Proactive Refresh for Accountable Threshold Signatures

Dan Boneh, Aditi Partap, Lior Rotem

Stanford University
{dabo,aditi712,lrotem}@cs.stanford.edu

Abstract. An *accountable* threshold signature (ATS) is a threshold signature scheme where every signature identifies the quorum of signers who generated that signature. They are widely used in financial settings where signers need to be held accountable for threshold signatures they generate. In this paper we initiate the study of *proactive refresh* for accountable threshold signatures. Proactive refresh is a protocol that lets the group of signers refresh their shares of the secret key, without changing the public key or the threshold. We give several definitions for this notion achieving different levels of security. We observe that certain natural constructions for an ATS cannot be proactively refreshed because the secret key generated at setup is needed for accountability. We then construct three types of ATS schemes with proactive refresh. The first is a generic construction that is efficient when the number of signers is small. The second is a hybrid construction that performs well for a large number of signers and satisfies a strong security definition. The third is a collection of very practical constructions derived from ATS versions of the Schnorr and BLS signature schemes; however these practical constructions only satisfy our weaker notion of security.

1 Introduction

A threshold signature scheme [27] protects the secret signing key by splitting it into n shares so that any t shares can sign. An *accountable* threshold signature scheme, also called an ATS, is a type of threshold scheme where the signature identifies the quorum set that generated the signature. In particular, there is a *tracing* algorithm that takes as input the public key pk, a message m, and a valid signature on m, and outputs a quorum $\mathcal{J} \subseteq [n]$ of size at least t that must have participated in generating the signature. More precisely, a set of signers $\mathcal{J}$ should be unable to cause the tracing algorithm to blame a signer outside of $\mathcal{J}$ for a signature generated by $\mathcal{J}$ (see Section 3 for the complete definition). Since every signature must encode the quorum that generated it, signature length must be at least $\lceil \log_2 \binom{n}{t} \rceil$ bits. We note that a non-accountable threshold signature scheme cannot be made accountable simply by requiring the signing quorum $\mathcal{J}$ to sign the pair $(m, \mathcal{J})$. The problem is that the signing quorum could lie and sign a pair $(m, \mathcal{J}')$ for some $\mathcal{J} \neq \mathcal{J}'$, thereby framing the quorum $\mathcal{J}'$ for a signature generated by $\mathcal{J}$.

Accountable threshold signatures (ATS) come up often in real-world settings: if a rogue transaction is signed by a threshold of trustees, the signature should identify the trustees responsible. For this reason, they are widely used in blockchain applications (e.g., [3]).

The trivial t-out-of-n ATS scheme is one where every signer locally generates a public-private key pair for a standard (non-threshold) signature scheme. The complete public key is the concatenation of all n local public keys. When t parties need to sign a message m, they each sign the message using their local secret key, and the final signature is the concatenation of all t signatures. The verifier accepts such an ATS signature if it contains t valid signatures. The tracing algorithm can trivially determine which parties participated in generating a given valid signature. This trivial ATS is used widely, for example in Bitcoin multisig transactions [3]. While the scheme has many benefits, signature size and verification time are at least linear in $t\lambda$, where λ is the security parameter. Several ATS constructions achieve much lower signature size and verification time [48, 7, 51, 14, 17, 16].

Proactive refresh. Consider an adversary that is able to corrupt one signing party every week and learn its key share. After t weeks the adversary will learn enough key shares to forge a signature on any message of its choice. To thwart such a dynamic adversary, Ostrovsky and Yung [52] introduced the concept of *proactive refresh.* Every epoch, say once a day, the n parties will engage in a protocol that refreshes their secret key shares without changing the public key. The requirement is that an adversary that corrupts fewer than t parties in every epoch will not be able to forge signatures, even though in aggregate the adversary may corrupt all n parties. Several subsequent works designed proactive refresh protocols for specific threshold systems [40, 39, 32, 31, 54, 5, 33, 21, 2, 42, 24, 44].

Our results. In this paper we initiate the study of proactive refresh for accountable threshold signatures (ATS). An ATS with proactive refresh, or $\mathsf{ATS\text{-}PR}$, is the same as an ATS with the addition of a share update protocol. At the beginning of every epoch all n parties participate in this update protocol to refresh their key shares, without changing the public key. The scheme must be unforgeable and accountable against an adversary that can corrupt a different set of parties at every epoch. We will define this more precisely in a minute.

At first, refreshing the secret keys of parties in an ATS may seem impossible. Each party's secret key is used to hold that party accountable for a rogue signature. If we refresh the party's secret key, the tracing algorithm will no longer be able to trace a rogue signature to that party. For example, in the trivial ATS described above it is not possible to refresh the secret keys without changing the public key: once an adversary learns the secret key of one party, it will always be able to forge signature shares on behalf of that party. Nevertheless, we show that by encoding a little more information in the signature, it is possible for the tracing algorithm to work correctly despite the fact that all the secret keys change at the beginning of every epoch.

In Section 3 we define a number of security models for an ATS-PR that capture multi-epoch unforgeability and accountability properties. We give two natural definitions of unforgeability, denoted $\mathsf{uf\text{-}0}$ and $\mathsf{uf\text{-}1}$, that are an adaption of the threshold unforgeability definitions of Bellare et al. [6] to the settings of proactive refresh. We next give two definitions of accountability, denoted $\mathsf{acc\text{-}0}$ and $\mathsf{acc\text{-}1}$. In $\mathsf{acc\text{-}1}$ the adversary can corrupt an arbitrary number of parties at every epoch, and can issue arbitrary signature queries to the parties at every epoch. Eventually, the adversary produces a message-signature pair (m^*, σ^*) that will trace to a signing set $\mathcal{J} \subseteq [n]$. We say that the adversary breaks accountability if in every epoch some party is incorrectly blamed for signing m^*. In more detail, if in some epoch e' the adversary obtained enough key shares and signature shares to sign m^* on behalf of the set $\mathcal{J}$, then the adversary did not break accountability — the set $\mathcal{J}$ effectively signed m^* at epoch e'. Therefore, to break accountability we require the adversary to satisfy the complementary condition: in every epoch e we require that there is some i_e in the set $\mathcal{J}$ for which the adversary did not corrupt party i_e in epoch e and did not ask party i_e to sign m^* in epoch e. In other words, the adversary wins if in every epoch some party is incorrectly blamed for signing m^*. The definition requires that no efficient adversary can satisfy this condition. We discuss this further in Section 3. Definition $\mathsf{acc\text{-}0}$ is weaker and requires that for some i in the set $\mathcal{J}$, the adversary never corrupted party i nor did it ever ask it to sign m^*, across all epochs. That is, the adversary wins under a more restricted condition: the same party i is incorrectly blamed for signing m^* across all epochs. We explore the differences between $\mathsf{acc\text{-}0}$ and $\mathsf{acc\text{-}1}$ in Appendix A. In summary, we obtain four notions of security denoted $(\mathsf{uf}\text{-}b \wedge \mathsf{acc}\text{-}b')$ for $b, b' \in \{0, 1\}$.

The security definitions in Section 3 require that the n parties honestly follow the update protocol. This captures security against an adversary that steals key shares, but does not otherwise corrupt the parties. It lets us focus on the main ideas needed to build an ATS with proactive refresh.

In Appendix B we consider a stronger adversary: we define security for an ATS-PR when some of the parties participating in the system are fully malicious. We then describe a generic compiler that lifts an ATS-PR that is only secure against semi-honest corruptions to an ATS-PR that is secure against malicious corruptions. The compiler makes use of techniques from maliciously secure multiparty computation. In Section 8 we discuss more efficient lifting techniques for our specific schemes.

Constructions. Next, we present five constructions. We begin with a generic combinatorial construction that performs well when $\binom{n}{t}$ is polynomial size. The scheme is built from any generic n-out-of-n threshold signature scheme (not necessarily accountable) that supports a proactive refresh. There are many examples built from RSA [40, 39, 32, 31, 54], Schnorr [39, 43, 47, 44], and BLS [19, 14]. It satisfies $\mathsf{uf\text{-}1} \wedge \mathsf{acc\text{-}1}$ security, our strongest notion of security.

In Section 5 we describe a generic construction that satisfies $\mathsf{uf\text{-}1} \wedge \mathsf{acc\text{-}1}$ security even when $\binom{n}{t}$ is large. The scheme is built by combining two schemes:

- a refreshable n-out-of-n threshold scheme $\mathcal{S}_1$ that is not accountable, and
- a t-out-of-n accountable threshold scheme $\mathcal{S}_2$ that is not-refreshable.

We build a two-level ATS-PR scheme where the scheme $\mathcal{S}_1$ is used to sign $\mathcal{S}_2$ public keys, and $\mathcal{S}_2$ is used to sign messages. At the beginning of epoch number i the parties do: (i) refresh their $\mathcal{S}_1$ secret keys, (ii) run a distributed key generation (DKG) protocol to generate fresh $\mathcal{S}_2$ secret keys and a public key pk_i; and (iii) sign the newly generated ATS public key pk_i using the scheme $\mathcal{S}_1$. A signature on a message m is a triple $(\mathsf{pk}_i, \sigma_1, \sigma_2)$, where σ_1 is the $\mathcal{S}_1$ signature on pk_i, and σ_2 is the $\mathcal{S}_2$ signature on m. To make this construction practical we need an ATS scheme $\mathcal{S}_2$ that has short public keys (so that our overall signature is short) and has an efficient DKG. In Section 6 we construct an ATS that has both properties: a constant size public key (i.e., its size is independent of t and n) and a simple DKG. Our construction makes use of techniques used to construct cryptographic accumulators.

We then turn to constructing a refreshable ATS from standard signature schemes such as Schnorr [56] and BLS [19]. In Section 7 we give very practical schemes that support proactive refresh for a Schnorr-ATS and a BLS-ATS. This leads to practical short ATS schemes that support proactive refresh. However, we describe an attack that shows that the schemes do not provide $\mathsf{acc\text{-}1}$ security. Instead, we prove that they provide $\mathsf{acc\text{-}0}$ and $\mathsf{uf\text{-}0}$ security (we discuss the requirements for proving $\mathsf{uf\text{-}1}$ in Section 7).

Future work. Our security definitions capture adaptive threshold adversaries (as in [45] and the references therein), where the adversary can choose to corrupt arbitrary parties before and after issuing signature queries in every epoch. Our constructions in Sections 4 and 5 inherit adaptive security from the underlying threshold schemes from which they are built. However, in Section 7, we analyze the practical Schnorr and BLS constructions in a semi-adaptive setting where the adversary must commit to its key queries at the beginning of every epoch, before issuing its adaptive signature queries. The reason is that achieving security against a fully adaptive adversary often leads to complex threshold signature schemes (as discussed in [45]). Future work can consider practical constructions for ATS-PR in the fully adaptive settings.

Additional related work. The notion of an accountable threshold signature (ATS) is closely related to the concept of a multisignature defined in [48] and further developed in [7, 17, 51, 41, 4, 16]. However, there are a number of differences. First, multisignatures are often viewed as a compression mechanism, compressing multiple signatures into one, not a threshold mechanism. The threshold is often left implicit. An ATS imposes an explicit threshold used by the verifier to decide if a signature is valid. Second, the syntax of an ATS allows for centralized key generation or an interactive distributed key generation protocol (DKG). Multisignatures often only allow for local key generation where every signer generates its key share by itself (however, there are some exceptions [16]).

Traditionally, threshold signatures come in two flavors: *fully private* (called PTS) where a signature reveals nothing about the threshold or the signing quorum, or *fully accountable* (called ATS), as in this paper. A recent proposal called TAPS [18] provides both properties: it is fully private to the public, but fully accountable to an authority that holds a secret tracing key.

2 Preliminaries

In this section we present the basic notions and cryptographic primitives that are used in this work. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \ldots, n\}$. For a distribution X we denote by $x \leftarrow\$ X$ the process of sampling a value x from the distribution X. Similarly, for a set $\mathcal{X}$ we denote by $x \leftarrow\$ \mathcal{X}$ the process of sampling a value x from the uniform distribution over $\mathcal{X}$.

2.1 Threshold Signatures

For simplicity of presentation, we begin by considering non-interactive threshold signature schemes, in which each signer can locally produce their signature shares without interacting with the other signers. In Appendix C we will formally consider interactive schemes as well.

Syntax. A non-interactive threshold signature scheme (see [20, 28, 19, 14, 43, 6] and the references therein) is defined with respect to some public parameters pp. Looking ahead, in our constructions pp will include the description of some cryptographic group. This description is typically generated as a function of the security parameter $\lambda \in \mathbb{N}$ by a group-generation algorithm, but in this work we will abstract this process away and fix the group as part of pp. To avoid over-cluttering in notation, we assume that pp are known to all algorithms without always providing it as an explicit input.

A threshold signature (TS) scheme is a 4-tuple $\mathsf{TS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf})$, where:

- KGen is the randomized key-generation algorithm. It takes in as input the public parameters pp, a number n of signers, and a threshold t, and outputs a public key pk, a public signature combination key pkc, and an n-vector of secret signing keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
- Sign is the randomized signing algorithm. It takes in as input a secret key sk_i and a message m, and outputs a signature share s_i.
- $\mathsf{Combine}$ is a deterministic signature combination algorithm. It takes in as input the combination key pkc and signature shares $s_{i_1}, \ldots, s_{i_\ell}$ and outputs either a signature σ or $\perp$. To simplify the notation, we will require throughout that every signature share encodes the signer from which it originated.
- Vf is the deterministic verification algorithm. It takes in the public key pk, a message m, and a signature σ, and outputs either 1 (implying acceptance) or 0 (implying rejection).

We postpone defining the correctness and security of TS schemes to Section 3, where we will present new generalizations of these definitions.

Accountable threshold signatures. An accountable threshold signature (ATS) scheme [49, 19, 4, 10, 16, 51] is a TS scheme equipped with an additional Trace algorithm. This algorithm takes as input the public key pk, a message m, and a signature σ. It outputs either a subset $\mathcal{J} \subseteq [n]$ of signers such that $|\mathcal{J}| \geq t$ (where t is the threshold provided to KGen) or $\perp$. Roughly speaking, the trace correctness property states that if a signature σ on m was produced by a subset $\mathcal{I} \subseteq [n]$ of the signers, then the subset $\mathcal{J}$ output by $\mathsf{Trace}(\mathsf{pk}, m, \sigma)$ is contained in $\mathcal{I}$. The accountability property asserts that no subset $\mathcal{I}$ of the signers can sign "on behalf" of a subset $\mathcal{J}$ which is not contained in $\mathcal{I}$ (i.e., it cannot make Trace output such a $\mathcal{J}$). We formally define these two properties in Section 3.

2.2 The Forking Lemma

In subsequent sections, we make use of the "forking lemma" of Bellare and Neven [7] (following Pointcheval and Stern [53]). Let $q \geq 1$ be an integer, let $\mathcal{H}$, $\mathcal{X}$ and $\mathcal{Y}$ be a sets such that $|\mathcal{H}| \geq 2$. Let $\mathcal{A}$ be a randomized algorithm that on input $(x, \vec{h}) \in \mathcal{X} \times \mathcal{H}^q$ returns either a pair $(i, y) \in [q] \times \mathcal{Y}$ or $\perp$. Let $\mathcal{F}_{\mathcal{A}}$ be an algorithm that takes an input $x \in \mathcal{X}$, and returns either an output $(y, y') \in \mathcal{Y}^2$ or $\perp$, and is defined as follows:

1. Sample random coins ρ for $\mathcal{A}$.
2. Sample $h_1, \ldots, h_q \leftarrow\$ \mathcal{H}$ and invoke $\mathsf{out}_1 \leftarrow \mathcal{A}(x, h_1, \ldots, h_q; \rho)$.
3. If $\mathsf{out}_1 = \perp$, return $\perp$. Otherwise, let $\mathsf{out}_1 = (i, y)$.
4. Sample $h'_i, \ldots, h'_q \leftarrow\$ \mathcal{H}$ and invoke $\mathsf{out}_2 \leftarrow \mathcal{A}(x, h_1, \ldots, h_{i-1}, h'_i, \ldots, h_q; \rho)$.
5. If $\mathsf{out}_2 = \perp$, return $\perp$. Otherwise, let $\mathsf{out}_2 = (i', y')$.
6. If $i' = i$ and $h_i \neq h'_i$ then return (y, y'). Otherwise, return $\perp$.

The forking lemma (Lemma 1 below) relates the probability that $\mathcal{F}_{\mathcal{A}}$ succesfully provides an output (other than $\perp$) to the probability that $\mathcal{A}$ provides an output.

Lemma 1 ([7]). *For all distributions D over $\mathcal{X}$, it holds that*

$$\Pr\left[\mathcal{F}_{\mathcal{A}}(x) \neq \perp \;:\; x \leftarrow\$ D\right] \geq \epsilon_D \cdot \left(\frac{\epsilon_D}{q} - \frac{1}{h}\right),$$

where ϵ_D is defined as

$$\epsilon_D := \Pr\left[\mathcal{A}(x, \vec{q}) \neq \perp \;:\; \begin{array}{c} x \leftarrow\$ D \\ \vec{h} \leftarrow\$ \mathcal{H}^q \end{array}\right].$$

3 Accountable Threshold Signatures with Proactive Refresh

In this section we present our definitions for ATS schemes with proactive refresh (ATS-PR). We start by providing the syntactic additions for such schemes (when compared to standard ATS schemes), then define the correctness properties that should be satisfied by them, and finally, we present new security notions. Recall that to simplify the presentation, we first focus on non-interactive schemes, and then formally consider interactive schemes in Appendix C.

3.1 Syntax and Correctness

The key-update procedure. An ATS-PR scheme is an ATS scheme that is additionally equipped with a key-update procedure, whose role is to refresh the signers' secret keys without modifying the public key in any way. We can envision the key-update procedure as dividing time into epochs. An epoch starts once one execution of the key-update procedure ends (or, for the first epoch right after the invocation key generation algorithm), and ends when the next execution of the key-update procedure ends.

Formally, the key-update procedure is a pair $\mathsf{Update} = (\mathsf{Update}_0, \mathsf{Update}_1)$ of algorithms:

– Update_0 is a randomized algorithm that takes in a secret key sk_i^e of signer i in epoch e and the public key pk, and outputs a vector $(\delta_{i,1}^e, ..., \delta_{i,n}^e)$ of update messages. Each signer i sends $\delta_{i,j}^e$ to the jth signer, for all $j \neq i$.
– Update_1 is a deterministic algorithm that takes in a secret key sk_i^e and n update messages $\delta_{1,i}^e, \ldots, \delta_{n,i}^e$. It outputs an updated secret key sk_i^{e+1} for epoch $e+1$ for signer i.

For succinctness, we may write $(\mathsf{sk}_1^{e+1}, \ldots, \mathsf{sk}_n^{e+1}) \leftarrow\$ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^e, \ldots, \mathsf{sk}_n^e)$ as a shorthand for the random process of first invoking $\mathsf{Update}_0(\mathsf{sk}_i^e, \mathsf{pk})$ for every $i \in [n]$ to randomly sample n^2 update messages $\{\delta_{i,j}\}_{i,j \in [n]}$; and then running $\mathsf{Update}_1(\mathsf{sk}_i^e, (\delta_{1,i}^e, \ldots, \delta_{n,i}))$ to obtain sk_i^{e+1} for every $i \in [n]$.

Correctness. Informally, the basic correctness requirement for ATS-PR schemes is that Vf should accept honestly-generated signatures in all epochs.

Definition 1 (correctness). *We say that an ATS-PR scheme* $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *is* **correct** *if for all public parameters* pp*, all messages m in the associated message space* $\mathcal{M}_{\mathsf{pp}}$*, all positive integers n, t and e such that $t \leq n$, and all subsets $\mathcal{J} \subseteq [n]$ of size at least t, it holds that*

$$\Pr\big[\mathsf{Vf}(\mathsf{pk}, m, \mathsf{Combine}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j^e, m)\}_{j \in \mathcal{J}})) = 1\big] = 1,$$

where the probability is over the random variables $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\$ \mathsf{KGen}(\mathsf{pp}, n, t)$*, and* $(\mathsf{sk}_1^{i+1}, \ldots, \mathsf{sk}_n^{i+1}) \leftarrow\$ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^i, \ldots, \mathsf{sk}_n^i)$ *for $i = 1, \ldots, e-1$, and the random coins of* Sign*.*

In addition to the traditional correctness property, an ATS-PR scheme should also provide *trace correctness*. That is, on input a public-key pk, a message m, and a signature σ, the tracing algorithm Trace should output a subset of the set of keys used to generate σ. This should hold irrespective of the epoch in which σ was generated.

Definition 2 (trace correctness). *We say that an ATS-PR scheme* $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *satisfies* **trace correctness** *if for all public parameters* pp*, all messages m in the associated message space* $\mathcal{M}_{\mathsf{pp}}$*, all positive integers n, t and e such that $t \leq n$, and all subsets $\mathcal{J} \subseteq [n]$ of size at least t, it holds that*

$$\Pr\big[\mathsf{Trace}(\mathsf{pk}, m, \mathsf{Combine}(\mathsf{pkc}, \{\mathsf{Sign}(\mathsf{sk}_j^e, m)\}_{j \in \mathcal{J}})) \subseteq \mathcal{J}\big] = 1,$$

where the probability is over the random variables $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\$ \mathsf{KGen}(\mathsf{pp}, n, t)$*, and* $(\mathsf{sk}_1^{i+1}, \ldots, \mathsf{sk}_n^{i+1}) \leftarrow\$ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^i, \ldots, \mathsf{sk}_n^i)$ *for $i = 1, \ldots, e-1$, and the random coins of* Sign*.*

3.2 Security Notions for ATS-PR Schemes

We now turn to present our notions of security for ATS-PR schemes. We start with a brief overview of the security notions and then provide formal definitions.

An ATS-PR scheme should satisfy two security requirements, unforgeability and accountability, which extend the traditional security notions of ATS schemes to the setting of proactive refreshes.

Unforgeability. The traditional unforgeability requirement for threshold signatures asserts that an adversary cannot produce a valid signature on a message m without observing either the secret key or a signature share on m of at least t different signers. In the proactive refresh setting, we require that this holds *per epoch*. That is, the adversary should not be able to produce a signature on a message m, unless *there is a specific epoch* in which it observed the secret keys or signature shares on m of at least t signers. Note that this means that the adversary is allowed to observe t or more secret keys or signature shares on m across epochs, and potentially even observe the secret keys of all signers at different points in time. However, in each specific epoch, the total number of secret keys and signature shares on m that the adversary observes should be strictly less than t. Following [6], we consider two flavors of this unforgeability definition, denoted uf-0 and uf-1, depending on whether or not the adversary is allowed to observe signature shares on the message m^* for which it forges a signature. We present constructions satisfying both notions with different trade-offs.

Accountability. The accountability property of ATS schemes states that an adversary should not be able to produce a valid signature on a message m on behalf of a subset $\mathcal{J}$ of signers without observing the secret key or signature share on m of all the signers in $\mathcal{J}$. In the proactive refresh setting we present a strong accountability definition that requires that this restriction on the adversary should only hold in each epoch (thus allowing them to observe secret keys/signature shares on m of all signers in $\mathcal{J}$ across different epochs). We also consider a milder accountability definition, which requires that for some signer $j \in \mathcal{J}$, the adversary never observes j's secret key or a signature share of j on m. Looking ahead, we will present different constructions of ATS-PR schemes satisfying the two notions. Appendix A explores the differences between these two notions.

Note that even under our strong accountability definition, if the adversary learns all the secret keys (or signature shares on m) of $\mathcal{J}$ in *the same epoch*, say epoch 1, then they can forever produce signatures on m, even in future epochs. This is inherent, since we want the public verification key to remain the same over time, rendering the verification algorithm oblivious to the epoch in which the message was signed.

Game-based security definitions. We use security games to define the above security notions for ATS-PR schemes, following the framework of Bellare and Rogaway [9]. A game $\mathbf{G}$ consists of an adversary $\mathcal{A}$ interacting with the challenger. The game is specified by a main procedure and possibly additional oracle procedures, which describe the manner in which the challenger replies to oracle queries issued by the adversary. We denote by $\mathbf{G}(\mathcal{A})$ the output of $\mathbf{G}$ when executed with an adversary $\mathcal{A}$. This $\mathbf{G}(\mathcal{A})$ is a random variable defined over the randomness of both $\mathcal{A}$ and the random choices of the game's main procedure and oracles.

For an ATS-PR scheme PRATS and public parameters pp, the above-described requirements are captured by the security games defined in Figure 1. All games are defined similarly, and the only difference between them is the winning condition for the adversary (i.e., the condition that results in the game outputting 1). In all games, the adversary first specifies the number n of overall signers, the threshold t, and the number E of epochs. This is followed by challenger sampling keys for all E epochs using the KGen and Update procedures of PRATS. The adversary then interacts with the challenger using two types of queries: Secret-key queries and signature queries. A secret-key query (e, i) reveals to the adversary the secret key of signer i in epoch e. A signature query (m, e, i) provides the adversary with an honestly-generated signature share on m with respect to signer i's secret key in epoch e. Finally, in all games the adversary should produce a valid forgery; that is, a message m^* and a signature σ^* that passes verification. Each game has additional restrictions on the adversary in light of our informal discussion above. Games $\mathbf{G}^{\text{uf-0}}_{\mathsf{PRATS}[\mathsf{pp}]}$ and $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRATS}[\mathsf{pp}]}$ capture two notions of unforgeability, whereas games $\mathbf{G}^{\text{acc-0}}_{\mathsf{PRATS}[\mathsf{pp}]}$ and $\mathbf{G}^{\text{acc-1}}_{\mathsf{PRATS}[\mathsf{pp}]}$ capture two notions of accountability. For $b, b' \in \{0, 1\}$, we also define the game $\mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}$, that captures schemes that satisfy both unforgeability and accountability. This will help us state and prove our theorem statements more succinctly.

Games $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS}[\mathsf{pp}]}, \mathbf{G}^{\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}, \mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}$

$$
\begin{array}{l}
\mathsf{flag}_{\text{uf-0}}, \mathsf{flag}_{\text{uf-1}}, \mathsf{flag}_{\text{acc-0}}, \mathsf{flag}_{\text{acc-1}} \leftarrow 0 \\
(\mathsf{st}, n, t, E) \leftarrow \mathcal{A}(\mathsf{pp}) \\
(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp}, n, t) \\
\textbf{for } e = \{2, \ldots, E\} \textbf{ do} \\
\quad (\mathsf{sk}_1^e, \ldots, \mathsf{sk}_n^e) \leftarrow\!\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^{e-1}, \ldots, \mathsf{sk}_n^{e-1}) \\
(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot), \mathsf{SignO}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc}) \\
\textbf{if } \mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 0 \textbf{ then} \\
\quad \textbf{return } 0 \\
\textbf{if } \forall e \in [E], |\mathcal{Q}_e^{\mathsf{sk}}| < t \wedge |\mathcal{Q}_e^{\mathsf{sig}}(m^*)| = 0 \textbf{ then } \mathsf{flag}_{\text{uf-0}} \leftarrow 1 \\
\textbf{if } \forall e \in [E], |\mathcal{Q}_e^{\mathsf{sk}} \cup \mathcal{Q}_e^{\mathsf{sig}}(m^*)| < t \textbf{ then } \mathsf{flag}_{\text{uf-1}} \leftarrow 1 \\
\textbf{if } \mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \cup_{e\in[E]} \left(\mathcal{Q}_e^{\mathsf{sk}} \cup \mathcal{Q}_e^{\mathsf{sig}}(m^*)\right) \textbf{ then } \mathsf{flag}_{\text{acc-0}} \leftarrow 1 \\
\textbf{if } \forall e \in [E], \mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \mathcal{Q}_e^{\mathsf{sk}} \cup \mathcal{Q}_e^{\mathsf{sig}}(m^*) \textbf{ then } \mathsf{flag}_{\text{acc-1}} \leftarrow 1 \\
\text{Only game } \mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS}[\mathsf{pp}]} : \textbf{return } \mathsf{flag}_{\text{uf-}b} \\
\text{Only game } \mathbf{G}^{\text{acc-}b}_{\mathsf{PRATS}[\mathsf{pp}]} : \textbf{return } \mathsf{flag}_{\text{acc-}b} \\
\text{Only game } \mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]} : \textbf{return } \mathsf{flag}_{\text{uf-}b} \vee \mathsf{flag}_{\text{acc-}b'}
\end{array}
$$

Oracle $\mathsf{skO}(e, i)$

$$
\begin{array}{l}
\mathcal{Q}_e^{\mathsf{sk}} \leftarrow \mathcal{Q}_e^{\mathsf{sk}} \cup \{i\} \\
\textbf{return } \mathsf{sk}_i^e
\end{array}
$$

Oracle $\mathsf{SignO}(m, e, i)$

$$
\begin{array}{l}
\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}_i^e, m) \\
\mathcal{Q}_e^{\mathsf{sig}}(m) \leftarrow \mathcal{Q}_e^{\mathsf{sig}}(m) \cup \{i\} \\
\textbf{return } \sigma_i
\end{array}
$$

Fig. 1. The security games $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS}[\mathsf{pp}]}, \mathbf{G}^{\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}, \mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}$ for $b, b' \in \{0, 1\}$ for an ATS-PR scheme $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ with public parameters pp. For a set $\mathcal{X}$ and an element x, we let $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$ be a shorthand for the following operation: If $\mathcal{X}$ was previously defined, then set $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$; if $\mathcal{X}$ is still undefined, then set $\mathcal{X} = \{x\}$.

Three remarks regarding the games in Figure 1 are in order:

- By convention, we assume that that $\bot \not\subseteq \mathcal{Q}$ for any set $\mathcal{Q}$. Hence, if the adversary successfully outputs a valid signature σ^* on a message m^* such that $\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) = \bot$, then the adversary breaks even our weak accountability notion (that is, wins the acc-0 security game).
- If we add the syntactic requirement that Trace never outputs $\bot$ on a signature that passes verification, then acc-1 security implies uf-1 security. This is because under this requirement, Trace always outputs a subset $\mathcal{J}$ of size at least t on valid signatures. If in each epoch the adversary corrupted at most $t-1$ signers, then in each epoch there must be at least one signer in $\mathcal{J}$ which is uncorrupted by the adversary.

– For simplicity of presentation, we start with a definition in which the challenger samples the keys for all epochs at the beginning of the games. The adversary cannot influence the key updates and receives no additional information about the key updates other than what is revealed by the answers to its secret key queries and signing queries. In Appendix B we consider stronger security notions, in which the adversary can corrupt signers (either semi-honestly or maliciously) during the key update protocol as well.

Definition 3 below defines the advantage of an adversary $\mathcal{A}$ in the eight games defined in Figure 1 as the probability that the games output 1 when executed with $\mathcal{A}$.

Definition 3. *Let* $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *be an ATS-PR scheme with public parameters* pp *and let* $\mathrm{prop} \in \{\text{uf-}b, \text{acc-}b, \text{uf-}b{\wedge}\text{acc-}b'\}_{b,b'\in\{0,1\}}$. *The advantage of an adversary* $\mathcal{A}$ *in* $\mathbf{G}^{\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ *is defined as*

$$\mathsf{Adv}^{\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \Pr\left[\mathbf{G}^{\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) = 1\right].$$

Threshold signatures without accountability or proactive refresh. In subsequent sections we will consider threshold signature (TS) schemes with proactive refresh but without accountability (i.e., without a Trace algorithm). These can be treated as a specific case of ATS-PR schemes in which the Trace algorithm is trivial (returns $\perp$ on all inputs). As such, games $\mathbf{G}^{\text{uf-0}}_{\mathsf{PRATS}[\mathsf{pp}]}$ and $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRATS}[\mathsf{pp}]}$ and Definition 3 readily captures the unforgeability property of such schemes. In addition, we will consider ATS schemes without proactive refresh (i.e., no Update procedure). These can also be treated as a special case of ATS-PR schemes in which the number of epochs is fixed at 1. The games defined in Figure 1 together with Definition 3 define the unforgeability and accountability of such schemes by having the game fix the number of epochs E to 1 (instead of receiving it from $\mathcal{A}$).

Semi-adaptive adversaries. The security games as defined in Figure 1 allow for fully-adaptive adversaries, in the sense that they do not pose any restrictions on the order in which the adversary decides on its oracle queries. Proving security against such adversaries is known to be a challenging task, already for non-accountable threshold signature schemes [45]. Since the problem of fully-adaptive adversaries is not at the focus of this work, we also consider semi-adaptive adversaries. For every epoch e, such adversaries are restricted to issuing all secret-key queries for that epoch before issuing their signature queries for this epoch. This is captured by modifying the security games as follows. The game will maintain a set $\mathcal{E}$ which will include all epochs for which a signature query has been issued by the adversary. On input (e, i), the oracle skO will first check if e is in $\mathcal{E}$. If so, it will ignore the query, returning $\perp$. Otherwise, it will continue as defined in Figure 1. This ensures that at every epoch e the adversary must issue all of its key queries for epoch e before issuing a signature query in epoch e.

For a ATS-PR scheme PRATS with public parameters pp, and for a security property $\mathrm{prop} \in \{\text{uf-}b, \text{acc-}b, \text{uf-}b \wedge \text{acc-}b'\}_{b,b'\in\{0,1\}}$, denote by $\mathbf{G}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ the semi-adaptive security game obtained from $\mathbf{G}^{\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ as described above. The advantage of an adversary in these games is defined similarly to the adversarial advantage in the fully-adaptive security games.

