On the Classic Protocol for MPC Schnorr Signatures

or Classic Schnorr checks all the boxes

Nikolaos Makriyannis*

Abstract

The classic MPC protocol for Schnorr Signatures [17, 20, 23] (*Classic Schnorr*) consists of a simple three-round process for the signing operation, and the protocol is essentially as efficient as the underlying non-MPC scheme (modulo the round-complexity). In particular, Classic Schnorr does *not* contain any ZK proofs, not even for key-generation, and the only cryptographic "machinery" it uses is the underlying hash function. In this paper, we show that Classic Schnorr UC-realizes the ideal threshold-signature functionality of Canetti, Makriyannis, and Peled (Manuscript'20) against adaptive adversaries for any number of corrupted parties. Furthermore, (1) the protocol does not impose any restrictions on the number of concurrent signings, (2) the protocol naturally supports identifiable abort, and (3) the protocol can be extended to achieve proactive security, almost for free. So, the main novelty of our work is showing that Classic Schnorr achieves the utmost security as a threshold-signatures protocol. We hold that the achieved security is truly surprising given how simple the protocol is.

On a technical level, we show the above by extending the proof technique of Canetti, Makriyannis, and Peled, recently generalized by Blokh, Makriyannis, and Peled (Manuscript'22) for arbitrary threshold-signature schemes, whereby the indistinguishability of the UC simulation is reduced to the unforgeability of the underlying signature scheme. Our results hold in the random oracle model under the discrete logarithm assumption.

*Fireblocks. Email: nikos@fireblocks.com

Contents

1 Introduction

The Schnorr signature scheme (Schnorr [22]) 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 applications in the Blockchain space, e.g. for verifying the validity and integrity of cryptocurrency transactions. In this paper, we focus on Schnorr signatures in the context of *Multi-Party Computation Signing* (MPC signing), defined next.

In MPC signing, mutually distrusting parties interact by means of an MPC protocol to produce a joint signature on a common document, and the resulting signature is indistinguishable from a standard, non-MPC, signature. Such signatures offer a number of advantages over standard signatures. For instance, they eliminate the single-point-of-failure problem because the signing capability is distributed among many parties, and thus the secret material of any given party becomes less sensitive. Furthermore, MPC signatures are more efficient than standard signatures because a *single* datum suffices to certify that a document was authorized by *many* parties.

MPC Schnorr. MPC protocols for Schnorr are truly abundant in the literature.[1] The earliest protocols [17, 20, 23] 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 they rely on cryptographic assumptions that are new, interactive, non-falsifiable, or some combination thereof.

In terms of security, most works in the literature follow one of two definitional paradigms (we discuss these later in Section 1.1) and they offer various security guarantees depending on the adversarial model; Passive vs Malicious, i.e. the adversary has eavesdropping capabilities vs full control over the corrupted parties, Static vs Adaptive, i.e. the adversary corrupts parties statically at the beginning of the protocol vs adaptively as the protocol progresses, Standalone vs Composable, i.e. the adversary operates within the confines of the protocol environment vs no such restriction is imposed on the adversary. As far as we know, no protocol achieves the highest level of security, i.e. malicious and adaptive security, with composability.

Motivation. In the spirit of Lindell [15], 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 MPC-signing paradigm, i.e. where the signatories agree on a common public key ahead of time (we elaborate on the different "flavours" in Section 1.1). 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 adaptive & malicious 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 Schnorr* henceforth, c.f. Figure 1). Our main contribution is showing that Classic Schnorr 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 (in fact most works do not even consider adaptive adversaries). Finally, we hold that the achieved level of security is truly surprising given how simple the protocol is.

On a technical level, we show that Classic Schnorr UC-realizes the ideal threshold-signature functionality $\mathcal{F}_{\mathsf{tsig}}$ of Canetti et al. [9] against adaptive and malicious adversaries. At a high-level, $\mathcal{F}_{\mathsf{tsig}}$ captures the "essence"

[1] Non-exhaustively, we mention [1, 2, 10, 12, 14, 15, 16, 17, 18, 19, 20, 23].

[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.

of a secure threshold-signatures scheme; i.e. authorized sets of parties can generate signatures *vs* unauthorized sets of parties cannot generate signatures. Furthermore, by instructing the parties to essentially re-execute the key-generation (with appropriate checks), we show that Classic Schnorr achieves full proactive security.[3] Before we present the technical merits of our results (c.f. Section 1.2), we discuss the necessary background.

