On the Classic Protocol for MPC Schnorr Signatures

or Classic Schnorr checks all the boxes

Nikolaos Makriyannis*

Abstract

In this paper, we prove that the classic three-round protocol for MPC Schnorr Signatures [20, 23, 26] is fully-adaptive UC-secure. Furthermore, we show that a simple variant of the Classic protocol achieves *tight* security, i.e. the security of the resulting, modified, protocol tightly reduces to the security of the underlying non-MPC scheme.

*Fireblocks. Email: nikos@fireblocks.com

Contents

1 Introduction

The Schnorr signature scheme (Schnorr [25]) is the simplest discrete-log based signature scheme. In recent years, along with its NIST-sponsored competitor, (EC)DSA, Schnorr signatures have received a lot of attention from industry and academia alike, mostly due to their new-found popularity in the Blockchain space. In this paper, we focus on Schnorr signatures in the context of *Multi-Party Computation Signing* (MPC Signing).

MPC Schnorr. MPC protocols for Schnorr are truly abundant in the literature.[1] The earliest protocols [20, 23, 26] are more than 20 years old. For the signing operation, all early protocols follow the same simple three-round template by which (1) the signatories reach consensus on a common random datum, i.e. the nonce, and (2) they locally calculate and release their respective signature shares. Interestingly, thanks to the inherent "MPC friendliness" of Schnorr,[2] the earliest protocols already achieve very good performance, i.e. they are almost as efficient as the underlying non-MPC scheme.

Recent protocols improve over the above either by reducing the number of rounds from three to two, or by realizing a deterministic variant of the signing operation where the nonce is pseudorandom. However, the recent protocols either sacrifice on efficiency, because, compared to the underlying scheme, the signing process is much more expensive, or, they sacrifice on so-called *conservative design*, because the protocols themselves are complex or the underlying cryptographic assumptions are new, interactive, non-falsifiable, or some combination thereof.

Furthermore, all previous works (old and new) impose the following restrictions on the security model: either the adversary is *static*, i.e. the adversary corrupts parties statically at the beginning of the protocol, or the adversarial model is *standalone*, i.e. the adversary is assumed to operate within the confines of the protocol environment and the protocol does not provide security guarantees when it is composed with other cryptographic components. As far as we know, no protocol achieves the highest level of security, i.e. fully adaptive security, with composability.

Motivation. In the spirit of Lindell [18], we strive to design and/or identify an MPC-Schnorr protocol that strikes the best balance between efficiency, security and conservative design for many applications of interest. Namely, we are motivated by applications to the "threshold flavour" of the MPC-signing paradigm, i.e. where the signatories agree on a common public key ahead of time. So, for this purpose, we formulate the following desiderata for our optimal protocol.

> *Efficiency.* The protocol is as costly as standard Schnorr and it supports concurrent signings.
>
> *Security.* The protocol is composable (e.g. in the UC framework) with full adaptive security.
>
> *Conservative Design.* The protocol is simple and the security reduces to standard Schnorr.

Our Results. We revisit the basic three-round signing protocol for Schnorr (dubbed *Classic S.* henceforth, c.f. Figure 1). Our first contribution is showing that *Classic S.* essentially "checks all the boxes" in terms of efficiency, security, and conservative design. As far as we know, no protocol was previously known to satisfy all our desiderata. Specifically, we show that *Classic S.* achieves fully adaptive UC-security where the quality of the security depends on the reconstruction threshold (ie. our analysis yields a security loss that is proportional to n^t, where t and n denote the reconstruction threshold and the total number of parties, respectively.)

Second, we show that a simple variant of the protocol (dubbed *Zero S.* henceforth) achieves fully adaptive UC-security and its security tightly reduces to the security of the underlying non-MPC scheme (i.e. the standard Schnorr signature scheme). We achieve this by modifying the protocol such that the parties' secret state is statistically hidden at all moments of the execution (we further discuss *Zero S.* in Section 1.1.1).

On a technical level, we prove that both protocols UC-realize the ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$ of Canetti et al. [9] against adaptive adversaries, where, at a high-level, $\mathcal{F}_{\mathsf{tsig}}$ captures the "essence" of a secure threshold-signatures scheme. Our results hold in the generalized UC model with a *strict* global random

[1] Non-exhaustively, we mention [1, 2, 10, 13, 16, 18, 19, 20, 21, 22, 23, 26].

[2] Viewed as an arithmetic circuit, the functionality for Schnorr does not contain any multiplication gates, which makes it far easier/cheaper to realize in a distributed way than, say, (EC)DSA.

oracle (RO).[3] We conclude this section with the following informal theorem-statements and we turn to our techniques.

Theorem (Informal)**.** *The following hold under the discrete log assumtion*

1. *If $n^t \in \mathsf{poly}$, it holds that Classic S. UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ in the* strict *global RO.*
2. *For any $n, t \in \mathbb{N}$, it holds that Zero S. UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ in the* strict *global RO.*

(letting n, t denote the total number of parties and the reconstruction threshold respectively)

FIGURE 1 (*Classic S.*)

Parameters. Group-generator-order tuple $(\mathbb{G}, g, q)$, hash function $\mathcal{H}$.

Key Generation.

1. Sample $x_i \leftarrow [q]$, set $X_i = g^{x_i}$ and broadcast $V_i = \mathcal{H}(\mathcal{P}_i, X_i)$.
2. When obtaining $(V_j)_{j \neq i}$, broadcast X_i.

   When obtaining $(X_j)_{j \neq i}$, verify $(V_j)_{j \neq i}$ and output $(X_1, \ldots, X_n; x_i)$.

Signing. On input $\mathrm{msg} \in \{0,1\}^*$, do:

1. Sample $k_i \leftarrow [q]$, set $R_i = g^{k_i}$ and broadcast $W_i = \mathcal{H}(\mathcal{P}_i, R_i)$.
2. When obtaining $(W_j)_{j \neq i}$, broadcast R_i.

   When obtaining $(R_j)_{j \neq i}$, verify $(W_j)_{j \neq i}$.
3. Calculate $R = \prod_{\ell=1}^{n} R_\ell$ and $e = \mathcal{H}(X, R, \mathrm{msg})$ and broadcast $\sigma_i = k_i + e x_i \mod q$.

   When obtaining $(\sigma_j)_{j \neq i}$, verify $(g^{\sigma_j} = R_j \cdot X_j^e)_{j \neq i}$ and output $(R, \sigma = \sum_{\ell=1}^{n} \sigma_\ell)$.

Figure 1: n-out-of-n Classic Schnorr from $\mathcal{P}_i$'s perspective. $\mathbb{G}$ denotes a prime-order group of size q generated by $g \in \mathbb{G}$ and $\mathcal{H}$ denotes the hash function. We recall that Schnorr signatures verify as follows: for public key $X \in \mathbb{G}$ and message $\mathrm{msg} \in \{0,1\}^*$, accept signature $(R, \sigma) \in \mathbb{G} \times [q]$ iff $g^\sigma = R \cdot X^e$ where $e = \mathcal{H}(X, R, \mathrm{msg})$. In the technical sections, we consider the t-out-of-n threshold setting, where any set of $t \leq n$ parties may sign. To avoid clutter, we have suppressed the use of identifiers $(sid, pid, ssid, \ldots)$ in the above.

A recent work on *Classic S.* Recently, Crites et al. [11] (to appear in CRYPTO'23) investigate the adaptive (non-composable) security of *Classic S.* when using stronger assumptions (namely the one-more discrete log assumption – OMDL), and when limiting the adversarial model to algebraic attacks (where the adversary provides the algebraic representation of any group element it outputs). They find that, assuming OMDL, *Classic S.* is secure as long as $2\alpha + \sigma < t$, where α and σ denote the number of adaptive and static corruptions of the adversary, and, in the algebraic model, *Classic S.* has full adaptive security (under OMDL).

We note that (while beyond the primary goals of the present paper) we can combine the result of [11] with our proof technique to show that partially-adaptive UC-security of *Classic S.* (under OMDL) and fully-adaptive UC-security of *Classic S.*(in the *global* algebraic group model). We provide an informal discussion of this claim in Appendix A and we intend to provide more details in the next iteration of our paper.

1.1 Our Techniques

We give an overview of our techniques by discussing *Zero S.* and presenting our security proof technique.

1.1.1 *Zero S.*

Intuitively, *Classic S.* admits a security loss because, baring some powerful resource (e.g. a dlog oracle), it is not possible for the simulator to "explain", i.e. calculate the secret state, for all not-yet-corrupted parties in the protocol (if it could, then it could also derive the master secret key). The main reason for this is that the

[3] We recall that in this model the simulator is *not* allowed to observe or program the oracle.

transcript of the protocol is binding (specifically the X_i's from Figure 1), meaning that for any given party in the protocol, there exists a single secret state that aligns with the information the adversary has acquired from observing the protocol.

Pedersen Commitments. To overcome the above, we use Pedersen commitments instead of the "naive" commitments $X_i = g^{x_i}$. (We recall that Pedersen commtments have the form $C_i = g^{x_i} h^{\mu_i}$ for random μ_i where $h \in \mathbb{G}$ denote an arbitrary element with unknown discrete log relation to g). This way, during the signing phase, it suffices for the parties to calculate additive shares of 0 on top the classic three-round signature process (the purpose of the additive shares of zero is to mask the signature share of each party because it leaks g^{x_i} which we want to avoid).

However, generating the C_i's in an oblivious way is not straightforward and we devise a novel protocol for this purpose. Our key idea is to blind the standard t-out-of-n verifiable secret sharing of the key x in the exponent of g with a $(n+1)$-out-of-$(n+1)$ secret sharing of a random value in the exponent of h. In more detail:

1. Each $\mathcal{P}_i$ samples two polynomials $\alpha_i(z) = \sum_{j=0}^{t-1} \alpha_{i,j} z^j$ and $\beta_i(z) = \sum_{j=0}^{n} \alpha_{i,j} z^j$ of degree $t-1$ and n respectively and they send $\{B_{i,k}\}_{k=0}^{n}$ to the other parties where

$$B_{i,k} = \begin{cases} g^{\alpha_{i,k}} h^{\beta_{i,k}} & \text{if } k \leq t-1 \\ h^{\beta_{i,k}} & \text{if } k \geq t \end{cases}$$

   (The above is enforced using dlog-style online extractable zero-knowledge proofs.)

