Security of Threshold Signatures

In this section, we describe and prove the two-round threshold signature scheme FROST secure under the Schnorr knowledge of exponent assumption (Assumption 4) and the binonce Schnorr computational assumption (Assumption 6). FROST uses the Pedersen distributed key generation protocol (DKG) [20] with proofs of possession in order to generate the joint public key. We call this protocol PedPoP and provide a description in Figure 9. The Pedersen DKG can be viewed as n parallel instantiations of Feldman verifiable secret sharing (VSS) [14], which itself is derived from Shamir secret sharing [36] but additionally requires each participant to provide a vector commitment $\boldsymbol{C}$ to ensure their received share is consistent with all other participants' shares. Following FROST [25], we require 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.

6.1 Definition of Security for Threshold Signatures

We build upon prior definitions of threshold signature schemes in the literature [19, 22, 18], 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 $\mathsf{Combine}$ algorithm to form the threshold signature σ. Note that $\mathsf{Combine}$ may be performed by one of the signers or an external party. The threshold signature σ on m is valid if $\mathsf{Verify}(\tilde{X}, m, \sigma) = 1$.

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 $\mathcal{O}^{\mathsf{Sign},\mathsf{Sign}',\mathsf{Sign}''}$ 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 $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$.

6.2 Two-Round Threshold Signature FROST

We review FROST key generation and signing in Figure 10. We introduce a modification to FROST that allows for improved efficiency during signing, reducing the number of exponentiations from at least t to one. We achieve this by using the same hash value for all signers.

First, the joint public key $\tilde{X}$ is generated using PedPoP (Fig. 9). 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:

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

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:

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

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 $z_i = r_i + a s_i + c \lambda_i f(\mathsf{id}_i)$.

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 $(\tilde{R}, z)$.

Theorem 6 (FROST). *FROST is secure under the binonce Schnorr computational assumption (Assumption 6) and the Schnorr knowledge of exponent assumption (Assumption 4).*

Proof. Let $\mathcal{A}$ be a PPT adversary attempting to break the EUF-CMA security of FROST. We construct a PPT reduction $\mathcal{B}$ playing $\mathsf{Game}_{\mathcal{B}}^{\mathsf{bischnorr}}(\lambda)$ such that whenever $\mathcal{A}$ outputs a valid forgery, $\mathcal{B}$ breaks the binonce Schnorr computational assumption.

The reduction $\mathcal{B}$ is responsible for simulating honest parties in PedPoP (Fig. 9), the two rounds of signing, and queries to $\mathsf{H}_{\mathsf{reg}}$, $\mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$. $\mathcal{B}$ receives as input group parameters $\mathcal{G} = (\mathbb{G}, p, g)$ and a challenge public key $\dot{X}$. 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 the joint public key $\tilde{X}$, $\mathcal{B}$ returns y such that $\tilde{X} = \dot{X}g^y$. Additionally, when PedPoP outputs public key share $\tilde{X}_k = g^{\bar{x}_k}$ for each honest party $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}$ returns (α_k, β_k) such that $\tilde{X}_k = \dot{X}^{\alpha_k} g^{\beta_k}$. This representation allows $\mathcal{B}$ to simulate FROST signing under each $\tilde{X}_k$.

Initialization. $\mathcal{B}$ may program $\mathsf{H}_{\mathsf{reg}}, \mathsf{H}_{\mathsf{non}}$, and $\mathsf{H}_{\mathsf{sig}}$, but not $\mathsf{H}_{\mathsf{bischnorr}}$ or $\mathsf{H}_{\mathsf{schnorr}}$ (because they are part of $\mathcal{B}$'s challenge). Let $\mathsf{Q}_{\mathsf{reg}}$ be the set of $\mathsf{H}_{\mathsf{reg}}$ queries and their responses, and similarly for $\mathsf{Q}_{\mathsf{non}}$ and $\mathsf{Q}_{\mathsf{sig}}$. Let Q_{Sign} be the set of $\mathcal{O}^{\mathsf{Sign}}$ queries and responses in the first round of signing, and let $Q_{\mathsf{Sign}'}$ be the set of $\mathcal{O}^{\mathsf{Sign}'}$ queries and responses in the second round. $\mathcal{B}$ initializes all of them to the empty set.

$\mathcal{B}$ computes α_k for each honest party $\mathsf{id}_k \in \mathsf{hon}$ as follows. First, $\mathcal{B}$ 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, $\mathcal{B}$ sets $\alpha_k \leftarrow L'_k(0)^{-1}$. (It will become clear why α_k is computed this way.)