FIGURE 1 (Classic Schnorr)

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 $\text{msg} \in \{0,1\}^*$, 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 $e = \mathcal{H}(X, R, \text{msg})$ 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 $\text{msg} \in \{0,1\}^*$, accept signature $(R, \sigma) \in \mathbb{G} \times [q]$ iff $g^\sigma = R \cdot X^e$ where $e = \mathcal{H}(X, R, \text{msg})$. In the technical sections, we generalize the above to the t-out-of-n threshold setting, where any set of $t \leq n$ parties may sign. Note that, to avoid clutter, we have suppressed the use of identifiers $(sid, pid, ssid, \ldots)$ in the above.

1.1 Background

In this section we discuss the different "flavours" of MPC signatures, the two definitional paradigms for security, and our choice of ideal functionality.

Threshold vs Multi-Signatures. MPC Signing can be broadly distinguished into threshold signatures (e.g. [9, 11, 12, 14, 15, 20, 23]) and multi-signatures (e.g. [1, 2, 16, 17, 18, 19]). As mentioned earlier, in the threshold case, the parties jointly agree on a common public key ahead of time. In contrast, in multi-signatures, each party has its own public key and there is minimal coordination prior to signing, if any. There is a lot of overlap between the two notions, both in techniques as well as results, and, for some applications, either notion may suffice. Interestingly, the two flavours of MPC signatures are further characterized by the security methodology they typical follow; most papers in the multi-signatures space use the historical paradigm, while most papers in the threshold space use the "Real vs Ideal" paradigm, defined next.

Simulation-Based vs Game-Based Security. The historical *aka* game-based paradigm for provable security can be summarized as follows: To prove that a given scheme/protocol is secure, the security analyst formulates an ad-hoc definition, typically a security *game*, and they show that the scheme satisfies the definition, e.g. via reduction where a hypothetical adversary that wins the security game can also be use to break some widely held cryptographic assumption. The scheme is then deemed secure assuming the security game captures certain desiderata.

In recent decades, with the advent of MPC, a new definitional paradigm emerged; the so-called "Real vs Ideal" *aka* simulation-based paradigm where security is defined as follows. The cryptographer (or security analyst) formulates an ideal experiment where all computation is performed by a trusted and incorruptible entity, and the real protocol is deemed secure if it emulates the ideal experiment, i.e. no distinguisher can

[3] A t-out-of-n threshold signing protocol achieves proactive security if the scheme remains unforgeable against any adaptive adversary that corrupts at most $t-1$ signatories in any given epoch, where the epochs are delineated by a key-refresh mechanism.

differentiate between the real protocol and the ideal experiment. In the MPC jargon, the ideal experiment is parameterized by some ideal *functionality*, and the protocol is said to *realize the ideal functionality* if it is secure.

Comparing the two notions, it is worth noting that the simulation-based paradigm is more expressive. For instance, it can capture notions that are external to the protocol environment, like composability; e.g. in the UC framework, the ideal experiment is augmented to account for such external threats and the protocol is said to *UC-realize the ideal functionality* if no distinguisher can differentiate between the real protocol and the (augmented) ideal experiment. In addition, the simulation-based paradigm is arguably more intuitive because security, i.e. the ideal experiment, is defined in terms of "trusted parties" instead of technical security games.

On the other hand, if the ideal experiment exceeds the security requirements of the targeted application, then the real protocol may incur unnecessary complexity costs; i.e., the simulation paradigm can "overdeliver" in a wasteful way. Therefore, it is preferable to design tailor-made solutions, in the spirit of the game-based approach, *within* the simulation-based paradigm. The discussion below illustrates this point.

$\mathcal{F}_{\mathsf{tsig}}$ vs Schnorr SFE. In [15], Lindell presents a three-round protocol[4] which UC-realizes the signing operation of Schnorr, i.e. the Schnorr functionality $\mathcal{F}_{\mathsf{schnorr}}$. By contrast, in this paper, we show that Classic Schnorr UC-realizes $\mathcal{F}_{\mathsf{tsig}}$, that is, a generic threshold-signatures functionality. Opting for $\mathcal{F}_{\mathsf{tsig}}$ is very rewarding for our targeted application; we achieve *adaptive* security compared to *static* security in [15], and our protocol is simpler and more competitive complexity-wise.