2. The parties exchange secret data in typical verifiable-secret-sharing fashion, i.e. Player $\mathcal{P}_i$ sends $\alpha_i(j)$ and $\beta_i(j)$ to $\mathcal{P}_j$ and in the end each $\mathcal{P}_i$ holds a decommitment (x_i, μ_i) of their "blind" public share $C_i = \prod_{j=1}^{n} \prod_{k=0}^{n} B_{j,k}^{i^k}$.

3. To calculate the public key, the parties reveal (in ZK) $\hat{A}_i = g^{\alpha_{i,0}+\delta_i}$ where $\delta_1, \ldots, \delta_n$ are additive shares of zero and $\sum_i \alpha_{i,0} = x$. The key-generation phase concludes with the parties outputting $X = \prod_{i=1}^{n} \hat{A}_i$ and their respective secret states.

Remark 1.1. We note that the more natural (but naive) approach of restricting the degree of β_i to $t-1$ does not seem to work because the transcript of the protocol still contains many binding relations (there are easy-to-find relations between $t+1$ public values, because any t values determine the rest).

1.1.2 Proof Technique

The present section assumes some familiarity with the UC framework.

Ideal Threshold-Signatures Functionality. Our main security claim is that *Classic/Zero S.* UC-realizes the ideal ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$ from [9].[4] As mentioned earlier, the purpose of $\mathcal{F}_{\mathsf{tsig}}$ is to capture the "essence" of a secure threshold-signatures scheme where, letting $\boldsymbol{P}$ denote the set of signatories, $\mathcal{F}_{\mathsf{tsig}}$ provides the following functionality/security:

1. *Key Generation.* Upon activation, the functionality requests a verification algorithm $\mathcal{V}$ from the ideal adversary; for us, $\mathcal{V}$ is simply the verification algorithm for Schnorr that depends on the public key.

2. *Signing.* When obtaining input msg $\in \{0,1\}^*$ from a subset $\boldsymbol{Q} \subseteq \boldsymbol{P}$ of size at least t, the functionality records the message msg as "signed".

3. *Verification.* When prompted on a message msg and a signature σ for verification, the functionality returns $\mathcal{V}(\text{msg}, \sigma) \in \{\texttt{true}, \texttt{false}\}$ if it has record of this message. If there is no record of msg, the functionality returns `false` regardless of $\mathcal{V}(\cdot)$.

[4] The results of [9] were published in [8] (CCS'20) as part of a combined work with Gennaro and Goldfeder [14].

Notice that the functionality will accept a pair (msg, σ) only if msg was authorized by a suitable a set of parties in Item 2 above and $\mathcal{V}(\text{msg}, \sigma) = \texttt{true}$. In any other case, i.e. either msg was not authorized or σ does not conform to $\mathcal{V}$, the functionality will reject the pair. Furthermore, we stress that $\mathcal{F}_{\mathsf{tsig}}$ does not hold any internal secrets so it does *not* know the secret key associated with the verification algorithm $\mathcal{V}$ (because $\mathcal{V}$ was supplied from the outside by the ideal adversary).

Simulatability & Unforgeability imply UC Security. We modify the key technique from [9] to show that *Classic/Zero S.* UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ by way of reduction to the assumed unforgeability of the underlying non-threshold scheme. This technique was recently generalized in [3] for general threshold-signatures protocols as follows: starting from a signature scheme Sig and a threshold protocol Σ for computing Sig, [3] show that if (1) Sig is unforgeable according to the usual game-based definition and (2) Σ can be standalone-simulated[5] using an oracle to Sig (the same oracle from the unforgeability game), then Σ UC-realizes the ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$.

In this paper, we model the internal hash function $\mathcal{H}$ of Schnorr as a *random oracle* (it is an interesting open problem to see if this can be avoided). In doing so, however, the theorem from [3] is no longer applicable.[6,7] To overcome this issue, we generalize the result of [3] to so-called *oracle-aided* signatures where the signing and/or verification process of Sig depends on message-dependent query-answer pairs to some oracle $\mathcal{O}$ (c.f. Section 2.3, Theorem 2.6).

As a corollary, we find that since Schnorr signatures are unforgeable in the random oracle model (assuming discrete log [24]), and *Classic/Zero S.* can be simulated against adaptive adversaries using a suitable signature oracle, it follows that *Classic/Zero S.* UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ against adaptive adversaries under the discrete log assumption.

No ZK? No Problem! A surprising aspect of *Classic S.* (and the signing phase of *Zero S.*) is the total absence of ZK proofs, especially given the advertised security. To explain in one sentence, *Classic S.* does not contain ZK proofs because the security analysis does not require extraction of the adversary's secrets. To elaborate further, typically the simulator extracts the adversary's secrets in order to calculate the honest party's simulated messages, e.g. for a Schnorr signature (R, σ) with $e = \mathcal{H}(X, R, \text{msg})$, the simulator calculates the adversary's share $(R_{\mathcal{A}}, \sigma_{\mathcal{A}}) = (g^{k_{\mathcal{A}}}, k_{\mathcal{A}} + e x_{\mathcal{A}})$ using the extracted secrets $k_{\mathcal{A}}$ and $x_{\mathcal{A}}$, and then it sets the (simulated) honest party's share as $(\hat{R}, \hat{\sigma}) = (R \cdot R_{\mathcal{A}}^{-1}, \sigma - \sigma_{\mathcal{A}})$.

In this work, we completely circumvent extraction (and any penalties it may induce) because we simulate the honest party's share directly by suitably programming the random oracle, i.e. the simulator is instructed to sample $\hat{\sigma}$ and e at random and set $\hat{R} = g^{\hat{\sigma}} \cdot \hat{X}^{-e}$, where $\hat{X}$ is the public-key share of the honest party. So, by programming the oracle accordingly, i.e. return e to the adversary when queried on (X, R, msg) for $R = \hat{R} \cdot R_{\mathcal{A}}$, the honest party's simulated share $\hat{\sigma}$ is identically distributed with the real one.[8]

Remark 1.2 (Conservative Design & Random Oracles). It may seem odd to use random oracles in the context of conservative design and the "cryptography purist" will proclaim that random oracles are not compatible with conservative design. To counter this obvious criticism, we offer the following arguments. First, random oracles are ubiquitous in practical real-world cryptography, e.g. there is no NIZK deployed in the real world that does not assume a random oracle, as far as we know. Thus, many Schnorr protocols in the wild implicitly assume random oracles (because many Schnorr protocols use NIZKs). Second, conservative design is also concerned with the simplicity of the protocol itself, i.e. a simple protocol in an idealized model may compare favorably to a complicated protocol under minimal assumptions. In this regard, *Classic S.* has no match. Finally, our security analysis uses the random oracle in the same fashion as the seminal work of Pointcheval and Stern [24], or, as mentioned in Footnote 8, the standard ZKPoK for discrete log. So, our use of the oracle is not innovative or sophisticated, and thus the principle of conservative design is not violated.

[5] i.e., the simulator has access to the adversary's code and it can provide simulated answers to random-oracle queries.

[6] Because the simulator may inadvertently create a non-suitable forgery when tinkering with the oracle in the reduction.

[7] [9] and [3] use a random oracle to show that the simulation is indistinguishable and they make no assumptions on the internal hash function of ECDSA.

[8] The astute reader will notice that we are simply running the simulator for the standard ZK Proof of Knowledge (ZKPoK) for Discrete Log.

2 Preliminaries

Notation. Throughout the paper $(\mathbb{G}, g, q)$ will denote the group-generator-order tuple for Schnorr. It is assumed that the description of $\mathbb{G}$ is efficiently generated by an algorithm group in input 1^κ. We let $\mathbb{Z}, \mathbb{N}$ denote the set of integer and natural numbers, respectively. We use sans-serif letters ($\mathsf{enc}, \mathsf{dec}, \ldots$) or calligraphic ($\mathcal{S}, \mathcal{A}, \ldots$) to denote algorithms. Secret values are always denoted with lower case letters ($x, \alpha, \ldots$) and public values are *usually* denoted with upper case letters ($A, X, \ldots$). Furthermore, for a tuple of both public and secret values, e.g. an RSA modulus and its factors (N, p, q), we use a semi-colon to differentiate public from secret values (so we write $(N; p, q)$ instead of (N, p, q)). Bold letters $\boldsymbol{X}, \boldsymbol{s}, \ldots$ denote sets and we write $2^{\boldsymbol{X}} = \{\boldsymbol{A} \text{ s.t. } \boldsymbol{A} \subseteq \boldsymbol{X}\}$ for the power set of $\boldsymbol{X}$. Bold letters may also denote random variables.

We write $x \leftarrow \boldsymbol{E}$ for sampling x uniformly from a set $\boldsymbol{E}$, and $x \leftarrow \mathcal{A}$ or $x \leftarrow \mathsf{gen}$ for sampling x according to (probabilistic) algorithms $\mathcal{A}$ or gen. A distribution ensemble $\{\boldsymbol{v}_\kappa\}_{\kappa \in \mathbb{N}}$ is a sequence of random variables indexed by the natural numbers. We say two ensembles $\{\boldsymbol{v}_\kappa\}$ and $\{\boldsymbol{u}_\kappa\}$ are indistingushable and we write $\{\boldsymbol{v}_\kappa\} \equiv \{\boldsymbol{u}_\kappa\}$ if $\Pr[\mathcal{D}(1^\kappa, \boldsymbol{u}_\kappa) = 1] - \Pr[\mathcal{D}(1^\kappa, \boldsymbol{v}_\kappa) = 1]$ is negligible for every efficient distinguisher $\mathcal{D}$. We write $\mathrm{SD}(\boldsymbol{u}, \boldsymbol{v})$ for the statistical distance of $\boldsymbol{u}$ and $\boldsymbol{v}$. Finally, we also define oracles and oracle-aided algorithms. An oracle $\mathcal{O}$ is a (not-necessarily-efficient) Turing machine and we say that $\mathcal{A}^{\mathcal{O}}$ is an oracle-aided algorithm (OA-algorithm) for oracle $\mathcal{O}$ if it can make queries and receive answers from $\mathcal{O}$; formally the PPTM $\mathcal{A}^{(\cdot)}$ has an additional oracle tape for this purpose.