Definition 4. *Let* $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *be an ATS-PR scheme with public parameters* pp *and let* $\mathrm{prop} \in \{\text{uf-}b, \text{acc-}b, \text{uf-}b{\wedge}\text{acc-}b'\}_{b,b'\in\{0,1\}}$. *The advantage of an adversary* $\mathcal{A}$ *in* $\mathbf{G}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ *is defined as*

$$\mathsf{Adv}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \Pr\left[\mathbf{G}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) = 1\right].$$

Our constructions in Sections 4 and 5 will assume a (non-accountable) TS scheme with proactive refresh as a building block. In terms of the adaptiveness of the adversary, our constructions will inherit the security guarantees of the assumed TS scheme. Hence, in these sections we will not address the question of adaptivity directly. In Section 7, we will present direct constructions of ATS-PR schemes, and prove them secure against semi-adaptive adversaries (we will remind the reader of this fact in these sections). We leave the task of extending these constructions to handle fully-adaptive adversaries as an interesting open question for future work.

Extending the definitions to the random oracle model. All of the syntactic and security definitions above extend to the random oracle model by granting all algorithms, including the adversary $\mathcal{A}$, oracle access to a function H chosen uniformly at random from a family $\mathcal{H}$ of functions. In the correctness and security definitions (Definition 3), all probabilities are then also taken over the choice of H.

4 A Combinatorial Construction for Few Signers

In this section we present a simple combinatorial construction of an ATS-PR scheme from any TS-PR scheme (that is, from any threshold signature scheme with proactive refresh but no accountability assurances). The ATS-PR scheme incurs an overhead of $\binom{n}{t}$ in the length of the public key and the secret keys, as well as in the running time of the key generation algorithm, where n is the total number of potential signers and t is the threshold. Hence, this scheme is of interest in the setting where n is relatively small.

Our combinatorial ATS-PR scheme, which we denote by CATS, uses as a building block a TS-PR scheme $\mathsf{PRTS} = (\mathsf{PRTS.KGen}, \mathsf{PRTS.Sign}, \mathsf{PRTS.Combine}, \mathsf{PRTS.Vf}, \mathsf{PRTS.Update} = (\mathsf{PRTS.Update}_0, \mathsf{PRTS.Update}_1))$. For ease of presentation, we assume that PRTS is a non-interactive scheme (recall Sections 2.1 and 3), which will result in our CATS being non-interactive as well. However, the construction and security proof readily extend to interactive schemes as well.[1]

In the presentation of CATS, we rely on the following notation: Let $\mathcal{S}_{t,n}$ denote the collection of all t-sized subsets of $[n]$. For $i \in [n]$, let $\mathcal{S}_{t,n}(i)$ denote the collection of all such subsets that include index i.

CATS: a combinatorial ATS-PR scheme (built from a TS-PR scheme PRTS)

$\underline{\mathsf{CATS.KGen}(\mathsf{pp}, n, t)}$:

1. Set $\mathsf{pk} \leftarrow \varepsilon$ and $\mathsf{sk}_i \leftarrow \varepsilon$ for each $i \in [n]$.
2. For each $\mathcal{J} \in \mathcal{S}_{t,n}$:
   (a) Sample $(\mathsf{pk}^{\mathcal{J}}, \mathsf{pkc}^{\mathcal{J}}, (\mathsf{sk}_1^{\mathcal{J}}, \ldots, \mathsf{sk}_t^{\mathcal{J}})) \leftarrow\!\!\$\ \mathsf{PRTS.KGen}(\mathsf{pp}, t, t)$.
   (b) Update $\mathsf{pk} \leftarrow \mathsf{pk} \| \mathsf{pk}^{\mathcal{J}}$ and $\mathsf{pkc} \leftarrow \mathsf{pkc} \| \mathsf{pkc}^{\mathcal{J}}$.
   (c) For each $j \in \mathcal{J}$, update $\mathsf{sk}_j \leftarrow \mathsf{sk}_j \| \mathsf{sk}_j^{\mathcal{J}}$.
3. Output $(\mathsf{pk} = (n, t, \mathsf{pk}), \mathsf{pkc} = (n, t, \mathsf{pkc}), (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\underline{\mathsf{CATS.Sign}(\mathsf{sk}_i, m)}$:

1. Parse sk_i as $\left(\mathsf{sk}_i^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.
2. For all $\mathcal{J} \in \mathcal{S}_{t,n}(i)$, compute $\sigma_i^{\mathcal{J}} \leftarrow\!\!\$\ \mathsf{PRTS.Sign}(\mathsf{sk}_i^{\mathcal{J}}, m)$.
3. Output $\sigma_i = (\sigma_i^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.

[1] We actually only need PRTS to be an n-out-of-n threshold signature scheme, which may be simpler to construct. To avoid introducing additional definitions, we present the construction assuming PRTS is a general TS-PR.

$\underline{\mathsf{CATS.Combine}(\mathsf{pkc}, \sigma_{i_1}, \ldots, \sigma_{i_\ell})}$:

1. Parse pkc as $\left(\mathsf{pkc}^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,n}}$ and σ_{i_j} as $(\sigma_{i_j}^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i_j)}$ for each $j \in [\ell]$.
2. Let $\mathcal{J}^* = \{j_1, \ldots, j_t\}$ be the lexicographically first t-sized subset of $\{i_1, \ldots, i_\ell\}$.
3. Compute $\sigma \leftarrow \mathsf{PRTS.Combine}(\mathsf{pkc}^{\mathcal{J}^*}, \sigma_{j_1}^{\mathcal{J}^*}, \ldots, \sigma_{j_t}^{\mathcal{J}^*})$.
4. Output $\sigma' = (\mathcal{J}^*, \sigma)$.

$\underline{\mathsf{CATS.Vf}(\mathsf{pk}, m, \sigma')}$:

1. Parse pk as (n, t, pk), pk as $\left(\mathsf{pk}^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,N}}$, and σ' as $(\mathcal{J}^*, \sigma)$.
2. If $\mathcal{J}^* < t$, output 0.
3. Output 1 if $\mathsf{PRTS.Vf}(\mathsf{pk}^{\mathcal{J}^*}, m, \sigma) = 1$.
   Otherwise, output 0.

$\underline{\mathsf{CATS.Trace}(\mathsf{pk}, m, \sigma)}$:

1. Parse σ' as $(\mathcal{J}^*, \sigma)$.
2. Output $\mathcal{J}^*$.

$\underline{\mathsf{CATS.Update}}$:

– $\mathsf{CATS.Update}_0(\mathsf{pk}, \mathsf{sk}_i)$:
  1. Parse pk as $\left(\mathsf{pk}^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,n}}$ and sk_i as $\left(\mathsf{sk}_i^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.
  2. For each $\mathcal{J} = (j_1, \ldots, j_t) \in \mathcal{S}_{t,n}(i)$:
     Sample $(\delta_{j_1}^{\mathcal{J}}, \ldots, \delta_{j_t}^{\mathcal{J}}) \leftarrow\!\!\$\ \mathsf{PRTS.Update}_0(\mathsf{pk}^{\mathcal{J}}, \mathsf{sk}_i^{\mathcal{J}})$.
  3. For each $j \in [n]$, set $\delta_{i,j} \leftarrow (\delta_j^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i) \cap \mathcal{S}_{t,n}(j)}$.
  4. Output $(\delta_{i,1}, \ldots, \delta_{i,n})$.

– $\mathsf{CATS.Update}_1(\mathsf{sk}_i, \delta_{1,i}, \ldots, \delta_{n,i})$:
  1. Parse sk_i as $\left(\mathsf{sk}_i^{\mathcal{J}}\right)_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$ and parse $\delta_{j,i}$ as $(\delta_i^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i) \cap \mathcal{S}_{t,n}(j)}$ for each $j \in [N]$.
  2. For each $\mathcal{J} = \{j_1, \ldots, j_t\} \in \mathcal{S}_{t,n}(i)$:
     Set $\widetilde{\mathsf{sk}_i^{\mathcal{J}}} \leftarrow\!\!\$\ \mathsf{PRTS.Update}_1(\mathsf{sk}_i^{\mathcal{J}}, \delta_{j_1}^{\mathcal{J}}, \ldots, \delta_{j_t}^{\mathcal{J}})$.
  3. Output $\widetilde{\mathsf{sk}_i} = (\widetilde{\mathsf{sk}_i^{\mathcal{J}}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.

Theorem 1 below, which is proven is Appendix D.1, reduces the security of CATS to that of the underlying scheme PRTS.

Theorem 1. *Let $b \in \{0, 1\}$. For any adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that for every public parameters pp it holds that*

$$\mathsf{Adv}^{\text{uf-b}\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}(\mathcal{A}) \leq \mathsf{bin}_{max} \cdot \mathsf{Adv}^{\text{uf-b}}_{\mathsf{PRTS[pp]}}(\mathcal{B}),$$

where bin_{max} is a bound on the binomial coefficient $\binom{n}{t}$ for n, t chosen by $\mathcal{A}$ in $\mathbf{G}^{\text{uf-}b\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}$.

On the tightness of the reduction. As evident in the statement of Theorem 1, the reduction used in its proof incurs a loss of factor bin_{max}. This may be reasonable when the total number n of potential signers is small, which is precisely the scenario in which one might use CATS. Nevertheless, we stress that this loss is a matter of definitional choices. In more detail, CATS uses $\binom{n}{t}$ independent instances PRTS. However, we chose to define our security games in Figure 1 in a manner that only captures single-instance security.[2] This allows us to more cleanly present the main contributions of the paper. As a result, the proof of Theorem 1 encompasses a multi-instance to single-instance

[2] Though we stress that our definition for interactive protocols (Appendix C) does capture concurrent executions of the signing protocol.

reduction for threshold signatures, resulting in a loss which is linear in the number of instances. If we were to define multi-instance security of TS-PR schemes (by a straightforward generalization of the games in Figure 1), then we would have a straightforward and tight reduction from the security of CATS to the multi-instance security of PRTS. In particular, if PRTS is instantiated using a scheme which has better-than-trivial multi-instance security, than the loss in Theorem 1 shrinks accordingly.

5 An Efficient Construction with Strong Security Guarantees

In this section we present a generic approach for obtaining ATS-PR schemes. The benefit of this approach over the one presented in Section 4 is that it has efficient instantiations even when $\binom{n}{t}$ is large. The ATS-PR scheme makes use of two basic schemes: An n-out-of-n TS-PR scheme (without accountability) and a t-out-of-n ATS scheme (without proactive refresh). The idea is to set the TS-PR public key as the public key of the new scheme. To refresh the secret keys of the new scheme, we refresh the secret keys of the TS-PR scheme and generate fresh epoch-specific keys for the ATS schemes. The epoch-specific ATS keys are then used to sign messages. To enforce consistency, in each update the new ATS public key is signed using the n-out-of-n TS-PR scheme. The epoch-specific ATS public key and the signature on it are then appended to signatures issued using the epoch-specific ATS signing keys.

We now formally present our generic ATS-PR scheme. The construction relies on the following two building blocks:

1. A threshold signature scheme with proactive refresh $\mathsf{PRTS} = (\mathsf{PRTS.KGen}, \mathsf{PRTS.Sign}, \mathsf{PRTS.Combine}, \mathsf{PRTS.Vf}, \mathsf{PRTS.Update})$.[3]
2. An ATS scheme $\mathsf{ATS} = (\mathsf{ATS.KGen}, \mathsf{ATS.Sign}, \mathsf{ATS.Combine}, \mathsf{ATS.Vf}, \mathsf{ATS.Trace})$. We assume that ATS is equipped with a distributed key generation protocol $\Pi_{\mathsf{ATS.KGen}}$ enabling signers to generate the keys for the scheme in a distributed manner.

When presenting our ATS scheme with proactive refresh we use the following notation. We write $\sigma \leftarrow\!\!\$\ \mathsf{PRTS.Sign}((\mathsf{sk}_1, \ldots, \mathsf{sk}_n), \mathsf{pkc}, m)$ to denote the process of simulating the execution of the (potentially interactive) signing protocol $\mathsf{PRTS.Sign}$, where the i-th signer runs on local input (sk_i, m) and σ is the result of applying $\mathsf{PRTS.Combine}$ onto the local outputs of the protocol with key pkc. When presenting the signing procedure, we do so in a general language that also captures interactive protocols. In particular, we also provide the (potentially interactive) Sign algorithm with the subset $\mathcal{J}$ of signers as input (we refer the reader to Section C for a formal definition of interactive ATS-PR schemes).

Our ATS-PR scheme, called PRATS, is then defined as follows.[4]

PRATS: A generic ATS scheme with proactive refresh (built from PRTS and ATS)

$\underline{\mathsf{PRATS.KGen}(\mathsf{pp}, n, t)}$:

[3] As in Section 4, we only need PRTS to support n-out-of-n signing, which may be easier to instantiate than general threshold signature schemes.

[4] Though the underlying PRTS and ATS scheme might be defined relative to a random oracle, we abstract this fact away for simplicity of presentation. The proof of security would remain essentially unchanged without this simplification.

1. Sample
   $(\mathsf{PRTS.pk}, \mathsf{PRTS.pkc}, (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n)) \leftarrow\$ \mathsf{PRTS.KGen}(\mathsf{pp}, n, n)$.
2. Sample $(\mathsf{ATS.pk}, \mathsf{ATS.pkc}, (\mathsf{ATS.sk}_1, \ldots, \mathsf{ATS.sk}_n)) \leftarrow\$ \mathsf{ATS.KGen}(\mathsf{pp}, n, t)$.
3. Compute $\sigma_{\mathsf{pk}} \leftarrow\$ \mathsf{PRTS.Sign}((\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n), \mathsf{PRTS.pkc}, \mathsf{ATS.pk})$.
4. For $i = 1, \ldots, n$ set $\mathsf{sk}_i \leftarrow (\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}})$.
5. Output $(\mathsf{pk} = (n, t, \mathsf{PRTS.pk}),\ \mathsf{pkc} = \mathsf{ATS.pkc},\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\underline{\mathsf{PRATS.Sign}(\mathsf{sk}_i, m, \mathcal{J})\text{:}}$

1. Parse sk_i as $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}})$.
2. Invoke $\mathsf{ATS.Sign}(\mathsf{ATS.sk}_i, m, \mathcal{J})$ and let s_m denote the output of the protocol.
3. Output $s_i = (\mathcal{J}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, s_m)$.

$\underline{\mathsf{PRATS.Combine}(\mathsf{pkc}, (s_{i_1}, \ldots, s_{i_\ell}))\text{:}}$

1. Parse each s_i as $(\mathcal{J}_i, \mathsf{ATS.pk}_i, \sigma_{\mathsf{pk},i}, s_{m,i})$.
2. Let $(\mathcal{J}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}}) \leftarrow (\mathcal{J}_{i_1}, \mathsf{ATS.pk}_{i_1}, \sigma_{\mathsf{pk},i_1})$.
   If for some $j \in [\ell]$ it holds that $(\mathcal{J}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}}) \neq (\mathcal{J}_{i_j}, \mathsf{ATS.pk}_{i_j}, \sigma_{\mathsf{pk},i_j})$, output $\perp$.
3. Invoke $\sigma_m \leftarrow \mathsf{ATS.Combine}(\mathsf{pkc}, s_{m,i_1}, \ldots, s_{m,i_\ell})$.
4. Output $\sigma = (\mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, \sigma_m)$.

$\underline{\mathsf{PRATS.Vf}(\mathsf{pk}, m, \sigma)\text{:}}$

1. Parse pk as $(n, t, \mathsf{PRTS.pk})$ and σ as $(\mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, \sigma_m)$.
2. Output 1 if $\mathsf{PRTS.Vf}(\mathsf{PRTS.pk}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}})$ and $\mathsf{ATS.Vf}(\mathsf{ATS.pk}, m, \sigma_m)$.
   Otherwise, output 0.

$\underline{\mathsf{PRATS.Trace}(\mathsf{pk}, m, \sigma)\text{:}}$

1. Parse σ as $(\mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, \sigma_m)$.
2. Verify that $\mathsf{PRTS.Vf}(\mathsf{PRTS.pk}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}}) = 1$ and otherwise, output $\perp$.
3. Output $\mathcal{J} = \mathsf{ATS.Trace}(\mathsf{ATS.pk}, m, \sigma_m)$.

$\underline{\mathsf{PRATS.Update}(\mathsf{sk}_i, \mathsf{pk})\text{:}}$

1. Parse sk_i as $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}})$ and pk as $(n, t, \mathsf{PRTS.pk})$.
2. Run the protocol $\mathsf{PRTS.Update}$ with signer i running on local input $(\mathsf{PRTS.sk}_i, \mathsf{PRTS.pk})$ and let $\mathsf{PRTS.sk}'_i$ be the output of signer i.
3. Invoke $\Pi_{\mathsf{ATS.KGen}}(n, t)$ and let $(\mathsf{ATS.pk}, \mathsf{ATS.sk}'_i)$ denote the output of signer i.
4. Invoke $\mathsf{PRTS.Sign}(\mathsf{PRTS.pk.PRTS.sk}'_i, \mathsf{ATS.pk})$ and let σ'_{pk} denote the output of the protocol.
5. Output $\mathsf{sk}'_i = (\mathsf{PRTS.sk}'_i, \mathsf{ATS.sk}'_i, \mathsf{ATS.pk}', \sigma'_{\mathsf{pk}})$.

On the efficiency of the scheme. The public key of PRATS consists solely of the public key of the non-accountable scheme PRTS (in addition to n and t), for which we instantiations with short public keys are known (the reader is referred to Section 1 and the references therein). Two main efficiency measures that depend on ATS are:

– Length of signatures, which consist of a public key of ATS, an ATS signature, and a PRTS signature. Known PRTS do enjoy short signatures.
– The complexity of updates, which is dominated by (1) The key refresh of PRTS; (2) the execution of the distributed key generation protocol $\Pi_{\mathsf{ATS.KGen}}$ to produce new ATS keys; and (3) invoking the signing algorithm of PRTS to collectively sign the new ATS keys. For items (1) and (3), existing PRTS schemes have efficient key updates and signing protocols.

In light of the above discussion, what is missing is an ATS scheme that simultaneously enjoys (1) a short public key; and (2) an efficient distributed key generation protocol. Although ATS schemes with short public keys are known (see for example [8]), their key generation procedures rely on a trapdoor, and hence do not admit efficient distributed analogs. To fill in this gap, in Section 6 we present a new construction of an ATS with short public keys and an efficient key distribution protocol.

Security. Theorem 2 below reduces the security of PRATS to that of PRTS and ATS. For concreteness, in the statement of the theorem and its proof we assume that PRTS and ATS satisfy uf-1 security. If either of them only satisfies uf-0 then essentially the same proof shows that PRATS satisfies uf-0 security.

Theorem 2. *For any adversary $\mathcal{A}$ there exist adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that for every set pp of public parameters it holds that*

$$\mathsf{Adv}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) \leq E \cdot \mathsf{Adv}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}(\mathcal{B}_2),$$

where E is a bound on the number of epochs requested by $\mathcal{A}$ in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$.

The proof of Theorem 2 is presented in Appendix D.2.

6 An ATS with Short Public Key and Efficient DKG

In this section we present a new ATS construction with a short public key and an efficient distributed key generation protocol. In conjunction with our generic construction from Section 5, this yields a concretely-efficient ATS-PR construction.

Our scheme relies on the strong RSA assumption. It is instructive to present our scheme vis-a-vis the ATS of Bellare and Neven [8], also relying on RSA. Loosely, their scheme fixes a "global" exponent e. The secret key of signer i is then $\mathsf{H}(\mathsf{nonce}, i)^{1/e}$ and signing with respect to a subset $\mathcal{J}$ of signers is done via proving knowledge of the e-th roots of all elements $\{\mathsf{H}(\mathsf{nonce}, j)\}_{j\in\mathcal{J}}$. As observed by Bellare and Neven, this can be done efficiently using a multi-prover variant of the GQ protocol [38]. The problem in our setting, however, is that deriving the secret keys in their scheme involves computing roots of random group element, thus requiring a trapdoor.[5]

Our construction. To address the aforesaid issue, we draw inspiration from cryptographic accumulators (see [22] as well as [11, 15] and the many references therein). Instead of associating different secret keys with roots of different group elements, we associate different keys with *different roots of the same group element*. That is, the public key consists of one group element Y, and the secret key of signer i is Y^{1/e_i} for some exponent e_i which is determinstically derived from the index i. As we will show, a careful generalization of the GQ protocol allows a subset $\mathcal{J}$ of signers to collectively prove knowledge of the $\left(\prod_{j\in\mathcal{J}} e_j\right)$-th root of Y, yielding an efficient ATS scheme. Moreover, we present an efficient 1-round (i.e., non-interactive) distributed key generation protocol for this scheme.

In detail, our construction is parameterized by a group $\mathbb{G}$ in which the strong RSA problem is conjectured to be hard. Possible instantiations include the group $\mathbb{Z}_N^*$ relative to a bi-prime modulus N, and class groups of imaginary quadratic fields. For each security parameter λ (implied by the description of the group $\mathbb{G}$) and integer $n = \mathsf{poly}(\lambda)$, we assume an efficiently-computable injective mapping from $[n]$ to primes greater than 2^λ, and we denote by e_i the prime corresponding to $i \in [n]$.

[5] Alternatively, one can settle on a long public key $Y_1, \ldots, Y_n \in \mathbb{G}^n$, and letting the secret key of signer i be the e-th root of Y_i. Such keys can be generated without a trapdoor, but the long public key is problematic in our setting.

The scheme makes use of two hash functions, treated as random oracles in the security proof. The first, $\mathsf{H}_{\mathsf{com}}$ maps pairs of subsets of signers and group elements to λ-bit strings. The second, $\mathsf{H}_{\mathsf{chal}}$, maps 4-tuples consisting of a message, a group element, a subset of signers, and an additional group element, to exponents.

$\mathsf{RSAATS}[\mathbb{G}]$: An RSA-based ATS scheme

$\mathsf{KGen}(\mathbb{G}, n, t)$:

1. For $i = 1, \ldots, n$: Sample $X_i \leftarrow\!\!\$\; \mathbb{G}$.
2. Compute $X \leftarrow \prod_{i=1}^{n} X_i$ and $Y \leftarrow X^{\prod_{i=1}^{n} e_i}$.
3. For each $i \in [n]$, set $\mathsf{sk}_i \leftarrow X^{\prod_{j \in [n] \setminus \{i\}} e_j}$ (so that $\mathsf{sk}_i^{e_i} = Y$).
4. Output $(\mathsf{pk} = (n, t, Y), \mathsf{pkc} = \perp, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\mathsf{Sign}(\mathsf{sk}_i, \mathsf{pk}, \mathcal{J}, m)$:

1. **First Round:**
   (a) Sample $Z_i \leftarrow\!\!\$\; \mathbb{G}$, and compute $R_i \leftarrow Z_i^{\prod_{j \in \mathcal{J}} e_j}$.
   (b) Compute $c_i \leftarrow \mathsf{H}_{\mathsf{com}}(\mathcal{J}, R_i)$.
   (c) Send c_i to each signer $j \in \mathcal{J} \setminus \{i\}$.
2. **Second Round:**
   (a) Upon receiving a message c_j from each $j \in \mathcal{J} \setminus \{i\}$, send R_i to all $j \in \mathcal{J} \setminus \{i\}$.
   (b) For each $j \in \mathcal{J} \setminus \{i\}$: Upon receiving R_j from signer j, verify that $c_j = \mathsf{H}_{\mathsf{com}}(\mathcal{J}, R_j)$. If not, abort the execution of the protocol.
   (c) Set $R \leftarrow \prod_{j \in \mathcal{J}} R_j$.
3. **Third Round:**
   (a) Set $h \leftarrow \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.
   (b) Compute $S_i \leftarrow \mathsf{sk}_i^h \cdot Z_i$.
   (c) Output (h, S_i).

$\mathsf{Combine}(\mathsf{pkc}, (\sigma_{i_1}, \ldots, \sigma_{i_k}))$:

1. Parse each σ_{i_j} as (h^j, S^j) and set $h \leftarrow h^1$. If $h^j \neq h$ for a $j \in [k]$, output $\perp$.
2. Let $\mathcal{J} = \{i_1, \ldots, i_k\}$, and compute $S \leftarrow \prod_{j \in [\mathcal{J}]} S_j$.
3. Output $\sigma = (\mathcal{J}, h, S)$.

$\mathsf{Vf}(\mathsf{pk}, m, \sigma)$:

1. Parse pk as (n, t, Y) and σ as $(\mathcal{J}, h, S)$.
2. Compute $R \leftarrow S^{\prod_{i \in \mathcal{J}} e_i} / Y^{h \cdot \sum_{i \in \mathcal{J}} \prod_{j \in \mathcal{J} \setminus \{i\}} e_j}$.
3. Compute $h' \leftarrow \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.
4. Output 1 if $h = h'$ and $|\mathcal{J}| \geq t$. Otherewise, output 0.

$\mathsf{Trace}(\mathsf{pk}, m, \sigma)$:

1. Parse σ as $(\mathcal{J}, h, S)$.
2. Output $\mathcal{J}$.

Correctness. Observe that for an honestly generated signature $(\mathcal{J}, h, S)$ it holds that

$$\begin{aligned} S^{\prod_{i\in\mathcal{J}} e_i} &= \prod_{j\in\mathcal{J}} \left(S_j^{\prod_{i\in\mathcal{J}} e_i}\right) \\ &= \prod_{j\in\mathcal{J}} \left(\left(\mathsf{sk}_j^h \cdot Z_j\right)^{\prod_{i\in\mathcal{J}} e_i}\right) \\ &= \prod_{j\in\mathcal{J}} \left(\mathsf{sk}_j^{h\cdot\prod_{i\in\mathcal{J}} e_i} \cdot R_j\right) \\ &= \prod_{j\in\mathcal{J}} \left(Y^{h\cdot\prod_{i\in\mathcal{J}\setminus\{j\}} e_i}\right) \cdot R. \end{aligned}$$

Rearranging, this implies that

$$R = \frac{S^{\prod_{i\in\mathcal{J}} e_i}}{Y^{h\cdot\sum_{i\in\mathcal{J}}\prod_{j\in\mathcal{J}\setminus\{i\}} e_j}},$$

and the verification goes through.

Distributed key generation. The pubic and secret keys of RSAATS can be generated via a simple distributed protocol. Recall that this is needed to instantiate ATS in the generic construction from Section 5. Concretely, this is done by the following steps:

1. Each signer $i \in [n]$ samples a uniformly random group element $X_i \leftarrow\$ \mathbb{G}$, computes $Y_i \leftarrow X_i^{e_i}$, and sends Y_i to all other signers.
2. Upon receiving $Y_1, \ldots, Y_n$ from the other signers, each signer sets the pubic key as $\mathsf{pk} \leftarrow \prod_{i\in[n]} Y_i^{\prod_{j\in[n]\setminus\{i\}} e_j}$, and its secret key as

$$\mathsf{sk}_i \leftarrow X_i^{\prod_{j\in[n]\setminus\{i\}} e_j} \cdot \prod_{k\in[n]\setminus\{i\}} Y_k^{\prod_{j\in[n]\setminus\{k,i\}} e_j} \in \mathbb{G}.$$

Observe that if we denote $X = \prod_{j\in[n]} X_j$, then $\mathsf{pk} = X^{\prod_{j\in[n]} e_j}$ and $\mathsf{sk}_i = X^{\prod_{j\in[n]\setminus\{i\}} e_j}$. Hence, sk_i is indeed the e_i-th root of pk for each i. Looking ahead, our security reduction for RSAATS will internally simulate precisely this key generation process, while planting a strong RSA challenge Y^* as one of the Y_i's and simulating the role of all other signers. Hence, it will readily prove the security of the scheme when the key generation algorithm KGen is replaced by an honest execution of the above distributed key generation protocol.

Security. The security of RSAATS is proven based on the hardness of the strong RSA problem, defined in Definition 5.

Definition 5. *Let $\mathbb{G}$ be a group and let $\mathcal{A}$ be an algorithm. We define the advantage of $\mathcal{A}$ in solving the* strong RSA problem *in $\mathbb{G}$ as*

$$\mathsf{Adv}_{\mathbb{G}}^{\mathrm{srsa}}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \left[X^e = Y \wedge e \notin \{-1, 1\} \;:\; \begin{array}{c} Y \leftarrow\$ \mathbb{G} \\ (X, e) \leftarrow\$ \mathcal{A}(\mathbb{G}, Y) \end{array}\right]$$

Theorem 3 below, whose proof can be found in Appendix D.3, reduces the security of RSAATS to the hardness of the strong RSA problem.

Theorem 3. *For any adversary $\mathcal{A}$ there exists an algorithm $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathrm{srsa}}_{\mathbb{G}}(\mathcal{B}) \geq \frac{\left(\mathsf{Adv}^{\mathrm{uf\text{-}1}\wedge\mathrm{acc\text{-}1}}_{\mathsf{RSSATS}[\mathbb{G}]}(\mathcal{A})\right)^2}{n_{\mathsf{max}}^2 \cdot (q_{\mathsf{sign}} + q_{\mathsf{chal}})} - \frac{q_{\mathsf{sign}}^2 + q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}} \cdot q_{\mathsf{chal}} + q_{\mathsf{sign}}}{|\mathbb{G}|} - \frac{2q_{\mathsf{com}}^2 + 3q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}}^2}{2^\lambda},$$

where n_{max} is a bound on the number of signers, and q_{sign}, q_{chal}, and q_{com} are bounds on the number of queries issued by $\mathcal{A}$ to its signing oracle, to $\mathsf{H}_{\mathsf{chal}}$, and to $\mathsf{H}_{\mathsf{com}}$, respectively.

On the necessity of strong RSA. We stress that though the security statement for our ATS scheme relies on the strong RSA assumption, our proof actually reduces the security of the scheme to a somewhat milder assumption. Concretely, the adversary that we construct in the reduction is restricted to computing the eth root of a randomly sampled group element Y, where e has to be chosen from a small set of pre-determined exponents $\{e_1, \ldots, e_n\}$ (where n is the number of signers). This should be contrasted with the strong RSA problem, in which the adversary is free to choose e however it pleases.

7 Shorter Signatures From BLS and Schnorr

In this Section we present very practical ATS-PR schemes in cyclic groups and in bilinear groups, building on Schnorr [56] and BLS [19] signatures, respectively. We start by providing an overview of the main ideas behind these constructions.

Proactive secret sharing. Ostrovsky and Yung [52] and Herzberg et al. [40] described how to proactively refresh Shamir's t-out-of-n secret sharing scheme [57]. Recall that in Shamir's scheme, to share a secret s in some finite field $\mathbb{F}$, the dealer samples a random polynomial f of degree $t-1$ over $\mathbb{F}$ such that $f(0) = s$. The share x_i of the ith party is then $f(i)$. Any subset of parties of size t can reconstruct the secret using Lagrange interpolation, while any $t-1$ shares are statistically independent of the secret s. To refresh, Ostrovsky and Yung and Herzberg et al. suggested the following procedure; the parties jointly sample a new polynomial f' of degree $t-1$ such that $f'(0) = 0$. Then, each party updates its share by $x'_i \leftarrow x_i + f'(i)$. Observe that regardless of the number of share refreshes, the ith party's share is always of the form $x_i + f''(i)$, where f'' is a polynomial of degree $t-1$ satisfying $f''(0) = 0$. Hence, by the linearity of Shamir's secret sharing scheme, the same reconstruction procedure still yields the correct secret s. At the same time, for any number E of epochs, and any $t-1$ shares $(x^e_{i^e_1}, \ldots, x^e_{i^e_{t-1}})$ in each epoch e, the distribution over $\{(x^e_{i^e_1}, \ldots, x^e_{i^e_{t-1}})\}_{e\in[E]}$ is uniformly random in $\mathbb{F}^{(t-1)\times E}$. Hence, an adversary observing $t-1$ shares of its choice in each epoch learns nothing about the secret s.

As observed in a follow-up work by Herzberg et al. [39], this proactive refresh naturally carries over to discrete-log-based (non-accountable) threshold signature schemes in which the signers' secret keys form a t-out-of-n Shamir secret sharing of some "global" secret key x.

From secret sharing to ATS. At first, these techniques do not seem to extend to the ATS constructions based on Schnorr and BLS signatures (e.g., [17, 7, 14, 16, 51]). In such constructions, the signers' secret keys $x_1, \ldots, x_n$ are sampled independently in $\mathbb{Z}_p$, where p is the order of the underlying groups. Then, each subset $\mathcal{J}$ of signers is naturally associated with a distinct corresponding signing key $x_{\mathcal{J}} = \sum_{j\in\mathcal{J}} x_j \in \mathbb{Z}_p$. The corresponding public key $g^{x_{\mathcal{J}}}$ can be computed from the individual public keys $g^{x_1}, \ldots, g^{x_n}$ by $g^{x_{\mathcal{J}}} = \prod_{j \in \mathcal{J}} g^{x_j}$, which enables accountability. However, if we now try to refresh the keys by adding a secret sharing of 0, the sum $\sum_{j \in \mathcal{J}} x'_j$ of the fresh keys will no longer correspond to the public key $g^{x_{\mathcal{J}}}$, and the signing procedure will produce invalid signatures.