We note, however, that $\mathcal{F}_{\mathsf{tsig}}$ and $\mathcal{F}_{\mathsf{schnorr}}$ are not interchangeable. For one, $\mathcal{F}_{\mathsf{schnorr}}$ implies $\mathcal{F}_{\mathsf{tsig}}$ only if non-MPC Schnorr is a good signature scheme.[5] Conversely, $\mathcal{F}_{\mathsf{tsig}}$ is not always the right abstraction for the Schnorr signature operation,[6] because it is agnostic to the signature-generation process (c.f. Section 1.2). By extension, protocols that realize $\mathcal{F}_{\mathsf{tsig}}$, e.g. Classic Schnorr, are not good substitutes for $\mathcal{F}_{\mathsf{schnorr}}$ in general. Rather, protocols that realize $\mathcal{F}_{\mathsf{tsig}}$ are good threshold-signatures protocols, and Classic Schnorr achieves the utmost security for this purpose; this is the main contribution of our work.

1.2 Our Techniques

The present section assumes some familiarity with the UC framework.

Ideal Threshold-Signatures Functionality. Our main security claim is that Classic Schnorr UC-realizes the ideal ideal threshold-signatures functionality $\mathcal{F}_{\mathsf{tsig}}$ from [9][7]. As mentioned earlier, the purpose of $\mathcal{F}_{\mathsf{tsig}}$ is to capture the "essence" of a secure threshold-signatures scheme. In more detail, letting $\boldsymbol{P}$ denote the set of signatories, $\mathcal{F}_{\mathsf{tsig}}$ provides the functionality/security for the t-out-of-n case:

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 $\mathrm{msg} \in \{0,1\}^*$ 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 $\mathcal{V}(\mathrm{msg}, \sigma) \in \{\texttt{true}, \texttt{false}\}$ if it has record of this message. If there is no record of msg, the functionality returns `false` regardless of $\mathcal{V}(\cdot)$.

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 $\mathcal{V}(\mathrm{msg}, \sigma) = \texttt{true}$. 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).

[4] The core protocol in [15] essentially corresponds to Classic Schnorr *with* ZK proofs.
[5] e.g. in a world where discrete log is easy, a protocol that realises $\mathcal{F}_{\mathsf{schnorr}}$ is a poor threshold-signatures protocol.
[6] For instance, $\mathcal{F}_{\mathsf{schnorr}}$ can be used to realize a distributed ZKPoK of discrete log, but $\mathcal{F}_{\mathsf{tsig}}$ is not useful for this purpose.
[7] The results of [9] were published in [8] (CCS'20) as part of a combined work with Gennaro and Goldfeder [13].

Simulatability & Unforgeability imply UC Security. We use the key technique from [9] to show that Classic Schnorr 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-signature protocols. In a nutshell, starting from a signature scheme Sig and a threshold protocol Σ for computing Sig, Blokh et al. [3] show that if (1) Sig is unforgeable according to the usual game-based definition and (2) Σ can be standalone-simulated[8] using an oracle to Sig (the same oracle from the unforgeability game), then Σ UC-realizes the ideal threshold-signature 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.[9,10] 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). In conclusion, since Schnorr signatures are unforgeable in the random oracle model (assuming discrete log [21]), and Classic Schnorr can be simulated against adaptive adversaries using a suitable signature oracle, it follows that Classic Schnorr UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ against adaptive adversaries in the random oracle model under the discrete log assumption.

No ZK? No Problem! A surprising aspect of Classic Schnorr is the total absence of ZK proofs, especially given the advertised security. To explain in one sentence, Classic Schnorr 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 $(R_{\mathcal{A}}, \sigma_{\mathcal{A}}) = (g^{k_{\mathcal{A}}}, k_{\mathcal{A}} + ex_{\mathcal{A}})$ 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.[11]

Remark 1.1 (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 with not-so-minimal assumptions may compare favorably to a complicated protocol with minimal assumptions. In this regard, Classic Schnorr has no match. Finally, our security analysis uses the random oracle in the same fashion as the seminal work of Pointcheval and Stern [21], or, as mentioned in Footnote 11, the standard ZKPoK for discrete log. Thus, our use of the oracle is not innovative or sophisticated, and so the principle of conservative design is once again upheld.