2.1 Oracle-Aided Signatures and Unforgeability

Definition 2.1 (OA-Signatures)**.** $\mathrm{Sig}^{\mathcal{O}} = (\mathsf{gen}, \mathsf{sign}, \mathsf{vrfy})$ is a tuple of OA-algorithms for oracle $\mathcal{O}$ s.t.

1. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$, where κ is the security parameter.
2. For $\mathrm{msg} \in \{0,1\}^*$, $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(\mathrm{msg})$.
3. For $\mathrm{msg}, \sigma \in \{0,1\}^*$, $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = b \in \{0,1\}$.

   Correctness. For $\sigma \leftarrow \mathsf{sign}_{\mathsf{sk}}(\mathrm{msg})$, it holds that $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$.

Existential Unforgeability. Next, we define security for OA-signature schemes. In Figure 2, we define a generic oracle for defining the security game in Figure 3 (and thus the security definition below, Definition 2.2, is parameterized by the oracle $\mathcal{G}$). Later, in Figure 5, we will define a specific oracle $\mathcal{G}^*$ for Schnorr and unforgeability will be defined with respect to $\mathcal{G}^*$.

FIGURE 2 (Augmented Signature Oracle $\mathcal{G}$)

Parameters. OA-Signature scheme $\mathrm{Sig}^{\mathcal{O}}$ and randomized OA-functionality $\mathcal{F}^{\mathcal{O}}$.

Operation.

1. On input $(\mathsf{gen}, 1^\kappa)$, generate a key pair $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{gen}(1^\kappa)$, initialize $\mathtt{state} = (\mathsf{sk}, \mathsf{pk})$, and return pk.
   Ignore future calls to gen.
2. On input $(\mathcal{F}^{\mathcal{O}}, x)$, sample $r \leftarrow \$$ and return $\tau = \mathcal{F}^{\mathcal{O}}(x, \mathtt{state}; r)$.
   Update $\mathtt{state} := \mathtt{state} \cup \{(x, \tau; r)\}$.

Figure 2: Augmented Signature Oracle $\mathcal{G}$

Definition 2.2 ($\mathcal{G}$-Existential Unforgeability.)**.** We say that $\mathrm{Sig}^{\mathcal{O}}$ satisfies $\mathcal{G}$-Existential unforgeability if there exists $\nu \in \mathsf{negl}(\kappa)$ such that for all $\mathcal{A}$, it holds that $\Pr[\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa) = 1] \leq \nu(\kappa)$, where $\mathcal{G}\text{-}\mathsf{EU}(\cdot)$ denotes the security game from Figure 3.

2.2 MPC and Universal Composability

We use the simplified variant of the UC framework (which is sufficient for our purposes because the identities of all parties are assumed to be fixed in advance). In this section we provide a quick reminder of the framework.

FIGURE 3 ($\mathcal{G}$-Existential Unforgeability Experiment $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$)

1. Call $\mathcal{G}$ on $(\mathsf{gen}, 1^\kappa)$ and hand pk to $\mathcal{A}$.
2. The adversary $\mathcal{A}$ makes $n(\kappa)$ adaptive calls to $\mathcal{G}$ and $\mathcal{O}$.
3. $\mathcal{A}$ outputs (m, σ) given its view (randomness and query-answer pairs to $\mathcal{G}$ and $\mathcal{O}$)

- **Output:** $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa) = 1$ if $\mathsf{vrfy}_{\mathsf{pk}}(m, \sigma) = 1$ and m was not queried by $\mathcal{A}$ when calling $\mathcal{G}$.

Figure 3: $\mathcal{G}$-Existential Unforgeability Experiment $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$

The model for n-party protocol Π. For the purpose of modeling the protocols in this work, we consider a system that consists of the following $n + 2$ *machines,* where each machine is a computing element (say, an interactive Turing machine) with a specified program and and identity. First, we have n machines with program Π and identities $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Next, we have a machine $\mathcal{A}$ representing the adversary an a machine $\mathcal{Z}$ representing the environment. All machines are initialized on a security parameter κ and are polynomial in κ. The environment $\mathcal{Z}$ is activated first, with an external input z. $\mathcal{Z}$ activates the parties, chooses their input and reads their output. $\mathcal{A}$ can corrupt parties and instruct them to leak information to $\mathcal{A}$ and to perform arbitrary instructions. $\mathcal{Z}$ and $\mathcal{A}$ communicate freely throughout the computation. The real process terminates when the environment terminates. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)$ denote the environment's output in the above process.

In this work we assume for simplicity that the parties are connected via an authenticated, synchronous broadcast channel. That is, the computation proceeds in rounds, and each message sent by any of of the parties at some round is made available to all parties at the next round. Formally, synchronous communication is modeled within the UC framework by way of $\mathcal{F}_{\text{syn}}$, the ideal synchronous communication functionality from [5, Section 7.3.3]. The broadcast property is modeled by having $\mathcal{F}_{\text{syn}}$ require that all messages are addressed at all parties.

Ideal Process. the ideal process is identical to the real process, with the exception that now the machines $\mathcal{P}_1, \ldots, \mathcal{P}_n$ do not run Π, Instead, they all forward all their inputs to a subroutine machine, called the *ideal functionality* $\mathcal{F}$. Functionality $\mathcal{F}$ then processes all the inputs locally and returns outputs to $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Let $\mathrm{EXEC}^{\mathcal{Z}}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ denote the environment's output in the above process.

Definition 2.3. We say that Π UC-realizes $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$ it holds that

$$\{\mathrm{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)\}_{z\in\{0,1\}^*} \equiv \{\mathrm{EXEC}^{\mathcal{Z}}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)\}_{z\in\{0,1\}^*}$$

The Adversarial Model. The adversary can corrupt parties adaptively throughout the computation. Once corrupted, the party reports all its internal state to the adversary, and from now on follows the instructions of the adversary. We also allow the adversary to *leave*, or *decorrupt* parties. A decorrupted party resumes executing the original protocol and is no longer reporting its state to the adversary. Still, the adversary knows the full internal state of the decorrupted party at the moment of decorruption. Finally, the real adversary is assumed to be *rushing*, i.e. it receives the honest parties messages before it sends messages on behalf of the corrupted parties.

Global Functionalities. It is possible to capture UC with global functionalities within the plain UC framework. Specifically, having Π UC-realize ideal functionality $\mathcal{F}$ in the presence of global functionality $\mathcal{G}$ is represented by having the protocol $[\Pi, \mathcal{G}]$ UC-realize the protocol $[\mathcal{F}, \mathcal{G}]$ within the plain UC framework. Here $[\Pi, \mathcal{G}]$ is the $n + 1$-party protocol where machines $\mathcal{P}_1, \ldots, \mathcal{P}_n$ run Π, and the remaining machine runs $\mathcal{G}$. Protocol $[\mathcal{F}, \mathcal{G}]$ is defined analogously, namely it is the $n + 2$-party protocol where the first $n + 1$ machines execute the ideal protocol for $\mathcal{F}$, and the remaining machine runs $\mathcal{G}$.

Secret Channels. We assume that the parties are connected with point-to-point secret channels. Formally, it is assumed that all pairs of parties admit a pairwise secret key for communicating secretly over the broadcast channel.

2.2.1 Threshold Signatures

The definition below is a restricted version of [3] because the protocol herein does not support *presigning* (we refer the reader to [3] for the definition of presigning).

Definition 2.4 (Threshold Signatures)**.** Let $\Sigma = (\Sigma_{\mathsf{kgen}}, \Sigma_{\mathsf{refr}}, \Sigma_{\mathsf{sign}})$ denote a protocol for parties in $\boldsymbol{P} = \{\mathcal{P}_0, \mathcal{P}_1, \ldots, \mathcal{P}_n\}$ parametrized by $\mathbb{Q} \subseteq 2^{\boldsymbol{P}}$. We say that Σ is a threshold signatures scheme for $\mathrm{Sig}^{\mathcal{O}} = (\ldots, \mathsf{vrfy})$ if it provides the following functionality.

1. Σ_{kgen} takes input 1^{κ} from $\mathcal{P}_i \in \boldsymbol{P}$ and returns (pk, s_i) to each $\mathcal{P}_i \in \boldsymbol{P}$.
2. Σ_{sign} takes input $\mathrm{msg} \in \{0,1\}^*$ and $(\mathsf{pk}, s_i, \boldsymbol{Q})$ from $\mathcal{P}_i \in \boldsymbol{Q} \in \mathbb{Q}$ and returns σ to (at least one) $\mathcal{P}_i$.

   Correctness. It holds that $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$ in an honest execution.

Sets $\boldsymbol{Q} \in \mathbb{Q}$ are called *quorums*.

A protocol Σ is said to be secure if it UC-realizes functionality $\mathcal{F}_{\mathrm{tsig}}$, defined below.

2.2.2 Ideal Threshold-Signatures Functionality

We use the ideal functionality $\mathcal{F}_{\mathsf{tsig}}$ of [9], which generalizes the non-threshold signature functionality of Canetti [6]. We briefly outline $\mathcal{F}_{\mathsf{tsig}}$ next and we refer the reader to the appendix (p. 19) for the full description.

For each signing request for a message msg, the functionality requests a signature string σ from the adversary, which is submitted from the outside, i.e. the signature string σ is not calculated internally from the ideal functionality. Once σ is submitted by the adversary, the functionality keeps record of (msg, σ). When a party submits a pair (msg', σ') for verification, the functionality simply returns *true* if it has record of that pair and *false* otherwise.

2.2.3 Global Random Oracle

We follow formalism of [4, 7] for incorporating the random oracle into the UC framework. In particular, we use the *strict global random oracle* paradigm which is the most restrictive way of defining a random oracle, defined in Figure 4.

FIGURE 4 (The Global Random Oracle Functionality $\mathcal{H}$)

Parameter: Output length h.

- On input $(\mathtt{query}, m)$ from machine $\mathcal{X}$, do:
  - If a tuple (m, a) is stored, then output $(\mathtt{answer}, a)$ to $\mathcal{X}$.
  - Else sample $a \leftarrow \{0,1\}^h$ and store (m, a).