Instead, our approach is to associate to each subset $\mathcal{J}$ the secret key

$$x_{\mathcal{J}} := \sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} \cdot x_j \quad \in \mathbb{Z}_p,$$

where $\lambda_j^{\mathcal{J}}$ is the jth Lagrange coefficient used by the subset $\mathcal{J}$ to reconstruct the secret in Shamir's secret sharing scheme. Now, suppose that at the beginning of an epoch we refresh the secret keys $\{x_i\}_{i \in [n]}$ by setting $x'_i \leftarrow x_i + \delta_i \in \mathbb{Z}_p$, where $\{\delta_i\}_{i \in [n]}$ is a random t-out-of-n secret sharing of 0. Then, for every subset $\mathcal{J}$ of size t, it holds that

$$\sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} x'_j = \sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} (x_j + \delta_j) = \sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} x_j = x_{\mathcal{J}}.$$

Hence, the "collective" secret key $x_{\mathcal{J}}$ of the subset $\mathcal{J}$ remains unchanged from epoch to epoch, despite the proactive refresh. We will see how to put this to use in a minute.

An attack on accountability. The schemes resulting from this idea only satisfy the weaker acc-0 accountability property defined in Section 3. This seems inherent to this linear key update mechanism. To see why, we describe an attack that shows that the schemes do not satisfy acc-1. Consider the simple case of $t = 3$. In the first epoch, the adversary corrupts signers $1, 2$ and 4, to learn the keys x_1, x_2, x_4 from which it computes $\lambda_1^{\{1,2,4\}} x_1 + \lambda_2^{\{1,2,4\}} x_2 + \lambda_4^{\{1,2,4\}} x_4$ (recall that $\lambda_i^{\mathcal{J}}$ is the coefficient of x_i in the Shamir secret sharing reconstruction for subset $\mathcal{J}$). In the second epoch, the adversary corrupts signers $1, 3$ and 4, and obtains the refreshed keys x'_1, x'_3, x'_4, from which it computes $\lambda_1^{\{1,3,4\}} x'_1 + \lambda_3^{\{1,3,4\}} x'_3 + \lambda_4^{\{1,3,4\}} x'_4$. This linear combination of refreshed keys is equal to the linear combination $\lambda_1^{\{1,3,4\}} x_1 + \lambda_3^{\{1,3,4\}} x_3 + \lambda_4^{\{1,3,4\}} x_4$ of the keys x_1, x_3, x_4 in the first epoch because this linear combination causes the refresh randomness $\delta_1, \delta_3, \delta_4$ to cancel (recall that $\delta_1, \delta_3, \delta_4$ are shares in a 3-out-of-n secret sharing of zero). The adversary now has two linear combinations of x_1, x_2, x_3, x_4 from which it can obtain a linear combination of x_1, x_2 and x_3. In epoch 3 the adversary corrupts signers $\{1, 2, 5\}$ and in epoch 4 it corrupts signers $\{1, 3, 5\}$. This gives another linear combination of x_1, x_2, x_3. Finally, in epochs 5 and 6 the adversary corrupts signers $\{1, 2, 6\}$ and then $\{1, 3, 6\}$ and this gives a third linear combination of x_1, x_2, x_3. Now, the adversary has a full rank linear system for x_1, x_2, x_3 which lets it find all three keys. Thus, after six epochs the adversary can sign any message of its choice on behalf of the set $\{1, 2, 3\}$, even though there was never an epoch in which the adversary simultaneously corrupted all three of these signers. This shows that the resulting schemes do not satisfy acc-1. We will show that they satisfy acc-0.

Although the schemes in this section satisfy this weaker notion of accountability, they introduce non-trivial efficiency gains. On the face of it, acc-0 can be achieved (in conjunction with unforgeability) by signing a message twice: Once with a (non-accountable) TS-PR scheme and once with a (non-refreshable) ATS scheme, as discussed in Appendix A. However, the downside of this approach is that it essentially doubles the length of signatures and the time to sign. In contrast, the schemes that we present in this section preserve the length and performance of their underlying schemes: Our BLS based scheme produces standard BLS signatures (consisting of just 1 group element), and our Schnorr-based produce standard Schnorr signatures (not counting the encoding of the subset $\mathcal{J}$ of signers, which, as discussed in the introduction, cannot be avoided).

Unforgeability: uf-0 vs. uf-1. The unforgeability notion we will prove for all of our constructions in this section is uf-0, rather than uf-1. Restricting ourselves to uf-0 allows us to reduce the security of our constructions to the security of their underlying basic signature scheme (BLS or Schnorr). In contrast, as observed by Bellare et al. [6], proving uf-1 requires stronger "one-more"-type assumptions (e.g., one-more discrete-log). Moreover, focusing on uf-0 security already captures our main ideas, and extending the proofs to handle uf-1 can be done using the ideas from [6].

7.1 A BLS-Based ATS-PR Scheme

Our BLS-Based ATS with proactive refresh is parameterized by a bilinear group $\mathcal{G}$ which is a tuple $(\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T, e, g_0, g_1, p)$, where $\mathbb{G}_0, \mathbb{G}_1$ are cyclic groups of order p generated by g_0 and g_1 respectively, and e is a non-degenerate bilinear map, mapping pairs in $\mathbb{G}_0 \times \mathbb{G}_1$ to $\mathbb{G}_T$. It relies on the existence of a hash function $\mathsf{H} : \mathcal{M} \to \mathbb{G}_0$, where $\mathcal{M}$ is the message space. We will implicitly assume that the number n of signers is upper bounded by the order p of the group and that all arithmetic is over $\mathbb{Z}_p$.

$\mathsf{BLSPR}[\mathcal{G}]$: A BLS-based ATS-PR scheme

$\mathsf{KGen}(\mathcal{G}, n, t)$:

1. For $i = 1, \ldots, n$: Sample $x_i \leftarrow\!\!\$\ \mathbb{Z}_p$, set $\mathsf{sk}_i \leftarrow x_i$, and $X_i \leftarrow g_1^{x_i} \in \mathbb{G}_1$.
2. Output $\big(\mathsf{pk} = (t, X_1, \ldots, X_n),\ \mathsf{pkc} = \perp,\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\big)$.

$\mathsf{Sign}(\mathsf{sk}_i, m)$:

1. Compute $\sigma_i \leftarrow \mathsf{H}(m)^{\mathsf{sk}_i} \in \mathbb{G}_0$.
2. Output σ_i.

$\mathsf{Combine}(\mathsf{pkc}, (\sigma_{i_1}, \ldots, \sigma_{i_{|\mathcal{J}|}}))$:

1. For $j \in \mathcal{J}$: Compute $\lambda_j^{\mathcal{J}} \leftarrow \prod_{i \in \mathcal{J} \setminus \{j\}} \frac{i}{i-j} \in \mathbb{Z}_p$.
2. Compute $Y \leftarrow \prod_{j \in \mathcal{J}} \sigma_j^{\lambda_j^{\mathcal{J}}} \in \mathbb{G}_0$.

   // observe that $Y = \mathsf{H}(m)^{x_{\mathcal{J}}}$ where $x_{\mathcal{J}} = \sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} x_j$ is the "collective" secret key of subset $\mathcal{J}$.
3. Output $\sigma = (\mathcal{J}, Y)$.

$\mathsf{Vf}(\mathsf{pk}, m, \sigma)$:

1. Parse pk as $(t, X_1, \ldots, X_n)$ and σ as $(\mathcal{J}, Y)$.
2. For $j \in \mathcal{J}$: Compute $\lambda_j^{\mathcal{J}} \leftarrow \prod_{i \in \mathcal{J} \setminus \{j\}} \frac{i}{i-j} \in \mathbb{Z}_p$.
3. Compute $X_{\mathcal{J}} \leftarrow \prod_{j \in \mathcal{J}} X_j^{\lambda_j^{\mathcal{J}}} \in \mathbb{G}_1$.
4. Output 1 if $e(Y, g_1) = e(\mathsf{H}(m), X_{\mathcal{J}})$ and $|\mathcal{J}| \geq t$. Else, output 0.

$\mathsf{Trace}(\mathsf{pk}, m, \sigma)$:

1. Parse σ as $(\mathcal{J}, Y)$.
2. Output $\mathcal{J}$.

$\mathsf{Update}_0(\mathsf{sk}_i, \mathsf{pk})$:

// Signer i samples a random polynomial f_i of degree $t-1$ and such that $f_i(0) = 0$ and sends $f_i(j)$ to signer j.

1. Parse pk as $(t, X_1, \ldots, X_n)$.

2. Sample $a_1, \ldots, a_{t-1} \leftarrow\$ \mathbb{Z}_p$.
3. For $j = 1, \ldots, n$: Compute $\delta_{i,j} \leftarrow \sum_{\ell=1}^{t-1} a_\ell \cdot j^\ell \in \mathbb{Z}_p$.
4. Output $(\delta_{i,1}, \ldots, \delta_{i,n})$,

$\underline{\mathsf{Update}_1(\mathsf{sk}_j, \delta_{1,j}, \ldots, \delta_{n,j})\text{:}}$

// Signer i adds $f(i)$ to its secret key, where $f = f_1 + \cdots + f_n$ is a random polynomial of degree $t-1$ satisfying $f(0) = 0$.

1. Compute $\mathsf{sk}'_j \leftarrow \mathsf{sk}_j + \sum_{i=1}^{n} \delta_{i,j} \in \mathbb{Z}_p$.
2. Output sk'_j.

Correctness. The correctness of the scheme follows from the fact that in each epoch k, the secret key of i is of the form $\mathsf{sk}_i = x_i + \delta_i^{(k)}$, where the values $\{\delta_1^{(k)}, \ldots, \delta_n^{(k)}\}$ form a Shamir t-out-of-n secret sharing of 0. Hence, it holds that

$$\begin{aligned} e(Y, g_1) &= e(\prod_{j \in \mathcal{J}} \sigma_j^{\lambda_j^{\mathcal{J}}}, g_1) = e(\mathsf{H}(m), g_1)^{\sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} \cdot \mathsf{sk}_j} \\ &= e(\mathsf{H}(m), g_1)^{\sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} \cdot (x_j + \delta_j^{(k)})} = e(\mathsf{H}(m), g_1)^{\sum_{j \in \mathcal{J}} \lambda_j^{\mathcal{J}} \cdot x_j} \\ &= \prod_{j \in \mathcal{J}} e(\mathsf{H}(m), X_j)^{\lambda_j^{\mathcal{J}}} \\ &= e(\mathsf{H}(m), \prod_{j \in \mathcal{J}} X_j^{\lambda_j^{\mathcal{J}}}). \end{aligned}$$

This means that at every epoch, a signature generated by $\mathcal{J}$, where $|\mathcal{J}| \geq t$, will be accepted.

Security. We now prove the unforgeability and accountability of our BLS-based scheme. Recall that we consider a semi-adaptive variant of our unforgeability definition, in which for each epoch the adversary is forced to issue all of its secret key queries prior to its signing queries. Theorem 4 below reduces the security of our scheme to the unforgeability of standard (single-signer) BLS signatures. Unforgeability for single-signer signature schemes is captured as a special case of our security definitions, by fixing the number n of signers (and the threshold t) to be 1, and the number E of epochs to be 1. For an adversary $\mathcal{B}$, we denote by $\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B})$ its advantage in breaking the unforgeabilty of BLS. We refer to [20, 19] for a definition of unforgeability of signature schemes, and a description of the BLS signature scheme.

Theorem 4. *Let $\mathcal{G}$ be a bilinear group. Then, for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathrm{sa\text{-}uf\text{-}0 \wedge acc\text{-}0}}_{\mathsf{BLSPR}[\mathcal{G}]}(\mathcal{A}) \leq 2n_{max} \cdot \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B}),$$

where n_{max} is an upper bound on the number of signers in the game $\mathbf{G}^{\mathrm{uf\text{-}0 \wedge acc\text{-}0}}_{\mathsf{BLSPR}[\mathcal{G}]}$.

Roadmap. We prove Theorem 4 by proving unforgeability and accountability seperately. Let $\mathsf{BLSPR\text{-}1}$ denote a 1-epoch variant of our BLS-based ATS-PR scheme (obtained by fixing the number E of epochs to be 1). Unforgeability is then proven in two steps. First, Lemma 2 reduces the unforgeability of $\mathsf{BLSPR\text{-}1}$ to that of BLS. Then, in Lemma 3 we reduce the unforgeability of our (multi-epoch) scheme to the 1-epoch variant.

Lemma 2. *For every adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}_1$ such that*

$$\mathsf{Adv}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}(\mathcal{A}) \leq n_{max} \cdot \mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B}_1).$$

Lemma 3. *For every adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}_2$ such that*

$$\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{BLSPR}[\mathcal{G}]}(\mathcal{A}) \leq \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}(\mathcal{B}_2).$$

The proof of Lemma 2 is similar to the proof of unforgeability for aggregated BLS signatures (see [19, 17, 20]) and is deferred to Appendix D.4. The proof of Lemma 3 can be found in Appendix D.5. *Accountability* of BLSPR is stated in Lemma 4 below, whose proof is deferred to Appendix D.6.

Lemma 4. *For every adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{BLSPR}[\mathcal{G}]}(\mathcal{A}) \leq n_{max} \cdot \mathsf{Adv}^{\text{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B}).$$

By definition of the $\mathbf{G}^{\text{sa-uf-0}\wedge\text{acc-0}}_{\mathsf{BLSPR}[\mathcal{G}]}$ security game, Theorem 4 immediately follows from lemmas 2, 3, and 4.

7.2 A 3-Round Schnorr-Based ATS-PR Scheme

Our three-round Schnorr-Based ATS with proactive refresh builds on the recent (non-accountable) threshold Schnorr scheme of Lindell [47]. We adapt the scheme to an ATS (making it accountable) and show how to add a proactive refresh. We first simplify Lindell's signing protocol: In his protocol, when each signer sends a group element R, it also provides alongside it a non-interactive zero-knowledge proof of knowledge for the discrete log of R. This is needed to prove simulation-based security, but (as we prove) is unnecessary to satisfy our game-based definitions. Then, we augment the protocol with a proactive refresh via the approach discussed at the beginning of this section. At the end of the section we explain how a similar approach can add a proactive refresh to the two-round Schnorr ATS schemes [51, 43, 25, 6].

Our ATS with proactive refresh is parameterized by a cyclic group $\mathbb{G} = \langle g \rangle$ of order p, and a pair of hash functions: $\mathsf{H}_{\mathsf{com}}$ mapping pairs of subsets of signers and group elements into λ-bit strings, and $\mathsf{H}_{\mathsf{chal}}$ mapping 4-tuples of the form (message, public key, signers subset, group element) into elements in $\mathbb{Z}_p$. We will implicitly assume that the number n of signers is upper bounded by the order p of the group, and that all finite field elements are in $\mathbb{Z}_p$.

$\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]$ A three-round Schnorr-based ATS-PR scheme

$\mathsf{KGen}(\mathbb{G}, n, t)$:

1. For $i = 1, \ldots, n$: Sample $x_i \leftarrow\$ \mathbb{Z}_p$, set $\mathsf{sk}_i \leftarrow x_i$, and $X_i \leftarrow g^{x_i} \in \mathbb{G}$.
2. Output $\big(\mathsf{pk} = (t, X_1, \ldots, X_n),\ \mathsf{pkc} = \perp,\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)\big)$.

$\mathsf{Sign}(\mathsf{sk}_i, \mathsf{pk}, \mathcal{J}, m)$:

- **First Round:**
  1. Sample $r_i \leftarrow\$ \mathbb{Z}_p$, and compute $R_i \leftarrow g^{r_i}$.
  2. Compute $c_i \leftarrow \mathsf{H}_{\mathsf{com}}(R_i)$.
  3. Send $\mathsf{msg}_{i,1} \leftarrow c_i$ to each signer $j \in \mathcal{J} \setminus \{i\}$.
- **Second Round:**
  1. Upon receiving a message c_j from each $j \in \mathcal{J} \setminus \{i\}$, send $\mathsf{msg}_{i,2} \leftarrow R_i$ to all $j \in \mathcal{J} \setminus \{i\}$.
- **Third Round:**
  1. For each $j \in \mathcal{J} \setminus \{i\}$: Upon receiving R_j from signer j, verify that $c_j = \mathsf{H}_{\mathsf{com}}(R_j)$. If not, abort.
  2. Set $R \leftarrow \prod_{j \in \mathcal{J}} R_j$.
  3. Set $h \leftarrow \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \in \mathbb{Z}_p$.
  4. Compute $\lambda_i^{\mathcal{J}} \leftarrow \prod_{j \in \mathcal{J} \setminus \{i\}} \frac{j}{j-i} \in \mathbb{Z}_p$ and $s_i \leftarrow \lambda_i^{\mathcal{J}} \cdot h \cdot \mathsf{sk}_i + r_i \in \mathbb{Z}_p$.
  5. Output s_i.

Combine(pkc, $\{s_j\}_{j\in\mathcal{J}}, \mathcal{J}, R$):

1. Upon receiving a message $s_j \in \mathbb{Z}_p$ from each $j \in \mathcal{J} \setminus \{i\}$, compute $s \leftarrow \sum_{j\in\mathcal{J}} s_j \in \mathbb{Z}_p$.
2. Output $\sigma = (\mathcal{J}, R, s)$.

   // observe that $s = h \cdot x_{\mathcal{J}} + r$ where $x_{\mathcal{J}} = \sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}} x_j$ is the "collective" secret key of subset $\mathcal{J}$ and $r = \sum_{j\in\mathcal{J}} r_j$.

Vf(pk, m, σ):

1. Parse pk as $(t, X_1, \ldots, X_n)$ and σ as $(\mathcal{J}, R, s)$.
2. Compute $h \leftarrow \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \in \mathbb{Z}_p$.
3. For $j \in \mathcal{J}$: Compute $\lambda_j^{\mathcal{J}} \leftarrow \prod_{i\in\mathcal{J}\setminus\{j\}} \frac{i}{i-j} \in \mathbb{Z}_p$.
4. Output 1 if $\left(\prod_{j\in\mathcal{J}} X_j^{\lambda_j^{\mathcal{J}}\cdot h}\right) \cdot R = g^s$ and $|\mathcal{J}| \geq t$. Otherwise, output 0.

Trace(pk, m, σ):

1. Parse σ as $(\mathcal{J}, R, s)$.
2. Output $\mathcal{J}$.

Update$_0$(sk$_i$, pk):

// Signer i samples a random polynomial f_i of degree $t-1$ and such that $f_i(0) = 0$ and sends $f_i(j)$ to signer j.

1. Parse pk as $(t, X_1, \ldots, X_n)$.
2. Sample $a_1, \ldots, a_{t-1} \leftarrow\!\!\$\ \mathbb{Z}_p$.
3. For $j = 1, \ldots, n$: Compute $\delta_{i,j} \leftarrow \sum_{\ell=1}^{t-1} a_\ell \cdot j^\ell$.
4. Output $(\delta_{i,1}, \ldots, \delta_{i,n})$,

Update$_1$(sk$_j, \delta_{1,j}, \ldots, \delta_{n,j}$):

// Signer i adds $f(i)$ to its secret key, where $f = f_1 + \cdots f_n$ is a random polynomial of degree $t-1$ satisfying $f(0) = 0$.

1. Compute $\mathsf{sk}'_j = \mathsf{sk}_j + \sum_{i=1}^n \delta_{i,j}$.
2. Output sk'_j.

Correctness. The correctness of the scheme follows from the fact that, in each epoch k, the secret key of signer i is of the form $\mathsf{sk}_i^{(k)} = x_i + \delta_i^{(k)}$, where the values $\{\delta_1^{(k)}, \ldots, \delta_n^{(k)}\}$ form a Shamir t-out-of-n secret sharing of 0. Hence, using $h \leftarrow \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$, for any epoch k, it holds that,

$$\begin{aligned}
g^s &= g^{\sum_{j\in\mathcal{J}} s_j} = g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot h\cdot \mathsf{sk}_j^{(k)} + r_j} = g^{\sum_{j\in\mathcal{J}} r_j} \cdot g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot h\cdot(x_j + \delta_j^{(k)})} \\
&= \prod_{j\in\mathcal{J}} R_j \cdot g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot h\cdot x_j} \cdot (g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot\delta_j^{(k)}})^h \\
&= R \cdot \prod_{j\in\mathcal{J}} (g^{x_j})^{\lambda_j^{\mathcal{J}}\cdot h} \cdot (g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot\delta_j^{(k)}})^h \\
&= R \cdot \prod_{j\in\mathcal{J}} X_j^{\lambda_j^{\mathcal{J}}\cdot h} \cdot (g^{\sum_{j\in\mathcal{J}} \lambda_j^{\mathcal{J}}\cdot\delta_j^{(k)}})^h
\end{aligned}$$

Since $\{\delta_1^k, \ldots, \delta_n^k\}$ form a Shamir t-out-of-n secret sharing of 0, we can use Lagrange interpolation over all $j \in \mathcal{J}$ (where $|\mathcal{J}| = t$) to get:

$$\sum_{j\in\mathcal{J}} \lambda_j \cdot \delta_j^{(k)} = 0.$$

Combining this with the previous equation, we get,

$$g^s = R \cdot \prod_{j \in \mathcal{J}} X_j^{\lambda_j^{\mathcal{J}} \cdot h} \cdot (g^{\sum_{j \in \mathcal{J}} \lambda_j \cdot \delta_j^{(k)}})^h = R \cdot \prod_{j \in \mathcal{J}} X_j^{\lambda_j^{\mathcal{J}} \cdot h} \cdot (g^0)^h = R \cdot \prod_{j \in \mathcal{J}} X_j^{\lambda_j^{\mathcal{J}} \cdot h}$$

This proves that signatures produced in all epochs are correct.

Security. We now prove the unforgeability and accountability of this scheme for semi-adaptive adversaries. Theorem 5 reduces the security of our scheme to the unforgeability of standard (single-signer) Schnorr signatures.

Theorem 5. *Let $\mathbb{G}$ be a cyclic group of order p. Then, for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\text{sa-uf-0}\wedge\text{acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \leq 3n_{max} \cdot \left(\mathsf{Adv}^{\text{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}) + \frac{q_S^2 + q_S q_H}{p} + \frac{2q_C^2}{2^\lambda} + \frac{q_S}{|\mathcal{M}|} \right)$$

where n_{max} is a bound on the number of signers, $\mathcal{M}$ is the message space, and q_S, q_H, and q_C are bounds on the number of signature queries, $\mathsf{H}_{\mathsf{chal}}$ queries, and $\mathsf{H}_{\mathsf{com}}$ queries issued by $\mathcal{A}$.

The proof for Theorem 5 follows a similar road map to the security proof of our BLS-based construction, and can be found in Appendix D.7.

From 3 to 2 rounds Schnorr. We believe that our 3-rounds Schnorr-based ATS-PR scheme can be compressed into a 2-rounds scheme using ideas from the recently-proposed multisignature scheme MuSig2 [51] (see also [43]). In detail, MuSig2 gets rid of the first commitment round (in which signer i commits to R_i using a random oracle) by introducing additional randomness into the protocol. In their scheme, each signer sends 4 random $R_{i,1} \leftarrow g^{r_{i,1}}, \ldots, R_{i,4} \leftarrow g^{r_{i,4}}$ group elements instead of one. The "collective" R is then computed as $R \leftarrow R_1 \cdot R_2^{\alpha} \cdot R_3^{\alpha^2} \cdot R_4^{\alpha^3}$, where $R_i \leftarrow \prod_{j \in \mathcal{J}} R_{j,i}$ and $\alpha \leftarrow \mathsf{H}(\mathsf{pk}, R_1, \ldots, R_4, m)$ for a random oracle H. In the second round, each signer i sends the sum $s_i \leftarrow x_i \cdot h + \sum_{j=1}^{4} \alpha^{j-1} \cdot r_{i,j}$. The aggregate signature is then (R, s) for $s = \sum_{i \in \mathcal{J}} s_i$; this can be verified as standard Schnorr signatures.

We believe that our 3-round scheme depicted above can be transformed into a 2-round scheme by relying on the same ideas. We leave the task of formally defining the resulting scheme and proving its security as an interesting question for future work.

8 Discussion and Extensions

In this section we explore several extensions and directions for future work.

Corruptions during key updates. In our security games, the adversary is oblivious to the key update process. It is reasonable to consider a strengthening in which the adversary can observe, and even control, some key-related information during key updates. We discuss this issue in detail in Appendix B, and give a brief overview here.

First, consider an adversary that is given the entire transcript of the key update protocol (that is, all messages $\delta_{i,j}$ produced by the parties using Update_0). Our discussion in Section 6 shows that the generic scheme from Section 5 remains secure under this strengthening. The situation is a bit more involved for the constructions in Section 7. There, exposing the adversary to the $\delta_{i,j}$ values by which the secret keys are shifted will render the key update useless. However, this is easily remedied by having the parties privately send their update messages (i.e., $\delta_{i,j}$ sent from party i to party j is encrypted using party j's public key). Then, security is maintained even if the adversary learns the update messages sent to at most $t-1$ parties in each key update, assuming that parties corrupted during an update protocol are "counted" as corrupted in both the previous and the subsequent epochs.

To see why the constructions in Section 7 remain secure in this setting, first consider a scenario in which the adversary corrupts at most $t-2$ parties during a key update. In this case, by the security of Shamir secret sharing, the $\delta_{i,j}$ values sent from honest to corrupted parties are just uniformly-random field elements in the view of the adversary. In particular, they are statistically independent of the values sent from honest parties to the other parties. Hence, since there is at least one uncorrupted party, each honest party adds to its secret key a value that is uniformly random in the view of the adversary. Now, if there are $t-1$ corruptions during the key update, then it is true that the adversary can completely reconstruct the values that all parties add to their secret keys, since it also knows that they are induced by a polynomial whose free coefficient is 0. However, this does not pose a problem, since all of these $t-1$ corruptions are counted towards both the previous and subsequent epochs. Thus, the adversarial "budget" for these epochs is exhausted and no more parties can be corrupted within them. This means that the adversary can calculate that some uncorrupted party i updated its secret key by a value δ, but this information is useless without some knowledge about the secret key sk_i in either of the previous or subsequent epochs.

Observe that the above discussion holds just the same, even if the adversary can choose the randomness of the parties that it corrupts during the update. In Appendix B we present formal definitions capturing security in this setting.

An even stronger security notion might allow the adversary to fully control the update messages of the (up to $t-1$) currently corrupted parties. In Appendix B.2 we formally define notions of security in the presence of such an active adversary. Then in Appendix B.3 We present a general compiler that takes any ATS-PR scheme which is secure against semi-honest corruptions during updates, and lifts it using MPC techniques into a scheme that is secure against malicious corruptions during key updates.

One can also consider more efficient techniques to immunize our specific schemes against malicious corruptions during updates. Our constructions from Section 7 readily extend to this setting by replacing the secret sharing step during key updates with a *verifiable* t-out-of-N linear secret sharing scheme (e.g., using Feldman's protocol [30, 40, 39]). As for the scheme obtained by combining our constructions from Sections 5 and 6, adopting a commit-and-open approach might be sufficient. We leave the task of exploring these more optimized techniques as an interesting direction for future work.

Distributed and local key generation. Our definitions and constructions are in a setting where key generation is carried out by a central key generation algorithm. When the set of potential signers is a-priori known, a possible extension which has been considered in closely-related scenarios, is to replace this algorithm with a distributed key-generation protocol. In our constructions from Section 7, the (honestly-generated) secret keys are completely independent. Hence, for these constructions, one might even consider a scenario in which the set of potential signers need not be a-priori fixed, and signers can join and locally generate their own key material over time.

An interesting direction for future research is generalizing our security notions to accommodate both of the above extensions, and adjusting our constructions to these settings. Note that this will require in particular employing mechanisms to fence of rouge key attacks, in which the adversary maliciously chooses the keys of corrupted parties based on what it knows about the keys of honest parties. In the simpler case where the identity of potential signers is known in advance, such mechanisms can be incorporated already in the distributed key generation protocol. In the setting where keys are locally and non-interactively generated by the parties, the signing protocols need to be adjusted to disallow such attacks. For more information on rogue key attacks and how to protect against them, see [49, 17, 14, 7, 16, 51] and the references therein.

Confirmation vs. tracing. In real-world settings it may be sufficient to use a *confirmation* algorithm instead of a *tracing* algorithm. A confirmation algorithm takes as input a rogue signature and a suspect quorum and outputs "yes" if the suspect quorum is the one that generated the given signature. This can lead to shorter accountable signatures because now it suffices to embed a commitment to the signing quorum in the signature, rather than explicitly encode the signing quorum in the signature.

Acknowledgments. This work was funded by NSF, DARPA, the Simons Foundation, UBRI, and NTT Research. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

1. Abdalla, M., Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D.: Sphf-friendly non-interactive commitments. In: Advances in Cryptology — ASIACRYPT 2013. p. 214–234 (2013)
2. Almansa, J.F., Damgård, I., Nielsen, J.B.: Simplified threshold rsa with adaptive and proactive security. In: Advances in Cryptology – EUROCRYPT' 06. p. 593–611 (2006)
3. Andresen, G.: Bitcoin m-of-n standard transactions (2011), BIP-0011
4. Bagherzandi, A., Cheon, J.H., Jarecki, S.: Multisignatures secure under the discrete logarithm assumption and a generalized forking lemma. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM CCS 2008. pp. 449–458. ACM Press (Oct 2008). https://doi.org/10.1145/1455770.1455827
5. Barak, B., Herzberg, A., Naor, D., Shai, E.: The proactive security toolkit and applications. In: Motiwalla, J., Tsudik, G. (eds.) ACM CCS 99. pp. 18–27. ACM Press (Nov 1999). https://doi.org/10.1145/319709.319713
6. Bellare, M., Crites, E., Komlo, C., Maller, M., Tessaro, S., Zhu, C.: Better than advertised security for non-interactive threshold signatures. In: CRYPTO'22. Springer-Verlag (2022)
7. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: CCS'06. pp. 390–399. ACM (2006)
8. Bellare, M., Neven, G.: Identity-based multi-signatures from RSA. In: Topics in Cryptology – CT-RSA 2007. pp. 145–162 (2007)
9. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: Advances in Cryptology – EUROCRYPT '06. pp. 409–426 (2006)
10. Bellare, M., Shoup, S.: Two-tier signatures, strongly unforgeable signatures, and fiat-shamir without random oracles. In: Public Key Cryptography – PKC 2007. p. 201–216 (2007)
11. Benaloh, J., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Advances in Cryptology – EUROCRYPT '93. pp. 274–285 (1993)
12. Blum, M., De Santis, A., Micali, S., Persiano, G.: Noninteractive zero-knowledge. SIAM Journal on Computing **20**(6), 1084–1118 (1991). https://doi.org/10.1137/0220068
13. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. p. 103–112. STOC '88 (1988). https://doi.org/10.1145/62212.62222
14. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Public Key Cryptography – PKC 2003. p. 31–46 (2003)
15. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to iops and stateless blockchains. In: Advances in Cryptology – CRYPTO '19. pp. 561–586 (2019)
16. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: ASIACRYPT'18. pp. 435–463 (2018)

17. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. Lecture Notes in Computer Science, vol. 2656, pp. 416–432. Springer (2003)
18. Boneh, D., Komlo, C.: Threshold signatures with private accountability. In: CRYPTO'22. LNCS, vol. 13509. Springer (2022)
19. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Advances in Cryptology — ASIACRYPT 2001. p. 514–532 (2001)
20. Boneh, D., Shoup, V.: A Graduate Course in Applied Cryptography, Draft 0.5. Cambridge University Press (2020)
21. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: Atluri, V. (ed.) ACM CCS 2002. pp. 88–97. ACM Press (Nov 2002). https://doi.org/10.1145/586110.586124
22. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Advances in Cryptology – CRYPTO '02. pp. 61–76 (2002)
23. Canetti, R., Fischlin, M.: Universally composable commitments. In: Advances in Cryptology — CRYPTO' 01. pp. 19–40 (2001)
24. Canetti, R., Gennaro, R., Goldfeder, S., Makriyannis, N., Peled, U.: UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1769–1787. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3423367
25. Crites, E., Komlo, C., Maller, M.: How to prove schnorr assuming schnorr: Security of multi- and threshold signatures. IACR Cryptol. ePrint Arch. (2021), https://eprint.iacr.org/2021/1375
26. De Santis, A., Di Crescenzo, G., Ostrovsky, R., Persiano, G., Sahai, A.: Robust non-interactive zero knowledge. In: Advances in Cryptology — CRYPTO 2001. pp. 566–598 (2001)
27. Desmedt, Y.: Society and group oriented cryptography: A new concept. In: Pomerance, C. (ed.) CRYPTO'87. LNCS, vol. 293, pp. 120–127. Springer, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2_8
28. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Brassard, G. (ed.) Advances in Cryptology — CRYPTO' 89. pp. 307–315 (1990)
29. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string. In: The 31st Annual Symposium on Foundations of Computer Science. pp. 308–317 (1990). https://doi.org/10.1109/FSCS.1990.89549
30. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th FOCS. pp. 427–437. IEEE Computer Society Press (Oct 1987). https://doi.org/10.1109/SFCS.1987.4
31. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Optimal resilience proactive public-key cryptosystems. In: 38th FOCS. pp. 384–393. IEEE Computer Society Press (Oct 1997). https://doi.org/10.1109/SFCS.1997.646127
32. Frankel, Y., Gemmell, P., MacKenzie, P.D., Yung, M.: Proactive RSA. In: Kaliski Jr., B.S. (ed.) CRYPTO'97. LNCS, vol. 1294, pp. 440–454. Springer, Heidelberg (Aug 1997). https://doi.org/10.1007/BFb0052254
33. Frankel, Y., MacKenzie, P.D., Yung, M.: Adaptively-secure optimal-resilience proactive RSA. In: Lam, K.Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT'99. LNCS, vol. 1716, pp. 180–194. Springer, Heidelberg (Nov 1999). https://doi.org/10.1007/978-3-540-48000-6_15
34. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game. In: Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing. p. 218–229. STOC '87, New York, NY, USA (1987). https://doi.org/10.1145/28395.28420
35. Goldreich, O.: The Foundations of Cryptography - Volume II: Basic Applications. Cambridge University Press (2004)
36. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (Dec 2006). https://doi.org/10.1007/11935230_29
37. Groth, J., Ostrovsky, R., Sahai, A.: Non-interactive zaps and new techniques for NIZK. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 97–111. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_6
38. Guillou, L.C., Quisquater, J.J.: A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In: Advances in Cryptology – CRYPTO '88. pp. 216–231 (1988)
39. Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H., Yung, M.: Proactive public key and signature systems. In: Graveman, R., Janson, P.A., Neuman, C., Gong, L. (eds.) ACM CCS 97. pp. 100–110. ACM Press (Apr 1997). https://doi.org/10.1145/266420.266442

40. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (Aug 1995). https://doi.org/10.1007/3-540-44750-4_27
41. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC research and development (1983)
42. Jarecki, S., Olsen, J.: Proactive RSA with non-interactive signing. In: Tsudik, G. (ed.) FC 2008. LNCS, vol. 5143, pp. 215–230. Springer, Heidelberg (Jan 2008)
43. Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Selected Areas in Cryptography. p. 34–65 (2020)
44. Kondi, Y., Magri, B., Orlandi, C., Shlomovits, O.: Refresh when you wake up: Proactive threshold wallets with offline devices. Cryptology ePrint Archive, Paper 2019/1328 (2019), https://eprint.iacr.org/2019/1328
45. Libert, B., Joye, M., Yung, M.: Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. Theor. Comput. Sci. **645**, 1–24 (2016)
46. Lindell, Y.: A simpler construction of cca2-secure public-key encryption under general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 241–254. Springer, Heidelberg (May 2003). https://doi.org/10.1007/3-540-39200-9_15
47. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. IACR Cryptol. ePrint Arch. (2022), https://eprint.iacr.org/2022/374
48. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. In: CCS'01. pp. 245–254. ACM (2001)
49. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. In: Reiter, M.K., Samarati, P. (eds.) ACM CCS 2001. pp. 245–254. ACM Press (Nov 2001). https://doi.org/10.1145/501983.502017
50. Naor, M.: Bit commitment using pseudorandomness. Journal of Cryptology **4**(2), 151–158 (1991)
51. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round schnorr multi-signatures. In: Advances in Cryptology – CRYPTO' 21. pp. 189–221 (2021)
52. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing. p. 51–59. PODC '91, Association for Computing Machinery, New York, NY, USA (1991). https://doi.org/10.1145/112600.112605, https://doi.org/10.1145/112600.112605
53. Pointcheval, D., Stern, J.: Security arguments for digital signatures and blind signatures. Journal of Cryptology **13**(3), 361–396 (2000)
54. Rabin, T.: A simplified approach to threshold and proactive RSA. In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 89–104. Springer, Heidelberg (Aug 1998). https://doi.org/10.1007/BFb0055722
55. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999). https://doi.org/10.1109/SFFCS.1999.814628
56. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology – CRYPTO '89. pp. 239–252 (1989)
57. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)

A A Separation Between acc-0 and acc-1

In Section 3.2 we defined two notions of accountability $\mathsf{acc}\text{-}0$ and $\mathsf{acc}\text{-}1$. In this section, we give an example that separates these two notions. We do so by presenting a simple ATS-PR construction that satisfies our $\mathsf{acc}\text{-}0$ definition, but is not $\mathsf{acc}\text{-}1$. The construction makes use of the following two building blocks:

- a threshold signature scheme with proactive refresh PRTS that need not be accountable, and
- an accountable threshold signature scheme ATS that need not support a proactive refresh.