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

$\underline{\mathsf{H}_{\mathsf{reg}}}$: When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{reg}}$ 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, $\mathcal{B}$ queries $\bar{c} \leftarrow \mathsf{H}_{\mathsf{pop}}(X, X, \bar{R})$, 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 $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$, $\mathcal{B}$ 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, $\mathcal{B}$ checks whether there exists some $k' \in \mathcal{S}$ such that $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$. If not, $\mathcal{B}$ 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 $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$, $\mathcal{B}$ 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 $\{\mathsf{id}_i\}_{i \in \mathcal{S}}$. $\mathcal{B}$ 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 $\mathcal{S}$. $\mathcal{B}$ then samples a random message $\hat{m}$ (to prevent trivial collisions), queries $\hat{a} \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}, \hat{m}, \{(\gamma_i, R_i, S_i)\}_{i \in \mathcal{S}})$, and appends $(X, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}}, \hat{m}, \hat{a})$ to $\mathsf{Q}_{\mathsf{non}}$. $\mathcal{B}$ 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, $\mathcal{B}$ aborts. Else, $\mathcal{B}$ queries $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \hat{m}, \hat{R})$ and appends $(\tilde{X}, m, \hat{R}, \hat{m}, \hat{c})$ to $\mathsf{Q}_{\mathsf{sig}}$. Finally, $\mathcal{B}$ returns $\hat{a}$.

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

Simulating the DKG. $\mathcal{B}$ runs PedPoP with $\mathcal{A}$ as follows. $\mathcal{B}$ embeds the challenge $\dot{X}$ as the public key of the honest party that the adversary queries first. Let this first honest party be id_τ. $\mathcal{B}$ 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 $\mathcal{B}$ must be able to simulate any of them.

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

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

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. $\mathcal{B}$ simulates honest party id_τ as follows.

1. $\mathcal{B}$ sets the public key $X_{\tau,0} \leftarrow \dot{X}$ and simulates a Schnorr proof of possession of $\dot{X}$ as follows. $\mathcal{B}$ 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 $\mathsf{Q}_{\mathsf{reg}}$.
2. $\mathcal{B}$ simulates a verifiable Shamir secret sharing of the discrete logarithm of $\dot{X}$ by performing the following steps.
   (a) $\mathcal{B}$ 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 $\mathsf{id}_j \in \mathsf{cor}$. $\mathcal{B}$ 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 $1 \leq i \leq t-1$, $\mathcal{B}$ computes
   $$A_{\tau,i} = \dot{X}^{L'_{0,i}} \prod_{\mathsf{id}_j \in \mathsf{cor}} g^{\bar{x}_{\tau,j} L'_{j,i}} \tag{2}$$
   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) $\mathcal{B}$ outputs $(\bar{R}_\tau, \bar{z}_\tau), \boldsymbol{C}_\tau = (A_{\tau,0} = X_{\tau,0}, A_{\tau,1}, \ldots, A_{\tau,t-1})$ for the broadcast round, and then sends shares $\bar{x}_{\tau,j}$ for each $j \in \mathsf{cor}$.
3. $\mathcal{B}$ 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, $\mathcal{B}$ 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 $\bar{x}_{\tau,k}$, $\mathcal{B}$ 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 $\mathsf{id}_k \in \mathsf{hon}, k \neq \tau$, $\mathcal{B}$ follows the protocol. $\mathcal{B}$ samples $f_k(Z) = a_{k,0} + a_{k,1}Z + \ldots + a_{k,t-1}Z^{t-1} \leftarrow\!\!\$\, \mathbb{Z}_p[Z]$ and sets $A_{k,i} \leftarrow g^{a_{k,i}}$ for all $0 \leq i \leq t-1$. $\mathcal{B}$ 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

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

if $(X_{j,0}, \bar{R}_j, \bar{z}_j)$ verifies (i.e., $\bar{R}_j X_{j,0}^{\mathsf{H}_{\mathsf{reg}}(X_{j,0}, X_{j,0}, \bar{R}_j)} = g^{\bar{z}_j}$), then $\mathcal{B}$ runs the extractor $\mathcal{E}(\mathsf{trans}_{\mathcal{A}})$ to obtain $a_{j,0}$ such that $X_{j,0} = g^{a_{j,0}}$. The Schnorr knowledge of exponent assumption guarantees that $\mathcal{E}$ succeeds at outputting $a_{j,0}$ with overwhelming probability.