    Output $(\mathtt{answer}, a)$ to $\mathcal{X}$.

Figure 4: The Global Random Oracle Functionality $\mathcal{H}$

2.3 Unforgeability & Simulatability imply UC Security

Write κ for the security parameter. For OA-signature scheme $\mathrm{Sig}^{\mathcal{O}}$ and associated threshold-protocol Σ, for adversary $\mathcal{A}$, write $\mathrm{Real}^{\Sigma}_{\mathcal{A}}(1^{\kappa}, z)$ for the adversary's view in an execution of Σ in the presence of an *adaptive* PPTM adversary $\mathcal{A}$ given external advice z. Without loss of generality assume that $\mathrm{Real}^{\Sigma}_{\mathcal{A}}(1^{\kappa}, z) = (\mathsf{pk}^{\Sigma}, \ldots)$, where pk^{Σ} denotes the public key resulting from the execution of Σ. Next, for an oracle-aided algorithm $\mathcal{S}$ with black-box access to $\mathcal{A}$ and oracle access to $\mathcal{G}$ and $\mathcal{O}$, write $\mathrm{Ideal}^{\mathcal{G}}_{\mathcal{S}}(1^{\kappa}, z) = (\mathsf{pk}^{\mathcal{G}}, \mathrm{Out}^{\mathcal{S}})$ for the pair of random variables consisting of the public key generated by $\mathcal{G}$ and the simulator's, $\mathcal{S}$, output given external input z.

Definition 2.5 (Simulatability)**.** Using the notation above, we say that Σ is $(\mathcal{G}, \tau)$-simulatable with $\mathcal{O}$-consistency if the following holds for every adversary $\mathcal{A}$. For every $z \in \{0,1\}^*$, there exists $\mathcal{S}$ with oracle access to $\mathcal{G}$ and $\mathcal{O}$ and black-box access to $\mathcal{A}$ such that

1. $\mathcal{G}$ is queried by $\mathcal{S}$ only on messages intended for signing as prescribed by Σ.

   (The simulator is not allowed query messages that are not intended for signing)

2. If $\mathcal{A}$ does not corrupt all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ simultaneously in any given epoch, then

   $$\{\text{Real}^{\Sigma}_{\mathcal{A}}(1^{\kappa}, z)\} \equiv \{\text{Ideal}^{\mathcal{G}}_{\mathcal{S}}(1^{\kappa}, z)\}.$$

   (The simulator is indistinguishable from the adversary's view with distinguishing advantage at most ε – and $\mathcal{S}$ may depend on ε)

3. All oracle queries for $\mathsf{vrfy}_{(\cdot)}(\text{msg})$ are consistent with $\mathcal{O}$, i.e. they are "real" oracle queries, or they are undefined by the simulation, *unless they are intended for signing as prescribed by* Σ.

   (i.e. the simulator is restricted when $\mathsf{vrfy}_{(\cdot)}(\text{msg})$ is invoked to avoid inadvertently creating "fake" forgeries for messages that the adversary attempts to forge.)

4. The running time of $\mathcal{S}$ is $\tilde{O}(\tau \cdot \mathsf{time}_{\Sigma})$ where time_{Σ} the running time of Σ.

The theorem below is a generalization of [3] for OA-signatures.

Theorem 2.6. *Let* $\text{Sig}^{\mathcal{O}}$ *denote an OA-signature scheme and let* Σ *denote a threshold protocol for* $\text{Sig}^{\mathcal{O}}$*. Let* $\mathcal{G}$ *denote an augmented signature oracle such that*

- $\text{Sig}^{\mathcal{O}}$ *is* $\mathcal{G}$*-existentially unforgeable.*
- Σ *is* $(\mathcal{G}, \tau)$*-simulatable with* $\mathcal{O}$*-consistency.*

If $\tau \in \mathsf{poly}$*, then* Σ *UC-realizes* $\mathcal{F}_{\text{tsig}}$ *in the presence of global functionality* $\mathcal{O}$*.*

Proof. Like in [3], the UC simulation is trivial; the simulator simply runs the code of the honest parties (we reiterate that UC simulation is trivial because the simulator "knows" all the honest parties' secrets – the simulator samples those secrets by itself). When interacting with the functionality in the UC simulation, the simulator does two things: (1) every time the honest parties output a signature, the simulator submits the resulting signature-string to the functionality, and, (2) depending on the environment's corruption pattern, the simulator registers parties as corrupted with the functionality.

It is not hard to see that the environment $\mathcal{Z}$ can distinguish real from ideal execution only if it can forge signatures in the protocol (i.e. in the real world) because the simulation is *perfect* otherwise. However, since Σ is $\mathcal{G}$-simulatable with $\mathcal{O}$-consistency, it follows by Definition 2.5 that the interaction between $\mathcal{Z}$ and the honest parties can be simulated using $\mathcal{G}$ and $\mathcal{O}$, and the simulation yields a "true" forgery because it is $\mathcal{O}$-consistent. In turn, this implies that $\mathcal{G}$ is useful for forging signatures of $\text{Sig}^{\mathcal{O}}$, in contradiction with the hypothesis of the theorem.

In other words, if there exists a PPTM $\mathcal{Z}_0$ that can forge signatures in the real protocol (so that the environment can distinguish between real and ideal), then, by $\mathcal{G}$-simulatability, we can construct (using the simulator from the $\mathcal{G}$-simulatability experiment) a PPTM $\mathcal{B}$ with black-box access to $\mathcal{Z}_0$ that breaks the unforgeability of the underlying non-threshold scheme $\text{Sig}^{\mathcal{O}}$, which yields a contradiction. Therefore, no such $\mathcal{Z}_0$ exists, and it holds that our UC simulator yields indeed a perfect simulation. □

2.4 Schnorr Signatures & Discrete Log

Definition 2.7 (Schnorr)**.** Let $(\mathbb{G}, g, q)$ denote the group-generator-order tuple.

Parameters: $(\mathbb{G}, q, g)$ and (random) oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{F}_q$.

1. $(X; x) \leftarrow \mathsf{gen}(\mathbb{G}, q, g)$ such that $x \leftarrow \mathbb{F}_q$ and $X = g^x$.
2. For $\text{msg} \in \boldsymbol{M}$, let $\mathsf{sign}_x(\text{msg}; k) = (R, \sigma) \in \mathbb{G} \times \mathbb{F}_q$, for $R = g^k$, $m = \mathcal{H}(X, R, \text{msg})$, $\sigma = k + mx \mod q$.

FIGURE 5 (Oracle $\mathcal{G}^*$ for Schnorr)

Parameters. Random Oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{F}_q$.

Operation.

1. On input $(\mathsf{gen}, (\mathbb{G}, g, q))$, sample $\mathsf{sk} = x \leftarrow \mathbb{F}_q$ and return $\mathsf{pk} = X = g^x$.

   Store $(\mathsf{sk}, \mathsf{pk})$ in memory and ignore future calls to gen.
2. Ignore all other prompts.

Figure 5: Oracle $\mathcal{G}^*$ for Schnorr

3. For $(R, \sigma) \in \mathbb{F}_q^2$, define $\mathsf{vrfy}_X(\mathrm{msg}, \sigma) = 1$ iff $g^\sigma = R \cdot X^m$ and $m = \mathcal{H}(X, R, \mathrm{msg})$.

Notice that Schnorr signatures are $\mathcal{G}^*$-existentially unforgeable if no adversary can forge signatures only given the public key; in particular, the adversary is *not* allowed to query the oracle for additional signatures. Thus, by the seminal result of Pointcheval and Stern (Theorem 2.9 below), Schnorr signatures are $\mathcal{G}^*$-existentially unforgeable under the discrete logarithm assumption. For completeness, we also give the definition for DLOG.

Definition 2.8. DLOG holds true if there exists a negligible function $\nu(\cdot)$ such that for every PPTM $\mathcal{A}$

$$\Pr[(\mathbb{G}, g, q) \leftarrow \mathsf{group}(1^\kappa) \wedge X \leftarrow \mathbb{G} \wedge \boldsymbol{\alpha} \leftarrow \mathcal{A}(1^\kappa, X) \wedge g^{\boldsymbol{\alpha}} = X] \leq \nu(\kappa)$$

Theorem 2.9 (Pointcheval and Stern [24])**.** *Assuming DLOG, letting $\mathcal{G}^*$ denote the oracle from Figure 5, it holds that Schnorr signatures are $\mathcal{G}^*$-existentially unforgeable.*

2.5 Pedersen Commitments and ZK-PoK

The present section is only relevant for the *Zero S.* protocol.

Definition 2.10 (Pedersen Commitments)**.** For group-generators-order tuple $(\mathbb{G}, g, h, q)$ define algorithm com that takes input $x \in \mathbb{F}_q \times \mathbb{G}$ and returns $(B; \mu)$ such that $B = h^\mu g^x$ for "randomizer" $\mu \leftarrow \mathbb{F}_q$, and we write $B = \mathsf{com}(x, \mu)$.

Definition 2.11 (Dlog PoK)**.** Fix $(\mathsf{aux}, \mathbb{G}, g, q)$ and define the following process $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$ parameterized by and $r, b \in \mathbb{N}$ and hash function $\mathcal{H} : \{0,1\} \to \{0,1\}^\kappa$. On common input (aux, B), auxiliary input aux, and $\mathcal{P}$ holding secret input μ such that $B = h^\mu$,

1. $\mathcal{P}$ samples $\delta_1, \ldots, \delta_r \leftarrow \mathbb{F}_q$ and sets $\boldsymbol{A} = (A_i)_{i=1}^r$, where $A_i = h^{\delta_i}$ for all $i \in [r]$.
2. Compute $e_1, \ldots, e_r \in \mathbb{F}_q$ s.t. $\mathcal{H}(\mathsf{aux}, B, \boldsymbol{A}, e_i, w_i) \in \{0\}^b \times \{0,1\}^{\kappa - b}$ and $w_i = \delta_i + e_i \mu$, for all $i \in [r]$.

   The e's are sampled randomly until suitable e's are found.