1.3 Extensions and Open Problems

Identifiable abort. We point out that every failure can be attributed to one of the participants if the protocol aborts prematurely, verifiably so. Namely, the protocol fails either because one of the decommitments does not verify, or, the signature share σ_j of $\mathcal{P}_j$ doe not verify according to the formula $g^{\sigma_j} = R_j \cdot X_j^e$. In either case, the exposed party may be identified as corrupted.

Non-Interactive Signing. Many recent protocols (for Schnorr and ECDSA) admit a preprocessing mode of operation where the message-independent parts of the protocol are executed before the message to be

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

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

[10] [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.

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

signed is presented, e.g. calculating the nonce. While Classic Schnorr also admits a non-interactive variant (by executing the first two rounds of signing ahead of time), the analysis herein does not extend to this variant and we speculate that achieving the same security guarantees requires either changes to the protocol or the underlying security assumptions, or both.

Removing the RO. In the protocol, the random oracle is used for two different purposes: (1) for the internal hash function of non-MPC Schnorr, and (2) for calculating commitments/decommitments during the computation of the nonce during signing. While the first item seems crucial for our security analysis, we note that the second item is merely a matter of choice; i.e. we could have opted to use a generic (non-malleable & extractable) commitment instead.[12] We leave it as an interesting open problem to see whether the ROM can be completely avoided, e.g. using correlation intractability.

Some perspective on the scope of our results. Our analysis assumes that the identity of the signatories is known in advance; therefore, our results do not extend to settings where the group of signatories (for the same public key) is massive and it evolves as the protocol progresses. Furthermore, we assume synchronous communication (where parties' messages are delivered with an *a priori* fixed maximum time delay), so our model is not realistic for such systems in any case.

Finally, we point out that the quality of adaptive security depends on the size of t or $n-t$, because the reduction is required to guess which (simulated) parties are corrupted as the experiment progresses. As such, the reduction gives rise to a non-trivial security loss when both t and $n-t$ are large, perhaps severe in some cases. It is stressed that there is no security loss for the static case, for any t and n.

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 $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \mathrm{msg}) = 1$.

[12] Using the RO for commitments conforms with standard practices and it makes the protocol description simpler.

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 for Schnorr and unforgeability will be defined with respect to that specific oracle.

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

Parameters. OA-Signature scheme $\text{Sig}^{\mathcal{O}}$ 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 $\mathcal{G}$

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 $\mathcal{G}\text{-}\mathsf{EU}(\mathcal{A}, 1^\kappa)$

Definition 2.2 ($\mathcal{G}$-Existential Unforgeability.)**.** We say that $\text{Sig}^{\mathcal{O}}$ 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.

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 $\text{EXEC}^{\mathcal{Z}}_{\Pi,\mathcal{A}}(1^\kappa, z)$ 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 $\text{EXEC}^{\mathcal{Z}}_{\mathcal{F},\mathcal{S}}(1^\kappa, z)$ 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

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

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 Proactive 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 proactive threshold signatures scheme for $\text{Sig}^{\mathcal{O}} = (\ldots, \mathsf{vrfy})$ 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_{refr} takes input (pk, s_i) from each $\mathcal{P}_i \in \boldsymbol{P}$ and returns (a fresh) value s_i to each $\mathcal{P}_i \in \boldsymbol{P}$.
3. Σ_{sign} takes input $\text{msg} \in \{0,1\}^*$ 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 $\mathsf{vrfy}_{\mathsf{pk}}(\sigma, \text{msg}) = 1$ in an honest execution.

Sets $\boldsymbol{Q} \in \mathbb{Q}$ are called *quorums*. The span between two consecutive executions of Σ_{refr} is referred to as an *epoch*. By convention, the span before the first execution of Σ_{refr} is the first epoch.

A protocol Σ is said to be secure if it UC-realizes functionality $\mathcal{F}_{\text{tsig}}$, 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. 17) 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.