Complaints. If $\mathcal{A}$ broadcasts a complaint, $\mathcal{B}$ 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 $\mathcal{B}$ 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 and $a_{j,0}$ is set to 0.

DKG Termination. When PedPoP terminates, the output is the joint public key

$$\tilde{X} = \prod_{i=0}^{n} X_{i,0}$$

$\mathcal{B}$ computes

$$y = \textstyle\sum_{i=1, i\neq\tau}^{n} a_{i,0}$$

Then

$$\tilde{X} = \dot{X} g^{y}$$

$\mathcal{B}$ 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 3 above, so $\alpha_k = \alpha'_k$ and $\beta_k = \beta'_k + \sum_{i=1, i\neq\tau}^{n} \bar{x}_{i,k}$. $\mathcal{B}$ returns y and $\{(\alpha_k, \beta_k)\}_{\mathsf{id}_k \in \mathsf{hon}}$.

Simulating FROST Signing. After completing simulation of PedPoP, $\mathcal{B}$ then simulates honest parties in the FROST signing protocol.

Signing Round 1 (Sign). When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{Sign}}$ on $\mathsf{id}_k \in \mathsf{hon}$, $\mathcal{B}$ 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 $(m, k', \{(\mathsf{id}_i, R_i, S_i)\}_{i\in\mathcal{S}})$, $\mathcal{B}$ first checks if $(R_i, S_i) = (R_j, S_j)$ for any $i \neq j$ and, if so, aborts. Then, $\mathcal{B}$ checks whether $(\mathsf{id}_{k'}, R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}}$ and, if not, returns $\perp$. $\mathcal{B}$ also checks whether $(R_{k'}, S_{k'}) \in Q_{\mathsf{Sign}'}$ and, if so, returns $\perp$.

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

Next, $\mathcal{B}$ 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 $\{\mathsf{id}_i\}_{i\in\mathcal{S}}$. $\mathcal{B}$ 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, $\mathcal{B}$ queries $\mathcal{O}^{\mathsf{bisign}}$ on $(\hat{m}', k', \{(\gamma_i, R_i, S_i)\}_{i\in\mathcal{S}})$ to get $z_{k'}$. Finally, $\mathcal{B}$ computes

$$\tilde{z}_{k'} = z_{k'} + \hat{c}' \lambda_{k'} \beta_{k'} \tag{3}$$

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

Output. When $\mathcal{A}$ returns $(\tilde{X}, m^*, \sigma^*)$ such that $\sigma^* = (\tilde{R}^*, z^*)$ and $\mathsf{Verify}(\tilde{X}, m^*, \sigma^*) = 1$, $\mathcal{B}$ computes its output as follows. $\mathcal{B}$ 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) $\mathcal{B}$ only aborts with negligible probability; (2) $\mathcal{A}$ cannot distinguish between a real run of the protocol and its interaction with $\mathcal{B}$; and (3) whenever $\mathcal{A}$ succeeds, $\mathcal{B}$ succeeds.

(1) $\mathcal{B}$ 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.

(2) As long as $\mathcal{B}$ does not abort, $\mathcal{B}$ 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 $\mathcal{B}$.

Indeed, $\mathcal{B}$'s simulation of PedPoP is perfect, as performing validation of each player's share (Step 4 in Fig. 9) holds, and by Equation 2, interpolation in the exponent correctly evaluates to the challenge $\dot{X}$.

When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{sig}}$ on (X, m, R), $\mathcal{B}$ queries $\hat{c} \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \hat{m}, R)$ 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 $c \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, m, R)$ 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. $\mathcal{B}$ 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

$$R_{k'} S_{k'}^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})} \tilde{X}_{k'}^{\lambda_{k'} \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \prod_{i \in \mathcal{S}} R_i S_i^{\mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})})} = g^{\tilde{z}_{k'}} \tag{4}$$

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

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

$\mathcal{B}$ computed the signature share $\tilde{z}_{k'}$ (Equation 3) as

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

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

$$R_{k'} S_{k'}^{\mathsf{H}_{\mathsf{bischnorr}}(\dot{X}, \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i \in \mathcal{S}})} \tilde{X}_{k'}^{\lambda_{k'} \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \hat{m}', \prod_{i \in \mathcal{S}} R_i S_i^{\mathsf{H}_{\mathsf{bischnorr}}(\dot{X}, \hat{m}', \{(\gamma_i, R_i, S_i)\}_{i \in \mathcal{S}})})} = g^{\tilde{z}_{k'}} \tag{5}$$