Both can be constructed directly from Schnorr or BLS threshold signatures.

The proposed ATS-PR construction simply signs every message twice: once with ATS and once with PRTS. The complete scheme, called PRATS_2, is presented in Figure 2. The public key is the concatenation of an ATS public key and a PRTS public key. Each party's secret key is made up of a secret key to each of the schemes. To sign a message, the signers sign it twice – once using the accountable scheme ATS, and once using the non-accountable scheme PRTS. Verification checks that both signatures are valid. The main point is that to refresh the keys, the parties refresh their PRTS keys, leaving their ATS keys unchanged.

PRATS_2: A simple ATS with proactive refresh

$\mathsf{PRATS}_2.\mathsf{KGen}(\mathsf{pp}, n, t)$:

1. Sample $(\mathsf{PRTS.pk},\ \mathsf{PRTS.pkc},\ (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n)) \leftarrow\!\!\$\ \mathsf{PRTS.KGen}(\mathsf{pp}, n, t)$.
2. Sample $(\mathsf{ATS.pk},\ \mathsf{ATS.pkc},\ (\mathsf{ATS.sk}_1, \ldots, \mathsf{ATS.sk}_n)) \leftarrow\!\!\$\ \mathsf{ATS.KGen}(\mathsf{pp}, n, t)$.
3. For $i = 1, \ldots, n$ set $\mathsf{sk}_i \leftarrow (\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i)$.
4. Output $(\mathsf{pk} = (n, t, \mathsf{PRTS.pk}, \mathsf{ATS.pk}),\ \mathsf{pkc} = (\mathsf{ATS.pkc}, \mathsf{PRTS.pkc}),\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\mathsf{PRATS}_2.\mathsf{Sign}(\mathsf{sk}_i, m)$:

1. Parse sk_i as $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i)$.
2. Invoke $\mathsf{ATS.Sign}(\mathsf{ATS.sk}_i, m)$ and $\mathsf{PRTS.Sign}(\mathsf{PRTS.sk}_i, m)$ and let $s^{\mathrm{acc}}, s^{\mathrm{pr}}$ denote the outputs respectively.
3. Output $s_i \leftarrow (s^{\mathrm{acc}}, s^{\mathrm{pr}})$.

$\mathsf{PRATS}_2.\mathsf{Combine}(\mathsf{pkc}, \{s_i\}_{i \in \mathcal{J}})$:

1. Parse pkc as $\mathsf{ATS.pkc}, \mathsf{PRTS.pkc}$ and each s_i as $(s_i^{\mathrm{acc}}, s_i^{\mathrm{acc}})$ for $i \in \mathcal{J}$.
2. Invoke $\sigma^{\mathrm{acc}} \leftarrow \mathsf{ATS.Combine}(\mathsf{ATS.pkc}, \{s_i^{\mathrm{acc}}\}_{i \in \mathcal{J}})$ and $\sigma^{\mathrm{pr}} \leftarrow \mathsf{PRTS.Combine}(\mathsf{PRTS.pkc}, \{s_i^{\mathrm{pr}}\}_{i \in \mathcal{J}})$.
3. Output $\sigma \leftarrow (\sigma^{\mathrm{acc}}, \sigma^{\mathrm{pr}})$.

$\mathsf{PRATS}_2.\mathsf{Vf}(\mathsf{pk}, m, \sigma)$:

1. Parse pk as $(n, t, \mathsf{PRTS.pk}, \mathsf{ATS.pk})$ and σ as $(\sigma^{\mathrm{acc}}, \sigma^{\mathrm{pr}})$.
2. Output 1 if $\mathsf{PRTS.Vf}(\mathsf{PRTS.pk}, m, \sigma^{\mathrm{pr}})$ and $\mathsf{ATS.Vf}(\mathsf{ATS.pk}, m, \sigma^{\mathrm{acc}})$. Otherwise, output 0.

$\mathsf{PRATS}_2.\mathsf{Trace}(\mathsf{pk}, m, \sigma)$:

1. Parse σ as $(\sigma^{\mathrm{acc}}, \sigma^{\mathrm{pr}})$ and pk as $(n, t, \mathsf{PRTS.pk}, \mathsf{ATS.pk})$.
2. Verify that $\mathsf{PRATS}_2.\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1$ and otherwise, output $\perp$.
3. Output $\mathcal{J} \leftarrow \mathsf{ATS.Trace}(\mathsf{ATS.pk}, m, \sigma^{\mathrm{acc}})$.

$\mathsf{PRATS}_2.\mathsf{Update}(\mathsf{sk}_i, \mathsf{pk})$:

1. Parse sk_i as $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i)$ and pk as $(n, t, \mathsf{PRTS.pk}, \mathsf{ATS.pk})$.
2. Run $\mathsf{PRTS.Update}(\mathsf{PRTS.sk}_i, \mathsf{PRTS.pk})$ and let $\mathsf{PRTS.sk}'_i$ be the output of signer i.
3. Output $\mathsf{sk}'_i \leftarrow (\mathsf{PRTS.sk}'_i, \mathsf{ATS.sk}_i)$.

Fig. 2. The ATS-PR $\mathbf{PRATS}_2$

Correctness. Correctness of PRATS_2 follows directly from the correctness of the two underlying schemes PRTS and ATS.

Security. The scheme clearly inherits the same notion of unforgeability (uf-$\mathsf{0}$ or uf-$\mathsf{1}$) as the underlying PRTS scheme. We next study its accountability properties. We show in Theorem 6 that PRATS_2 is acc-$\mathsf{0}$ if the (non-refreshable) ATS scheme is accountable. Recall that a non-refreshable ATS scheme is accountable if it satisfies our acc-$\mathsf{0}$ definition when there is only one epoch. We then show in Theorem 7 that PRATS_2 does not satisfy acc-$\mathsf{1}$. This separation shows that acc-$\mathsf{0}$ is much easier to achieve than $\mathsf{acc\text{-}1}$. We also note that our practical $\mathsf{acc\text{-}0}$ schemes in Section 7 are much more efficient than the generic PRATS_2 construction in Figure 2.

Theorem 6. PRATS_2 *is* $\mathsf{acc\text{-}0}$ *secure assuming* ATS *is* $\mathsf{acc\text{-}0}$ *secure. In particular, for every adversary $\mathcal{A}$, there exists another adversary $\mathcal{B}$, such that for every set* pp *of public parameters it holds that*

$$\mathsf{Adv}^{\text{acc-0}}_{\mathsf{PRATS}_2[\mathsf{pp}]}(\mathcal{A}) = \mathsf{Adv}^{\text{acc-0}}_{\mathsf{ATS}[\mathsf{pp}]}(\mathcal{B}),$$

Proof. Consider the following adversary $\mathcal{B}$ playing the game $\mathbf{G}^{\text{acc-1}}_{\mathsf{ATS}[\mathsf{pp}]}$. $\mathcal{B}$ invokes $\mathcal{A}(\mathsf{pp})$ and simulates $\mathbf{G}^{\text{acc-0}}_{\mathsf{PRATS}_2[\mathsf{pp}]}$ as follows:

1. Receive (n, t, E) from $\mathcal{A}$. Forward (n, t) to the challenger in $\mathbf{G}^{\text{acc-1}}_{\mathsf{ATS}[\mathsf{pp}]}$.
2. Receive $\mathsf{ATS.pk}^*$ from the challenger in $\mathbf{G}^{\text{acc-1}}_{\mathsf{ATS}[\mathsf{pp}]}$.
   Sample $(\mathsf{PRTS.pk}, \mathsf{PRTS.pkc}, (\mathsf{PRTS.sk}^1_1, \ldots, \mathsf{PRTS.sk}^1_n)) \leftarrow\$ \mathsf{PRTS.KGen}(n, t)$ and send the public key $\mathsf{PRATS}_2.\mathsf{pk} = (n, t, \mathsf{PRTS.pk}, \mathsf{ATS.pk}^*)$ to $\mathcal{A}$.
3. For $e \in 2, .., E$, run $(\mathsf{PRTS.sk}^e_1, \ldots, \mathsf{PRTS.sk}^e_n) \leftarrow \mathsf{PRTS.Update}(\mathsf{PRTS.pk}, \{\mathsf{PRTS.sk}^{e-1}_i\}_{i\in[n]})$.
4. Answer $\mathcal{A}$'s signing and secret key queries as follows:
   - $\mathsf{skO}(e, i)$: $\mathcal{B}$ forwards the query $\mathsf{skO}(i)$ to its challenger and gets the response $\mathsf{ATS.sk}_i$. Then $\mathcal{B}$ sends $(\mathsf{PRTS.sk}^e_i, \mathsf{ATS.sk}_i)$ to $\mathcal{A}$.
   - $\mathsf{SignO}(m, e, i)$: $\mathcal{B}$ forwards the query $\mathsf{SignO}(m, i)$ to its challenger and gets back s^{acc}_i. It then runs $s^{\text{pr}}_i \leftarrow \mathsf{PRTS.Sign}(\mathsf{PRTS.sk}^e_i, m)$ and sends $(s^{\text{acc}}_i, s^{\text{pr}}_i)$ to $\mathcal{A}$.
5. Finally $\mathcal{B}$ receives a forgery (m^*, σ^*) from $\mathcal{A}$ where $\sigma^* = (\sigma^*_{\text{acc}}, \sigma^*_{\text{pr}})$. Our $\mathcal{B}$ outputs $(m^*, \sigma^*_{\text{acc}})$.

Observe that whenever the output of the simulated $\mathbf{G}^{\text{acc-0}}_{\mathsf{PRATS}_2[\mathsf{pp}]}$ game is 1, the forgery outputted by $\mathcal{A}$ is a valid one, and in particular σ^*_{acc} is a valid ATS signature on m^* with respect to $\mathsf{ATS.pk}^*$. Hence, the forgery output by $\mathcal{B}$ is a valid ATS message-signature pair with respect to $\mathsf{ATS.pk}^*$.

Next, since (m^*, σ^*) is a valid acc-0 forgery with respect to PRATS_2, there exists an index i^* in $\mathsf{PRATS}_2.\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*)$, but i^* is not in $\cup_{e\in[E]} \left(\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)\right)$. This means that i^* is in $\mathsf{ATS.Trace}(\mathsf{ATS.pk}, m^*, \sigma^*_{\text{acc}})$, but $\mathcal{B}$ never forwards any query of the form $\mathsf{skO}(i^*)$ or $\mathsf{SignO}(m^*, i^*)$ to its challenger. Hence, $\mathcal{B}$ produces a valid acc-0 forgery for the ATS scheme. ◻

Theorem 7. *The construction* PRATS_2 *is not* $\mathsf{acc\text{-}1}$ *secure. In particular, there is a PPT adversary $\mathcal{A}$ such that*

$$\mathsf{Adv}^{\text{acc-1}}_{\mathsf{PRATS}_2[\mathsf{pp}]}(\mathcal{A}) = 1.$$

Proof. We construct $\mathcal{A}$ as follows. $\mathcal{A}$ chooses some $0 < t < n$ and sets the number of epochs E to $E = 2$. Next, $\mathcal{A}$ chooses an arbitrary message m^*, and in the first epoch, asks signers $1, \ldots, t$ for their signature shares on m^*. In the second epoch, $\mathcal{A}$ asks signer $t + 1$ for its signature share on m^*. It then constructs the signature forgery $\sigma^* = (\sigma^*_{\text{acc}}, \sigma^*_{\text{pr}})$ as follows:

- $\mathcal{A}$ calls $\mathsf{PRTS.Combine}$ on the t partial PRTS signatures it obtained in the first epoch, to get a valid PRTS signature on m^*.
- Since ATS is not refreshed across epochs, $\mathcal{A}$ calls $\mathsf{ATS.Combine}$ on $t - 1$ ATS signature shares from the first epoch, say shares $2, \ldots, t$, along with the single signature share from the second epoch. This gives $\mathcal{A}$ a valid ATS sign on m^*, combined over the set of signers $\{2, \ldots, t, t + 1\}$.

Hence, $\mathcal{A}$ obtains a valid PRATS_2 signature: verification succeeds for both halves of the signature σ^*. Now, $\mathsf{PRATS}_2.\mathsf{Trace}(m^*, \sigma^*)$ calls $\mathsf{ATS}.\mathsf{Trace}(m^*, \sigma^*_{\mathrm{acc}})$ which returns the set $\{2, \ldots, t, t+1\}$. This set is not a subset of $\mathcal{Q}_e^{\mathsf{sk}} \cup \mathcal{Q}_e^{\mathsf{sig}}(m^*)$ for either one of the two epochs $e = 1$ or $e = 2$. Therefore, (m^*, σ^*) is a valid acc-1 forgery. $\square$

B Handling Malicious Corruptions During Key Refreshes

In this section, we generalize our security definitions from Section 3.2 to capture malicious corruptions during key updates. We then present a generic compiler that lifts a construction from the semi-honest updates setting to the malicious updates setting.

The communication model. Following previous works that considered malicious corruptions during key updates [52, 40, 39, 32, 31, 54, 5, 33, 21, 2, 42, 24, 44], we assume that all signers are connected by an authenticated and synchronous broadcast channel. As observed, for example, by [24], the use of authenticated communication is necessary for obtaining proactive security. In the absence of authenticated communication, an adversary that formally "left" a previously corrupted party and controls all the communication between that party and the rest of the network can continue to impersonate that party indefinitely.

We assume for simplicity that all communication is public, and every message that is sent will be received by all parties in the next round, or not at all. Note that secret point-to-point channels can still be emulated by encrypting messages to their intended receivers. In more detail, at the beginning of the update protocol, each party i samples a fresh key pair $(\mathsf{ek}_i, \mathsf{dk}_i)$ for a semantically-secure public-key encryption scheme, to be used solely during this execution of the update protocol. They then broadcast the encryption key ek_i to all other parties. When party j wishes to send a private message to i, it simply encrypts it under ek_i and broadcasts the ciphertext. Note that semantic security is sufficient because the broadcast channel is authenticated.

Updated syntax for Update_0. We need to slightly change the syntax of the update protocol from Section 3 to accommodate the fact that all communication is now done via a broadcast channel. The update protocol is still specified by a pair $(\mathsf{Update}_0, \mathsf{Update}_1)$ of algorithms that are defined as in Section 3 except that now, Update_0 outputs a single update message δ_i^e. Each signer i broadcasts δ_i^e to all other signers, and each signer computes their new secret key by invoking Update_1 on sk_i^e and $\delta_1^e, \ldots, \delta_n^e$. As explained in the previous paragraph, point-to-point communication is still possible with appropriate use of encryption.

B.1 Security in the Face of Semi-Honest Corruptions

Before presenting our security definition that accommodates maliciously corrupting parties during updates, we first present an intermediate definition that considers semi-honest corruptions during key updates. By semi-honestly corrupting a subset of the parties during the update protocol, the adversary learns their secret state, and can determine their randomness for the update protocol (that is, their randomness for Update_0, which without loss of generality is the only randomized algorithm specifying the update protocol). Note that if the update protocol uses private communication via the use of public-key encryption as described above, then this allows the adversary to know their decryption keys and hence observe all (decrypted) incoming communication to these parties.

The security definitions of unforgeability and accountability in the face of semi-honest corruptions during updates are a natural generalization of the definitions found in Section 3. The difference is this – recall that in our earlier definitions, the challenger honestly generated the secret signing keys of all signers for all epochs in the beginning of the game. Instead, we now provide the adversary with an additional oracle for incrementing the current epoch, while choosing a subset C of signers to corrupt during the key update and choosing their random coins for the update protocol. The challenger then emulates the update protocol conditioned on these random coins, resulting in secrets for the next epoch and a transcript for the protocol. The latter is given to the adversary.

As typical in the setting of proactive security, we treat all signers in C as corrupted for both the preceding and subsequent epochs. This is necessary since, given their secret key from the previous epoch, their randomness for the update protocol, and the transcript of the update protocol, the adversary can compute their secret key for the subsequent epoch. To make sure that signers in C are "counted" as corrupted in the preceding epoch, we enforce that the adversary has asked for the secret keys of all signers in C before it queries the update oracle on C. Note that this does not restrict the power of the adversary since they can always request the secret keys of all signers in C before querying the new update oracle. Finally, the signers in C are "marked" as corrupted for the subsequent epoch as well.

The security games capturing this updated definition are presented in Fig. 3. For simplicity of presentation, the games in Fig. 3 consider only non-interactive updates (per the syntax from Section 3), but they naturally generalize to consider interactive update protocols as well.

Observe that our constructions remain secure even in the face of semi-honest corruptions, with essentially the same proofs of security. We discuss this fact in more detail in Section 8.

Allowing for skewed randomness. Note that our notion of semi-honest security gives more leeway to the adversary than what is typical when considering semi-honest security: It allows the adversary to maliciously choose the randomness used to generate its messages during the protocol. Typically, semi-honest adversaries are assumed to honestly sample their randomness. We choose to define semi-honest security this way since it enables us to use a simpler and more efficient compiler to handle malicious corruptions (starting from a more secure protocol means less work for the compiler). We can do that since our bare-bones protocols remain secure even under this stronger notion of semi-honest security. It is worth mentioning, though, that the compiler can be augmented to handle protocols that assume the adversary also honestly samples the randomness for corrupted parties using standard techniques [34, 35].

B.2 Maliciously-Corrupted Updates

We now define security in settings where the adversary can maliciously corrupt signers during key updates.

Two-round key updates. The syntax from Section 3 considered non-interactive updates. Looking ahead, our generic compiler will add one round of interaction to the key update protocol. Hence, in our security definitions considering maliciously-corrupted updates, we restrict ourselves to two-round update protocols. However, our definitions readily extend to capture arbitrary update protocols with more rounds.

Concretely, a two-round update protocol is specified by a tuple of three algorithms, $\mathsf{Update} = (\mathsf{Update}_0, \mathsf{Update}_1, \mathsf{Update}_2)$:

– $\mathsf{Update}_0(\mathsf{sk}_i^e, \mathsf{pk}) \to (\mathsf{st}_{i,0}, \delta_{i,0}^e)$ is a randomized algorithm that takes in a secret key sk_i^e of signer i in epoch e and the public key pk, and outputs a state $\mathsf{st}_{i,0}$ and a first message $\delta_{i,0}^e$. Each signer i broadcasts $\delta_{i,0}^e$ to all other signers.

Games $\mathbf{G}^{\text{shu,uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{shu,acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{shu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$

$\mathsf{flag}_{\text{shu,uf-0}}, \mathsf{flag}_{\text{shu,uf-1}}, \mathsf{flag}_{\text{shu,acc-0}}, \mathsf{flag}_{\text{shu,acc-1}} \leftarrow 0$
$(\mathsf{st}, n, t, E) \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp}, n, t)$
$e_{\text{curr}} \leftarrow 1$ // the current epoch number
for $e \in [E] \wedge x \in [n] \wedge y \in [n]$ **do**
  $\delta^e_{x,y,0} \leftarrow \perp, \delta^e_{x,y,1} \leftarrow \perp$
$(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot),\mathsf{SignO}(\cdot,\cdot,\cdot),\mathsf{UpdateO}(\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc})$
if $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 0$ **then**
  return 0
if $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e| < t \wedge |\mathcal{Q}^{\mathsf{sig}}_e(m^*)| = 0$ **then** $\mathsf{flag}_{\text{shu,uf-0}} \leftarrow 1$
if $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)| < t$ **then** $\mathsf{flag}_{\text{shu,uf-1}} \leftarrow 1$
if $\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \cup_{e\in[E]} \left(\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)\right)$ **then** $\mathsf{flag}_{\text{shu,acc-0}} \leftarrow 1$
if $\forall e \in [E], \mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)$ **then** $\mathsf{flag}_{\text{shu,acc-1}} \leftarrow 1$
Only game $\mathbf{G}^{\text{shu,uf-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{shu,uf-}b}$
Only game $\mathbf{G}^{\text{shu,acc-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{shu,acc-}b}$
Only game $\mathbf{G}^{\text{shu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{shu,uf-}b} \vee \mathsf{flag}_{\text{shu,acc-}b'}$

Oracle $\mathsf{skO}(e, i)$

if $e > e_{\text{curr}}$ **then**
  return $\perp$
$\mathcal{Q}^{\mathsf{sk}}_e \leftarrow \mathcal{Q}^{\mathsf{sk}}_e \cup \{i\}$
return sk^e_i

Oracle $\mathsf{SignO}(m, e, i)$

if $e > e_{\text{curr}}$ **then**
  return $\perp$
$\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}^e_i, m)$
$\mathcal{Q}^{\mathsf{sig}}_e(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}_e(m) \cup \{i\}$
return σ_i

Oracle $\mathsf{UpdateO}(C, \{r_i\}_{i\in C})$

if $C \not\subseteq \mathcal{Q}^{\mathsf{sk}}_{e_{\text{curr}}} \vee e_{\text{curr}} = E$ **then**
  return $\perp$
for $i \in C$ **do**
  $\delta_i \leftarrow \mathsf{Update}_0(\mathsf{pk}, \mathsf{sk}^{e_{\text{curr}}}_i; r_i)$
for $i \in [n] \setminus C$ **do**
  $\delta_i \leftarrow\!\!\$\ \mathsf{Update}_0(\mathsf{pk}, \mathsf{sk}^{e_{\text{curr}}}_i)$
for $i \in [n]$ **do**
  $\mathsf{sk}^{e_{\text{curr}}+1}_i \leftarrow \mathsf{Update}_1(\mathsf{sk}^{e_{\text{curr}}}_i, \{\delta_j\}_{j\in[n]})$
$e_{\text{curr}} \leftarrow e_{\text{curr}} + 1$
$Q^{\mathsf{sk}}_{e_{\text{curr}}} \leftarrow Q^{\mathsf{sk}}_{e_{\text{curr}}} \cup C$
return $\{\delta_j\}_{j\in[n]\setminus C}$

Fig. 3. The security games $\mathbf{G}^{\text{shu,uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{shu,acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{shu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$ for $b, b', b'' \in \{0, 1\}$ for an ATS-PR scheme $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ with public parameters pp. For a set $\mathcal{X}$ and an element x, we let $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$ be a shorthand for the following operation: If $\mathcal{X}$ was previously defined, then set $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$; if $\mathcal{X}$ is still undefined, then set $\mathcal{X} = \{x\}$.

– $\mathsf{Update}_1\big(\mathsf{st}_{i,0}, \{\delta^e_{i,0}\}_{i\in[n]}\big) \to (\mathsf{st}_{i,1}, \delta^e_{i,1})$ is a deterministic algorithm, that takes in a state $\mathsf{st}_{i,0}$ and n update messages $\delta^e_{1,0}, \dots, \delta^e_{n,0}$, and outputs an updated state $\mathsf{st}_{i,1}$ and a second message $\delta^e_{i,1}$. Each signer i broadcasts $\delta^e_{i,1}$ to all other signers. Note that a randomized Update_1 can be always made deterministic by including its randomness as part of $\mathsf{st}_{i,0}$.
– $\mathsf{Update}_2\big(\mathsf{st}_{i,1}, \{\delta^e_{i,1}\}_{i\in[n]}\big) \to \mathsf{sk}^{e+1}_i$ is a deterministic algorithm that takes in a state $\mathsf{st}_{i,1}$ and n update messages $\delta^e_{1,1}, \dots, \delta^e_{n,1}$. It outputs an updated secret key sk^{e+1}_i for epoch $e+1$ for signer i.

The definitions of correctness and trace correctness are straightforward generalizations of the definitions found in Section 3, and are obtained by replacing the honest executions of the non-interactive update protocol with honest executions of the two-round update protocol.

Security. Unforgeability and accountability in the face of malicious corruptions during key updates are defined through the security games in Fig. 4. These games are obtained by endowing the adversary with the ability to arbitrarily choose the messages of corrupted parties during the update protocol. We assume the adversary is rushing; that is, in each round, they can choose the messages of corrupted parties after observing the messages of the honest parties for that round.

Concretely, the security games in Fig. 4 are obtained by augmenting the semi-honest corruption security games with oracles that allow the adversary to observe the messages of honest parties and then choose the message of the corrupted parties in a round-by-round manner.

The following definition extends the notion of adversarial advantage to the setting where the adversary can corrupt signers during key updates, either semi-honestly or maliciously. The relevant games are defined in Figures 3 and 4.

Definition 6. *Let* $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ *be an ATS-PR scheme with public parameters* pp*, let* $\mathrm{type} \in \{\mathrm{shu}, \mathrm{mu}\}$*, and let* $\mathrm{prop} \in \{\text{uf-}b, \text{acc-}b', \text{uf-}b \wedge \text{acc-}b'\}_{b,b'\in\{0,1\}}$*. The advantage of an adversary* $\mathcal{A}$ *in* $\mathbf{G}^{\mathrm{type},\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ *is defined as*

$$\mathsf{Adv}^{\mathrm{type},\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \Pr\left[\mathbf{G}^{\mathrm{type},\mathrm{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}(\mathcal{A}) = 1\right].$$

Detecting deviation. We assume that whenever an honest party detects a deviation from the protocol during the update protocol, it updates its secret key to $\perp$. Hence, in this case, the output of the security game is 0. The question of what to do upon such a detection may depend on the system within which the ATS-PR scheme is used, and is orthogonal to this work. However, looking ahead, it is worth mentioning that our compiler will enjoy the property that if one honest party identifies another party as malicious, then so do all other honest parties.

B.3 The General Compiler: Construction & Security

We now present the general compiler. The compiler transforms any ATS-PR scheme that is secure facing semi-honest corruptions during updates into a scheme which is secure against malicious corruptions. The compiler is based on two building blocks: A commitment scheme and a zero-knowledge proof system. We briefly present our requirements of these primitives here. For readability, we defer the formal definitions to the end of this appendix. Concretely, our compiler makes use of:

1. An extractable statistically-binding non-interactive commitment scheme [23, 50, 1] $\mathsf{COM} = (\mathsf{COM.Setup}, \mathsf{COM.Commit}, \mathsf{COM.Verify}, \mathsf{COM.Ext})$. That is, with overwhelming probability over the choice of $\mathsf{crs}_{\mathsf{COM}} \leftarrow\!\$\ \mathsf{COM.Setup}$, *there are no values* $\mathsf{com}, m, m', \mathsf{decom}, \mathsf{decom}'$ such that $m \neq$

Games $\mathbf{G}^{\text{mu,uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{mu,acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{mu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$

$\mathsf{flag}_{\text{mu,uf-0}}, \mathsf{flag}_{\text{mu,uf-1}}, \mathsf{flag}_{\text{mu,acc-0}}, \mathsf{flag}_{\text{mu,acc-1}}\mathsf{flag}_{\text{detected}} \leftarrow 0$
$(\mathsf{st}, n, t, E) \leftarrow \mathcal{A}(\mathsf{pp})$
$(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\$ \mathsf{KGen}(\mathsf{pp}, n, t)$
$e_{\text{curr}} \leftarrow 1$ // the current epoch number
$\text{ind-upd} \leftarrow 0$
for $e \in [E] \wedge x \in [n] \wedge y \in [n]$ **do**
  $\delta^e_{x,y,0} \leftarrow \bot, \delta^e_{x,y,1} \leftarrow \bot$
$(m^*, \sigma^*) \leftarrow\$ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot),\mathsf{SignO}(\cdot,\cdot,\cdot),\mathsf{UpdateO}_0(\cdot),\mathsf{UpdateO}_1(\cdot,\cdot),\mathsf{UpdateO}_2(\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc})$
if $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 0$ **then return** 0
if $\mathsf{flag}_{\text{detected}} = 1$ **then return** 0
if $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e| < t \wedge |\mathcal{Q}^{\mathsf{sig}}_e(m^*)| = 0$ **then** $\mathsf{flag}_{\text{mu,uf-0}} \leftarrow 1$
if $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)| < t$ **then** $\mathsf{flag}_{\text{mu,uf-1}} \leftarrow 1$
if $\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \cup_{e\in[E]} \left(\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)\right)$ **then** $\mathsf{flag}_{\text{mu,acc-0}} \leftarrow 1$
if $\forall e \in [E], \mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)$ **then** $\mathsf{flag}_{\text{mu,acc-1}} \leftarrow 1$
Only game $\mathbf{G}^{\text{mu,uf-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{mu,uf-}b}$
Only game $\mathbf{G}^{\text{mu,acc-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{mu,acc-}b}$
Only game $\mathbf{G}^{\text{mu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{mu,uf-}b} \vee \mathsf{flag}_{\text{mu,acc-}b'}$

Oracle $\mathsf{skO}(e, i)$

if $e > e_{\text{curr}}$ **then**
  return $\bot$
$\mathcal{Q}^{\mathsf{sk}}_e \leftarrow \mathcal{Q}^{\mathsf{sk}}_e \cup \{i\}$
return sk^e_i

Oracle $\mathsf{SignO}(m, e, i)$

if $e > e_{\text{curr}}$ **then**
  return $\bot$
$\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}^e_i, m)$
$\mathcal{Q}^{\mathsf{sig}}_e(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}_e(m) \cup \{i\}$
return σ_i

Oracle $\mathsf{UpdateO}_0(C)$

if $C \not\subseteq \mathcal{Q}^{\mathsf{sk}}_{e_{\text{curr}}} \vee e_{\text{curr}} = E \vee \text{ind-upd} \neq 0$ **then**
  return $\bot$
$\text{ind-upd} \leftarrow 1$
for $i \in [n] \setminus C$ **do**
  $(\mathsf{st}^{e_{\text{curr}}}_{i,0}, \delta^{e_{\text{curr}}}_{i,0}) \leftarrow\$ \mathsf{Update}_0(\mathsf{pk}, \mathsf{sk}^{e_{\text{curr}}}_i)$
return $\{\delta^{e_{\text{curr}}}_{i,0}\}_{i\in[n]\setminus C}$

Oracle $\mathsf{UpdateO}_1(\{\delta^{e_{\text{curr}}}_{i,0}\}_{i\in C})$

if $\text{ind-upd} \neq 1$ **then**
  return $\bot$
$\text{ind-upd} \leftarrow 2$
for $i \in [n] \setminus C$ **do**
  $(\mathsf{st}^{e_{\text{curr}}}_{i,1}, \delta^{e_{\text{curr}}}_{i,1}) \leftarrow\$ \mathsf{Update}_1(\mathsf{st}^{e_{\text{curr}}}_{i,0}, \{\delta^{e_{\text{curr}}}_{j,0}\}_{j\in[n]})$
return $\{\delta^{e_{\text{curr}}}_{i,1}\}_{i\in[n]\setminus C}$

Oracle $\mathsf{UpdateO}_2(\{\delta^{e_{\text{curr}}}_{i,1}\}_{i\in C})$

if $\text{ind-upd} \neq 2$ **then**
  return $\bot$
for $i \in [n]$ **do**
  $\mathsf{sk}^{e_{\text{curr}}+1}_i \leftarrow \mathsf{Update}_2(\mathsf{st}^{e_{\text{curr}}}_{i,1}, \{\delta^{e_{\text{curr}}}_{j,1}\}_{j\in[n]})$
  if $\mathsf{sk}^{e_{\text{curr}}+1}_i = \bot$ **then** $\mathsf{flag}_{\text{detected}} \leftarrow 1$
$e_{\text{curr}} \leftarrow e_{\text{curr}} + 1$
$Q^{\mathsf{sk}}_{e_{\text{curr}}} \leftarrow Q^{\mathsf{sk}}_{e_{\text{curr}}} \cup C$
$\text{ind-upd} \leftarrow 0$

Fig. 4. The security games $\mathbf{G}^{\text{mu,uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{mu,acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{mu,uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$ for $b, b', b'' \in \{0, 1\}$ for an ATS-PR scheme $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ with public parameters pp.

m',
$\mathsf{COM.Verify}(\mathsf{crs}_{\mathsf{COM}}, \mathsf{com}, m, \mathsf{decom}) = 1$ and $\mathsf{COM.Verify}(\mathsf{crs}_{\mathsf{COM}}, \mathsf{com}, m', \mathsf{decom}') = 1$. By "extractable", we mean that it is possible to generate a CRS for COM in a way that endows the generator with a trapdoor. Knowledge of this trapdoor can be used in order to extract committed values.