   Output. $\psi = (A_i, e_i, w_i)_{i=1}^r$

Furthermore, we say that $(\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}, x, \psi)$ is accepting iff

$$\forall i \in [r], \quad \begin{cases} h^{w_i} = A_i \cdot B^{e_i} \\ \mathcal{H}(\mathsf{aux}, B, \boldsymbol{A}, e_i, w_i) \in \{0\}^b \times \{0,1\}^{\kappa - b} \end{cases} .$$

Notation 2.12. We write $\psi \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{dlog}}(B, x; \mu)$ for the prover's output resulting from the above process (recall that $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$ is defined by $\mathcal{H}$ and the tuple $(\mathsf{aux}, \mathbb{G}, g, h, q)$).

Remark 2.13. The above corresponds to the so-called Fischlin transform [12] of the standard three-round zero-knowledge proof-of-knowldge of discrete log. We present the Fischlin transform in the "vanilla version" and we ignore many optimizations that are beyond the scope of our paper, e.g. batching [15] or finding collisions instead of pre-images of zero [17].

Next, we state the security properties of $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$, without proof.

Theorem 2.14 (Security of $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$)**.** *It holds that* $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$ *satisfies the following security properties.*

> ***Knowledge Soundness.*** *For random oracle* $\mathcal{H}$*, let* $\mathcal{A}^{(\cdot)}$ *denote an oracle-aided non-uniform PPTM taking advice* $z \in \{0,1\}^*$ *and let* b, r *such that* $b \cdot r \in \omega(\log(\kappa))$*, where* κ *denotes the security parameter. With all-but-negligible probability over* $\psi \leftarrow \mathcal{A}^{\mathcal{H}}(\mathsf{aux}, \mathbb{G}, g, h, q, B, z)$*, if* $(\Pi^{\mathrm{Fi}}, B, \psi)$ *is accepting, then there exists* $\mathcal{E}$ *given* $\mathcal{A}$*'s oracle tape that outputs* $w \in \mathbb{F}_q$ *s.t.* $B = h^w \in \mathbb{G}$
>
> ***Zero-Knowledge.*** *Define* SIM *s.t.* $(A_i, e_i, w_i)_{i=1}^r \leftarrow \mathsf{SIM}$ *where* $(e_i, w_i)_{i=1}^r \leftarrow \mathbb{F}_q^r$ *and* $A_i = h^{w_i} \cdot B^{-e_i}$*. Then, there exists* $\mathcal{S}$ *taking input* $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$*,* B *and* $\psi \in \{0,1\}^*$ *s.t. for every (OA) distinguisher* $\mathcal{D}^{(\cdot)}$*, it holds that*
>
> $$\Pr[\mathcal{D}^{\mathcal{H}}(\boldsymbol{\psi}) = 1 \wedge \boldsymbol{\psi} \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{dlog}}(B, x; \mu)] - \Pr[\mathcal{D}^{\mathcal{S}(\boldsymbol{\psi})}(\boldsymbol{\psi}) = 1 \wedge \boldsymbol{\psi} \leftarrow \mathsf{SIM}] \in \mathsf{negl}(\kappa).$$

2.5.1 Pedersen Equality ZK-PoK.

We note that $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$ can easily be extended to any "dlog-style" relation. Three such relation of interest which (we will use in our protocol) are defined as follows: Let ped_1 consist of all tuples $(\mathbb{G}, g, h, q, B; x, \mu)$ such that $B = \mathsf{com}(x, \mu) = g^x h^\mu$. Let ped_2 consist of all tuples $(\mathbb{G}, g, h, q, B, C; x, \mu)$ such that $B = \mathsf{com}(x, \mu) = g^x h^\mu$ and $C = g^x$.

Definition 2.15 (Pederen Eq. PoK)**.** Analogously to $\Pi^{\mathrm{Fi}}_{\mathsf{dlog}}$, let $\Pi^{\mathrm{Fi}}_{\mathsf{ped}_1}$ and $\Pi^{\mathrm{Fi}}_{\mathsf{ped}_2}$ denote the Fischlin transform of the three-round proof-of-knowledge for relation ped_1 and ped_2, and we write $\psi \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{ped}_{(\cdot)}}(\ldots)$ for the prover's output resulting from each process.

3 Protocol(s)

In this section, we define the two protocols $\Sigma_{\mathtt{classic}}$ and $\Sigma_{\mathtt{zero}}$. Since the protocols are very similar to each other, we opt to present the two together, c.f. Figure 6 for the key-generation Σ_{kgen} and Figure 7 for the signing phase Σ_{sign}, were suitable color coding is used to distinguish between the two (we use brick red for *Classic S.*).

Notation and Conventions. Prior to key-generation, the parties hold a common input that specifies the supper-session identifier (*ssid*) which, in particular, specifies the parties' identities (*pid*'s in $\boldsymbol{P}$) and the access structure $\mathbb{Q}$. Similarly, during signing, it is assumed that the parties start with the same message to be signed as the session identifier (*sid*) for that signing request; i.e., in this paper, we are agnostic about how the parties reach consensus on the relevant identifiers, or the message to be signed in each signature request. In the protocol description below, all the aforementioned data is in the common input $\mathsf{aux} \in \{0,1\}^*$ which is appropriately updated for each phase of the protocol. Finally, it is assumed that the parties store their respective output of each phase of the protocol and they erase all other data.

In the second round of the key-generation, we write $\mathsf{enc}_{(\cdot)}(\beta)$ to signify that β is sent over the secret channel (or it is encrypted using the appropriate key and sent over the broadcast channel). Finally, let $\lambda_i(\boldsymbol{Q})$ denote the Lagrange coefficient for $\mathcal{P}_i$ with respect to set $\boldsymbol{Q}$, i.e. $\lambda_i(\boldsymbol{Q}) = \prod_{j \in \boldsymbol{Q}}(-j) / \prod_{j \in \boldsymbol{Q} \setminus \{i\}}(i - j)$.

3.1 Identifiable Abort.

We note that all errors in both protocols can be attributed to a well defined corrupted party *except when a signature fails*. Indeed, a quick inspection of the protocol(s) reveal that in such a case errors occur because of a failed decommitment or ZK proof verification (for *Zero S.*). Thus, it suffices to explain how identifiable abort is achieved when the signature fails during signing. For this purpose, we will augment the protocol(s) as follows.

FIGURE 6 (Classic/Zero Schnorr: Key Generation – Σ_{kgen})

On input $(\mathsf{kgen}, ssid = (\mathbb{G}, g, h, q, \ldots, \mathbb{Q}, \boldsymbol{P}), \mathcal{P}_i)$ for $\mathcal{P}_i \in \boldsymbol{P}$, do:

1. Sample $\{\alpha_{i,k} \leftarrow \mathbb{F}_q\}_{k=0}^{t-1}$ and $\{\mu_{i,k}, \gamma_{i,k}, \rho_{i,k} \leftarrow \mathbb{F}_q\}_{k=1}^{n}$, and compute:
   - $B_{i,k} = g^{\alpha_{i,k}} h^{\mu_{i,k}} = \mathsf{com}(\alpha_{i,k}, \mu_{i,k})$ for $k \leq t-1$ and $B_{i,k} = h^{\mu_{i,k}} = \mathsf{com}(0, \mu_{i,k})$ for $k \geq t$.
     Generate $\{\psi_{i,j} \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{ped}_1}(B_{i,j}; \alpha_{i,j}, \mu_{i,j})\}_{k=0}^{t-1}$ and $\{\psi_{i,j} \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{dlog}}(B_{i,j}; \mu_{i,j})\}_{k=t}^{n}$
   - $D_{i,j} = \mathsf{com}(\gamma_{i,j}, \rho_{i,j})$ and $C_{i,j} = \prod_{j=0}^{n} B_{i,k}^{j^k} = \mathsf{com}(\beta_{i,j}, \nu_{i,j})$ for $j \in [n]$
     For *Classic S.*, ignore the above and set $A_{i,k} = \mathsf{com}(\alpha_{i,k}, 0)$ for $k \leq t-1$.

   Broadcast $(ssid, \mathcal{P}_i, V_i)$ for $V_i = \mathcal{H}(ssid, \mathcal{P}_i, U_i)$ s.t.

$$U_i = \begin{cases} (A_{i,0}, A_{i,1}, \ldots, A_{i,t-1}) & \text{for } \textit{Classic S.} \\ (B_{i,0}, \psi_{i,0} \ldots, B_{i,n}, \psi_{i,n}) & \text{for } \textit{Zero S.} \end{cases}$$

2. When obtaining $(ssid, \mathcal{P}_j, V_j)$ from all $\mathcal{P}_j$ send $(ssid, \mathcal{P}_i, U_i, W_{i,j})$ to each $\mathcal{P}_j$ where

$$W_{i,j} = \begin{cases} \mathsf{enc}_j(\beta_{i,j}) & \text{for } \textit{Classic S.} \\ \mathsf{enc}_j(\beta_{i,j}, \nu_{i,j}, \gamma_{i,j}, \rho_{i,j}) & \text{for } \textit{Zero S.} \end{cases}$$

3. When obtaining $(ssid, \mathcal{P}_j, U_j, W_{j,i})$ from $\mathcal{P}_j$, verify $V_j = \mathcal{H}(ssid, \mathcal{P}_j, U_j)$, and do:
   - Set $\{\hat{A}_j = A_{j,0}\}_{j=1}^{n}$ and check $g^{\beta_{j,i}} = \hat{A}_j \cdot \prod_{k=1}^{t-1} A_{j,k}^{i^k}$ (*Classic S.* ends here).
   - Check $g^{\beta_{j,i}} h^{\nu_{j,i}} = C_{j,i}$ and $g^{\gamma_{j,i}} h^{\rho_{j,i}} = D_{j,i}$ for *Zero S.*.

   When passing the above verification for all $j \in [n]$, do:
   - Set $\hat{D}_i = B_{i,0} \cdot \prod_{j=1}^{n} (D_{i,j} \cdot D_{j,i}^{-1}) = \mathsf{com}(\hat{\alpha}_i, \hat{\mu}_i)$ and $\hat{A}_i = g^{\hat{\alpha}_i}$
   - Generate proof $\theta_i \leftarrow \Pi^{\mathrm{Fi}}_{\mathsf{ped}_2}(\hat{D}_i, \hat{A}_i; \hat{\alpha}_i, \hat{\mu}_i)$

   Send $(ssid, \mathcal{P}_i, \hat{A}_i, \theta_i)$ to all.