$\mathcal{B}$ has programmed the hash values in Equations 4 and 5 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) $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \hat{m}^*, \prod_{i \in \mathcal{S}^*} R_i^* (S_i^*)^{\hat{a}^*})$ 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 $\hat{a}^* \leftarrow \mathsf{H}_{\mathsf{bischnorr}}(\dot{X}, \hat{m}^*, (\gamma_i^*, R_i^*, S_i^*)_{i \in \mathcal{S}^*})$, or (2) $\dot{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \dot{m}^*, \tilde{R}^*)$ without having queried $\mathsf{H}_{\mathsf{non}}$ first. In either case, $\mathcal{B}$ has a record $(\tilde{X}, m^*, \tilde{R}^*, \hat{m}^*, \hat{c}^*) \in \mathsf{Q}_{\mathsf{sig}}$ such that $\hat{c}^* \leftarrow \mathsf{H}_{\mathsf{schnorr}}(\dot{X}, \hat{m}^*, \tilde{R}^*)$. (Note that $\mathcal{B}$ can check which case occurred by looking for $\hat{m}^*$ in its $\mathsf{Q}_{\mathsf{non}}$ records.) Thus, $\mathcal{A}$'s forgery satisfies:

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

and $\mathcal{B}$'s output $(\hat{m}^*, \tilde{R}^*, z^* - \hat{c}^* y)$ under $\dot{X}$ is correct. If $\mathcal{A}$ outputs a valid forgery for FROST with non-negligible probability, then $\mathcal{B}$ breaks the binonce Schnorr computational assumption.

7 Conclusion

We present an improved method for proving the security of multi- and threshold signature schemes, building upon three novel but intuitive assumptions. We demonstrate the strength of this methodology by proving the security of a range of multi-party schemes, from simple three-round constructions to variants of the more recent two-round MuSig2 [29] and FROST [25] signatures. Our goal 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 the Blockchain Technology Laboratory at the University of Edinburgh and funded by Input Output Global.

References

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

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

[3] 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.

[4] 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.

[5] 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.

[6] 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, Proceedings*. Ed. by S. Vaudenay. Vol. 4004. LNCS. Springer, 2006, pp. 409–426.

[7] 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, Proceedings, Part I*. Ed. by A. Canteaut and F. Standaert. Vol. 12696. LNCS. Springer, 2021, pp. 33–53.

[8] 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, Proceedings*. Ed. by Y. Desmedt. Vol. 2567. LNCS. Springer, 2003, pp. 31–46.

[9] D. Boneh, M. Drijvers, and G. Neven. "Compact Multi-signatures for Smaller Blockchains". In: *ASIACRYPT 2018, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II*. Ed. by T. Peyrin and S. D. Galbraith. Vol. 11273. LNCS. Springer, 2018, pp. 435–464.

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

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

[12] 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.

[13] 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.

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

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

[16] 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, Proceedings, Part II*. Ed. by H. Shacham and A. Boldyreva. Vol. 10992. LNCS. Springer, 2018, pp. 33–62.

[17] 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.

[18] 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.

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

[20] 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, Proceedings*. Ed. by M. Joye. Vol. 2612. LNCS. Springer, 2003, pp. 373–390.

[21] 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.

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

[23] 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.

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

[25] C. Komlo and I. Goldberg. "FROST: Flexible Round-Optimized Schnorr Threshold Signatures". In: *SAC 2020, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers*. Ed. by O. Dunkelman, M. J. J. Jr., and C. O'Flynn. Vol. 12804. LNCS. Springer, 2020, pp. 34–65.