2. A non-interactive zero-knowledge proof system [13, 12, 29, 55, 26, 46, 36, 37] $\mathsf{NIZK} = (\mathsf{NIZK.Setup}, \mathsf{NIZK.P}, \mathsf{NIZK.V})$ for the language $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ defined as:

$$\mathcal{L}_\lambda = \left\{ (\mathsf{pp}, \mathsf{crs}_{\mathsf{COM}}, \mathsf{pk}, \mathsf{com}, \delta) \ : \ \begin{array}{c} \exists(\mathsf{sk}, r, \mathsf{decom}) \text{ s.t.} \\ \mathsf{COM.Verify}(\mathsf{crs}_{\mathsf{COM}}, \mathsf{com}, \mathsf{decom}, (\mathsf{sk}, r)) = 1 \text{ and} \\ \delta = \mathsf{SHU\text{-}ATS\text{-}PR.Update}_0(\mathsf{sk}, \mathsf{pk}; r) \end{array} \right\}.$$

We require that the scheme provides adaptive zero knowledge and simulation soundness. Adaptive zero knowledge states that zero-knowledge should be preserved even for (true) statements that are chosen as a function of the common reference string. Informally speaking, simulation soundness guarantees that a prover cannot produce a valid proof for a false statement, even after observing many simulated proofs (on statements of their choice).
Note that even though our ATS-PR constructions are in the random-oracle model, our update protocols (and in particular, our constructions for the Update_0 algorithm) are not. Hence, the language $\mathcal{L}$ is indeed in NP.

The compiler takes in an ATS-PR scheme SHU-ATS-PR[pp] that is secure against semi-honest updates. It builds an ATS-PR secure against malicious updates by utilizing the ideas behind the GMW compiler [34, 35]. At the beginning of each update protocol, all signers commit to their current secret key and the randomness to be used during the update protocol. Then, they append to each message sent during the protocol a NIZK proof certifying that this message is indeed consistent with the committed values and previously-transmitted messages. The compiled scheme MU-ATS-PR[pp] is defined below. We specify only the KGen algorithm and Update protocol, since all other algorithms remain unchanged.

MU-ATS-PR[pp]: An ATS-PR scheme supporting malicious updates

$\underline{\mathsf{KGen}(\mathsf{pp}, n, t)}$:

1. Invoke $(\mathsf{pk}', \mathsf{pkc}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)) \leftarrow\$ \mathsf{SHU\text{-}ATS\text{-}PR.KGen}(\mathsf{pp}, n, t)$.
2. Sample $\mathsf{crs}_{\mathsf{COM}} \leftarrow\$ \mathsf{COM.Setup}(1^\lambda)$ and $\mathsf{crs}_{\mathsf{NIZK}} \leftarrow\$ \mathsf{NIZK.Setup}(1^\lambda)$.
3. Set $\mathsf{pk} \leftarrow (\mathsf{pk}', \mathsf{crs}_{\mathsf{COM}}, \mathsf{crs}_{\mathsf{NIZK}})$.
4. Output $(\mathsf{pk}, \mathsf{pkc}, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

$\underline{\mathsf{Update}_0(\mathsf{sk}_i, \mathsf{pk})}$:

1. Parse pk as $(\mathsf{pk}', \mathsf{crs}_{\mathsf{COM}}, \mathsf{crs}_{\mathsf{NIZK}})$
2. Sample random coins r for $\mathsf{SHU\text{-}ATS\text{-}PR.Update}_0$.
3. Compute $(\mathsf{com}, \mathsf{decom}) \leftarrow\$ \mathsf{COM.Commit}(\mathsf{crs}_{\mathsf{COM}}, (\mathsf{sk}_i, r))$.
4. Set $\mathsf{st}_{i,0} \leftarrow (r, \mathsf{sk}_i, \mathsf{pk}, \mathsf{decom})$ and $\delta_{i,0} \leftarrow \mathsf{com}$.
5. Output $(\mathsf{st}_{i,0}, \delta_{i,0})$.

$\underline{\mathsf{Update}_1(\mathsf{st}_{i,0}, \{\delta_{j,0}\}_{j \in [n]})}$:

1. Parse $\mathsf{st}_{i,0}$ as $(r, \mathsf{sk}_i, \mathsf{pk}, \mathsf{decom})$, pk as $(\mathsf{pk}', \mathsf{crs}_{\mathsf{COM}}, \mathsf{crs}_{\mathsf{NIZK}})$, and $\delta_{j,0}$ as com_j for every $j \in [n]$.
2. Compute $(\mathsf{st}'_{i,1}, \delta'_{i,1}) \leftarrow \mathsf{SHU\text{-}ATS\text{-}PR.Update}_0(\mathsf{sk}_i, \mathsf{pk}; r)$.
3. Compute a proof $\pi \leftarrow\$ \mathsf{NIZK.P}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pp}, \mathsf{crs}_{\mathsf{COM}}, \mathsf{pk}', \mathsf{com}_i, \delta'_{i,1}), (\mathsf{sk}_i, r, \mathsf{decom}))$.

4. Set $\delta_{i,1} \leftarrow (\delta'_{i,1}, \pi)$ and $\mathsf{st}_{i,1} \leftarrow (\mathsf{pk}, \{\mathsf{com}_j\}_{j\in[n]}, \mathsf{st}'_{i,1})$.
5. Output $(\mathsf{st}_{i,1}, \delta_{i,1})$.

$\underline{\mathsf{Update}_2(\mathsf{st}_{i,1}, \{\delta_{j,1}\}_{j\in[n]})}$:

1. Parse $\mathsf{st}_{i,1}$ as $(\mathsf{pk}, \{\mathsf{com}_j\}_{j\in[n]}, \mathsf{st}'_{i,1})$, pk as $(\mathsf{pk}', \mathsf{crs}_{\mathsf{COM}}, \mathsf{crs}_{\mathsf{NIZK}})$, and $\delta_{j,1}$ as $(\delta'_{j,1}, \pi_j)$ for every $j \in [n] \setminus \{i\}$.
2. For every $j \in [n] \setminus \{i\}$: Assert that
$$\mathsf{NIZK.Vf}(\mathsf{crs}_{\mathsf{NIZK}}, (\mathsf{pp}, \mathsf{crs}_{\mathsf{COM}}, \mathsf{pk}', \mathsf{com}_j, \delta'_{j,1}), \pi_j) = 1.$$
If for any $j \in [n] \setminus \{i\}$ this check fails, set $\mathsf{sk}_i \leftarrow \perp$, outputs $\perp$, and terminate.
3. Compute $\mathsf{sk}_i \leftarrow \mathsf{SHU\text{-}ATS\text{-}PR.Update}_1(\mathsf{st}'_{i,1}, \{\delta'_{j,1}\}_{j\in[n]})$.
4. Output sk_i.

The security of the compiled protocol is captured by the following theorem, which states that if SHU-ATS-PR is secure against semi-honest corruptions during updates, then MU-ATS-PR is secure against malicious corruptions.

Theorem 8. *Let $b, b' \in \{0,1\}$. For any adversary $\mathcal{A}$ there exist an adversary $\mathcal{B}$ and a negligible function $\nu(\cdot)$, such that for every set $\mathsf{pp} = \{\mathsf{pp}_\lambda\}_\lambda$ of public parameters it holds that*
$$\mathsf{Adv}^{\text{mu,uf-b}\wedge\text{acc-b}'}_{\mathsf{MU\text{-}ATS\text{-}PR[pp]}}(\mathcal{A}) \leq \mathsf{Adv}^{\text{shu,uf-b}\wedge\text{acc-b}'}_{\mathsf{SHU\text{-}ATS\text{-}PR[pp]}}(\mathcal{A}) + \nu(\lambda),$$
for all sufficiently large $\lambda \in \mathbb{N}$.

As our compiler can be seen as a restricted application of the general GMW compiler [34],[6] Theorem 8 follows from the security of the latter. A detailed description of the GMW compiler and a proof of its security can be found in [35]. Nevertheless, for completeness, we briefly sketch the proof of Theorem 8 below.

Proof sketch. Let $\mathbf{G}_0 = \mathbf{G}^{\text{mu,uf-}b\wedge\text{acc-}b'}_{\mathsf{MU\text{-}ATS\text{-}PR[pp]}}$. Consider the game $\mathbf{G}_1$ obtained from $\mathbf{G}_0$ by the following revision revision: When replying to Update_1 queries by A, the challenger swaps the honestly generated NIZK proofs with simulated proofs. We argue that the probability that A wins in $\mathbf{G}_1$ is negligibly close to the probability that it wins in the $\mathbf{G}_0$ game. This is due to adaptive zero-knowledge property of NIZK. On input $\mathsf{crs}_{\mathsf{NIZK}}$ that is either honestly generated or simulated, the distinguisher D in the reduction will simulate either $\mathbf{G}_0$ or $\mathbf{G}_1$ to A. It will forward $\mathsf{crs}_{\mathsf{NIZK}}$ as part of the public key, generating all other components of the public key on its own. It will reply to all queries by A honestly, with the exception of generating the NIZK proofs as part of the replies to Update_1 queries: For that, it will query its oracle on the corresponding instance-witness pairs, and will obtain either honestly-generated proofs or simulated proofs. Finally, the distinguisher will output 1 if A wins in the simulated game and 0 otherwise. The probability that D outputs 1 given access to honestly-generated proofs is equal to the probability that A wins in $\mathbf{G}_0$, while the probability that D outputs 1 given access to simulated proofs is equal to the probability that A wins in $\mathbf{G}_1$.

Now consider the game $\mathbf{G}_2$ which is obtained from $\mathbf{G}_1$ by the following modification: When replying to Update_0 queries by A, the challenger replies with commitments to the all-zero strings as the first message of the honest parties. This is instead of honestly generating the commitments

[6] We apply the ideas from the GMW compiler to the specific case of key update protocols, whereas the GMW compiler can generally be applied to any semi-honest secure MPC protocol. Moreover, the full-fledged GMW compiler includes a randomness generation component which we do not need.

as commitments to the actual secret keys of the parties and randomness used for computing their $\mathsf{SHU\text{-}ATS\text{-}PR.Update}_0$ messages. We claim that the probability that A wins in this revised game is negligibly close to the probability that it wins in $\mathbf{G}_1$. This follows from a standard reduction to the hiding property of the commitment scheme.

The proof is concluded by presenting an adversary B that breaks the security of $\mathsf{SHU\text{-}ATS\text{-}PR}[\mathsf{pp}]$ with respect to *semi-honest updates*, with essentially the same probability as A wins in $\mathbf{G}_2$. The adversary B simulates the game $\mathbf{G}_2$ to A. To do so, it adds to the public key an honestly-generated CRS for COM and a simulated CRS for the NIZK proof system. To simulate oracle responses, B can forward A's queries to its own oracles, but it needs to add the commitments and NIZK proofs to the update messages returned to A. To this end, in each such commitment, B first commits to the all-zero string. Then, it uses the simulator of NIZK in order to produce the NIZK proofs. Note that even though the simulator is used to produce potentially false statements, a standard argument shows that this should go undetected by A, due to the hiding property of the commitment scheme.

Suppose that the CRS for COM is such that COM is perfectly binding (this is the case with overwhelming probability). Further, assume that all NIZK proofs produced by A are accepted by the verifier; otherwise, the output of the game is 0. Now, consider two cases: Either A produces a NIZK for a false statement, or it does not.

- **Case I:** Assume A produces a NIZK for a false statement. In this case, by the simulation soundness of NIZK, the probability that A wins in $\mathbf{G}_2$ is negligible.
- **Case II:** Assume A never produces a NIZK for a false statement. This means that message sent by the signers corrupted by B during updates can be explained by the secret keys and randomness to which these signers have committed (note that these are uniquely defined by the commitments since we are assuming a scenario in which COM is perfectly binding). We would like for B to use A's choice of randomness for the corrupted parties' update messages as randomness for their messages in the game $\mathbf{G}^{\text{shu,uf-}b\wedge\text{acc-}b'}_{\mathsf{SHU\text{-}ATS\text{-}PR}[\mathsf{pp}]}$. To do that, we use the fact that B generated the CRS for COM. Hence, B knows a trapdoor to this CRS and can extract the randomness to which A has committed to. B can thus forward this randomness as its input to the Update oracle.

Finally, when A outputs a forgery, B outputs the same forgery. The proof is then concluded by noting that whenever A wins in $\mathbf{G}_2$, B wins in $\mathbf{G}^{\text{mu,uf-}b\wedge\text{acc-}b'}_{\mathsf{MU\text{-}ATS\text{-}PR}[\mathsf{pp}]}$.

B.4 Building Blocks: Deferred Definitions

For completeness, we present formal definitions of the building blocks used by our compiler.

Extractable non-interactive statistically-binding commitment schemes. We rely on the following notion of an extractable non-interactive statistically-binding commitment scheme [23, 50, 1]:

Definition 7. *Let* $\epsilon_{\mathsf{binding}} = \epsilon_{\mathsf{binding}}(\lambda)$ *and* $\epsilon_{\mathsf{ext}} = \epsilon_{\mathsf{ext}}(\lambda)$ *be functions of the security parameter* $\lambda \in \mathbb{N}$*. An* ϵ_{ext}-extractable non-interactive $\epsilon_{\mathsf{binding}}$-statistically-binding commitment scheme *for a message space* $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ *is a 4-tuple of probabilistic polynomial-time algorithms* $\Pi = (\mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify}, \mathsf{Ext})$ *with the following properties:*[7]

[7] Often, the definition for extractable non-interactive commitments considers two separate setup procedures, an "honest" one and an alternative setup procedure that admits a trapdoor. The CRS generated by both procedures should be indistinguishable. Since, in the context of threshold signatures, we are assuming a trusted setup anyway, we use a simplified definition that considers a single setup algorithm that admits a trapdoor.

1. **Perfect completeness:** *For every* $\lambda \in \mathbb{N}$ *and* $m \in \mathcal{M}_\lambda$ *it holds that*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}, m) = 1 \;\middle|\; \begin{array}{c}(\mathsf{crs}, \tau) \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}(\mathsf{crs}, m)\end{array}\right] = 1$$

where the probability is taken over the internal randomness of Setup*,* Commit *and* Verify*.*

2. **Statistical binding:** *For every* $\lambda \in \mathbb{N}$ *it holds that*

$$\Pr\left[\begin{array}{c}\exists \mathsf{com}, m \neq m', \mathsf{decom}, \mathsf{decom}' \ s.t. \\ \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}, m) = \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}', m') = 1\end{array}\right] \leq \epsilon_{\mathsf{binding}}(\lambda)$$

where the probability is taken over the choice of $(\mathsf{crs}, \tau) \leftarrow \mathsf{Setup}(1^\lambda)$*.*

3. **Computational hiding:** *For every probabilistic polynomial-time algorithm* $A = (A_1, A_2)$ *there exists a negligible function* $\nu(\cdot)$ *such that*

$$\mathsf{Adv}^{\mathsf{hiding}}_{\Pi,A}(\lambda) \stackrel{\mathsf{def}}{=} \left|\Pr\left[\mathsf{Expt}^{\mathsf{hiding}}_{\Pi,A}(0, \lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{hiding}}_{\Pi,A}(1, \lambda) = 1\right]\right| \leq \nu(\lambda)$$

for all sufficiently large n*, where for each* $b \in \{0, 1\}$ *the experiment* $\mathsf{Expt}^{\mathsf{hiding}}_{\Pi,A}(b, \lambda)$ *is defined as:*

   (a) $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$.
   (b) $(m_0, m_1, \mathsf{state}) \leftarrow A_1(1^\lambda, \mathsf{crs})$.
   (c) $(\mathsf{com}, \mathsf{decom}) \leftarrow \mathsf{Commit}(\mathsf{crs}, m_b)$.
   (d) $b' \leftarrow A_2(\mathsf{crs}, \mathsf{com}, \mathsf{state})$.
   (e) Output b'*.*

4. **Extractability:** *For every* $\lambda \in \mathbb{N}$ *and for every* $\mathsf{com} \in \{0, 1\}^*$ *it holds that*

$$\Pr\left[\begin{array}{c}\exists \mathsf{decom} \ s.t. \ \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}, m) = 1 \ or \\ \forall \mathsf{decom}', m' \mathsf{Verify}(\mathsf{crs}, \mathsf{com}, \mathsf{decom}', m') = 0\end{array} \;\middle|\; \begin{array}{l}(\mathsf{crs}, \tau) \leftarrow\!\$\ \mathsf{Setup}(1^\lambda) \\ m \leftarrow \mathsf{Ext}(\mathsf{crs}, \tau, \mathsf{com})\end{array}\right]$$

is at least $1 - \epsilon_{\mathsf{ext}}(\lambda)$*, where the probability is taken over the choice of* $(\mathsf{crs}, \tau) \leftarrow \mathsf{Setup}(1^\lambda)$*.*

Non-interactive zero-knowledge proof systems. We rely on the following standard notion of a non-interactive simulation-sound adaptive zero-knowledge proof system [13, 12, 29, 55, 26, 46, 36, 37]:

Definition 8. *A* non-interactive simulation-sound zero-knowledge proof system *for a language* $\mathcal{L} = \{\mathcal{L}_\lambda\}_{\lambda \in \mathbb{N}}$ *with a witness relation* $R(\mathcal{L}) = \{R_\lambda\}_{\lambda \in \mathbb{N}}$ *is a tuple of probabilistic polynomial-time algorithms* $\Pi = (\mathsf{Setup}, \mathsf{P}, \mathsf{V})$*. We require the existence of a pair* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *of probabilistic polynomial-time algorithms such that the following properties hold:*

1. **Perfect completeness:** *For every* $\lambda \in \mathbb{N}$ *and* $(x, w) \in R_\lambda$ *it holds that*

$$\Pr\left[\mathsf{V}(1^\lambda, \mathsf{crs}, x, \pi) = 1 \;\middle|\; \begin{array}{c}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda) \\ \pi \leftarrow \mathsf{P}(1^\lambda, \mathsf{crs}, x, w)\end{array}\right] = 1$$

where the probability is taken over the internal randomness of Setup*,* P *and* V*.*

2. **Adaptive zero knowledge:** *For every probabilistic polynomial-time algorithm* A *there exists a negligible function* $\nu(\cdot)$ *such that*

$$\mathsf{Adv}^{\mathsf{zk}}_{\Pi,A}(\lambda) \stackrel{\mathsf{def}}{=} \left|\Pr\left[\mathsf{Expt}^{\mathsf{zk}}_{\Pi,A}(\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{zk}}_{\Pi,A,\mathsf{Sim}_1,\mathsf{Sim}_2}(\lambda) = 1\right]\right| \leq \nu(\lambda)$$

for all sufficiently large n*, where the experiment* $\mathsf{Expt}^{\mathsf{zk}}_{\Pi,A}(\lambda)$ *is defined as:*

(a) $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$.
(b) $b \leftarrow A^{\mathsf{P}(1^\lambda,\mathsf{crs},\cdot,\cdot)}(1^\lambda, \mathsf{crs})$.
(c) Output b.
and the experiment $\mathsf{Expt}^{\mathsf{zk}}_{\Pi,A,\mathsf{Sim}_1,\mathsf{Sim}_2}(\lambda)$ *is defined as:*
(a) $(\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda)$.
(b) $b \leftarrow A^{\mathsf{Sim}'_2(1^\lambda,\tau,\cdot,\cdot)}(1^\lambda, \mathsf{crs})$, *where* $\mathsf{Sim}'_2(1^\lambda, \tau, x, w) = \mathsf{Sim}_2(1^\lambda, \tau, x)$ *if* $(x, w) \in R_\lambda$ *and* $\mathsf{Sim}'_2(1^\lambda, \tau, x, w) = \perp$ *otherwise.*
(c) output b.

3. **Simulation soundness:** *For every probabilistic polynomial-time algorithm* A *there exists a negligible function* $\nu(\cdot)$ *such that*

$$\mathsf{Adv}^{\mathsf{ss}}_{\Pi,A}(\lambda) \stackrel{\mathsf{def}}{=} \Pr\left[\mathsf{Expt}^{\mathsf{ss}}_{\Pi,A}(\lambda) = 1\right] \leq \nu(\lambda)$$

for all sufficiently large $\lambda \in \mathbb{N}$, *where the experiment* $\mathsf{Expt}^{\mathsf{ss}}_{\Pi,A}(\lambda)$ *is defined as:*
(a) $(\mathsf{crs}, \tau) \leftarrow \mathsf{Sim}_1(1^\lambda)$.
(b) $(x, \pi) \leftarrow A^{\mathsf{Sim}_2(1^\lambda,\tau,\cdot)}(1^\lambda, \mathsf{crs})$.
(c) Denote by Q *the set of* Sim_2*'s query-response pairs.*
(d) Output 1 *if and only if* $x \notin \mathcal{L}_\lambda$, $(x, \pi) \notin Q$, *and* $\mathsf{V}(1^\lambda, \mathsf{crs}, x, \pi) = 1$.

C Interactive ATS-PR

In this section, we present our definitions for interactive ATS-PR schemes. We focus on schemes in which the signing protocol is made up of three rounds of communication among the signers. These definitions capture our (non-refreshable) ATS construction from Section 6 and our ATS-PR construction from Section 7.2.

Three-round threshold signature schemes. In addition to threshold signature schemes with non-interactive signing procedures (with or without a proactive refresh), we also consider schemes in which signing is done via an interactive protocol of three rounds or less. The syntax for such schemes closely follows that of non-interactive schemes, with the following exception. In interactive schemes, the signature algorithm Sign is now an interactive protocol, made up of three sub-algorithms $(\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3)$, where:

– Sign_1 is a randomized algorithm which takes as input a secret key sk_i, a message m, and a subset $\mathcal{J}$ of indices. It outputs a state $\mathsf{st}_{i,1}$ and a first message $\mathsf{msg}_{i,1}$ to be sent to all in round 1 of the protocol to all signers in $\mathcal{J} \setminus \{i\}$.
– Sign_2 is a deterministic algorithm which takes as input a state $\mathsf{st}_{i,1}$ and incoming messages $\{\mathsf{msg}_{j,1}\}_{j \in \mathcal{J}\setminus\{i\}}$. It outputs a state $\mathsf{st}_{i,2}$ and a second message $\mathsf{msg}_{i,2}$ to be sent to all in round 2 of the protocol to all signers in $\mathcal{J} \setminus \{i\}$.
– Sign_3 is a deterministic algorithm which takes as input a state $\mathsf{st}_{i,2}$ and incoming messages $\{\mathsf{msg}_{j,2}\}_{j \in \mathcal{J}\setminus\{i\}}$. It outputs a signature share s_i.

The syntax above requires knowledge of m and $\mathcal{J}$ at the beginning of the protocol (these are given as inputs to Sign_1). This captures the standard scenario in which a subset $\mathcal{J}$ of signers initiates the signing protocol in a coordinated manner in order to sign a particular message m. Defining the syntax in this manner also has the advantage of being general enough to capture protocols that require knowledge of m and $\mathcal{J}$ already in the onset of the protocol. It should be noted, however, that signers in our interactive protocols (Sections 6 and 7) do not require knowledge of m until the last message is computed, and require knowledge of $\mathcal{J}$ during the all-but-last rounds only in order to know to whom to send their all-but-last messages.

The correctness definitions for an interactive threshold siganture scheme are naturally extended from the non-interactive case, by replacing the non-interactive signing algorithm with an honest execution of the interactive signing protocol.

Security. Accountability and unforgeability of three-round threshold signatures are defined via a natural generalization of the analogous definitions for non-interactive schemes, presented in Figure 5. Importantly, the signing oracle is now replaced with three seperate oracles, one for each of the sub-routines making up the signing protocol. The adversary may query any of these oracles on inputs and ordering of its choice. In particular, this allows the adversary to interact with honest signers in many concurrent sessions of the signing protocol, arbitrarily interleaving between them.

For a three-round threshold signatures scheme with proactive refresh $\mathsf{PRTS} = (\mathsf{KGen}, \mathsf{Sign} = (\mathsf{Sign}_1, \mathsf{Sign}_2, \mathsf{Sign}_3), \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update} = (\mathsf{Update}_0, \mathsf{Update}_1))$, the security games capturing its unforgeability and accountability are defined below. As in the non-interactive case, the scheme and corresponding games are parameterized by public parameters pp. To avoid over-cluttering in notation, we assume that pp are included in the public key outputted by KGen.

D Deferred Proofs

D.1 Proof of Theorem 1

Proof. Let $b \in \{0,1\}$ and let $\mathcal{A}$ be an adversary participating in the $\mathbf{G}^{\text{uf-}b\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}$ security game. Consider the following algorithm $\mathcal{B}$ that takes part in the $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRTS[pp]}}$ security game. $\mathcal{B}$ takes as input the public parameters pp and simulates the game $\mathbf{G}^{\text{uf-}b\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}$ to $\mathcal{A}$ as follows:

1. Invoke $\mathcal{A}(\mathsf{pp})$ to obtain (n,t,E). Send (t,t,E) to the challenger in $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRTS[pp]}}$, and receive from the challenger $(\mathsf{pk}^*, \mathsf{pkc}^*)$.
2. Sample $\mathcal{J}^* \leftarrow\!\!\$\ \mathcal{S}_{t,n}$ and set $\mathsf{pk}^{\mathcal{J}^*} \leftarrow \mathsf{pk}^*$ and $\mathsf{pkc}^{\mathcal{J}^*} \leftarrow \mathsf{pkc}^*$.
3. For each $\mathcal{J} = \{j_1, \ldots, j_t\} \in \mathcal{S}_{t,n} \setminus \{\mathcal{J}^*\}$:
   (a) Sample $(\mathsf{pk}^{\mathcal{J}}, \mathsf{pkc}^{\mathcal{J}}, (\mathsf{sk}_{j_1}^{\mathcal{J},1}, \ldots, \mathsf{sk}_{j_t}^{\mathcal{J},1})) \leftarrow \mathsf{PRTS.KGen}(\mathsf{pp}, t, t)$.
   (b) For $e = 1, \ldots, E-1$, compute $(\mathsf{sk}_{j_1}^{\mathcal{J},e+1}, \ldots, \mathsf{sk}_{j_t}^{\mathcal{J},e+1}) \leftarrow\!\!\$\ \mathsf{PRTS.Update}(\mathsf{pk}^{\mathcal{J}}, \mathsf{sk}_{j_1}^{\mathcal{J},e}, \ldots, \mathsf{sk}_{j_t}^{\mathcal{J},e})$.
4. Set $\mathsf{pk} \leftarrow (\mathsf{pk}^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}}$ and $\mathsf{pkc} \leftarrow (\mathsf{pkc}^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}}$.
5. Pass pk and pkc to $\mathcal{A}$ and reply to its oracle queries as follows:
   – When $\mathcal{A}$ issues a secret-key query (e,i) to skO: If $i \in \mathcal{J}^*$, then forward this query to the secret-key oracle in the game $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRTS[pp]}}$. Denote the response by $\mathsf{sk}_i^{\mathcal{J}^*,e}$. Reply to $\mathcal{A}$ with $\mathsf{sk}_i^e = (\mathsf{sk}_i^{\mathcal{J},e})_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.
   – When $\mathcal{A}$ issues a signature share query (m,e,i): If $i \in \mathcal{J}^*$, then forward the query to the signature oracle in $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRTS[pp]}}$ and get back a partial signature s_i; denote it as $s_i^{\mathcal{J}^*}$. For $\mathcal{J} \in \mathcal{S}_{t,n}(i) \setminus \{\mathcal{J}^*\}$, compute $s_i^{\mathcal{J}} \leftarrow\!\!\$\ \mathsf{PRTS.Sign}(\mathsf{sk}_i^{\mathcal{J},e}, m)$. Reply to $\mathcal{A}$ with $s_i = (s_i^{\mathcal{J}})_{\mathcal{J} \in \mathcal{S}_{t,n}(i)}$.