   Output. If no error is detected, output $(X; x_i)$ where $X = \prod_{j=1}^{n} \hat{A}_j$ and $x_i = \sum_{j=1}^{n} \beta_{j,i}$.

Figure 6: Classic/Zero Schnorr: Key Generation – Σ_{kgen}

FIGURE 7 (Classic/Zero Schnorr: Signing – Σ_{sign})

On input $(\mathsf{sign}, sid = (ssid, \ldots, \boldsymbol{Q}, \mathrm{msg}), \mathcal{P}_i)$ for $\mathcal{P}_i \in \boldsymbol{Q}$, set $w_i = \lambda_i(\boldsymbol{Q}) \cdot x_i$, and do:

1. Sample $k_i \leftarrow \mathbb{F}_q$ and set $R_i = g^{k_i}$.
   Broadcast $(ssid, \mathcal{P}_i, V_i)$ for $V_i = \mathcal{H}(ssid, \mathcal{P}_i, R_i)$.
2. When obtaining $(ssid, \mathcal{P}_j, V_j)$ from all $\mathcal{P}_j$, send $(ssid, \mathcal{P}_i, R_i, S_{i,j})$ to each $\mathcal{P}_j$ s.t.

$$S_{i,j} = \begin{cases} \perp & \text{for } \textit{Classic S.} \\ \mathsf{enc}_j(\delta_{i,j}) & \text{where } \delta_{i,j} \leftarrow \mathbb{F}_q \text{ for } \textit{Zero S.} \end{cases}$$

3. When obtaining $(ssid, \mathcal{P}_j, R_j, S_{j,i})$ from all $\mathcal{P}_j$, send $(ssid, \mathcal{P}_i, \sigma_i)$ to all $\mathcal{P}_j$ s.t.

$$\sigma_i = \begin{cases} k_i + w_i \cdot \mathcal{H}(X, R, \mathrm{msg}) \mod q & \text{for } \textit{Classic S.} \\ k_i + w_i \cdot \mathcal{H}(X, R, \mathrm{msg}) + \sum_{j \in \boldsymbol{Q} \setminus \{i\}} (\delta_{i,j} - \delta_{j,i}) \mod q & \text{for } \textit{Zero S.} \end{cases}$$

   Output. When obtaining $(ssid, \mathcal{P}_j, \sigma_j)$ from all $\mathcal{P}_j$, output (R, σ) where $\sigma = \sum_{j \in \boldsymbol{Q}} \sigma_j$ iff

$$g^{\sigma} = R \cdot X^{e} \quad \text{and} \quad e = \mathcal{H}(X, R, \mathrm{msg})$$

Figure 7: Classic/Zero Schnorr: Signing – Σ_{sign}

Classic S. In this case, the process is essentially trivial and it suffices for the parties to store the "public-key shares" $\{X_i\}_{i=1}^n$ where

$$X_i = \prod_{k=0}^{t-1} (\prod_{j=1}^{n} A_{j,k})^{i^k} = g^{x_i}. \tag{1}$$

Then, when a quorum $\boldsymbol{Q} \in \mathbb{Q}$ is formed for signing msg $\in \{0,1\}^*$, using the notation from Figure 7 the parties check that $g^{\sigma_j} = R_j \cdot X_j^{f_j}$ for $f_j = \lambda_j(\boldsymbol{Q}) \cdot \mathcal{H}(X, R, m)$, for every $j \in \boldsymbol{Q}$.

Remark 3.1. We emphasize that all parties can calculate $\{X_i\}_{i=1}^n$ given their respective views of the key-generation phase using the left-hand side of Equation (1).

Zero S. Achieving identifiable abort for *Zero S.* is slightly more involved. First, similarly to *Classic S.*, we instruct the parties to store the values $(C_1, \ldots, C_n)$ during key-generation where

$$C_i = \cdot \prod_{k=0}^{n} (\prod_{j=1}^{n} B_{j,k})^{i^k} = g^{x_i} h^{\nu_i}. \tag{2}$$

Then, when a signature fails, the parties are instructed to follow the following identification process.

1. Each $\mathcal{P}_i$ calculates Pedersen commitments $Z_{i,j} = \mathsf{com}(\delta_{i,j}, \rho_{i,j})$ for random $\rho_{i,j} \leftarrow \mathbb{F}_q$ for all $j \in \boldsymbol{Q}$.

   The parties are instructed to broadcast $\{Z_{i,j}\}_{i,j\in[n]}$ and each $\mathcal{P}_j$ receives $\{\rho_{i,j}\}_{i\in[n]}$ over the secret channel. We note that if a corrupted party sends $Z_{i,j} = \mathsf{com}(\delta'_{i,j}, \rho_{i,j})$, for $\delta'_{i,j} \neq \delta_{i,j}$, then $\mathcal{P}_i$ may be reported as corrupted, e.g. by "opening" $S_{i,j}$ and revealing $\delta_{i,j}, \delta'_{i,j}, \rho_{i,j}$ to all parties.[9]

2. When completing the above for all parties in $\mathcal{Q}$, for $e = \mathcal{H}(X, R, \text{msg})$, each $\mathcal{P}_i$ proves that $Y_i = g^{-\sigma_i} \cdot C_{i,j}^{\lambda_i(\boldsymbol{Q})\cdot e} \cdot \prod_{j=1}^{n} Z_{i,j} \cdot Z_{j,i}^{-1}$ is a 0-commitment. That is, P_i sends $\psi_i \leftarrow \Pi_{\mathsf{dlog}}^{\mathrm{Fi}}(Y_i; \eta_i)$ for $\eta_i = \lambda_i(\boldsymbol{Q}) \cdot e\gamma_i + \sum_{j=1}^{n}(\rho_{i,j} - \rho_{j,i})$, and the other parties report $\mathcal{P}_i$ as corrupted if the tuple $(\psi_i, Y_i, 0)$ is rejecting (notice that Y_i is a 0-commitment iff $\mathcal{P}_i$ sends the right σ_i at the end of the signing phase).

4 Security

Next, we state our main security claims.

Theorem 4.1. *For t-out-of-n $\Sigma_{\mathtt{classic}}$, the following holds under the discrete log assumption (wlog $t < n - t$). If $n^t \in \mathsf{poly}(\kappa)$, then $\Sigma_{\mathtt{classic}}$ UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of a global random oracle $\mathcal{H}$.*

The above theorem is a corollary of Theorems 2.6, 2.9 and 4.3. In more detail, in Section 4.1 we show that $\Sigma_{\mathtt{classic}}$ is $\mathcal{G}^*$-simulatable with $\mathcal{H}$-consistency according to Definition 2.5 against adaptive adversaries. Thus, since Schnorr signatures are $\mathcal{G}^*$-existentially unforgeable (Pointcheval and Stern [24]), it follows that $\Sigma_{\mathtt{classic}}$ UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ against adaptive adversaries in the random oracle model.

Theorem 4.2. *For t-out-of-n $\Sigma_{\mathtt{zero}}$, the following holds under the discrete log assumption. For every $n, t \in \mathsf{poly}(\kappa)$, it holds that $\Sigma_{\mathtt{zero}}$ UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of a global random oracle $\mathcal{H}$.*

The above theorem is a corollary of Theorems 2.6, 2.9 and 4.4.

4.1 Simulatability of $\Sigma_{\mathtt{classic}}$

Theorem 4.3. *It holds that $\Sigma_{\mathtt{classic}}$ is $(\mathcal{G}^*, n^t)$-simulatable with $\mathcal{H}$-consistency. (wlog $t < n - t$)*

Proof. At the beginning of the simulation (The simulator is described in Figure 8), our simulator chooses $n - t$ honest parties randomly which are called the special parties and all other parties are simulated by running their code as prescribed. To deal with adaptive corruptions, we assume that the special parties are chosen afresh every time Σ_{refr} is simulated (the simulation is reset, via rewinding, to the last key refresh if

[9] For example, $\delta_{i,j}$ may be signed when sent over the secret channel.

FIGURE 8 ($\mathcal{G}^*$-simulation for $\Sigma_{\mathtt{classic}}$)

Parameters. Adversary $\mathcal{A}$, RO $\mathcal{H}$.

Operation.

init. Call $\mathcal{G}^*$ on input $(\mathbb{G}, g, q)$. Obtain $\mathsf{pk} = X$.

– (Σ_{kgen}) Choose $\boldsymbol{B} \subseteq \boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C}$ of size $n - t + 1$ and do:

1. For $b \in \boldsymbol{B}$, hand over $V_b \leftarrow \{0,1\}^*$ to $\mathcal{A}$.
2. When obtaining $(V_j)_{j \notin \boldsymbol{B}}$, retrieve $(A_{j,0}, \ldots, A_{j,t-1})_{j \notin \boldsymbol{B}}$ and do:
   (a) Set $X_{\boldsymbol{B}} = X \cdot (\prod_{j \notin \boldsymbol{B}} A_{j,0})^{-1}$ and sample $\{A_{b,0}\}_{b \in \boldsymbol{B}}$ subject to $\prod_{b \in \boldsymbol{B}} A_{b,0} = X_{\boldsymbol{B}}$.
   (b) Sample $\{\beta_{b,j} \leftarrow \mathbb{F}_q\}_{b \in \boldsymbol{B}, j \notin \boldsymbol{B}}$ and set $\{A_{b,1}, \ldots, A_{b,t-1}\}_{b \in \boldsymbol{B}}$ s.t. $\prod_{k=1}^{t-1} A_{b,k}^{j^k} = A_{b,0}^{-1} \cdot g^{\beta_{b,j}}$.
   Use the Vandermonde matrix for the above.
   (c) Calculate all other values as prescribed.
   Hand over $(A_{b,0}, \ldots, A_{b,t-1})_{b \in \boldsymbol{B}}$ and $(\beta_{b,j})_{b \in \boldsymbol{B}, j \in \boldsymbol{C}}$ to $\mathcal{A}$.