[26] 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.
[27] 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.
[28] J. Nick. *Insecure Shortcuts in MuSig*. 2019. URL: https://medium.com/blockstream/insecure-shortcuts-in-musig-2ad0d38a97da.
[29] J. Nick, T. Ruffing, and Y. Seurin. "MuSig2: Simple Two-Round Schnorr Multi-signatures". In: *CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*. Ed. by T. Malkin and C. Peikert. Vol. 12825. LNCS. Springer, 2021, pp. 189–221.
[30] 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.
[31] D. Pointcheval and J. Stern. "Security Arguments for Digital Signatures and Blind Signatures". In: *J. Cryptol.* 13.3 (2000), pp. 361–396.
[32] D. Pointcheval and J. Stern. "Security Proofs for Signature Schemes". In: *EUROCRYPT 1996, Saragossa, Spain, May 12-16, 1996, Proceedings*. Ed. by U. M. Maurer. Vol. 1070. LNCS. Springer, 1996, pp. 387–398.
[33] 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, Proceedings*. Ed. by M. Naor. Vol. 4515. LNCS. Springer, 2007, pp. 228–245.
[34] C. Schnorr. "Efficient Signature Generation by Smart Cards". In: *J. Cryptol.* 4.3 (1991), pp. 161–174.
[35] Y. Seurin. "On the Exact Security of Schnorr-Type Signatures in the Random Oracle Model". In: *EUROCRYPT 2012, Cambridge, UK, April 15-19, 2012. Proceedings*. Ed. by D. Pointcheval and T. Johansson. Vol. 7237. LNCS. Springer, 2012, pp. 554–571.
[36] A. Shamir. "How to Share a Secret". In: *Commun. ACM* 22.11 (1979), pp. 612–613.
[37] 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.
[38] M. Zochowski. *Benchmarking Hash and Signature Algorithms*. 2019. URL: https://medium.com/logos-network/benchmarking-hash-and-signature-algorithms-6079735ce05.

A Background on Knowledge of Exponent Assumptions

Here we frame our Schnorr knowledge of exponent assumption 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 [10]. In arguing the security of a practical CCA encryption scheme using assumptions that more closely resemble real restrictions on the adversary, Damgard [11] introduced the knowledge of exponent assumption (KoE). KoE says that for every algorithm $\mathcal{A}$ given a generator g and random power $X = g^x$, if $\mathcal{A}$ outputs (A, B) such that $B = A^x$, then there exists an extractor algorithm $\mathcal{E}$ 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.

MAIN $\mathsf{Game}^{\mathsf{koe}}_{\mathcal{A},\mathcal{E}}(\lambda)$

$\mathcal{G} \leftarrow \mathsf{GroupGen}(1^\lambda)$
$x \leftarrow\$ \mathbb{F}; X \leftarrow g^x$
$(A, B) \leftarrow\$ \mathcal{A}(\mathcal{G}, X)$
$\alpha \leftarrow \mathcal{E}(\mathsf{trans}_{\mathcal{A}})$
// $\mathsf{trans}_{\mathcal{A}}$ is the transcript of $\mathcal{A}$
if $B = A^x \wedge g^\alpha \neq A$ return 1
// $\mathcal{A}$ wins if $(A, B) = (g^\beta, X^\beta)$ but $\beta \neq \alpha$
else return 0

Fig. 5. The knowledge of exponent game.

Definition 3 (The Knowledge of Exponent Assumption). *Let* $\mathsf{GroupGen}$ *be a group generator that outputs* $\mathcal{G} = (\mathbb{G}, p, g)$. *Let* $\mathsf{Adv}^{\mathsf{koe}}_{\mathcal{A},\mathcal{E}}(\lambda) = \Pr[\mathsf{Game}^{\mathsf{koe}}_{\mathcal{A},\mathcal{E}}(\lambda) = 1]$, *where* $\mathsf{Game}^{\mathsf{koe}}_{\mathcal{A},\mathcal{E}}(\lambda)$ *is defined in Figure 5. The knowledge of exponent assumption holds with respect to* $\mathcal{G}$ *if for all PPT adversaries* $\mathcal{A}$, *there exists a PPT extractor* $\mathcal{E}$ *and a negligible function* ν *such that* $\mathsf{Adv}^{\mathsf{koe}}_{\mathcal{A},\mathcal{E}}(\lambda) < \nu(\lambda)$.

Knowledge of exponent assumptions have subsequently been generalized, with Abdolmaleki et al. [1] extending them to bilinear groups and Groth [24] 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 [23]. One extreme variation is the algebraic group model, which suggests that all natural KoE assumptions hold but that security proofs should still reduce to computational assumptions.

We now provide a proof of Theorem 1, which states that our $\mathsf{schnorr\text{-}koe}$ assumption (Fig. 2) is implied by the discrete logarithm assumption in the algebraic group model.