When $\mathcal{A}$ outputs a message-signature pair (m^*, σ^*), $\mathcal{B}$ outputs the same. Observe that $\mathcal{B}$ perfectly simulates the game $\mathbf{G}^{\text{uf-}b\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}$ to $\mathcal{A}$. Moreover, by the definition of CATS, whenever $\mathbf{G}^{\text{uf-}b\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}(\mathcal{A}) = 1$, this implies that $\sigma^* = (\mathcal{J}, \sigma)$ is a valid signature on m^* with respect to PRTS, and that in each epoch e there is at least one user $j_e \in \mathcal{J}$ for which $\mathcal{A}$ did not query for $\mathsf{sk}_{j_e}^e$, nor for the signature

Games $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$

1 : $\mathsf{flag}_{\text{uf-0}}, \mathsf{flag}_{\text{uf-1}}, \mathsf{flag}_{\text{acc-0}}, \mathsf{flag}_{\text{acc-1}} \leftarrow 0$
2 : $(\mathsf{st}, n, t, E) \leftarrow \mathcal{A}(\mathsf{pp})$
3 : $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}^1_1, \ldots, \mathsf{sk}^1_n) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp}, n, t)$
4 : **for** $e = \{2, \ldots, E\}$ **do**
5 : $\quad (\mathsf{sk}^e_1, \ldots, \mathsf{sk}^e_n) \leftarrow\!\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}^{e-1}_1, \ldots, \mathsf{sk}^{e-1}_n)$
6 : **for** $i = \{1, \ldots, n\}$ **do**
7 : $\quad \mathsf{sid}_i \leftarrow 0, \mathcal{S}_{i,1} \leftarrow \emptyset, \mathcal{S}_{i,2} \leftarrow \emptyset$
8 : $(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot),\mathsf{Sign}(\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc})$
9 : **if** $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 0$ **then**
10 : $\quad$ **return** 0
11 : **if** $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e| < t \wedge |\mathcal{Q}^{\mathsf{sig}}_e(m^*)| = 0$ **then**
12 : $\quad \mathsf{flag}_{\text{uf-0}} \leftarrow 1$
13 : **if** $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)| < t$ **then**
14 : $\quad \mathsf{flag}_{\text{uf-1}} \leftarrow 1$
15 : **if** $\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \cup_{e\in[E]} \left(\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)\right)$ **then**
16 : $\quad \mathsf{flag}_{\text{acc-0}} \leftarrow 1$
17 : **if** $\forall e \in [E], \mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) \not\subseteq \mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)$ **then**
18 : $\quad \mathsf{flag}_{\text{acc-1}} \leftarrow 1$
19 : Only game $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{uf-}b}$
20 : Only game $\mathbf{G}^{\text{acc-}b}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{acc-}b}$
21 : Only game $\mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$: **return** $\mathsf{flag}_{\text{uf-}b} \vee \mathsf{flag}_{\text{acc-}b'}$

Oracle $\mathsf{Sign_1O}(e, i, \mathcal{J}, m)$:

1 : $\mathsf{sid}_i \leftarrow \mathsf{sid}_i + 1$
2 : $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \cup \{\mathsf{sid}_i\}$
3 : $(\mathsf{st}^{\mathsf{sid}_i}_{i,1}, \mathsf{msg}^{\mathsf{sid}_i}_{i,1}) \leftarrow\!\!\$\ \mathsf{Sign}_1(\mathsf{sk}^e_i, m, \mathcal{J})$
4 : **return** $\mathsf{msg}^{\mathsf{sid}_i}_{i,1}$

Oracle $\mathsf{Sign_2O}(e, i, \mathsf{sid}, (\widetilde{\mathsf{msg}_{i_1,1}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,1}}))$:

1 : **if** $\mathsf{sid} \notin \mathcal{S}_{i,1}$ **then**
2 : $\quad$ **return** $\perp$
3 : **fi**
4 : $(\mathsf{st}^{\mathsf{sid}}_{i,2}, \mathsf{msg}^{\mathsf{sid}}_{i,2}) \leftarrow\!\!\$\ \mathsf{Sign}_2(\mathsf{st}^{\mathsf{sid}}_{i,1}, \widetilde{\mathsf{msg}_{i_1,1}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,1}})$
5 : $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \setminus \{\mathsf{sid}\}$
6 : $\mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \cup \{\mathsf{sid}\}$
7 : **return** $\mathsf{msg}^{\mathsf{sid}}_{i,2}$

Oracle $\mathsf{Sign_3O}(e, i, \mathsf{sid}, (\widetilde{\mathsf{msg}_{i_1,2}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,2}}))$:

1 : **if** $\mathsf{sid} \notin \mathcal{S}_{i,2}$ **then**
2 : $\quad$ **return** $\perp$
3 : **fi**
4 : $\mathcal{Q}^{\mathsf{Sig}}_e(m) \leftarrow \mathcal{Q}^{\mathsf{Sig}}_e(m) \cup \{i\}$
5 : $s^{\mathsf{sid}}_i \leftarrow\!\!\$\ \mathsf{Sign}_3(\mathsf{st}^{\mathsf{sid}}_{i,2}, \widetilde{\mathsf{msg}_{i_1,2}}, \ldots, \widetilde{\mathsf{msg}_{i_\ell,2}})$
6 : $\mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \setminus \{\mathsf{sid}\}$
7 : **return** s^{sid}_i

Oracle $\mathsf{skO}(e, i)$:

1 : $\mathcal{Q}^{\mathsf{sk}}_e \leftarrow \mathcal{Q}^{\mathsf{sk}}_e \cup \{i\}$
2 : **return** sk^e_i

Fig. 5. The security games $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{acc-}b'}_{\mathsf{PRATS[pp]}}, \mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS[pp]}}$ for $b, b' \in \{0,1\}$ for a three-round ATS-PR scheme $\mathsf{PRATS} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Combine}, \mathsf{Vf}, \mathsf{Trace}, \mathsf{Update})$ with public parameters pp. In line 8, we write $\mathsf{Sign}(\cdot)$ as a short hand for denoting that $\mathcal{A}$ has oracle access to the three oracles $\mathsf{Sign_1O}(\cdot,\cdot,\cdot,\cdot), \mathsf{Sign_2O}(\cdot,\cdot,\cdot,\cdot)$ and $\mathsf{Sign_3O}(\cdot,\cdot,\cdot,\cdot)$. As in Figure 1, for a set $\mathcal{X}$ and an element x, we let $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$ be a shorthand for the following operation: If $\mathcal{X}$ was previously defined, then set $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$; if $\mathcal{X}$ is still undefined, then set $\mathcal{X} = \{x\}$.

share of j_e on m^*. Hence, in the corresponding epoch, $\mathcal{B}$ also did not query for $\mathsf{sk}_{j_e}^{\mathcal{J}^*,e}$ nor for a signature share of j_e on m^* with respect to this key. Therefore, if it additionally holds that $\mathcal{J} = \mathcal{J}^*$, then $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRTS[pp]}}(\mathcal{B}) = 1$ as well. Since the view of $\mathcal{A}$ is independent of the choice of $\mathcal{J}^*$, this implies that

$$\mathsf{Adv}^{\text{uf-b}}_{\mathsf{PRTS[pp]}}(\mathcal{B}) \geq \frac{1}{\mathsf{bin}_{max}} \cdot \mathsf{Adv}^{\text{uf-b}\wedge\text{acc-1}}_{\mathsf{CATS[pp]}}(\mathcal{A}),$$

concluding the proof of Theorem 1.

D.2 Proof of Theorem 2

Proof. Let $\mathcal{A}$ be an adversary playing game $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ and let (M^*, S^*) denote the message and signature $\mathcal{A}$ outputs at the end of the game. Assume without loss of generality that with probability 1, the signature S^* is a 3-tuple containing a public key for ATS, a PRTS signature, and an ATS signature. Let $S^* = (PK^*, S_0^*, S_1^*)$, and let S_{pk} denote the set of ATS public keys sampled by the challenger in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$, either by calling $\mathsf{ATS.KGen}$ as a subroutine of $\mathsf{PRATS.KGen}$ or by simulating $\Pi_{\mathsf{ATS.KGen}}$ as part of $\mathsf{PRATS.Update}$.

By total probability,

$$\begin{aligned}\mathsf{Adv}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) &= \Pr\left[\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \in S_{\mathsf{pk}}\right] \\ &\quad + \Pr\left[\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \notin S_{\mathsf{pk}}\right].\end{aligned}$$

Theorem 2 then follows from Lemma 6 and Lemma 5 below.

Lemma 5. *There exist an adversary $\mathcal{B}_1$ such that for all* pp

$$\Pr\left[\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \in S_{\mathsf{pk}}\right] \leq E \cdot \mathsf{Adv}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}(\mathcal{B}_1)$$

Proof (of Lemma 5). Consider the following adversary $\mathcal{B}_1$ playing in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$. The adversary $\mathcal{B}_1$ invokes $\mathcal{A}(\mathsf{pp})$ and simulates $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ as follows:

1. Receive (n, t, E) from $\mathcal{A}$. Forward (n, t) to the challenger in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$.
2. Receive $\mathsf{ATS.pk}^*$ from the challenger in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$.
   Sample $(\mathsf{PRTS.pk}, (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n)) \leftarrow\$ \mathsf{PRTS.KGen}(n, n)$ and send the public key $\mathsf{PRATS.pk} = (n, t, \mathsf{PRTS.pk})$ to $\mathcal{A}$.
3. Sample $e^* \leftarrow\$ [E]$ and answer the signing and secret key queries of $\mathcal{A}$ as in the following manner:
   - For $e \in [E] \setminus \{e^*\}$: Sample $(\mathsf{ATS.pk}^{(e)}, (\mathsf{ATS.sk}_1^{(e)}, \ldots, \mathsf{ATS.sk}_n^{(e)})) \leftarrow\$ \mathsf{ATS.KGen}(n, t)$ and compute

     $$\sigma_{\mathsf{pk}}^{(e)} \leftarrow\$ \mathsf{PRTS.Sign}(\mathsf{PRTS.pk}, (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n), \mathsf{ATS.pk}^{(e)}).$$

     For each secret key query of the form (e, i), send $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i^{(e)}, \mathsf{ATS.pk}_i^{(e)}, \sigma_{\mathsf{pk}}^{(e)})$ to $\mathcal{A}$. For each query to one of the signing oracles, simulate the response of the oracle using the knowledge of the secret keys $\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n$ and $\mathsf{ATS.sk}_1^{(e)}, \ldots, \mathsf{ATS.sk}_n^{(e)}$.

– For $e = e^*$: Compute

$$\sigma_{\mathsf{pk}}^{(e^*)} \leftarrow\!\!\$\ \mathsf{PRTS.Sign}(\mathsf{PRTS.pk}, (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n), \mathsf{ATS.pk}^*).$$

For each secret key query of the form (e, i), send the index i as the input to a secret key query in the game $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$ and receive a key $\mathsf{ATS.sk}_i^*$ in response. Send $(\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i^*, \mathsf{ATS.pk}^*, \sigma_{\mathsf{pk}}^{(e^*)})$ to $\mathcal{A}$ as the response to the secret key query.

For each query to a signing oracle, forward the query to the corresponding oracles in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$ (omitting the epoch index e^* from the input). If the query is to $\mathsf{Sign_1O}$ or $\mathsf{Sign_2O}$, relay the response to $\mathcal{A}$. If it is to $\mathsf{Sign_3O}$, then the oracle in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{ATS[pp]}}$ responds with a signature σ_m on a message m on behalf of a subset $\mathcal{J}$; reply to $\mathcal{A}$ with $(\mathsf{ATS.pk}^*, \sigma_{\mathsf{pk}}^{(e^*)}, \sigma_m)$.

4. Receive a message m^* and a signature $\sigma^* = (\mathsf{pk}^*, \sigma_0^*, \sigma_1^*)$. If the output of the simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ is 1 and $\mathsf{pk}^* = \mathsf{ATS.pk}^*$, then output (m^*, σ_1^*). Otherwise, output $\perp$.

Observe that whenever the output of the simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ game is 1, the forgery outputted by $\mathcal{A}$ is a valid one, and in particular σ_1^* is a valid ATS signature on m^* with respect to pk^*. This also means that either it holds that $\mathsf{PRATS.Trace}(\mathsf{PRATS.pk}, m^*, \sigma^*) = \perp$, or in epoch e^* there is an index $j^* \in \mathcal{J}^*$ for which $\mathcal{A}$ did not query for its secret key or its signature share on m^*. In the former case, this implies that $\mathsf{ATS.Trace}(\mathsf{pk}^*, m^*, \sigma_1^*) = \perp$, and in the latter, this implies that $\mathcal{B}_1$ did not issue the corresponding queries as well.

Finally, whenever $\mathbf{G}^{\mathit{uf}\text{-}1\wedge \mathit{acc}\text{-}1}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \in S_{\mathsf{pk}}$ occurs and $\mathcal{B}_1$ guesses correctly e^* as the epoch relative to which $\mathcal{A}$ outputs the forgery (note that this is well defined when $PK^* \in S_{\mathsf{pk}}$), it is also the case that $\mathbf{G}^{\mathit{uf}\text{-}1\wedge \mathit{acc}\text{-}1}_{\mathsf{ATS[pp]}}(\mathcal{B}_1)$. Conditioned on $PK^* \in S_{\mathsf{pk}}$, the probability that $\mathsf{pk}^* = \mathsf{ATS.pk}^*$ is $1/E$, since the view of $\mathcal{A}$ is independent of the epoch e^* which is chosen uniformly at random. This implies the lemma.

Lemma 6. *There exist an adversary $\mathcal{B}_2$ such that for all* pp

$$\Pr\left[\mathbf{G}^{\mathit{uf}\text{-}1\wedge \mathit{acc}\text{-}1}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \notin S_{\mathsf{pk}}\right] \leq \mathsf{Adv}^{\text{uf-b}\wedge\text{uf-1}}_{\mathsf{PRTS[pp]}}(\mathcal{B}_2)$$

Proof (of Lemma 6). Consider the following adversary $\mathcal{B}_2$ playing in $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}$. The adversary $\mathcal{B}_2$ invokes $\mathcal{A}(\mathsf{pp})$ and simulates $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ as follows:

1. Receive (n, t, E) from $\mathcal{A}$. Forward (n, n, E) to the challenger in $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}$.
2. Receive $\mathsf{PRTS.pk}$ from the challenger in $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}$ and forward $\mathsf{PRTS.pk}$ to $\mathcal{A}$ as the public key in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$.
3. Answer $\mathcal{A}$'s oracle queries as follows:
   – For $i = 1, \ldots, E$: Sample

$$(\mathsf{ATS.pk}^{(i)}, (\mathsf{ATS.sk}_1^{(i)}, \ldots, \mathsf{ATS.sk}_n^{(i)})) \leftarrow\!\!\$\ \mathsf{ATS.KGen}(n, t).$$

   and compute an n-out-of-n signature with respect to epoch i on $\mathsf{ATS.pk}^{(i)}$, via access to the signing oracle in $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}$. Denote the resulting signature by $\sigma_{\mathsf{pk}}^{(i)}$ for each $i \in [E]$.
   – In response to a secret-key query for index $\ell \in [n]$ and epoch $e \in [E]$: Forward the query to the secret-key oracle of $\mathbf{G}^{\text{uf-1}}_{\mathsf{PRTS[pp]}}$ and obtain a secret key $\mathsf{PRTS.sk}_\ell^{(e)}$. Respond to $\mathcal{A}$ with $\mathsf{PRATS.sk}_\ell^{(e)} = (\mathsf{PRTS.sk}_\ell^{(e)}, \mathsf{ATS.sk}_\ell^{(e)}, \mathsf{ATS.pk}^{e)}, \sigma_{\mathsf{pk}}^{(e)})$.

– In response to a signature queries for epoch $e \in [E]$, simulate the signing oracles of $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ using knowledge of $\mathsf{ATS.sk}_1^{(e)}, \ldots, \mathsf{ATS.sk}_n^{(e)}$ and the values $\mathsf{ATS.pk}^{(e)}$ and $\sigma_{\mathsf{pk}}^{(e)}$.

4. Receive a message m^* and a signature $\sigma^* = (\mathsf{pk}^*, \sigma_0^*, \sigma_1^*)$. If the output of the simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ is 1 and $\mathsf{pk}^* \notin \{\mathsf{ATS.pk}^{(1)}, \ldots, \mathsf{ATS.pk}^{(E)}\}$, then output $(\mathsf{pk}^*, \sigma_0^*)$. Otherwise, output $\perp$.

When the output of the simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRATS[pp]}}$ game is 1, it holds the forgery outputted by $\mathcal{A}$ is a valid one, and in particular σ_0^* is a valid PRTS signature on pk^* with respect to $\mathsf{PRTS.pk}$. This also means that in each epoch $\mathcal{A}$ issued at most $t-1$ secret key queries, and so the same is true for $\mathcal{B}_2$. Finally, conditioned on $PK^* \notin S_{\mathsf{pk}}$, $\mathcal{B}_2$ never queried its signing oracle on pk^*. Hence, the event $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{PRTS[pp]}}(\mathcal{B}_2)$ is contained in the conjunction $\mathbf{G}^{\textit{uf-1}\wedge\textit{acc-1}}_{\mathsf{PRATS[pp]}}(\mathcal{A}) = 1 \wedge PK^* \in S_{\mathsf{pk}}$, implying the lemma.

D.3 Proof of Theorem 3

Proof. Let $\mathcal{A}$ be an adversary participating in the game $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$ and q_{chal}, q_{sign} be bounds on the number of $\mathsf{H}_{\mathsf{chal}}$ queries and signature queries issued by $\mathcal{A}$ in $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$, respectively, and let $q = q_{\mathsf{chal}} + q_{\mathsf{sign}}$. Assume without loss of generality that $\mathcal{A}$ does not issue the same query to $\mathsf{H}_{\mathsf{com}}$ or to $\mathsf{H}_{\mathsf{chal}}$ more than once. Consider the following algorithm $\mathcal{B}_0$. On inputs $(\mathbb{G}, Z)$ and $(h_1, \ldots, h_q)$, the algorithm $\mathcal{B}_0$ simulates $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$ to $\mathcal{A}$ as follows:

1. Invoke $\mathcal{A}(\mathbb{G})$ and receive (n, t) from $\mathcal{A}$.
2. Sample $i^* \leftarrow\$ [n]$. For $\ell \in [n] \setminus \{i^*\}$ sample $X_\ell \leftarrow\$ \mathbb{G}$ and compute $\mathsf{sk}_\ell \leftarrow Z^{\prod_{j\in[n]\setminus\{i^*,\ell\}} e_\ell} \cdot \left(\prod_{j\in[n]\setminus\{i^*\}} X_j\right)^{\prod_{k\in[n]\setminus\{\ell\}} e_k}$.
3. Compute $Y \leftarrow Z^{\prod_{j\in[n]\setminus\{i^*\}} e_j} \cdot \left(\prod_{j\in[n]\setminus\{i^*\}} X_j\right)^{\prod_{\ell\in[n]} e_\ell}$ and pass $\mathsf{pk} = (n, t, Y)$ to $\mathcal{A}$.
4. Answer oracle queries as follows:
   – Initialize a dictionary $\mathcal{L}_{\mathsf{com}}$ of input-output pairs for $\mathsf{H}_{\mathsf{com}}$. When $\mathcal{A}$ issues a query q to $\mathsf{H}_{\mathsf{com}}$, if $\mathcal{L}_{\mathsf{com}}[q]$ is defined, then let $c = \mathcal{L}[q]$. If not, let $c \leftarrow \{0,1\}^\lambda$. If there exists a query q' in $\mathcal{L}_{\mathsf{com}}$ such that $\mathcal{L}_{\mathsf{com}}[q'] = c$, then output $\perp$ and terminate. Otherwise, record $\mathcal{L}_{\mathsf{com}}[q] = c$. Reply with c.
   – Initialize a counter $t = 0$ and a dictionary $\mathcal{L}_{\mathsf{chal}}$ of input-output pairs for $\mathsf{H}_{\mathsf{chal}}$. When $\mathcal{A}$ issues a query $q = (m, \mathsf{pk}, \mathcal{J}, R)$ to $\mathsf{H}_{\mathsf{chal}}$, if $\mathcal{L}_{\mathsf{chal}}[q]$ is defined, then let $h = \mathcal{L}_{\mathsf{chal}}[q]$. Otherwise let $t = t + 1$ and $h = h_t$ and record $\mathcal{L}_{\mathsf{chal}}[q] = h$. Reply with h.
   – In response to a secret-key query for index $\ell \in [n]$ reply as follows. If $\ell = i^*$, abort the simulation and output $\perp$. Otherwise, reply with sk_ℓ.
   – In response to signing queries: If the session identifier provided by $\mathcal{A}$ is inconsistent with previous queries, return $\perp$ to $\mathcal{A}$. Otherwise, simulate users $j \neq i^*$ honestly using the knowledge of sk_j, simulating the random oracles and recording their responses.
     For $j = i^*$, do:
     (a) *First round:* Sample $c_{i^*} \leftarrow \{0,1\}^\lambda$ and send c_{i^*} as the first message of user i^* in the protocol.
     (b) *Second round:* Let $t = t + 1$ and $h = h_t$, sample $S_{i^*} \leftarrow \mathbb{G}$, and compute $R_{i^*} = S_{i^*}^{\prod_{j\in\mathcal{J}} e_j} \cdot Y^{-h\cdot\prod_{j\in\mathcal{J}\setminus\{i^*\}} e_j}$. If $\mathcal{L}_{\mathsf{com}}[R_{i^*}]$ has already been defined, then output $\perp$ and terminate. Otherwise, record $\mathcal{L}_{\mathsf{com}}[R_{i^*}] = c_{i^*}$.

Given $\{c_j\}_{j\in\mathcal{J}\setminus\{i^*\}}$, for each j find the group element R_j such that $\mathcal{L}_{\mathsf{com}}[R_j] = c_j$. If for some j there is no such value, then skip to Step 4c. If for some j there exist two elements R_j and R'_j such that $\mathcal{L}_{\mathsf{com}}[R_j] = c_j = \mathcal{L}_{\mathsf{com}}[R'_j]$, then output $\perp$ and terminate the simulation. Let $R = \prod_{j\in\mathcal{J}} R_j$. If $\mathcal{L}_{\mathsf{chal}}[(m, \mathsf{pk}, \mathcal{J}, R)]$ is already defined then output $\perp$ and terminate. Otherwise, record $\mathcal{L}_{\mathsf{chal}}[(m, \mathsf{pk}, \mathcal{J}, R)] = h$ and send R_{i^*} as the second message in the protocol.

(c) *Third round:* Receive all openings $\{R_j\}_{j\in\mathcal{J}\setminus\{i^*\}}$ provided as the second-round messages. For all $j \in \mathcal{J} \setminus \{i^*\}$: If $\mathcal{L}_{\mathsf{com}}[R_j]$ is undefined, then sample $c \leftarrow \{0,1\}^\lambda$ and record $\mathcal{L}_{\mathsf{com}}[R_j] = c$. Verify that for all $j \in \mathcal{J} \setminus \{i^*\}$ it holds that $\mathcal{L}_{\mathsf{com}}[R_j]$ is consistent with the commitment c_j from the first round. If for some j this is not the case, reply $\perp$ to $\mathcal{A}$. Otherwise, reply with S_{i^*} sampled in Step 4b.

At the end of the simulation, the forger $\mathcal{A}$ outputs a message m^* and a signature $\sigma^* = (\mathcal{J}^*, R^*, S^*)$. If $i^* \notin \mathcal{J}^*$, if $\mathcal{A}$ never queried $\mathsf{H}_{\mathsf{chal}}$ with $q = (m^*, \mathsf{pk}, \mathcal{J}^*, R^*)$ or if the output of the simulated game $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$ is 0, then $\mathcal{B}_0$ outputs $\perp$ and terminates. Otherwise, let ℓ^* denote the index of the $\mathsf{H}_{\mathsf{chal}}$-query in which $\mathcal{A}$ queried for $\mathsf{H}_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, R^*)$. Then, $\mathcal{B}_0$ outputs $(\ell^*, (h_{\ell^*}, \sigma^*))$.

We say that $\mathcal{B}_0$ *succeeds* if its output is different than $\perp$, and we turn to bound the probability that $\mathcal{B}_0$ succeeds. Recall that $\mathcal{B}_0$ outputs $\perp$ if one of the following events occurs:

- The simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$ game comes to a conclusion and its output is 0. Denote this event by E_1.
- $\mathcal{A}$ queries the secret key of i^* or outputs a forgery with respect to a subset that does not include i^*. Denote this event by E_2.
- In a q query to $\mathsf{H}_{\mathsf{com}}$ that was not previously defined, $\mathcal{B}_0$ samples a value c which a previous query was mapped to. Denote this event by E_3. Since c is sampled uniformly at random from $\{0,1\}^\lambda$, and $\mathcal{L}_{\mathsf{com}}$ are recorded either during signature or $\mathsf{H}_{\mathsf{com}}$ queries, the probability that E_3 occurs is bounded by $q_{\mathsf{com}} \cdot (q_{\mathsf{com}} + q_{\mathsf{sign}}) \cdot 2^{-\lambda}$.
- In some signing query, $\mathcal{L}_{\mathsf{com}}[R_{i^*}]$ was already defined before R_{i^*} was computed in Step 4b. Denote this event by E_4. Observe that the distribution of R_{i^*} is uniform in $\mathbb{G}$. Hence, the probability that E_3 occurs is bounded by $q_{\mathsf{sign}} \cdot (q_{\mathsf{com}} + q_{\mathsf{sign}}) / |\mathbb{G}|$.
- In some signing query, $\mathcal{A}$ provided a commitment c_j which is a collision between (at least) two values in $\mathcal{L}_{\mathsf{com}}$. Denote this event by E_5. Since all c_j values stored in $\mathcal{L}_{\mathsf{com}}$ are chosen uniformly at random, the birthday bound implies that the probability for E_5 is at most $(q_{\mathsf{com}} + q_{\mathsf{sign}})^2 \cdot 2^{-\lambda}$.
- In some signing query, $\mathcal{L}_{\mathsf{chal}}[(m, \mathsf{pk}, \mathcal{J}, R)]$ was already defined for R computed in Step 4b. Denote this event by E_6. Since R is distributed uniformly in $\mathbb{G}$, and $\mathcal{L}_{\mathsf{chal}}$ values are recorded in either $\mathsf{H}_{\mathsf{chal}}$ queries or signature queries, it holds that the probability of E_6 is bounded by $q_{\mathsf{sign}} \cdot (q_{\mathsf{chal}} + q_{\mathsf{sign}}) / |\mathbb{G}|$.

Taking everything together, we obtain that the probability that $\mathcal{B}_0$ succeeds is bounded by

$$\begin{aligned}
\Pr[\mathcal{B}_0 \text{ succeeds}] &\geq \Pr\left[\overline{\mathsf{E}}_1 \wedge \overline{\mathsf{E}}_2 \wedge \overline{\mathsf{E}}_3 \wedge \overline{\mathsf{E}}_4 \wedge \overline{\mathsf{E}}_5 \wedge \overline{\mathsf{E}}_6\right] \\
&\geq \Pr\left[\overline{\mathsf{E}_1} \wedge \overline{\mathsf{E}_2}\right] - \Pr[\mathsf{E}_3] - \Pr[\mathsf{E}_4] - \Pr[\mathsf{E}_5] - \Pr[\mathsf{E}_6] \\
&\geq \Pr\left[\overline{\mathsf{E}_1} \wedge \overline{\mathsf{E}_2}\right] - \frac{q_{\mathsf{sign}}^2 + q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}} \cdot q_{\mathsf{chal}} + q_{\mathsf{sign}}}{|\mathbb{G}|} \\
&\quad - \frac{2q_{\mathsf{com}}^2 + 3q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}}^2}{2^\lambda}.
\end{aligned}$$

Observe that the event $\overline{\mathsf{E}_1} \wedge \overline{\mathsf{E}_2}$ occurs when the output of the simulated $\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]}$ game is 1, $\mathcal{A}$ never queries for the secret key of i^*, and the forgery outputted by $\mathcal{A}$ is with respect a subset $\mathcal{J}^*$ that contains i^*. Note that for the output of the game to be 1, there has to be at least one index $j^* \in \mathcal{J}^*$ such that $\mathcal{A}$ never queries for the secret key of j^*. Hence, and since i^* is chosen uniformly at random from $[n]$, we get that

$$\Pr\left[\overline{\mathsf{E}_1} \wedge \overline{\mathsf{E}_2}\right] \geq \frac{1}{n_{\mathsf{max}}} \cdot \Pr\left[\mathbf{G}^{\text{uf-1}\wedge\text{acc-1}}_{\mathsf{RSAATS}[\mathbb{G}]} = 1\right] = \frac{1}{n_{\mathsf{max}}} \cdot \mathsf{Adv}^{uf\text{-}1\wedge acc\text{-}1}_{\mathsf{RSSATS}[\mathbb{G}]}(\mathcal{A}).$$

Putting things together, it holds that

$$\begin{aligned} &\Pr\left[\mathcal{B}_0 \text{ succeeds}\right] \\ &\geq \frac{\mathsf{Adv}^{uf\text{-}1\wedge acc\text{-}1}_{\mathsf{RSSATS}[\mathbb{G}]}(\mathcal{A})}{n_{\mathsf{max}}} - \frac{q_{\mathsf{sign}}^2 + q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}} \cdot q_{\mathsf{chal}} + q_{\mathsf{sign}}}{|\mathbb{G}|} \\ &\quad - \frac{2q_{\mathsf{com}}^2 + 3q_{\mathsf{sign}} \cdot q_{\mathsf{com}} + q_{\mathsf{sign}}^2}{2^\lambda}. \end{aligned} \tag{1}$$

Now consider the algorithm $\mathcal{B}$ attempting to solve the strong RSA problem. On input $(\mathbb{G}, Y)$, $\mathcal{B}$ runs the forking algorithm $\mathcal{F}_{\mathcal{B}_0}$ for $\mathcal{B}_0$ guaranteed by Lemma 1. If the output of $\mathcal{F}_{\mathcal{B}_0}$ is $\perp$, then $\mathcal{B}$ outputs $\perp$ and terminate. Otherwise, $\mathcal{B}$ obtains from $\mathcal{F}_{\mathcal{B}_0}$ a pair $((h, \sigma = (\mathcal{J}, R, S)), (h', \sigma' = (\mathcal{J}', R', S')))$ such that $h \neq h'$. Moreover, by the definition of $\mathcal{B}_0$ and that the following equations hold:

$$S^{\prod_{i\in\mathcal{J}} e_i} = R \cdot Y^{h \cdot \sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j} \tag{2}$$

$$S'^{\prod_{i\in\mathcal{J}'} e_i} = R' \cdot Y^{h' \cdot \sum_{i\in\mathcal{J}'} \prod_{j\in\mathcal{J}'\setminus\{i\}} e_j} \tag{3}$$

Additionally, by the definition of $\mathcal{B}_0$, we also know that $R = R'$, $\mathcal{J} = \mathcal{J}'$, and $i^* \in \mathcal{J}$. Assume without loss of generality that $h > h'$ (otherwise, the proof is symmetric). Hence, dividing Eq. (2) by (3) and rearranging, we obtain that

$$\left(S/S'\right)^{\prod_{i\in\mathcal{J}} e_i} = Y^{(h-h')\cdot\left(\sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right)} \tag{4}$$

We argue that $\prod_{i\in\mathcal{J}} e_i$ and $(h - h') \cdot \left(\sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right)$ are coprime.

Claim 9 $\mathsf{GCD}\left(\prod_{i\in\mathcal{J}} e_i, (h-h') \cdot \left(\sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right)\right) = 1.$

We postpone the proof of the claim to the end of the section. We now explain how $\mathcal{B}$ decides on its output. First, by Claim 9 and Bezout's theorem, there exist integers a and b, efficiently computable using Euclid's extended algorithm, such that

$$a \cdot \prod_{i\in\mathcal{J}} e_i + b \cdot (h - h') \cdot \left(\sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right) = 1$$

This implies that

$$\begin{aligned} Y &= Y^{a\cdot\prod_{i\in\mathcal{J}} e_i + b\cdot(h-h')\cdot\left(\sum_{i\in\mathcal{J}} \prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right)} \\ &= \left(Y^a \cdot \left(S/S'\right)^b\right)^{\prod_{i\in\mathcal{J}} e_i}, \end{aligned} \tag{5}$$

where Eq. (5) follows from Eq. (4). Denote $W = Y^a \cdot (S/S')^b$ and $X_{-i^*} = \prod_{j\in[n]\setminus\{i^*\}} X_j$. Then, plugging in $Y = Z^{\prod_{j\in[n]\setminus\{i^*\}} e_j} \cdot \left(\prod_{j\in[n]\setminus\{i^*\}} X_j\right)^{\prod_{\ell\in[n]} e_\ell}$ and rearranging, we obtain that

$$W^{\prod_{i\in\mathcal{J}} e_i} = Z^{\prod_{j\in[n]\setminus\{i^*\}} e_j} \cdot X_{-i^*}^{\prod_{\ell\in[n]} e_\ell}.$$

Rearranging,

$$\left(\frac{W}{X_{-i^*}^{\prod_{\ell\in[n]\setminus\mathcal{J}} e_\ell}}\right)^{\prod_{\ell\in\mathcal{J}} e_\ell} = Z^{\prod_{j\in[n]\setminus\{i^*\}} e_j}.$$

Denote $T = \frac{W}{X_{-i^*}^{\prod_{\ell\in[n]\setminus\mathcal{J}} e_\ell}}$. Since, by definition, $\{e_i\}_{i\in[n]}$ are coprime to the order of $\mathbb{G}$, this implies that

$$T^{e_{i^*}} = Z^{\prod_{j\in[n]\setminus\mathcal{J}} e_j}.$$

Since $i^* \in \mathcal{J}$, it holds that $\mathsf{GCD}(e_{i^*}, \prod_{j\in[n]\setminus\mathcal{J}} e_j) = 1$. Hence, using the extended Euclid's algorithm, we can find integers a' and b' such that

$$Z = Z^{a'\cdot e_{i^*} + b'\cdot\prod_{j\in[n]\setminus\mathcal{J}} e_j} = \left(Z^{a'} \cdot T^{b'}\right)^{e_{i^*}}.$$

Hence, $\mathcal{B}$ computes $U = Z^{a'} \cdot T^{b'}$ and outputs (U, e_{i^*}) as its output to the strong RSA problem. By the above reasoning, whenever $\mathcal{B}$ outputs an output other than $\perp$ it succeeds in solving the strong RSA problem. By Lemma 1, this happens with probability at least

$$\Pr[\mathcal{B}_0 \text{ succeeds}] \cdot \left(\frac{\Pr[\mathcal{B}_0 \text{ succeeds}]}{q_{\mathsf{sign}} + q_{\mathsf{chal}}} - \frac{1}{2^\lambda}\right).$$

Using Eq. (1), we obtain Theorem 3.

We conclude by proving Claim 9.

Proof (Claim 9). Note that $0 < h - h' < 2^\lambda < \min\{e_i\}_{i\in\mathcal{J}}$ and hence

$$\mathsf{GCD}\left(\prod_{i\in\mathcal{J}} e_i, h - h'\right) = 1.$$

We are left with arguing that $\mathsf{GCD}\left(\prod_{i\in\mathcal{J}} e_i, \sum_{i\in\mathcal{J}}\prod_{j\in\mathcal{J}\setminus\{i\}} e_j\right) = 1$. Assume towards contradiction that this is not the case. Then, there exists some $\ell \in \mathcal{J}$ such that e_ℓ divides $\sum_{i\in\mathcal{J}}\prod_{j\in\mathcal{J}\setminus\{i\}} e_j$. However, we can write

$$\alpha = \frac{\sum_{i\in\mathcal{J}}\prod_{j\in\mathcal{J}\setminus\{i\}} e_j}{e_\ell} = \sum_{i\in\mathcal{J}\setminus\{\ell\}}\prod_{j\in\mathcal{J}\setminus\{i,\ell\}} e_j + \frac{\prod_{j\in\mathcal{J}\setminus\{\ell\}} e_j}{e_\ell}$$

This implies that e_ℓ divides $\prod_{j\in\mathcal{J}\setminus\{\ell\}} e_j$ in contradiction to the fact that $\{e_i\}_{i\in\mathcal{J}}$ are distinct primes.