For proactive security, the functionality admits an additional interface for recording corrupted and decorrupted parties. When a party is decorrupted, the functionality records that party as *quarantined* until it is instructed to purge that record (via a special key-refresh interface). If all parties are corrupted and/or quarantined at any given time, then the functionality enters a pathological mode of operation and it ignores the message-signature repository it holds internally.

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

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

2.3 Unforgeability & Simulatability imply UC Security

For OA-signature scheme $\text{Sig}^{\mathcal{O}}$ and associated threshold-protocol Σ, for adversary $\mathcal{A}$, write $\text{Real}_{\mathcal{A}}$ for the adversary's view in an execution of Σ in the presence of an *adaptive* PPTM adversary $\mathcal{A}$. Without loss of generality assume that $\text{Real}_{\mathcal{A}} = (\mathsf{pk}^{\Sigma}, \ldots)$, 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 $\mathcal{H}$, write $\text{Ideal}_{\mathcal{S}} = (\mathsf{pk}^{\mathcal{G}}, \text{Out}^{\mathcal{S}})$ for the pair of random variables consisting of the public key generated by $\mathcal{G}$ and the simulator's, $\mathcal{S}$, output.

Definition 2.5 (Simulatability)**.** Using the notation above, we say that Σ is $\mathcal{G}$-simulatable with $\mathcal{O}$-consistency if the following holds for every adversary $\mathcal{A}$. 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, (2) if $\mathcal{A}$ does not corrupt all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ simultaneously in any given epoch, then $\{\text{Real}_{\mathcal{A}}\} \equiv \{\text{Ideal}_{\mathcal{S}}\}$, and, (3) unless $\text{msg} \in \{0,1\}^*$ was queried to $\mathcal{G}$, it holds that the 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.

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* $\mathcal{G}$*-simulatable with* $\mathcal{O}$*-consistency.*

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. Furthermore, every time the honest parties output a signature, then the simulator submits the resulting signature-string to the functionality, and, depending on the adversary's corruption pattern and the protocol's

key-refresh schedule, the simulator registers parties as corrupted and/or quarantined. 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). 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 $R = g^k$, $m = \mathcal{H}(X, R, \text{msg})$, $\sigma = k + mx \mod q$.
3. For $(R, \sigma) \in \mathbb{F}_q^2$, define $\mathsf{vrfy}_X(\text{msg}, \sigma) = 1$ iff $g^\sigma = R \cdot X^m$ and $m = \mathcal{H}(X, R, \text{msg})$.

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

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 [21])**.** *Assuming DLOG, letting $\mathcal{G}^*$ denote the oracle from Figure 5, it holds that Schnorr signatures are $\mathcal{G}^*$-existentially unforgeable.*

3 Protocol

In this section we define our proactive threshold-Schnorr protocol $\Sigma_{\mathtt{schnorr}} = (\Sigma_{\mathsf{kgen}}, \Sigma_{\mathsf{refr}}, \Sigma_{\mathsf{sign}})$, c.f. Figure 6 for the key-generation Σ_{kgen} and Figure 7 for the signing phase Σ_{sign}. The key-refresh phase Σ_{refr} is described in Section 3.1 by pointing out the difference with Σ_{kgen} (the key-generation and the key-refresh phase are essentially the same protocol).