Proof. ($\mathsf{dl} \Rightarrow \mathsf{schnorr\text{-}koe}$) The algebraic adversary $\mathcal{A}$ takes as input the group description $\mathcal{G} = (\mathbb{G}, p, g)$. Let $Q = \{(X_1, R_1, z_1), \ldots, (X_{q_s}, R_{q_s}, z_{q_s})\}$ denote the responses made by the signing oracle $\mathcal{O}^{\mathsf{schnorr\text{-}pop}}$ in $\mathsf{Game}^{\mathsf{schnorr\text{-}koe}}_{\mathcal{A},\mathcal{E}}$. If $\mathcal{A}$ returns a verifying $(X^*, R^*, z^*) \notin Q$, it also outputs an algebraic representation of X^* and R^*. Let $\mathcal{E}$ be the extractor that returns x^*. We argue that there exists a reduction $\mathcal{B}$ such that whenever $\mathcal{E}$ does not succeed, i.e., $X^* \neq g^{x^*}$, then $\mathcal{B}$ returns the solution to a discrete logarithm instance. $\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{schnorr\text{-}pop}}$ and $\mathsf{H}_{\mathsf{pop}}$, which $\mathcal{B}$ may program. $\mathcal{B}$ sees $\mathcal{A}$'s algebraic representations but does not rewind the adversary.

DL Input. $\mathcal{B}$ takes as input the discrete logarithm challenge X and aims to output x such that $X = g^x$.

Hash Queries. When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{pop}}$ 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 $\bar{c}$.

$\mathcal{O}^{\mathsf{schnorr\text{-}pop}}$ Queries. When $\mathcal{A}$ queries $\mathcal{O}^{\mathsf{schnorr\text{-}pop}}$ for the j^{th} time, $\mathcal{B}$ samples $a_j, c_j, z_j \leftarrow\!\$\, \mathbb{Z}_p$ and sets $X_j \leftarrow X^{a_j}$ and $R_j \leftarrow g^{z_j} X_j^{-c_j}$. $\mathcal{B}$ appends (X_j, X_j, R_j, c_j) to $\mathsf{Q}_{\mathsf{reg}}$, (X_j, R_j, z_j) to Q, and returns (X_j, R_j, z_j).

Extracting the Discrete Logarithm Solution. $\mathcal{B}$ initializes the sets $Q, \mathsf{Q}_{\mathsf{reg}}$ to the empty set and runs $\mathcal{A}(\mathcal{G})$. Suppose $\mathcal{A}$ terminates with (X^*, R^*, z^*) and that $c^* = \mathsf{H}_{\mathsf{pop}}(X^*, X^*, R^*)$. If $\mathcal{A}$ succeeds, then $c^* \neq c_j$ for $1 \leq j \leq q_s$ and $R^* X^{c^*} = g^{z^*}$. $\mathcal{A}$ also outputs a representation

$$(\alpha_0, \gamma_0, \alpha_1, \beta_1, \gamma_1, \delta_1, \ldots, \alpha_{q_s}, \beta_{q_s}, \gamma_{q_s}, \delta_{q_s})$$
$$X^* = g^{\alpha_0} \prod_{j=1}^{q_s} X_j^{\alpha_j} R_j^{\beta_j} \quad R^* = g^{\gamma_0} \prod_{j=1}^{q_s} X_j^{\gamma_j} R_j^{\delta_j}$$

Then $\mathcal{B}$ computes

$$f(W) = (\alpha_0 + \sum_{j=1}^{q_s} \beta_j z_j) + \sum_{j=1}^{q_s} a_j(\alpha_j - c_j\beta_j)W = f_0 + f_1 W$$
$$h(W) = (\gamma_0 + \sum_{j=1}^{q_s} \delta_j z_j) + \sum_{j=1}^{q_s} a_j(\gamma_j - c_j\delta_j)W = h_0 + h_1 W$$

where W is some indeterminate value, $X^* = g^{f(x)}$, and $R^* = g^{h(x)}$.

Now because $R^*(X^*)^{c^*} = g^{z^*}$, we have that $z^* = h(x) + c^* f(x)$ and

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

Thus, $\mathcal{B}$ returns x successfully as long as $h_1 + c^* f_1 \neq 0$. The probability that $h_1 + c^* f_1 = 0$ for h_1, f_1 not both 0 and for $(R^*, X^*, c^*) \notin Q$ is $1/p$. Since the adversary can make no more than q_H queries to $\mathsf{H}_{\mathsf{pop}}$, the probability that this holds for h_1, f_1 not both zero is bounded by q_H/p.

If $h_1 = f_1 = 0$, then $f(x) = f_0 = (\alpha_0 + \sum_{j=1}^{q_s} \beta_j z_j)$, which is exactly the output of the extractor. This completes the proof.

B Proof of the Schnorr Computational Assumption

We now provide a proof of Theorem 2, which states that our schnorr assumption (Fig. 3) is implied by the discrete logarithm assumption in the random oracle model.