D.4 Proof of Lemma 2

Proof. Consider the following adversary $\mathcal{B}_1$ playing the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}$. $\mathcal{B}_1$ can issue signature queries and random oracle queries to its BLS challenger. To distinguish the random oracle in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}$ from the one in the 1-epoch BLS ATS, we denote queries to the former by $\hat{\mathsf{H}}(m)$.

On getting a challenge public key pk^* from its BLS challenger, the adversary $\mathcal{B}_1$ invokes $\mathcal{A}(\mathcal{G})$ and simulates $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$ as follows:

1. Receive (n, t) from $\mathcal{A}$.
2. Guess $i^* \leftarrow\$ [n]$. Sample $\{\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n\} \leftarrow\$ \mathbb{Z}_p$. Compute $\mathsf{pk}_j \leftarrow g^{\alpha_j}$ for all $i \neq i^*$.
3. Set $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}^*$, i.e. $\mathcal{B}_1$ embeds the challenge public key in position i^*.
4. Send $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \mathsf{pkc} = \perp$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries, and $\mathcal{B}_1$ replies as follows:

- $\mathsf{H}(m)$: $\mathcal{B}_1$ queries its challenger for $h \leftarrow \hat{\mathsf{H}}(m)$, sets $\mathsf{H}(m) \leftarrow h$ and returns h to $\mathcal{A}$.
- $\mathsf{skO}(1, i)$: If $i = i^*$, $\mathcal{B}_1$ aborts. Otherwise, $\mathcal{B}_1$ returns α_i.
- $\mathsf{SignO}(m, 1, i)$: We assume without loss of generality that the adversary always queries $\mathsf{H}(m)$ before any signing query on m. If $i = i^*$, $\mathcal{B}_1$ answers by querying its signing oracle for a signature. Otherwise, $\mathcal{B}_1$ returns $\mathsf{H}(m)^{\alpha_i}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, (\mathcal{J}^*, \sigma^*))$. If $i^* \notin \mathcal{J}^*$ or if $i^* \in \mathcal{Q}^{\mathsf{sig}}_1(m^*)$, then $\mathcal{B}_1$ aborts (outputs $\perp$ and terminates). Let α^* denote the secret key corresponding to the BLS challenge pk^* (that is, $\mathsf{pk}^* = g^{\alpha^*}$). Observe that if the forgery outputted by $\mathcal{A}$ is valid and if $\mathcal{B}_1$ hasn't aborted, it means that

$$\sigma^* = (\mathsf{H}(m)^{\alpha^* \lambda_{i^*}}) \prod_{i \in \mathcal{J}^*, i \neq i^*} \mathsf{H}(m)^{\lambda_i \alpha_i}$$

where $\lambda_j = \prod_{i \in \mathcal{J}^* \setminus \{j\}} \frac{i}{i-j}$.

Then, $\mathcal{B}_1$ computes

$$\sigma' \leftarrow \left(\frac{\sigma^*}{\prod_{i \in \mathcal{J}^*, i \neq i^*} H(m)^{\lambda_i \alpha_i}} \right)^{\frac{1}{\lambda_{i^*}}}$$

and outputs the pair (m^*, σ'). Note that if $\mathcal{A}$ succesfully outputted a forgery, then (m^*, σ') is a valid message-signature pair with respect the challenge public key pk^*. Also note that if $\mathcal{A}$ did not query for a signature share on m^* with respect to user i^*, then by definition of $\mathcal{B}_1$, $i^* \notin \mathcal{Q}^{\mathsf{sig}}_1(m^*)$.

Finally, note that whenever $\mathcal{A}$ wins in the unforgeability game $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$, there must be an index $j^* \in \mathcal{J}^*$ for which $\mathcal{A}$ never queries the secret key or a signature share for the forgery message m^*. If $\mathcal{B}_1$ correctly guesses this value ($i^* = j^*$), then it also outputs a succesful forgery. Overall, since $\mathcal{B}_1$ perfectly simulates $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$ to $\mathcal{A}$ and the view of $\mathcal{A}$ is independent of i^*, it holds that

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B}_1) \geq \frac{1}{n_{max}} \cdot \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}(\mathcal{A}).$$

This proves the lemma.

D.5 Proof of Lemma 3

Proof. Consider the following adversary $\mathcal{B}_2$ playing the game $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$. $\mathcal{B}_2$ can issue signature queries and random oracle queries to its 1-epoch BLS ATS challenger. To distinguish the random oracle in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$ from the one in the general BLS ATS, we denote queries to the former by $\hat{\mathsf{H}}(m)$.

$\mathcal{B}_2$ invokes $\mathcal{A}(\mathcal{G})$ and simulates $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR}[\mathcal{G}]}$ as follows:

1. Receive (n, t, E) from $\mathcal{A}$, and forward (n, t) to the challenger in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$.
2. Receive $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n), \mathsf{pkc}$ from the challenger and forward them to $\mathcal{A}$.
3. Reply to oracle queries by $\mathcal{A}$ as follows:
   - Whenever $\mathcal{A}$ queries H on a message m: $\mathcal{B}_2$ queries $h \leftarrow \hat{\mathsf{H}}(m)$, sets $\mathsf{H}(m) \leftarrow h$ and returns h.
   - Whenever $\mathcal{A}$ queries its secret-key oracle on (k, i) (that is, the key of user i in epoch k): If $k = 1$, then $\mathcal{B}_2$ forwards this query to its own secret-key oracle, sets sk_i^1 as the response. If $k > 1$, then $\mathcal{B}_2$ samples a random secret key $\mathsf{sk}_i^k \leftarrow\$\ \mathbb{Z}_p$. For each epoch k, $\mathcal{B}_2$ maintains a set R_k, encoding all the signers for which $\mathcal{B}_2$ determined the secret key in epoch k. $\mathcal{B}_2$ sets $R_k \leftarrow R_k \cup \{i\}$ and returns sk_i^k to $\mathcal{A}$.
   - Signing queries: Assume without loss of generality that $\mathcal{A}$ always queries the random oracle for $\mathsf{H}(m)$ before issuing a signing query of the form (m, k, i) (a signature share on m relative to i's secret key in the kth epoch). Whenever $\mathcal{A}$ issues a signing query (m, k, i), if $k = 1$ then $\mathcal{B}_2$ forwards the query (m, i) to its signing oracle, and replies with the response. Otherwise, if $k > 1$, $\mathcal{B}_2$ decides on its response as follows:
     - If $i \in R_k$, then $\mathcal{B}_2$ replies with $\mathsf{H}(m)^{\mathsf{sk}_i^k}$.
     - If $i \notin R_k$ and $|R_k| < t - 1$, then $\mathcal{B}_2$ samples $\mathsf{sk}_i^k \leftarrow\$\ \mathbb{Z}_p$, updates $R_k \leftarrow R_k \cup \{i\}$, and replies with $\mathsf{H}(m)^{\mathsf{sk}_i^k}$.
     - If $i \notin R_k$ and $|R_k| = t - 1$, then we use the fact that $\forall j \in R_k, \mathsf{sk}_j^k = \mathsf{sk}_j^1 + \delta_j^k$ for some δ_j^k, where $\{\delta_j^k\}$ are Shamir secret shares of 0. In other words, in a random execution of the protocol, there should be a polynomial f of degree $t-1$ such that $\delta_j^k = f(j) \forall j \in R_k$ and $f(0) = 0$. Hence, by Lagrange interpolation, $\delta_i^k = \sum_{j \in R_k} \lambda_j(i) \cdot (\mathsf{sk}_j^k - \mathsf{sk}_j^1)$, where $\lambda_j(i) = \prod_{\ell \in R_k \setminus \{j\}} \frac{\ell - i}{\ell - j}$.

       To compute the response, $\mathcal{B}_2$ computes $a \leftarrow \mathsf{H}(m)^{\sum_{j \in R_k} \lambda_j(i) \cdot \mathsf{sk}_j^k}$. For each $j \in (R_k \cup \{i\}) \cap R_1$ it then computes a term b_j as $b_j \leftarrow \mathsf{H}(m)^{\mathsf{sk}_j^1}$. For each $j \in (R_k \cup \{i\}) \setminus R_1$, it requests a signature share on m with respect to signer j from its own signing oracle, and sets b_j to be the response. Finally, $\mathcal{B}_2$ replies to $\mathcal{A}$ with the partial signature $a \cdot b_i \cdot \prod_{j \in R_k} b_j^{-\lambda_j(i)}$. The above discussion shows that this preserves the perfect simulation of $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR}[\mathcal{G}]}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, (\mathcal{J}^*, \sigma^*))$. $\mathcal{B}_2$ outputs this forgery as well. We claim that if the output of the simulated $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR}[\mathcal{G}]}$ is 1 then so is the output of $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$. This is because in this case it holds that

- $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 1$.
- For each $k \in [E]$ it holds that $|\mathcal{Q}_k^{\mathsf{sk}}| < t$. Hence, $|\mathcal{Q}_1^{\mathsf{sk}}| < t$. Also, $\mathcal{B}_2$ makes no secret key queries for epochs $k > 1$.

– For all $k \in [E]$ it holds that $\mathcal{Q}_k^{\mathsf{sig}}(m^*) = \emptyset$. In particular, this means that $\mathcal{B}_2$ forwards no signing queries for m^* to its signing oracle.

Hence, all the requirements are satisfied for the output $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}$ to be 1. Since

$$\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{BLSPR}[\mathcal{G}]}(\mathcal{A}) \leq \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{BLSPR\text{-}1}[\mathcal{G}]}(\mathcal{B}_2).$$

This completes the proof of the lemma.

D.6 Proof of Lemma 4

Proof. Consider the following adversary $\mathcal{B}$ playing the game $\mathbf{G}^{\text{uf}}_{\mathsf{BLS}[\mathcal{G}]}$. We use α to denote the challenge secret key in the BLS game $\mathbf{G}^{\text{uf}}_{\mathsf{BLS}[\mathcal{G}]}$. On getting a challenge public key $\mathsf{pk}^* \leftarrow g^\alpha$ from its BLS challenger, $\mathcal{B}$ invokes $\mathcal{A}(\mathcal{G})$ and simulates $\mathbf{G}^{\text{sa-acc-0}}_{\mathsf{BLSPR}[\mathcal{G}]}$ as follows:

– Receive (n, t, E) from $\mathcal{A}$.
– Guess $i^* \leftarrow\!\!\$\ [n]$. Sample $\{\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n\} \leftarrow\!\!\$\ \mathbb{Z}_p$. Compute $\mathsf{pk}_j \leftarrow g^{\alpha_j}$ for all $i \neq i^*$.
– Set $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}^*$ (i.e. $\mathcal{B}$ embeds the challenge public key at position i^*).
– For all $k \in \{2, \ldots, E\}$, sample $a_{1,k}, \ldots, a_{t-1,k} \leftarrow\!\!\$\ \mathbb{F}_p$. Then, for all $i \in [n]$ compute $\delta_i^k \leftarrow \sum_{l=1}^{t-1} a_{l,k} \cdot i^l$. (i.e. we generate new Shamir secret shares of 0 for every epoch).
– Send $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \mathsf{pkc} = \perp$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries. We assume without loss of generality that $\mathcal{A}$ always queries $\mathsf{H}(m)$ before issuing any signing query on m. We now specify how $\mathcal{B}$ answers each of those queries:

– $\mathsf{H}(m)$: $\mathcal{B}$ queries the random oracle of its BLS challenger and forwards the answer to $\mathcal{A}$. $\mathcal{B}$ also maintains a mapping of messages to their corresponding hashes.
– $\mathsf{skO}(k, i)$: If $i = i^*$, $\mathcal{B}$ aborts. Otherwise, if $k = 1$, $\mathcal{B}$ returns α_i, and if $k > 1$, $\mathcal{B}$ returns $\alpha_i + \sum_{j=2}^{k} \delta_i^j$.
– $\mathsf{SignO}(m, k, i)$: If $i \neq i^*$, $\mathcal{B}$ returns $\mathsf{H}(m)^{\alpha_i^k}$, where $\alpha_i^k = \alpha_i$ if $k = 1$ and $\alpha_i^k = \alpha_i + \sum_{j=2}^{k} \delta_i^j$ if $k > 1$. If $i = i^*$, $\mathcal{B}$ queries its BLS challenger for $\sigma_{m,i^*} \leftarrow\!\!\$\ \mathsf{Sign}(m)$. Then, if $k = 1$, $\mathcal{B}$ returns σ_{m,i^*}, and if $k > 1$, $\mathcal{B}$ returns $\sigma_{m,i^*} \cdot \mathsf{H}(m)^{\sum_{j=2}^{k} \delta_{i^*}^j}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, (\mathcal{J}^*, \sigma^*))$. $\mathcal{B}$ aborts if $i^* \notin \mathcal{J}^*$ or if there exists some $k \in [E]$ such that $i^* \in \mathcal{Q}_k^{\mathsf{sig}}(m^*)$. Observe that by a similar analysis to that found in the proof of Lemma 2, if $\mathcal{B}$ does not abort, it wins the unforgeability game of BLS. Moreover, by the same analysis, it holds that

$$\mathsf{Adv}^{\text{uf}}_{\mathsf{BLS}[\mathcal{G}]}(\mathcal{B}) \geq \frac{1}{n_{max}} \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{BLSPR}[\mathcal{G}]}(\mathcal{A}).$$

This concludes the lemma.

D.7 Proof of Theorem 5

The proof of Theorem 5 follows the same roadmap as the proof for our BLS-based scheme, proving unforgeability and accountability separately. Let us use $\mathsf{Schnorr3\text{-}PR\text{-}1}$ to denote a 1-epoch variant of our Schnorr-based 3-round ATS-PR scheme (obtained by fixing the number E of epochs to 1). Similar to the BLS based scheme, unforgeability is proven in two steps. First, we reduce the unforgeability of $\mathsf{Schnorr3\text{-}PR\text{-}1}$ to that of Schnorr signatures in Lemma 7. Then, in Lemma 8, we reduce the unforgeability of our multi-epoch scheme to the 1-epoch variant.

Lemma 7. *For every adversary $\mathcal{A}$ for the game $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1[pp]}}$, there exists an adversary $\mathcal{B}_1$, such that for all groups $\mathbb{G}$,*

$$\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{A}) \le n_{max} \cdot (q_H + q_S) \cdot \left(\mathsf{Adv}^{\text{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}_1) + \frac{q_S \cdot (q_H + q_S)}{p} + \frac{2q_C^2}{2^\lambda} \right).$$

Lemma 8. *For every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}_2$ such that,*

$$\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \le (q_S + q_H) \cdot \left(\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{B}_2) + \frac{q_S(q_S + q_H)}{p} + \frac{2q_C^2}{2^\lambda} \right).$$

Lemma 9. *For every adversary $\mathcal{A}$ there exists an adversary $\mathcal{B}$, such that,*

$$\mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \le n_{max} \cdot (q_S + q_H) \cdot \left(\mathsf{Adv}^{\text{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}) + \frac{q_S(q_S + q_H)}{p} + \frac{2q_C^2}{2^\lambda} \right).$$

Proof of Lemma 7

Proof. We assume without loss of generality that $\mathcal{A}$ does not issue the same query to $\mathsf{H_{com}}$ or to $\mathsf{H_{chal}}$ more than once.

Consider the following adversary $\mathcal{B}_1$ playing the game $\mathbf{G}^{\text{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}$. $\mathcal{B}_1$ can issue signature queries and random oracle queries. To distinguish the random oracle in $\mathbf{G}^{\text{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}$ from the one in the 1-epoch Schnorr ATS, we denote queries to the former by $\hat{\mathsf{H}}_{\mathsf{sig}}(m, \mathsf{pk}, R)$.

On getting a challenge public key pk^* from its challenger, $\mathcal{B}_1$ invokes $\mathcal{A}(\mathbb{G})$ and simulates $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$ as follows:

- Receive (n, t) from $\mathcal{A}$.
- Guess $i^* \leftarrow\!\!\$\ [n]$. Then, sample $\{\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n\} \leftarrow\!\!\$\ \mathbb{Z}_p$. Compute $\mathsf{pk}_j \leftarrow g^{\alpha_j}\ \forall j \neq i^*$.
- Set $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}^*$. Send $(\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \mathsf{pkc} = \perp)$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries. We use q_S, q_H, q_C to denote a bound on the number of signing queries, random oracle queries on $\mathsf{H_{chal}}$ and random oracle queries on $\mathsf{H_{com}}$, issued by $\mathcal{A}$ across all signing sessions, respectively.

$\mathcal{B}_1$ also samples $j^* \leftarrow\!\!\$\ [q_H + q_S]$, guessing which of the $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ queries will correspond to the forgery that $\mathcal{A}$ will return. Note that j^* is sampled from $[q_H + q_S]$ (rather than from $[q_H]$), because we assume w.l.o.g. that $\mathcal{A}$ always queries $\mathsf{H_{chal}}$ before sending a corresponding $\mathsf{Sign_3O}$ query, meaning that the total number of $\mathsf{H_{chal}}$ queries is upper bounded by $q_H + q_S$.

$\mathcal{B}_1$ initializes (the simulated oracles) $\mathsf{H_{com}}(R) \leftarrow \perp, \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow \perp$ for all values of m, R and all possible subsets of signers $\mathcal{J}$. $\mathcal{B}_1$ also maintains (i) a counter t to track the number of $\mathsf{H_{chal}}$ queries, initialized with 0, and (ii) a dictionary Q_{chal} to track auxiliary information used to answer $\mathsf{H_{chal}}$ queries. It is initialized as empty, i.e. $Q_{\mathsf{chal}} \leftarrow \emptyset$.

For signing queries, for each signer $j \in [n]$, $\mathcal{B}_1$ maintains four mappings: (i). $R_j : [q_S] \rightarrow \{0,1\}^\lambda$ to record nonces generated to respond to $\mathsf{Sign_1O}(1, j, \mathcal{J}, m)$ queries, (ii) S_j to store auxiliary information used for answering signing queries, and (iii) $M_{j,1}$ to record all the messages sent by $\mathcal{A}$ in a $\mathsf{Sign_2O}(1, j, \mathsf{sid}, \{\mathsf{msg}_{i,1}\})$ query, (iv) $\mathcal{S}'_j$ to store the $(m, \mathcal{J}, e)$ values for every signing session.

All these mappings are initialized as empty at the beginning of simulation. These are in addition to the variables $\mathsf{sid}_i, \mathcal{S}_{i,1}, \mathcal{S}_{i,2}$ defined and initialized as per the security game in Figure 5.

We now discuss how $\mathcal{B}_1$ responds to each of $\mathcal{A}$'s queries.

$\mathsf{H_{com}}(R)$. If $\mathsf{H_{com}}(R)$ has been determined, i.e. if $\mathsf{H_{com}}(R) \neq \perp$, then $\mathcal{B}_1$ returns $\mathsf{H_{com}}(R)$. Otherwise, $\mathcal{B}_1$ samples a random string $c \leftarrow\!\!\$\ \{0,1\}^\lambda$ and returns c. $\mathcal{B}_1$ sets $\mathsf{H_{com}}(R) \leftarrow c$.

$\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.

- If $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ has been determined, i.e. $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \neq \perp$, then $\mathcal{B}_1$ returns the value $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.
- Otherwise, $\mathcal{B}_1$ sets $t \leftarrow t+1$. And, if $i^* \notin \mathcal{J}$, then (i) if this is the j^*th query (i.e. $t = j^*$), $\mathcal{B}_1$ aborts, (ii) else, $\mathcal{B}_1$ samples a random element $h \leftarrow\!\!\$\ \mathbb{Z}_p$. $\mathcal{B}_1$ sets $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$ and returns h to $\mathcal{A}$.
- Or, if $i^* \in \mathcal{J}$ and $t = j^*$ then $\mathcal{B}_1$ samples a random message $m' \leftarrow\!\!\$\ \mathcal{M}$, and queries its Schnorr challenger for $h \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(m', \mathsf{pk}^*, R^{\frac{1}{\lambda_{i^*}^{\mathcal{J}}}})$, where $\lambda_{i^*}^{\mathcal{J}} = \prod_{j \in \mathcal{J} \setminus \{i^*\}} \frac{j}{j - i^*}$. $\mathcal{B}_1$ then sets $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$ and returns h to $\mathcal{A}$. $\mathcal{B}_1$ also records the value $Q_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow m'$.
- Otherwise, if $i^* \in \mathcal{J}$ and $t \neq j^*$, then, $\mathcal{B}_1$ iterates over all session ids in $\mathcal{S}_{i^*,2}$. For each $\mathsf{sid} \in \mathcal{S}_{i^*,2}$, $\mathcal{B}_1$ first checks if $(m, \mathcal{J})$ match the values in $\mathcal{S}'_{i^*}(\mathsf{sid})$. Next, for all $j \in \mathcal{J} \setminus \{i^*\}$, $\mathcal{B}_1$ finds a value R_j s.t. $\mathsf{H_{com}}(R_j) = \mathsf{msg}_{j,1}$, with $\mathsf{msg}_{j,1}$ stored in $M_{i^*,1}(\mathsf{sid})$. If there's more than one such R_j for some j, $\mathcal{B}_1$ aborts. If there is exactly one such value for all j then, $\mathcal{B}_1$ calculates $R(\mathsf{sid}) \leftarrow R_{i^*}(\mathsf{sid}) \cdot \prod_{j \in \mathcal{J} \setminus \{i^*\}} R_j$. If $R = R(\mathsf{sid})$, then, $\mathcal{B}_1$ uses h stored in $S_{i^*}(\mathsf{sid})$, sets $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$, and returns this h.
- If there is no such sid that meets all the conditions above, then $\mathcal{B}_1$ samples $h \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$ and returns h.

$\mathsf{skO}(1, i)$. If $i = i^*$ then $\mathcal{B}_1$ aborts. Otherwise, $\mathcal{B}_1$ returns α_i.

$\mathsf{Sign_1O}(1, i, \mathcal{J}, m)$. $\mathcal{B}_1$ sets $\mathsf{sid}_i \leftarrow \mathsf{sid}_i + 1$, $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \cup \{\mathsf{sid}_i\}$, $\mathcal{S}'_i(\mathsf{sid}_i) \leftarrow (m, \mathcal{J}, 1)$.

- If $i \neq i^*$, then $\mathcal{B}_1$ samples a random $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and sets $R_i(\mathsf{sid}_i) \leftarrow g^{r_i}$. If $\mathsf{H_{com}}(R_i(\mathsf{sid}_i)) \neq \perp$, then $\mathcal{B}_1$ returns $\mathsf{msg}_{i,1} \leftarrow \mathsf{H_{com}}(R_i(\mathsf{sid}_i))$. Otherwise, $\mathcal{B}_1$ samples a random string $c \leftarrow\!\!\$\ \{0,1\}^\lambda$, sets $\mathsf{H_{com}}(R_i(\mathsf{sid}_i)) \leftarrow c$. $\mathcal{B}_1$ returns c and sets $S_i(\mathsf{sid}_i) \leftarrow (r_i, c)$
- If $i = i^*$, then $\mathcal{B}_1$ samples a random string c_{i^*} and returns this. $\mathcal{B}_1$ also stores $S_{i^*}(\mathsf{sid}_{i^*}) \leftarrow c_{i^*}$.

$\mathsf{Sign_2O}(1, i, \mathsf{sid}, \{\mathsf{msg}_{j,1}\}_{j \in \mathcal{J} \setminus \{i\}})$. Assuming that the session is valid, i.e. $\mathsf{sid} \in \mathcal{S}_{i,1}$, and denoting $(m, \mathcal{J}, 1) \leftarrow \mathcal{S}'_i(\mathsf{sid})$, for each $j \in \mathcal{J}$, $\mathcal{B}_1$ finds the element R_j such that $\mathsf{H_{com}}(R_j) = \mathsf{msg}_{j,1}$. If for some j, there is more than one such, then $\mathcal{B}_1$ aborts. If for some $j \neq i^*$ this value is undefined, $\mathcal{B}_1$ sets a flag $\mathsf{flag} \leftarrow 1$, returns $R_i(\mathsf{sid})$, and stores all the messages received, $M_{i,1}(\mathsf{sid}) \leftarrow \{\mathsf{msg}_{j,1}\}_{j \in \mathcal{J} \setminus \{i\}}$. Also, for $j = i^*$, if there is no $\mathsf{sid}_{i^*,i}$ s.t. $S_{i^*}(\mathsf{sid}_{i^*,i}) = \mathsf{msg}_{i^*,1}$, then $\mathcal{B}_1$ sets $\mathsf{flag} \leftarrow 1$.

Otherwise, if $i \neq i^*$, $\mathcal{B}_1$ simply returns $R_i(\mathsf{sid})$. But if $i = i^*$, $\mathcal{B}_1$ first samples $s_{i^*} \leftarrow\!\!\$\ \mathbb{Z}_p$, and a random $h \leftarrow\!\!\$\ \mathbb{Z}_p$. Then, $\mathcal{B}_1$ sets $R_{i^*}(\mathsf{sid}) \leftarrow \frac{g^{s_{i^*}}}{(\mathsf{pk}^*)^{h \cdot \lambda_{i^*}^{\mathcal{J}}}}$, and sets $\mathsf{H_{com}}(R_{i^*}(\mathsf{sid})) \leftarrow c_{i^*}$, where c_{i^*} is stored in $S_{i^*}(\mathsf{sid})$. $\mathcal{B}_1$ additionally sets $S_{i^*}(\mathsf{sid}) \leftarrow (S_{i^*}(\mathsf{sid}), s_{i^*}, h)$.

Let $(m, \mathcal{J}, 1) \leftarrow \mathcal{S}'_i(\mathsf{sid})$.

Then, $\mathcal{B}_1$ computes $R = R_{i^*}(\mathsf{sid}) \prod_{j \in \mathcal{J} \setminus \{i^*\}} R_j$, wherein $\mathsf{H_{com}}(R_j) = \mathsf{msg}_{j,1} \forall j \in \mathcal{J} \setminus \{i^*\}$. Then, If $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ was ever queried by $\mathcal{A}$ before or was programmed by $\mathcal{B}_1$ in a sign query in another session, then $\mathcal{B}_1$ aborts.

In all cases, $\mathcal{B}_1$ stores all the messages received, $M_{i,1}(\mathsf{sid}) \leftarrow \{\mathsf{msg}_{j,1}\}_{j \in \mathcal{J} \setminus \{i\}}$. Similar to the game, $\mathcal{B}_1$ sets $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \setminus \{\mathsf{sid}\}, \mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \cup \{\mathsf{sid}\}$.

$\mathsf{Sign_3O}(1, i, \mathsf{sid}, \{\mathsf{msg}_{j,2}\}_{j\in\mathcal{J}\setminus\{i\}})$. Assuming that the session is valid, i.e. $\mathsf{sid} \in \mathcal{S}_{i,2}$, and denoting $(m, \mathcal{J}, 1) \leftarrow \mathcal{S}'_i(\mathsf{sid})$, $\mathcal{B}_1$ first verifies all commitments that were sent by $\mathcal{A}$ in signing rounds 1 and 2. If, for some $j \in \mathcal{J} \setminus \{i\}, \mathsf{H_{com}}(\mathsf{msg}_{j,2}) = \bot$, then $\mathcal{B}$ samples a random number $c \leftarrow\!\!\$ \{0,1\}^\lambda$ and sets $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \leftarrow c$. Next, If for some $j \in \mathcal{J} \setminus \{i\}$, $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \neq \mathsf{msg}_{j,1}$ (using the $\mathsf{msg}_{j,1}$ stored in $M_{i,1}(\mathsf{sid})$), then $\mathcal{B}_1$ returns $\bot$. $\mathcal{B}_1$ also checks that for $\mathsf{msg}_{i^*,1}$, if there is any other value $R \neq \mathsf{msg}_{i^*,2}$, s.t. $\mathsf{H_{com}}(R) = \mathsf{msg}_{i^*,1}$, then $\mathcal{B}_1$ aborts.

If $\mathsf{flag} = 1$, $\mathcal{B}_1$ aborts. If not aborted or returned $\bot$, $\mathcal{B}_1$ computes $R = R_i(\mathsf{sid}) \prod_{j\in\mathcal{J}\setminus\{i\}} \mathsf{msg}_{j,2}$.

If the j^*th $\mathsf{H_{chal}}$ query has already been made, then $\mathcal{B}_1$ checks if the j^*th $\mathsf{H_{chal}}$ query had inputs $(m, \mathsf{pk}, \mathcal{J}, R)$, if so, then $\mathcal{B}_1$ aborts since for $\mathcal{A}$ to return a valid uf-0 forgery on $(m, \mathsf{pk}, \mathcal{J}, R)$, $\mathcal{A}$ is not allowed to query partial signature of any signer on these inputs.

Note that by our assumption that $\mathcal{A}$ always queries $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ before calling $\mathsf{Sign_3O}$, $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \neq \bot$.

$\mathcal{B}_1$ then returns partial signatures as follows:

- If $i \neq i^*$, $\mathcal{B}_1$ returns $s_i = r_i + \lambda_i^{\mathcal{J}} \cdot \alpha_i \cdot \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ (using $r_i \leftarrow S_i(\mathsf{sid})$).
- Otherwise, if $i = i^*$, $\mathcal{B}_1$ returns s_{i^*} from $S_{i^*}(\mathsf{sid})$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, \mathcal{J}^*, z^*, R^*)$. If $i^* \notin \mathcal{J}^*$ or if $m^*, \mathsf{pk}, \mathcal{J}^*, R^*$ are not the j^*th $\mathsf{H_{chal}}$ query, then $\mathcal{B}_1$ aborts. We assume w.l.o.g that $\mathsf{H_{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, \mathcal{R}^*) \neq \bot$, and denote it as $h^* \leftarrow \mathsf{H_{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, \mathcal{R}^*)$. $\mathcal{B}_1$ gets $m'^* \leftarrow Q_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, \mathcal{R}^*)$, which was used to program $\mathsf{H_{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, \mathcal{R}^*)$.

If not aborted, $\mathcal{B}_1$ outputs

$$\left(m'^*, R'^* = (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}, z'^* = \frac{1}{\lambda_{i^*}^{\mathcal{J}^*}} \cdot \left(z^* - h^* \cdot (\sum_{j\in\mathcal{J}^*\setminus\{i^*\}} \lambda_j^{\mathcal{J}^*} \cdot \alpha_j)\right)\right).$$

We claim that this is a valid Schnorr forgery. This is because, (i). $\mathcal{B}_1$ never sent a Sign query to the Schnorr challenger, and (ii). $\mathsf{Vf}(\mathsf{pk}, m^*, (\mathcal{J}^*, R^*, z^*)) = 1$ which means that,

$$g^{z^*} = R^* \cdot \left(\prod_{j\in\mathcal{J}} X_j^{h^*\cdot\lambda_j}\right)$$

This means that,

$$\begin{aligned}
g^{z'^*} &= \left(g^{z^*-h^*\cdot(\sum_{j\in\mathcal{J}^*\setminus\{i^*\}}\lambda_j^{\mathcal{J}^*}\cdot\alpha_j)}\right)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\\
&= \left(\frac{g^{z^*}}{g^{h^*(\sum_{j\in\mathcal{J}^*\setminus\{i^*\}}\lambda_j^{\mathcal{J}^*}\cdot\alpha_j)}}\right)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\\
&= \left(\frac{R^*\cdot\prod_{j\in\mathcal{J}^*}X_j^{h^*\lambda_j}}{\prod_{j\in\mathcal{J}^*\setminus\{i^*\}}(X_j)^{h^*\cdot\lambda_j}}\right)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\\
&= \left(R^*X_{i^*}^{h^*\lambda_{i^*}}\right)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\\
&= (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\cdot X_{i^*}^{h^*}\\
&= R'^*\cdot X_{i^*}^{h^*}
\end{aligned}$$

Lastly, $\mathcal{B}_1$ programmed $h^* = \mathsf{H}_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, R^*)$ to be equal to the value of $\hat{\mathsf{H}}_{\mathsf{sig}}$ on input $\left(m'^*, \mathsf{pk}^*, (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\right)$. This follows from how $\mathcal{B}_1$ responds to all $\mathsf{H}_{\mathsf{chal}}$ queries with $i^* \in \mathcal{J}$, and how $\mathcal{B}_1$ programs the j^*th $\mathsf{H}_{\mathsf{chal}}$ query from within the signing queries. This gives us:

$$g^{z'^*} = R'^* \cdot X_{i^*}^{\hat{\mathsf{H}}_{\mathsf{chal}}\left(m'^*, pk^*, (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}\right)}$$

Hence, $\left(R'^* = \frac{R^*}{\lambda_{i^*}^{\mathcal{J}^*}}, z'^* = \frac{1}{\lambda_{i^*}^{\mathcal{J}^*}} \cdot (z^* - h^* \cdot (\sum_{j\in\mathcal{J}^*\setminus\{i^*\}} \lambda_j^{\mathcal{J}^*} \cdot \alpha_j))\right)$ is a valid Schnorr signature for message m'^*.