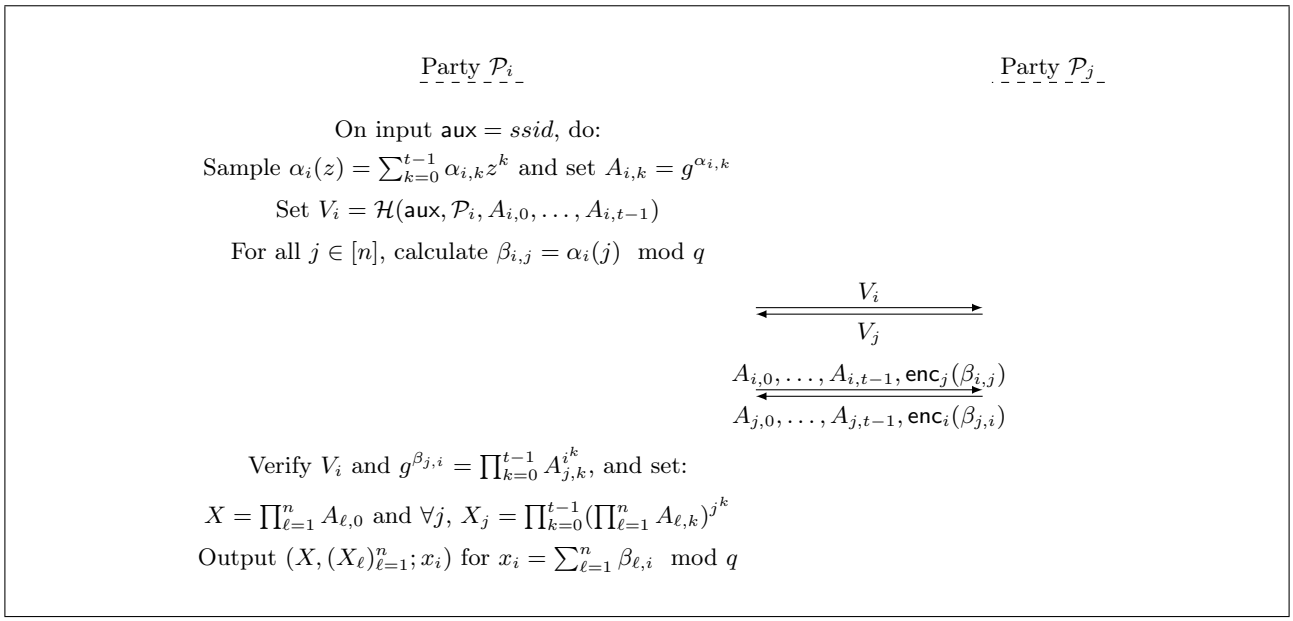

Figure 6: Threshold Schnorr: Key Generation (Σ_{kgen})

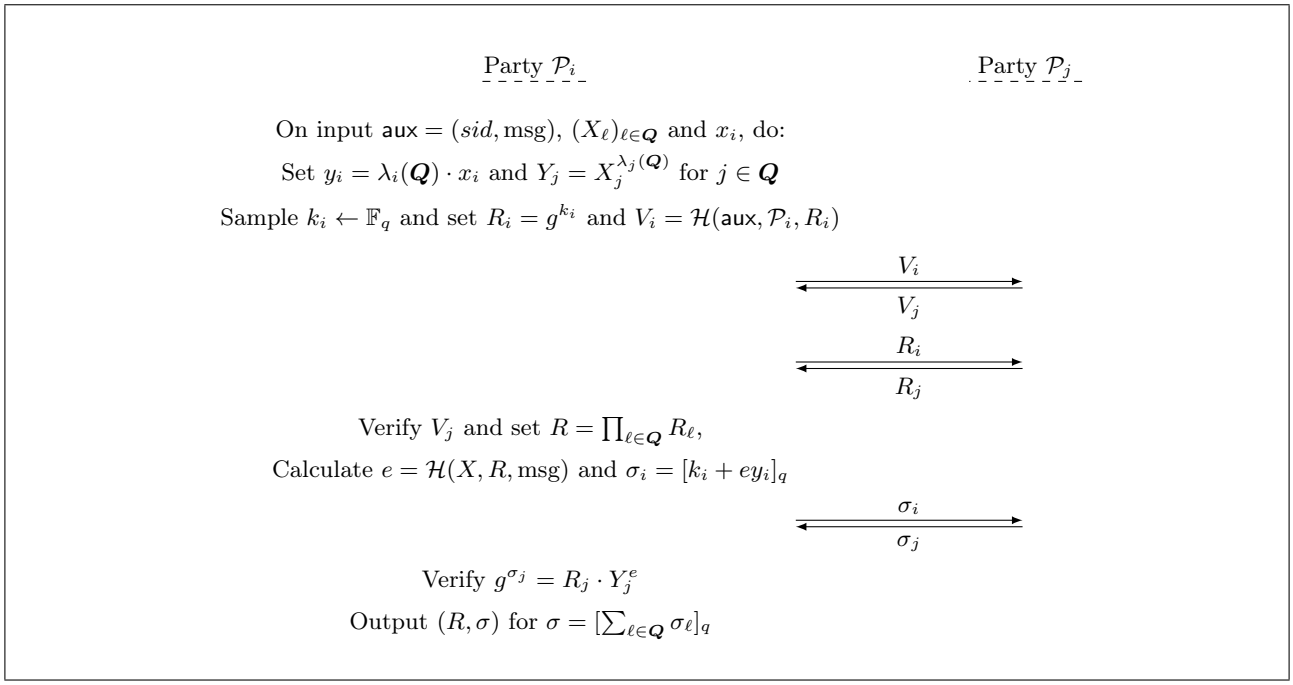

Figure 7: Threshold Schnorr: Signing (Σ_{sign})