– (Σ_{refr}) Reasign $\boldsymbol{B} \subseteq \boldsymbol{H} = \boldsymbol{P} \setminus \boldsymbol{C}$ of size $n - t + 1$ and do
Run simulator for Σ_{kgen} using $\{A_{b,0} = X_b^{\lambda_b(\boldsymbol{P})}\}_{b \in \boldsymbol{B}}$ in Item 2a and verify $\{A_{j,0} = X_j^{\lambda_j(\boldsymbol{P})}\}_{j \notin \boldsymbol{B}}$.

– (Σ_{sign}) If $|\boldsymbol{Q}| \geq t$, do: Letting $\{Y_i = X_i^{\lambda_i(\boldsymbol{Q})}\}_{i \in \boldsymbol{Q}}$,

1. For $b \in \boldsymbol{Q} \cap \boldsymbol{B}$, hand over $V_b \leftarrow \{0,1\}^*$ to $\mathcal{A}$.
2. When obtaining $(V_j)_{j \notin \boldsymbol{B}}$, retrieve $(R_j)_{j \notin \boldsymbol{B}}$ and do:
   (a) Sample $e \leftarrow \mathbb{F}_q$.
   (b) For $b \in \boldsymbol{Q} \cap \boldsymbol{B}$, sample $\sigma_b \leftarrow \mathbb{F}_q$ and set $R_b = g^{\sigma_b} \cdot Y_b^{-e}$.
   (c) Calculate all other values as prescribed and hand over $(R_b)_{b \in \boldsymbol{Q} \cap \boldsymbol{B}}$ to $\mathcal{A}$.
3. When obtaining $(R_j)_{j \in \boldsymbol{Q} \setminus \boldsymbol{B}}$, do:
   Hand over $(\sigma_b)_{b \in \boldsymbol{Q} \cap \boldsymbol{B}}$

Figure 8: $\mathcal{G}^*$-simulation for $\Sigma_{\mathtt{classic}}$. In the above, every time $\mathcal{S}$ "retrieves" a value, we mean that it obtains the relevant value from $\mathcal{A}$'s queries. Furthermore, it assumed $\mathcal{S}$'s messages are consistent with the simulated oracle (by programming the simulated random oracle accordingly whenever needed). So, e.g., for R_i chosen by $\mathcal{S}$ for special party $\mathcal{P}_i$ during signing, if the adversary queries $\mathcal{H}$ on input $(\mathsf{aux}, \mathcal{P}_i, R_i)$, then the simulator provides "answer" that leads to an error-free execution, namely V_i chosen by the simulator in the first round.

the adversary decides to corrupt any of the special parties). Furthermore, $\mathcal{S}$ simulates the random oracle and thus $\mathcal{A}$ "queries" $\mathcal{S}$ when it requests to query $\mathcal{H}$ and $\mathcal{S}$ returns answers according to the "real" oracle $\mathcal{H}$ *unless* these were programmed by the simulator itself (specifically the V's in Item 1 of key-generation and Item 1 of signing respectively, or the e's in Item 2a of signing). Thus, the simulation is consistent with the oracle. The reader is referred to Figure 8 for the full description of the simulation.

It is not hard to see that the simulation is statistically close to the real distribution. To conclude, we note that the simulation concludes with overwhelming probability in time $\binom{n}{t} \cdot \log(\kappa) \cdot \mathsf{time}_\Sigma$ and time_Σ is the running time of Σ (because the simulator is required to guess the correct identities of the simulated honest parties). □

4.2 Simulatability of $\Sigma_{\mathtt{zero}}$

Theorem 4.4. *Assuming discrete log, it holds that $\Sigma_{\mathtt{classic}}$ is $(\mathcal{G}^*, 1)$-simulatable with $\mathcal{H}$-consistency.*

Proof. Contrary to the *Classic S.* cas, the simulator(s) we will define next do *not* perform any guesswork and there is no rewinding when something "bad" happens (because there is no such event in this case). Similarly to Figure 8, we write $\boldsymbol{H}$, $\boldsymbol{C}$ for the set of honest and corrupted parties (these sets change as the interaction progresses).

We consider a two experiments (hybrids) where the first experiment is indistinguishable from the real execution (assuming discrete log) and the second experiment corresponds to the $\mathcal{G}^*$-simulation. In the second

hybrid, we will assume that the simulator is given the discrete log of h relative to g (this additional input may be viewed as a trapdoor of the CRS).

Hybrid 1. In the first experiment, the simulator proceeds as follows: The simulator merely runs the code of the honest parties as prescribed with the following caveats:

1. The simulator extracts the adversary's tuples $(\alpha_{j,k}, \mu_{j,k})_{j\in\boldsymbol{C},k\geq 0}$ and $(\hat{\alpha}_j, \mu_j)$ from the key-generation using the Fischlin extractor (Theorem 2.14) and the simulator calculates the corrupted parties' secret shares $\{(x_j, \rho_j)\}_{j\in\boldsymbol{C}}$ where $\rho_j = \sum_{k\in[n]} \rho_{k,j}$.
2. During signing, instead of calculating the honest parties' shares as prescribed, $\mathcal{S}$ simulates the signature using $Y = \prod X \cdot (\prod_{j\in\boldsymbol{Q}\cap\boldsymbol{C}} g^{\lambda_j(\boldsymbol{Q})\cdot x_j})^{-1}$ analogously to Item 2b of the signing simulation in Figure 8. That is, the simulator calculates $\hat{R} = g^{-\hat{\sigma}} \cdot Y^{-e}$ for $(\hat{\sigma}, e) \leftarrow \mathbb{F}_q^2$ and returns (independent) sharings $(\hat{R}_i)_{i\in\boldsymbol{H}}$ of $\hat{R}$ and $\{\hat{\tau}_i\}_{i\in\boldsymbol{H}}$ of $\hat{\tau}$ such that, letting $\{\delta_{i,j}\}_{i,j}$ denote the shifts exposed to the adversary during the experiment:
$$\hat{\tau} = \hat{\sigma} + \sum_{i\in\boldsymbol{H},j\in\boldsymbol{C}} \delta_{i,j} - \delta_{j,i} \mod q \tag{3}$$

   New Corruptions. When the adversary decides to corrupt $\mathcal{P}_i \in \boldsymbol{H}$, the simulator returns corresponding secret state (sampled/calculated by $\mathcal{S}$ during the simulation).

It straightforward to see that the above experiment is distinguishable from the real experiment only if the simulator calculates the wrong Y when calculating the signature above (all other values are identically distributed). In that case, however, it means that there is a discrepancy between the values extracted from the adversary and the "honestly" generated values. So, since the adversary controls all "honest" values, it can deduce a non-trivial relation between g and h using the witnesses extracted in round 1 witness extracted in round 3 (if they are consistent, i.e. $\{B_{i,k}\}$ and $\{\hat{A}_i\}$ yield the same g-component, then there is no distinguishing opportunity).

Hybrid 2 ($\mathcal{G}^*$-Simulation). For the second and last experiment, the simulator $\mathcal{S}$ is given trapdoor input $w \in \mathbb{F}_q$ such that $h = g^w \in \mathbb{G}$. We describe the simulator by explaining how it interacts with $\mathcal{G}^*$ and how it calculates all the "honest" simulated values. First, the simulator requests a public key pk from $\mathcal{G}^*$ and it sets $X = \mathsf{pk}$. Then, $\mathcal{S}$ samples $\{C_i = g^{y_i}\}_{i=1}^n$ for $y_i \leftarrow \mathbb{F}_q$ and sets

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 1 & \cdots & 1^n \\ 1 & 2 & 4 & \cdots & 2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & n & n^2 & \cdots & n^n \end{pmatrix}^{-1} \cdot \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ \vdots \\ C_n \end{pmatrix}$$

1. In the first round of the key generation, the simulator retrieves $\{B_{j,k}\}_{k=0}^n$ from the oracle queries and it extracts the tuples $(\alpha_{j,k}, \mu_{j,k})_{j\in\boldsymbol{C},k\geq 0}$ using the Fischlin extractor. Then, $\mathcal{S}$ calculates $\{B_{i,k}\}_{i\in\boldsymbol{H},k\geq 0}$ where each tuple $(B_{i,k})_{i\in\boldsymbol{H}}$ for fixed k is a random secret sharing of $B_j \cdot \prod_{j\in\boldsymbol{C}} B_{j,k}^{-1}$.

   By construction, $\mathcal{S}$ knows the discrete logarithms with respect to g of all $B_{j,k}$.
2. When sending $(\beta_{i,j}, \mu_{i,j}, \gamma_{i,j}, \rho_{i,j})_{i\in\boldsymbol{H},j\in\boldsymbol{C}}$, the simulator chooses the tuple as follows:
   - Calculate $y_{i,j}$ such that $g^{y_{i,j}} = \prod_{k=0}^n B_{i,k}^{j^k}$ and set $(\beta_{i,j}, \gamma_{i,j}, \rho_{i,j}) \leftarrow \mathbb{F}_q^3$ and $\mu_{i,j} = z^{-1} \cdot (y_{i,j} - \beta_{i,j})$.
3. When sending $(\hat{A}_i, \theta_i)_{i\in\boldsymbol{H}}$, letting $\{\gamma_{i,j}\}$ denote the γ's exposed to the adversary, choose $(\hat{A}_i)_{i\in\boldsymbol{H}}$ as a random sharing of $g^{\hat{y}} \cdot X$ where
$$\hat{y} = -\sum_{j\in\boldsymbol{C}} \lambda_j x_j + \sum_{i\in\boldsymbol{H},j\in\boldsymbol{C}} \gamma_{i,j} - \gamma_{i,j}.$$
   (the simulator calculates $\{x_j\}_{j\in\boldsymbol{C}}$ at the end of the previous simulated round using the Fishlin extractor.)

4. The signing process is handled exactly like the previous experiment.

   New Corruptions. When the adversary decides to corrupt $\mathcal{P}_i \in \boldsymbol{H}$, the simulator samples $x_i \leftarrow \mathbb{F}_q$ and returns (x_i, ν_i) as $\mathcal{P}_i$'s secret state, where $\nu_i = z^{-1} \cdot (y_i - x_i)$.