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

$$\mathsf{Adv}^{\mathsf{schnorr}}_{\mathcal{A}}(\lambda) \leq q_H \mathsf{Adv}^{\mathsf{dl}}_{\mathcal{B}}(\lambda)$$

The reduction $\mathcal{B}$ runs $\mathcal{A}$ two times in total. On the second iteration, $\mathcal{B}$ programs $\mathsf{H}_{\mathsf{schnorr}}$ to output a different random value on a single point so that it can extract a discrete logarithm from $\mathcal{A}$'s outputs. $\mathcal{B}$ perfectly simulates $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}$. However, $\mathcal{B}$ can only extract a discrete logarithm if $\mathcal{A}$'s output (m^*, R^*, z^*) at the end of each iteration includes the same nonce R^*. This happens with probability at least equal to $1/q_H$, resulting in a tightness loss of q_H.

$\mathcal{B}$ is responsible for simulating oracle responses for queries to $\mathcal{O}^{\mathsf{schnorr}}$ as well as $\mathsf{H}_{\mathsf{schnorr}}$. Let Q be the set of $\mathcal{O}^{\mathsf{schnorr}}$ queries as in $\mathsf{Game}^{\mathsf{schnorr}}_{\mathcal{A}}$. Let Q_{schnorr} be the set of $\mathsf{H}_{\mathsf{schnorr}}$ queries and responses.

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

Simulating Hash Queries. When $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{schnorr}}$ on (X, m, R), $\mathcal{B}$ checks whether $(X, m, R, c) \in Q_{\mathsf{schnorr}}$ and, if so, returns c. Else, $\mathcal{B}$ samples $c \leftarrow\!\$\, \mathbb{Z}_p$, appends (X, m, R, c) to Q_{schnorr}, 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 $R_i \leftarrow g^{z_i} X^{-c_i}$. $\mathcal{B}$ appends (X_i, m_i, R_i, c_i) to Q_{schnorr} and m_i to Q and returns (R_i, z_i).

Extracting the Discrete Logarithm from the Adversary. $\mathcal{B}$ initializes Q and Q_{schnorr} to the empty set. It then runs $\mathcal{A}(X; r)$ on the challenge X with randomness r.

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

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 $x = \frac{z^* - z'}{c^* - c'}$ such that $X = g^x$. If $(m', R') \neq (m^*, R^*)$, then $\mathcal{B}$ must abort. The probability that $\mathcal{B}$ aborts is strictly less than $1/q_H$ because: (1) $\mathcal{A}$ makes no more than q_H queries to $\mathsf{H}_{\mathsf{schnorr}}$ in total; and (2) the statistical probability of $\mathcal{A}$ succeeding if it did not query $\mathsf{H}_{\mathsf{schnorr}}$ on (X, m^*, R^*) or (X, m', R') is less than $\frac{2}{p} < \frac{1}{q_H}$. Thus, the probability that $\mathcal{B}$ extracts x such that $X = g^x$ is at least $1/q_H$. This completes the proof.

C Background on the Two-Nonce Fix

In Section 4.3, we introduce the binonce Schnorr computational assumption, which we later use to prove the security of SpeedyMuSig and FROST. We now expand on the use of two nonces in SpeedyMuSig and FROST.

Why two nonces are necessary. There is a danger when reducing two-round Schnorr multisignatures secure 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_s (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.[13], many previous security proofs [3, 26, 37, 27] 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. designed a concurrent ROS attack against many schemes with broken security proofs [7]. 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 $R_s S_s^a$, where a 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 at the point of choosing their own nonce. Preventing the forger from cancelling a requires two nonces; otherwise, the forger could simply incorporate a^{-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.[8]

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 Figures for Definitions and Constructions

Here we provide figures for the multisignature EUF-CMA security game as well as the multi- and threshold signature constructions.