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 Key-Refresh

Next we describe the key-refresh phase Σ_{refr}. On input $(X_\ell)_{\ell\in\boldsymbol{P}}$ and x_i, execute the Key-Generation (Figure 6) with the following changes:

Round 1. Sample $\alpha_i(z)$ s.t. $\alpha_i(0) = \lambda_i(\boldsymbol{P})\cdot x_i \mod q$.

Round 2. Check that $A_{j,0} = X_j^{\lambda_j(\boldsymbol{P})}$, for every $j \in [n]$.

At the end of the execution, if no error was detected, reassign $(X_\ell)_{\ell\in\boldsymbol{P}}$ and x_i as prescribed by the protocol.

4 Security

Theorem 4.1. *Assuming DLOG, it holds that $\Sigma_{\mathtt{schnorr}}$ UC-realizes functionality $\mathcal{F}_{\mathsf{tsig}}$ in the presence of a global random oracle functionality $\mathcal{H}$.*

The above theorem is a corollary of Theorems 2.6, 2.9 and 4.2. In more detail, in Section 4.1 we show that $\Sigma_{\mathtt{schnorr}}$ 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 [21]), it follows that $\Sigma_{\mathtt{schnorr}}$ UC-realizes $\mathcal{F}_{\mathsf{tsig}}$ against adaptive adversaries in the random oracle model.

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

Theorem 4.2. *It holds that $\Sigma_{\mathtt{schnorr}}$ is $\mathcal{G}^*$-simulatable with $\mathcal{H}$-consistency.*

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 the adversary decides to corrupt any of the special parties). Furthermore, $\mathcal{S}$ simulates the random oracle as well and thus $\mathcal{A}$ "queries" $\mathcal{S}$ when it needs to query $\mathcal{H}$ and we point out that $\mathcal{S}$ returns answers according to the "real" oracle $\mathcal{H}$ *unless* these were programmed by the simulator itself; namely, 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 $\tau\cdot n^\tau \log(n)\cdot \mathsf{time}_\Sigma$ for $\tau = \min(t, n-t)$ and time_Σ is the running time of Σ (because the simulator is required to guess the correct identities of the simulated honest parties). □

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

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 $(\beta_{j,b})_{j \notin \boldsymbol{B}, b \in \boldsymbol{B}}$ and do:
   (a) Set $X_{\boldsymbol{B}} = X \cdot (\prod_{j \notin \boldsymbol{B}} A_{j,0})^{-1}$
   (b) Sample $\{A_{b,0}\}_{b \in \boldsymbol{B}}$ subject to $\prod_{b \in \boldsymbol{B}} A_{b,0} = X_{\boldsymbol{B}}$.
   (c) 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.
   (d) 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 2b 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 $\Sigma_{\mathtt{schnorr}}$. 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.

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] J. Doerner, Y. Kondi, E. Lee, and a. shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019. doi: 10.1109/SP.2019.00024.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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 $\mathcal{X}$.

Key-Refresh:

Upon receiving `key-refresh` from all $\mathcal{P}_i \in \boldsymbol{P}$, send `key-refresh` to $\mathcal{S}$, and do:

- If not all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ are corrupted/quarantined, erase all records of $(\texttt{quarantine}, \ldots)$.

Corruption/Decorruption:

1. Upon receiving $(\texttt{corrupt}, \mathcal{P}_j)$ from $\mathcal{S}$, record $\mathcal{P}_j$ is corrupted.
2. Upon receiving $(\texttt{decorrupt}, \mathcal{P}_j)$ from $\mathcal{S}$:
   - If not all parties in some $\boldsymbol{Q} \in \mathbb{Q}$ are corrupted/quarantined do:
     If there is record that $\mathcal{P}_j$ is corrupted, erase it and record $(\texttt{quarantine}, \mathcal{P}_j)$.
   - Else do nothing.

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