It is easy to see that the two hybrids are identically distributed. This concludes the proof.

□

References

[1] H. K. Alper and J. Burdges. Two-round trip schnorr multi-signatures via delinearized witnesses. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 157–188, Virtual Event, Aug. 2021. Springer, Heidelberg. doi: 10.1007/978-3-030-84242-0_7.

[2] M. Bellare and G. Neven. Multi-signatures in the plain public-key model and a general forking lemma. In A. Juels, R. N. Wright, and S. De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, Oct. / Nov. 2006. doi: 10.1145/1180405.1180453.

[3] C. Blokh, N. Makriyannis, and U. Peled. Efficient asymmetric threshold ecdsa for mpc-based cold storage. Cryptology ePrint Archive, Paper 2022/1296, 2022. https://eprint.iacr.org/2022/1296.

[4] J. Camenisch, M. Drijvers, T. Gagliardoni, A. Lehmann, and G. Neven. The wonderful world of global random oracles. In J. B. Nielsen and V. Rijmen, editors, *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 280–312. Springer, Heidelberg, Apr. / May 2018. doi: 10.1007/978-3-319-78381-9_11.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001. doi: 10.1109/SFCS.2001.959888.

[6] R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. https://eprint.iacr.org/2003/239.

[7] R. Canetti, A. Jain, and A. Scafuro. Practical UC security with a global random oracle. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, Nov. 2014. doi: 10.1145/2660267.2660374.

[8] R. Canetti, R. Gennaro, S. Goldfeder, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, Nov. 2020. doi: 10.1145/3372297.3423367.

[9] R. Canetti, N. Makriyannis, and U. Peled. UC non-interactive, proactive, threshold ECDSA. Cryptology ePrint Archive, Report 2020/492, 2020. https://eprint.iacr.org/2020/492.

[10] E. Crites, C. Komlo, and M. Maller. How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Report 2021/1375, 2021. https://eprint.iacr.org/2021/1375.

[11] E. Crites, C. Komlo, and M. Maller. Fully adaptive schnorr threshold signatures. Cryptology ePrint Archive, Paper 2023/445, 2023. URL https://eprint.iacr.org/2023/445. https://eprint.iacr.org/2023/445.

[12] M. Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Heidelberg, Aug. 2005. doi: 10.1007/11535218_10.

[13] F. Garillot, Y. Kondi, P. Mohassel, and V. Nikolaenko. Threshold Schnorr with stateless deterministic signing from standard assumptions. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 127–156, Virtual Event, Aug. 2021. Springer, Heidelberg. doi: 10.1007/978-3-030-84242-0_6.

[14] R. Gennaro and S. Goldfeder. One round threshold ECDSA with identifiable abort. Cryptology ePrint Archive, Report 2020/540, 2020. https://eprint.iacr.org/2020/540.

[15] R. Gennaro, D. Leigh, R. Sundaram, and W. S. Yerazunis. Batching Schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. In P. J. Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 276–292. Springer, Heidelberg, Dec. 2004. doi: 10.1007/978-3-540-30539-2_20.

[16] C. Komlo and I. Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In O. Dunkelman, M. J. J. Jr., and C. O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Heidelberg, Oct. 2020. doi: 10.1007/978-3-030-81652-0_2.

[17] Y. Kondi and A. Shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In S. Agrawal and D. Lin, editors, *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part II*, volume 13792 of *Lecture Notes in Computer Science*, pages 279–309. Springer, 2022.

[18] Y. Lindell. Simple three-round multiparty schnorr signing with full simulatability. Cryptology ePrint Archive, Report 2022/374, 2022. https://eprint.iacr.org/2022/374.

[19] G. Maxwell, A. Poelstra, Y. Seurin, and P. Wuille. Simple schnorr multi-signatures with applications to bitcoin. *Designs, Codes and Cryptography*, 87, 09 2019.

[20] S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: Extended abstract. In M. K. Reiter and P. Samarati, editors, *ACM CCS 2001*, pages 245–254. ACM Press, Nov. 2001. doi: 10.1145/501983.502017.

[21] J. Nick, T. Ruffing, Y. Seurin, and P. Wuille. MuSig-DN: Schnorr multi-signatures with verifiably deterministic nonces. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *ACM CCS 2020*, pages 1717–1731. ACM Press, Nov. 2020. doi: 10.1145/3372297.3417236.

[22] J. Nick, T. Ruffing, and Y. Seurin. MuSig2: Simple two-round Schnorr multi-signatures. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part I*, volume 12825 of *LNCS*, pages 189–221, Virtual Event, Aug. 2021. Springer, Heidelberg. doi: 10.1007/978-3-030-84242-0_8.

[23] A. Nicolosi, M. N. Krohn, Y. Dodis, and D. Mazières. Proactive two-party signatures for user authentication. In *NDSS 2003*. The Internet Society, Feb. 2003.

[24] D. Pointcheval and J. Stern. Security proofs for signature schemes. In U. M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Heidelberg, May 1996. doi: 10.1007/3-540-68339-9_33.

[25] C. Schnorr. Efficient signature generation by smart cards. *J. Cryptol.*, 4(3):161–174, 1991.

[26] D. R. Stinson and R. Strobl. Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates. In V. Varadharajan and Y. Mu, editors, *ACISP 01*, volume 2119 of *LNCS*, pages 417–434. Springer, Heidelberg, July 2001. doi: 10.1007/3-540-47719-5_33.

FIGURE 9 (Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$)

Key-generation:

1. Upon receiving $(\texttt{keygen}, ssid)$ from some party $\mathcal{P}_i$, interpret $ssid = (\ldots, \boldsymbol{P}, \mathbb{Q})$, where $\boldsymbol{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_n)$.
   - If $\mathcal{P}_i \in \boldsymbol{P}$, send to $\mathcal{S}$ and record $(\texttt{keygen}, ssid, \mathcal{P}_i)$.
   - Otherwise ignore the message.
2. Once $(\texttt{keygen}, ssid, j)$ is recorded for all $\mathcal{P}_j \in \boldsymbol{P}$, send $(\texttt{pubkey}, ssid)$ to the adversary $\mathcal{S}$ and do:
   (a) Upon receiving $(\texttt{pubkey}, ssid, X, \mathcal{V})$ from $\mathcal{S}$, record $(ssid, X, \mathcal{V})$.
   (b) Upon receiving $(\texttt{pubkey}, ssid)$ from $\mathcal{P}_i \in \boldsymbol{P}$, output $(\texttt{pubkey}, ssid, X)$ if it is recorded. Else ignore the message.

Signing:

1. Upon receiving $(\texttt{sign}, sid = (ssid, \ldots), m)$ from $\mathcal{P}_i$, send to $\mathcal{S}$ and record $(\texttt{sign}, sid, m, i)$.
2. Upon receiving $(\texttt{sign}, sid = (ssid, \ldots), m, j)$ from $\mathcal{S}$, record $(\texttt{sign}, sid, m, j)$ if $\mathcal{P}_j$ is corrupted. Else ignore the message.
3. Once $(\texttt{sign}, sid, m, i)$ is recorded for all $\mathcal{P}_i \in \boldsymbol{Q} \subseteq \boldsymbol{P}$ and $\boldsymbol{Q} \in \mathbb{Q}$, send $(\texttt{sign}, sid, m)$ to $\mathcal{S}$ and do:
   (a) Upon receiving $(\texttt{signature}, sid, m, \sigma)$ from $\mathcal{S}$,
      - If the tuple $(sid, m, \sigma, 0)$ is recorded, output an error.
      - Else, record $(sid, m, \sigma, 1)$.
   (b) Upon receiving $(\texttt{signature}, sid, m)$ from $\mathcal{P}_i \in \boldsymbol{Q}$:
      - If $(sid, m, \sigma, 1)$ is recorded, output $(\texttt{signature}, sid, m, \sigma)$ to $\mathcal{P}_i$.
      - Else ignore the message.

Verification:

Upon receiving $(\texttt{sig-vrfy}, sid, m, \sigma, X)$ from a party $\mathcal{X}$, do:

- If a tuple (m, σ, β') is recorded, then set $\beta = \beta'$.
- Else, if m was never signed and not all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ are corrupted/quarantined, set $\beta = 0$.

"*Unforgeability*"

- Else, set $\beta = \mathcal{V}(m, \sigma, X)$.

Record (m, σ, β) and output $(\texttt{istrue}, sid, m, \sigma, \beta)$ to $\mathcal{X}$.

Figure 9: Ideal Threshold Signature Functionality $\mathcal{F}_{\mathsf{tsig}}$

A (OM) Discrete Log & Algebraic Model

Using OMDL. It is easy to show that *Classic S.* is adaptive UC-secure using the result of [11] for the case of $2\alpha + \sigma < t$ where α and σ represent the adaptive or static number of corruptions respectively. To see why, we note that by augmenting signing oracle $\mathcal{G}^*_{\mathsf{dlog}}$ to include discrete-log queries for previously generated group elements (representing the parties' public-key shares), we apply Crites et al. [11]'s result to claim that Schnorr signatures are $\mathcal{G}^*_{\mathsf{dlog}}$-unforgeable under the OMDL assumption. On the other hand, *Classic S.* is $\mathcal{G}^*_{\mathsf{dlog}}$-simulatable without any security loss (since the discrete log oracle removes all the guesswork).

Using AGM. In a second result, [11] show that in the algebraic group model achieves full adaptivity under the OMDL. The intuition is quite simple: in the reduction to (OM) discrete log, the adversary outputting a forgery also outputs a non-trivial discrete log relation when it supplies the representation of the forgery, thus breaking OMDL.

We believe that the above results easily extends to the UC case *by forcing adversaries to be algebraic via a strict global functionality* (like the RO in our work). So, using such an algebraic oracle (AO), we can extend our theorem to show that *Classic S.* UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ in the presence of the global AO and RO.

Remark A.1. We want to emphasize that the strictness of the AO, rendering it non-observable and non-programmable, has no impact on our analysis. In fact, it strengthens the composability result. The security of *Classic S.* remains preserved even when it is composed with other protocols *utilizing the same group.* As far as we know, this guarantee of composability fails when the AO is not strict (i.e., observable or programmable).