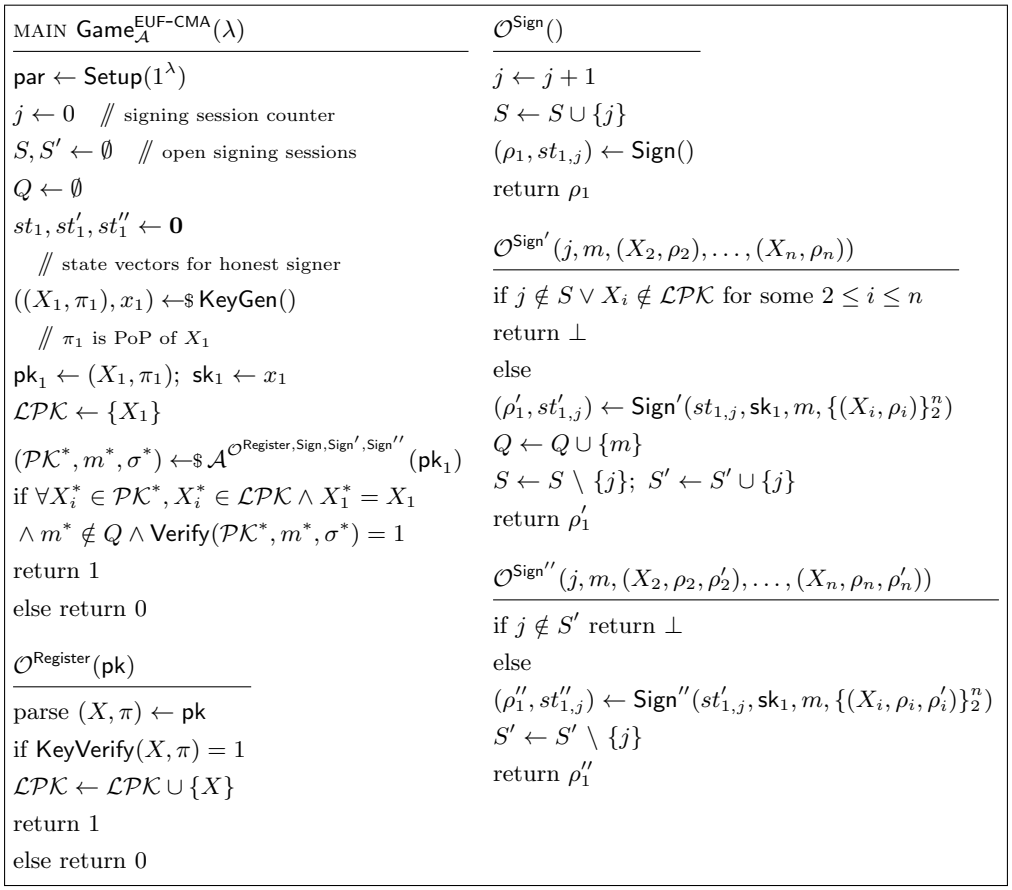

Fig. 6. The EUF-CMA security game for a multisignature scheme with proofs of possession. The public parameters par are implicitly given as input to all algorithms, and ρ represents messages defined within the construction. Note that the winning condition cannot be $(\mathcal{PK}^*, m^*) \notin Q$ because $\mathcal{A}$ can find $\mathcal{PK}^* \neq \mathcal{PK}$ such that $\tilde{X}^* = \tilde{X}$.

[8] https://github.com/mmaller/multi_and_threshold_signature_reductions

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

$\mathcal{LPK} \leftarrow \emptyset$ // registered public keys
$(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
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)

$\mathsf{Sign}''(st'_1, \mathsf{sk}_1, m, (X_2, \rho_2, \rho'_2), \ldots, (X_n, \rho_n, \rho'_n))$

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

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

$\mathcal{LPK} \leftarrow \emptyset$ // registered public keys
$(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
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 $\perp$
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. 8. The two-round SpeedyMuSig multisignature scheme with proofs of possession. The public parameters par are implicitly given as input to all algorithms.

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

1. Each party P_i chooses a random polynomial $f_i(Z)$ over $\mathbb{F}$ 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 Schnorr signature on $X_{i,0}$ as follows. They sample $\bar{r}_i \leftarrow\$\, \mathbb{F}$ 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 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. The output is the joint public key $\tilde{X} = \prod_{i=1}^{n} X_{i,0}$.

Fig. 9. PedPoP: Pedersen's distributed key generation protocol with proofs of possession.

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

$(\mathbb{G}, p, g) \leftarrow \mathsf{GroupGen}(1^\lambda)$
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 $\perp$
else
$a \leftarrow \mathsf{H}_{\mathsf{non}}(\tilde{X}, m, \{(\mathsf{id}_i, R_i, S_i)\}_{i \in \mathcal{S}})$
$\tilde{R} \leftarrow \prod_{i \in \mathcal{S}} R_i S_i^a$
$c \leftarrow \mathsf{H}_{\mathsf{sig}}(\tilde{X}, m, \tilde{R})$
$z_k \leftarrow r_k + a s_k + c \lambda_k x_k$
// 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 FROST threshold signature scheme. The public parameters par are implicitly given as input to all algorithms. FROST 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.