$\mathcal{B}_1$ is able to produce a valid forgery whenever it does not abort. Let Abort denote the event in which $\mathcal{B}_1$ prematurely aborts the simulation. This means that $\Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = 1 \mid \overline{\mathsf{Abort}}\right] = \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{A})$. Hence, we get:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}_1) &= \Pr\left[\overline{\mathsf{Abort}} \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = 1)\right] + \Pr\left[\mathsf{Abort} \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = 1)\right]\\
&= \Pr\left[\overline{\mathsf{Abort}} \wedge (\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = 1)\right]\\
&= \Pr\left[\overline{\mathsf{Abort}}\right] \cdot \Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = \mathbf{1} \mid \overline{\mathsf{Abort}}\right]
\end{aligned}$$

Let us use E_{1a} to refer to the event that $\mathcal{B}_1$ guessed i^* correctly (i.e. $i^* \in \mathcal{J}^*$ and $i^* \notin Q_1^{SK}$), let E_{1b} represent the event that $\mathcal{B}_1$ correctly guessed j^* Let E_{2a} be the event that, for all q_S signing queries, $\mathcal{A}$ had not queried $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ before calling $\mathsf{Sign_2O}(m, 1, i, \mathcal{J})$ for some i, and E_{2b} be the event that, for all q_S signing queries, $\mathcal{B}_1$ had not already programmed $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ before $\mathcal{A}$ calls $\mathsf{Sign2O}(\cdot)$. Let E_3 denote the event in which $\mathcal{B}_1$ does not abort due to a collision in $\mathsf{H}_{\mathsf{com}}$ and E_4 the event in which $\mathcal{B}_1$ does not abort due to flag being equal to 1.

Next, observe that $\Pr[\mathsf{E}_{1a}] \geq \frac{1}{n}$, $\Pr[\mathsf{E}_{1b}] \geq \frac{1}{q_H+q_S}$, and the probability that $\mathcal{B}_1$ guesses both i^*, j^* correctly is atleast $\frac{1}{n(q_S+q_H)}$. Also, $\Pr\left[\overline{\mathsf{E}}_{2b}\right] \leq \frac{q_S^2}{p}$, since for any signing query, a collision can occur with probability max q_S/p. Next, $\Pr[\mathsf{E}_{2a}] \geq (1-\frac{q_S}{p})^{q_H}$. This is true since, conditioned on $\mathcal{A}$ not observing an $\mathsf{H}_{\mathsf{com}}$ query whose output was $\hat{\mathsf{H}}_{\mathsf{com}}(R_i(\mathsf{sid}))$, its view is independent of the value R in the input $(m, \mathsf{pk}, \mathcal{J}, R)$ to $\mathsf{H}_{\mathsf{chal}}$. We simplify the expression to get $\Pr[\mathsf{E}_{2a}] \geq 1 - \frac{q_S q_H}{p}$, meaning $\Pr\left[\overline{\mathsf{E}}_{2a}\right] \leq \frac{q_S q_H}{p}$. Similarly $\Pr[\overline{\mathsf{E}_3}] \leq q_C^2/2^\lambda$ and $\Pr[\overline{\mathsf{E}_4}] \leq q_C/2^\lambda$, since the responses of $\mathcal{B}_1$ to $\mathsf{H}_{\mathsf{com}}$ queries are uniformly random in $\{0,1\}^\lambda$. Then, we have,

$$\begin{aligned}
\Pr\left[\overline{\mathsf{Abort}}\right] &= \Pr[\mathsf{E}_{1a} \wedge \mathsf{E}_{1b} \wedge \mathsf{E}_{2a} \wedge \mathsf{E}_{2b} \wedge \mathsf{E}_3 \wedge \mathsf{E}_4] \\
&\geq \Pr[\mathsf{E}_{1a} \wedge \mathsf{E}_{1b}] - \Pr\left[\overline{\mathsf{E}}_{2a}\right] - \Pr\left[\overline{\mathsf{E}}_{2b}\right] - \Pr[\overline{\mathsf{E}_3}] - \Pr[\overline{\mathsf{E}_4}] \\
&\geq \frac{1}{n(q_H+q_S)} - \frac{q_S q_H}{p} - \frac{q_S^2}{p} - \frac{q_C^2}{2^\lambda} - \frac{q_C}{2^\lambda} \\
&\geq \frac{1}{n(q_H+q_S)} - \frac{q_S(q_S+q_H)}{p} - \frac{2q_C^2}{2^\lambda}.
\end{aligned}$$

Since conditioned on $\overline{\mathsf{Abort}}$, $\mathcal{B}_1$ perfectly simulates the game to $\mathcal{A}$, we get,

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}_1) &\geq \left(\frac{1}{n(q_H+q_S)} - \frac{q_S(q_H+q_S)}{p} - \frac{2q_C^2}{2^\lambda}\right) \cdot \mathsf{Adv}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{A}) \\
&\geq \frac{\mathsf{Adv}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{A})}{n(q_H+q_S)} - \frac{q_S \cdot (q_H+q_S)}{p} - \frac{2q_C^2}{2^\lambda}.
\end{aligned}$$

This completes the proof.

Proof of Lemma 8

Proof. Consider the following adversary $\mathcal{B}_2$ playing the game $\mathbf{G}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$. $\mathcal{B}_2$ can issue signature queries and random oracle queries to its challenger. To distinguish the random oracles in $\mathbf{G}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$ from the ones in the Schnorr3-PR game, we denote queries to the former by $\hat{\mathsf{H}}_{\mathsf{com}}(R)$ and $\hat{\mathsf{H}}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$. $\mathcal{B}_2$ invokes $\mathcal{A}(\mathbb{G})$ and simulates $\mathbf{G}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$ as follows:

1. Receive (n, t, E) from $\mathcal{A}$, and forward (n, t) to the challenger in the game $\mathbf{G}^{\mathrm{sa\text{-}uf\text{-}0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$
2. Receive $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, pkc from the challenger and forward them to $\mathcal{A}$.
3. Initialize (the simulated oracles) $\mathsf{H}_{\mathsf{com}}(R) \leftarrow \perp, \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow \perp$ for all values of m, R and all possible subsets of signers $\mathcal{J}$. $\mathcal{B}_2$ also maintains a counter t to track the number of $\mathsf{H}_{\mathsf{chal}}$ queries, initialized with 0.
4. For signing queries, for each signer $j \in [n]$, $\mathcal{B}_2$ maintains the following mappings: (i) $\mathcal{R}_j : [S] \rightarrow \{0,1\}^\lambda$ to record nonces generated to respond to $\mathsf{Sign_1O}(m, e, j, \mathcal{J})$ queries, (ii) $Q_{j,2} : [S] \rightarrow \mathcal{M} \times [E] \times 2^N \times \{\{0,1\}^\lambda\}$ to record all the $\mathsf{Sign_2O}$ queries (with their corresponding inputs) that $\mathcal{A}$ sends to $\mathcal{B}_2$, (iii) $\mathcal{S}'_j : [S] \rightarrow \mathcal{M} \times [E] \times 2^N$ to track $\mathsf{Sign_1O}$ queries, and (iv) $Q'_{j,1}, Q'_{j,2}$ to store auxiliary information for answering $\mathsf{Sign_1O}$ and $\mathsf{Sign_2O}$ queries respectively. All these mappings are initialized as empty at the beginning of simulation. These are in addition to the variables $\mathsf{sid}_j, \mathcal{S}_{i,1}, \mathcal{S}_{i,2}$ defined and initialized as per the security game in Figure 5.

5. We use q_S, q_H, q_C to denote a bound on the number of signing queries, random oracle queries on $\mathsf{H}_{\mathsf{chal}}$ and random oracle queries on $\mathsf{H}_{\mathsf{com}}$, issued by $\mathcal{A}$ across all epochs and signing sessions, respectively. $\mathcal{B}_2$ guesses $j^* \leftarrow\!\!\$\ [q_H + q_S]$, denoting which of the $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ queries will correspond to the forgery that $\mathcal{A}$ will return. j^* is upper bounded by $q_H + q_S$, because we assume w.l.o.g. that $\mathcal{A}$ queries $\mathsf{H}_{\mathsf{chal}}$ before calling $\mathsf{Sign}_3\mathsf{O}$.
6. Reply to oracle queries by $\mathcal{A}$ as follows:
   - $\mathsf{H}_{\mathsf{com}}(R)$: $\mathcal{B}_2$ forwards query $\hat{\mathsf{H}}_{\mathsf{com}}(R)$ to the challenger in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, and sends the response to $\mathcal{A}$, where we use $\hat{\mathsf{H}}_{\mathsf{com}}$ to denote the corresponding random oracle in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$. $\mathcal{B}_2$ sets $\mathsf{H}_{\mathsf{com}}(R) \leftarrow \hat{\mathsf{H}}_{\mathsf{com}}(R)$.
   - $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$: If $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \neq \perp$, then $\mathcal{B}_2$ just returns that. Otherwise, $\mathcal{B}_2$ increments $t \leftarrow t + 1$.
     If $t = j^*$, then $\mathcal{B}_2$ queries its oracle, and sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow \hat{\mathsf{H}}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.
     Otherwise, $\mathcal{B}_2$ iterates over all the $\mathsf{Sign}_2\mathsf{O}$ queries that $\mathcal{A}$ has made so far, i.e. for all j, for all $\mathsf{sid}_j \in Q_{j,2}$, with $(m_j, e_j, \mathcal{J}_j, \{\mathsf{msg}^j_{k,1}\}) \leftarrow Q_{j,2}(\mathsf{sid}_j)$ s.t. $j \notin Q^{SK}_{e_j} \cup Q^{SK'}_{e_j}$ and $m_j = m, \mathcal{J}_j = \mathcal{J}$, $\mathcal{B}_2$ does the following: for all $k \in \mathcal{J}_j \setminus \{j\}$, $\mathcal{B}_2$ finds the element $R_{k,j}$ s.t. $\mathsf{H}_{\mathsf{com}}(R_{k,j}) = \mathsf{msg}^j_{k,1}$. For any k if there's more than one such, $\mathcal{B}_2$ aborts. Otherwise, if there's exactly one $R_{k,j}$ value for all k, then $\mathcal{B}_2$ computes $R^j = \mathcal{R}_j(\mathsf{sid}_j) \prod_{k \in \mathcal{J} \setminus \{j\}} R_{k,j}$, and if $R^j = R$, then $\mathcal{B}_2$ sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$, where h is taken from $Q'_{j,2}(\mathsf{sid}_j)$.
     Lastly, if no such $\mathsf{Sign}_2\mathsf{O}$ query is found, $\mathcal{B}_2$ sends query $\hat{\mathsf{H}}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ to the challenger in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, and sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow \hat{\mathsf{H}}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$.
     In all cases, $\mathcal{B}_2$ sends $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ to $\mathcal{A}$, after setting its value.
   - $\mathsf{skO}(e, i)$: For $e = 1$, $\mathcal{B}_2$ forwards this to the challenger in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, and sends the response to $\mathcal{A}$. For $e > 1$, $\mathcal{B}_2$ samples a random $\mathsf{sk}^e_i \leftarrow\!\!\$\ \mathbb{Z}_p$, and returns sk^e_i to $\mathcal{A}$. $\mathcal{B}_2$ sets $Q^{SK}_e \leftarrow Q^{SK}_e \cup \{i\}$.
   - $\mathsf{Sign}_1\mathsf{O}(e, i, \mathcal{J}, m)$: For $e = 1$, $\mathcal{B}_2$ forwards this to the challenger in $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, and sends the response to $\mathcal{A}$. For $e > 1$, if this is the first signing query for this epoch, and if $\mathcal{A}$ has made less than $t - 1$ skO queries for this epoch, then $\mathcal{B}_2$ samples a random set $Q^{SK'}_e \subset [n] \setminus Q^{SK}_e$ of signers of size $t - 1 - |Q^{SK}_e|$, and for all $j \in Q^{SK'}_e$, $\mathcal{B}_2$ samples a random $\mathsf{sk}^e_j \leftarrow\!\!\$\ \mathbb{Z}_p$. From now on, we can use lagrange interpolation over the elements in $Q^{SK}_e \cup Q^{SK'}_e$ to respond to signing queries for epoch e.
     Then, $\mathcal{B}_2$ sets $\mathsf{sid}_i \leftarrow \mathsf{sid}_i + 1$, $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \cup \{\mathsf{sid}_i\}$ and $\mathcal{S}'_i(\mathsf{sid}_i) \leftarrow (m, \mathcal{J}, e)$. $\mathcal{B}_2$ samples a random element, $c_i \leftarrow\!\!\$\ \{0,1\}^\lambda$, sets $Q'_{i,1}(\mathsf{sid}_i) \leftarrow c_i$ and returns $\mathsf{msg}_{i,1} \leftarrow c_i$ and sid_i to $\mathcal{A}$.
   - $\mathsf{Sign}_2\mathsf{O}(e, i, \mathsf{sid}_i, \{\mathsf{msg}_{j,1}\})$: If $\mathsf{sid}_i \notin \mathcal{S}_{i,1}$ or if e does not match the epoch number in $\mathcal{S}'_i(\mathsf{sid}_i)$, then $\mathcal{B}_2$ returns $\perp$. Otherwise, we denote $(m, \mathcal{J}, e) \leftarrow \mathcal{S}'_i(\mathsf{sid}_i)$. For each $j \in \mathcal{J}$, $\mathcal{B}_2$ finds the element R_j such that $\mathsf{H}_{\mathsf{com}}(R_j) = \mathsf{msg}_{j,1}$. If, for some j, there is more than one such value, then $\mathcal{B}_2$ aborts.
     If $e = 1$, then $\mathcal{B}_2$ just forwards this to its challenger.
     Otherwise, $\mathcal{B}_2$ sets $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \setminus \{\mathsf{sid}_i\}, \mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \cup \{\mathsf{sid}_i\}$. Next, if $i \in Q^{SK}_e \cup Q^{SK'}_e$, then $\mathcal{B}_2$ samples $r_i \leftarrow\!\!\$\ \mathbb{Z}_p$, sets $R_i \leftarrow g^{r_i}$, and saves $Q'_{i,2}(\mathsf{sid}_i) \leftarrow (r_i, R_i)$.
     Otherwise, $\mathcal{B}_2$ calculates $\mathsf{pk}^e_i \leftarrow \mathsf{pk}_i \prod_{j \in Q^{SK}_e \cup Q^{SK'}_e} (\frac{g^{\mathsf{sk}^e_j}}{\mathsf{pk}_j})^{\lambda_j(i)}$, where $\lambda_j(i) = \prod_{k \in Q^{SK}_e \cup Q^{SK'}_e \setminus \{j\}} \frac{k-i}{k-j}$.
     Next, for each $j \in \mathcal{J} \setminus \{i\}$, $\mathcal{B}_2$ finds the $\mathsf{sid}_{j,i}$ s.t. $Q'_{j,1}(\mathsf{sid}_{j,i}) = \mathsf{msg}_{j,1}$.
     - If for any j, there is no such $\mathsf{sid}_{j,i}$, or if $\mathcal{S}'_j(\mathsf{sid}_{j,i}) \neq (m, \mathcal{J}, e)$, $\mathcal{B}_2$ samples $s_i \leftarrow\!\!\$\ \mathbb{Z}_p, h_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes $R_i \leftarrow \frac{g^{s_i}}{(\mathsf{pk}^e_i)^{h_i \lambda^{\mathcal{J}}_i}}$, where $\lambda^{\mathcal{J}}_i = \prod_{j \in \mathcal{J} \setminus \{i\}} \frac{j}{j-i}$. $\mathcal{B}_2$ then sets $Q'_{i,2}(\mathsf{sid}_i) \leftarrow (s_i, h_i, R_i)$. If no such $\mathsf{sid}_{j,i}$ is found for some j, then $\mathcal{B}_2$ sets $\mathsf{flag} \leftarrow 1$.

- Otherwise, for all j for which $\mathcal{B}_2$ found a $\mathsf{sid}_{j,i}$, $\mathcal{B}_2$ checks if $\mathcal{A}$ has called $\mathsf{Sign_2O}(e, j, \mathsf{sid}_{j,i}, \{\mathsf{msg}^j_{k,1}\})$ yet, i.e. $\mathcal{B}_2$ checks if $\mathsf{sid}_{j,i} \in Q_{j,2}$. If this has not been called for any j i.e. $\mathsf{sid}_{j,i} \notin Q_{j,2} \forall j$, then, $\mathcal{B}_2$ samples $s_i \leftarrow\!\!\$\ \mathbb{Z}_p, h_i \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes $R_i \leftarrow \frac{g^{s_i}}{(\mathsf{pk}^e_i)^{h_i \lambda^{\mathcal{J}}_i}}$. $\mathcal{B}_2$ stores $Q'_{i,2}(\mathsf{sid}_i) \leftarrow (s_i, h_i, R_i)$.
- If there is a j such that $\mathsf{sid}_{j,i} \in Q_{j,2}$ i.e. $\mathcal{A}$ has already called $\mathsf{Sign_2O}(m, e, j, \mathcal{J}, \{\mathsf{msg}^j_{k,1}\})$ such that $j \notin Q^{SK}_e \cup Q^{SK'}_e$, then, $\mathcal{B}_2$ compares $\mathsf{msg}_{k,1}$ with $\mathsf{msg}^j_{k,1}$ for all $k \in \mathcal{J} \setminus \{i, j\}$. If they are all equal, and if $\mathsf{msg}^j_{i,1} = Q'_{i,1}(\mathsf{sid}_i)$, then $\mathcal{B}_2$ samples a random $s_i \leftarrow\!\!\$\ \mathbb{Z}_p$ but uses the same h_j from $Q'_{j,2}(\mathsf{sid}_{j,i})$. $\mathcal{B}_2$ computes $R_i \leftarrow \frac{g^{s_i}}{(\mathsf{pk}^e_i)^{h_j \lambda^{\mathcal{J}}_i}}$. $\mathcal{B}_2$ stores $Q'_{i,2}(\mathsf{sid}_i) \leftarrow (s_i, h_j, R_i)$.
- If there is no such j (i.e. for which all the messages in the $\mathsf{Sign_2O}(m, e, j, \mathcal{J}, \mathsf{sid}_{j,i}, \{\mathsf{msg}^j_{k,1}\})$ call match this $\mathsf{Sign_2O}(m, e, i, \mathcal{J}, \mathsf{sid}_i, \{\mathsf{msg}_{j,1}\})$ call, and which is also not in $Q^{SK}_e \cup Q^{SK'}_e$), then $\mathcal{B}_2$ samples a random $s_i \leftarrow\!\!\$\ \mathbb{Z}_p, h_i \leftarrow\!\!\$\ \mathbb{Z}_p$, and sets $R_i \leftarrow \frac{g^{s_i}}{(\mathsf{pk}^e_i)^{h_i \lambda^{\mathcal{J}}_i}}$. $\mathcal{B}_2$ stores $Q'_{i,2}(\mathsf{sid}_i) \leftarrow (s_i, h_i, R_i)$.

In all cases, $\mathcal{B}_2$ sends R_i to $\mathcal{A}$, and sets $Q_{i,2}(\mathsf{sid}_i) \leftarrow (m, e, \mathcal{J}, \{\mathsf{msg}_{j,1}\})$, $\mathsf{H_{com}}(R_i) \leftarrow Q'_{i,1}(\mathsf{sid}_i)$, $\mathcal{R}_i(\mathsf{sid}_i) \leftarrow R_i$.
Lastly, $\mathcal{B}_2$ iterates over all the $\mathsf{Sign_2O}$ queries that have been made so far, i.e. for all j, for all $\mathsf{sid}_j \in Q_{j,2}$, with $Q_{j,2}(\mathsf{sid}_j) = (m_j, e_j, \mathcal{J}_j, \{\mathsf{msg}^j_{k,1}\})$ s.t. $j \notin Q^{SK}_{e_j} \cup Q^{SK'}_{e_j}$ and $i \in \mathcal{J}_j$ and $\mathsf{msg}^j_{i,1} = Q'_{i,1}(\mathsf{sid}_i)$, $\mathcal{B}_2$ does the following: for all $k \in \mathcal{J}_j \setminus \{j\}$, $\mathcal{B}_2$ finds the element $R_{k,j}$ s.t. $\mathsf{H_{com}}(R_{k,j}) = \mathsf{msg}^j_{k,1}$. For any k if there's more than one such, $\mathcal{B}_2$ aborts. Otherwise, if there's exactly one $R_{k,j}$ value for all k, then $\mathcal{B}_2$ computes $R^j = \mathcal{R}_j(\mathsf{sid}_j) \prod_{k \in \mathcal{J} \setminus \{j\}} R_{k,j}$, and does the following:
- If $\mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R^j)$ has been set before, and was not the j^*th $\mathsf{H_{chal}}$ query, then $\mathcal{B}_2$ aborts.

– $\mathsf{Sign_3O}(e, i, \mathsf{sid}_i, \{\mathsf{msg}_{j,2}\})$: If $e = 1$, $\mathcal{B}_2$ forwards this to its challenger. If $\mathsf{sid}_i \notin \mathcal{S}_{i,2}$, then $\mathcal{B}_2$ returns $\perp$. Otherwise, let us denote $(m, \mathcal{J}, e) \leftarrow \mathcal{S}'_i(\mathsf{sid}_i)$.
Upon receiving $\mathsf{msg}_{j,2}$ for all $j \in \mathcal{J} \setminus \{i\}$, $\mathcal{B}_2$ first checks that all the commitments are valid. If, for some $j \in \mathcal{J} \setminus \{i\}, \mathsf{H_{com}}(\mathsf{msg}_{j,2}) = \perp$, then $\mathcal{B}_2$ sets $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \leftarrow \hat{\mathsf{H}}_{\mathsf{com}}(\mathsf{msg}_{j,2})$.
Next, if for some $j \in \mathcal{J} \setminus \{i\}$, $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \neq \mathsf{msg}_{j,1}$ ($\mathcal{B}_2$ uses the $\mathsf{msg}_{j,1}$ stored in $Q_{i,2}(\mathsf{sid}_i)$ to compare), then $\mathcal{B}_2$ returns $\perp$. Note that this perfectly simulates the game $\mathbf{G}^{\text{uf-}\mathbf{0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$, since the protocol returns null if the messages received are malformed.
If $\mathsf{flag} = 1$, $\mathcal{B}_2$ aborts.
If not aborted or returned $\perp$ up to this point, $\mathcal{B}_2$ computes

$$R \leftarrow \mathcal{R}_i(\mathsf{sid}_i) \prod_{j \in \mathcal{J} \setminus \{i\}} \mathsf{msg}_{j,2}.$$

$\mathcal{B}_2$ denotes $h_{m,\mathcal{J},e} \leftarrow \mathsf{H_{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$. Note that this should not correspond to the j^*th query since $\mathcal{A}$ is not allowed to query any partial sign on m^* for its forgery to be valid. i.e. if $(m, \mathsf{pk}, \mathcal{J}, R)$ are the inputs to the j^*th $\mathsf{H_{chal}}$ query, then $\mathcal{B}_2$ aborts.
Note that $h_{m,\mathcal{J},e} \neq \perp$ due to our assumption that $\mathcal{A}$ always calls $\mathsf{H_{chal}}$ oracle before the corresponding $\mathsf{Sign_3O}$ query. Then, if $i \in Q^{SK}_e \cup Q^{SK'}_e$, $\mathcal{B}_2$ returns $r_i + h_{m,\mathcal{J},e} \cdot \mathsf{sk}^e_i \cdot \lambda^{\mathcal{J}}_i$, using r_i from $Q'_{i,2}(\mathsf{sid}_i)$. Otherwise, $\mathcal{B}_2$ uses the s_i value in $Q'_{i,2}(\mathsf{sid}_i)$ and returns it as the partial signature.
Lastly, $\mathcal{B}_2$ does $\mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \setminus \{\mathsf{sid}_i\}$.

$\mathcal{B}_2$ preserves perfect simulation of $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*, (\mathcal{J}^*, R^*, \sigma^*))$. If $(m^*, \mathsf{pk}, \mathcal{J}^*, R^*)$ are not the j^*th $\mathsf{H}_{\mathsf{chal}}$ query, then $\mathcal{B}_2$ aborts. Otherwise, $\mathcal{B}_2$ outputs this forgery as well. We claim that if the output of the simulated $\mathbf{G}^{\text{sa-uf-}\mathbf{0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$ is 1, then so is the output of $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, assuming $\mathcal{B}_2$ does not abort. This is because,

- $\mathsf{Vf}(\mathsf{pk}, m^*, (\mathcal{J}^*, R^*, \sigma^*)) = 1$ implies that this forgery will be valid w.r.t. the 1-epoch game as well.
- $(m^*, (\mathcal{J}^*, R^*, \sigma^*))$ being a valid forgery means that $\mathcal{A}$ didn't query any partial sign on m^*, in any epoch. Since $\mathcal{B}_2$ forwards signing queries to its challenger only in the first epoch, this means that $\mathcal{B}_2$ made no signing queries on m^* as well.
- Since $\mathcal{A}$ returns a valid UF-0 forgery, for each $k \in [E]$, it holds that $|Q_e^{SK}| < t$. This implies, $|Q_1^{SK}| < t$. Also, $\mathcal{B}_2$ forwards no secret key queries in any epoch $e > 1$.

Hence, all the requirements are satisfied for the output of $\mathbf{G}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$ to be 1. Using law of total probability and Bayes theorem, we get:

$$\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{B}_2) \geq \Pr\left[\overline{\mathsf{Abort}}\right] \cdot \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A})$$

Let us use E_1 to denote the event that $\mathcal{B}_2$ guessed j^* correctly, E_{2a} to denote the event that for all q_S signing queries, $\mathcal{A}$ had not queried $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$, before $\mathcal{A}$ calls $\mathsf{Sign_2O}(m, e, i, \mathcal{J}, \cdot)$ for some e, i, and $\mathsf{E}_{2\mathsf{b}}$ to denote the event that, for all signing queries, $\mathcal{B}_2$ had not already programmed $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$, before $\mathcal{A}$ calls $\mathsf{Sign_2O}(m, e, i, \mathcal{J}, \cdot)$ for some e, i. Let E_3 be the event that $\mathcal{B}_2$ doesn't abort due to a collision in $\mathsf{H}_{\mathsf{com}}$ (which is checked in $\mathsf{Sign_2O}(\cdot)$ queries) , and E_4 be the event that $\mathcal{B}_2$ doesn't abort because $\mathsf{flag} = 1$. Then, we have,

$$\begin{aligned}\Pr\left[\overline{\mathsf{Abort}}\right] &= \Pr[\mathsf{E}_1 \wedge \mathsf{E}_{2a} \wedge \mathsf{E}_{2b} \wedge \mathsf{E}_3 \wedge \mathsf{E}_4] \\ &\geq \Pr[\mathsf{E}_1] - \Pr\left[\overline{\mathsf{E}}_{2a}\right] - \Pr\left[\overline{\mathsf{E}}_{2b}\right] - \Pr[\overline{\mathsf{E}_3}] - \Pr[\overline{\mathsf{E}_4}].\end{aligned}$$

Similar to Proof D.7, $\Pr[\mathsf{E}_1] \geq \frac{1}{q_S + q_H}$. $\Pr[\overline{\mathsf{E}_3}] \leq q_C^2/2^\lambda$ and $\Pr[\overline{\mathsf{E}_4}] \leq q_C/2^\lambda \leq q_C^2/2^\lambda$, since the responses of $\mathcal{B}_1$ to $\mathsf{H}_{\mathsf{com}}$ queries are uniformly random in $\{0,1\}^\lambda$.

$\Pr\left[\overline{\mathsf{E}}_{2b}\right] \leq \frac{q_S^2}{p}$, since for every signing query, a collision can occur with probability atmost q_S/p. Next, $\Pr[\mathsf{E}_{2a}] \geq (1 - \frac{q_S}{p})^{q_H}$. This is true since conditioned on $\mathcal{A}$ not observing an $\mathsf{H}_{\mathsf{com}}$ query whose output was $\hat{\mathsf{H}}_{\mathsf{com}}(R_i(m, \mathcal{J}, e))$, its view is independent of the value R in the input $(m, \mathsf{pk}, \mathcal{J}, R)$ to $\mathsf{H}_{\mathsf{chal}}$. We simplify the expression to get $\Pr[\mathsf{E}_{2a}] \geq 1 - \frac{q_S q_H}{p}$, meaning $\Pr\left[\overline{\mathsf{E}}_{2a}\right] \leq \frac{q_S q_H}{p}$.

Combining everything, we get,

$$\begin{aligned}&\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{B}_2) \\ &\qquad \geq \left(\frac{1}{q_S + q_H} - \frac{q_S q_H}{p} - \frac{q_S^2}{p} - \frac{q_C^2}{2^\lambda} - \frac{q_C^2}{2^\lambda}\right) \cdot \mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \\ &\qquad \geq \frac{\mathsf{Adv}^{\text{sa-uf-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A})}{q_S + q_H} - \frac{q_S(q_S + q_H)}{p} - \frac{2q_C^2}{2^\lambda}\end{aligned}$$

This completes the proof.

Proof of Lemma 9

Proof. Consider the following adversary $\mathcal{B}$ playing the game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}$, wherein $\mathcal{B}$ can issue signature queries and random oracle queries. To distinguish the random oracle in $\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}$ from the one in the accountability game of 3-round Schnorr ATS-PR, we denote queries to the former by $\hat{\mathsf{H}}_{\mathsf{sig}}(m, \mathsf{pk}, R)$. We use α to denote the challenge secret key in the Schnorr game $\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}$. On getting the challenge public key $\mathsf{pk}^* \leftarrow g^{\alpha}$ from its Schnorr challenger, $\mathcal{B}$ invokes $\mathcal{A}(\mathbb{G})$ and simulates $\mathbf{G}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$ as follows:

- Receive (n, t, E) from $\mathcal{A}$.
- Guess $i^* \leftarrow\!\!\$\, [n]$. Sample $\{\alpha_1, \ldots, \alpha_{i^*-1}, \alpha_{i^*+1}, \ldots, \alpha_n\} \leftarrow\!\!\$\, \mathbb{Z}_p$. Compute $\mathsf{pk}_j \leftarrow g^{\alpha_j}$ for all $j \neq i^*$.
- Set $\mathsf{pk}_{i^*} \leftarrow \mathsf{pk}^*$ (i.e. $\mathcal{B}$ embeds the challenge public key at position i^*)
- For all $k \in \{2, \ldots, E\}$, sample $a_{1,k}, \ldots, a_{t-1,k} \leftarrow\!\!\$\, \mathbb{F}_p$. Then, for all $i \in [n]$, compute $\delta_i^k \leftarrow \sum_{l=1}^{t-1} a_{l,k} \cdot i^l$ (i.e. we generate new Shamir secret shares of 0 for every epoch)
- Send $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_n\}, \mathsf{pkc} = \perp$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of queries. We use q_S, q_H, q_C to denote a bound on the number of signing queries, random oracle queries on $\mathsf{H}_{\mathsf{chal}}$ and random oracle queries on $\mathsf{H}_{\mathsf{com}}$ across all epochs and signing sessions respectively. $\mathcal{B}$ guesses j^*, denoting which of the $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ queries will correspond to the forgery that $\mathcal{A}$ will return. j^* is upper bounded by $q_H + q_S$, because we assume w.l.o.g. that $\mathcal{A}$ always queries $\mathsf{H}_{\mathsf{chal}}$ before sending a corresponding $\mathsf{Sign}_3\mathsf{O}$ query.

$\mathcal{B}$ initializes (the simulated oracles) $\mathsf{H}_{\mathsf{com}}(R) \leftarrow \perp, \mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow \perp$ for all values of m, R and all possible subsets of signers $\mathcal{J}$. $\mathcal{B}$ also maintains (i) a counter t to track the number of $\mathsf{H}_{\mathsf{chal}}$ queries, and (ii) a dictionary Q_{chal} to track auxiliary information used to answer $\mathsf{H}_{\mathsf{chal}}$ queries. It is initialized as empty, i.e. $Q_{\mathsf{chal}} \leftarrow \emptyset$.

For signing queries, for each signer $j \in [n]$, $\mathcal{B}$ maintains four mappings: (i). $\mathcal{R}_j : [q_S] \to \{0,1\}^\lambda$ to record nonces generated to respond to $\mathsf{Sign}_1\mathsf{O}(e, j, \mathcal{J}, m)$ queries, (ii) Next, S_j to store auxiliary information for answering signing queries, (iii) $M_{j,1}$ to record all the messages sent by $\mathcal{A}$ in a $\mathsf{Sign}_2\mathsf{O}(e, j, \mathsf{sid}, \{\mathsf{msg}_{i,1}\})$ query and (iv) $\mathcal{S}'_j$, to store $(m, \mathcal{J}, e)$ corresponding to session ids. All these mappings are initialized as empty at the beginning of simulation. We now specify how $\mathcal{B}$ answers each of $\mathcal{A}$'s queries:

- $\mathsf{H}_{\mathsf{com}}(R)$: If $\mathsf{H}_{\mathsf{com}}(R)$ has been determined, i.e. if $\mathsf{H}_{\mathsf{com}}(R) \neq \perp$, then $\mathcal{B}_1$ returns $\mathsf{H}_{\mathsf{com}}(R)$. Otherwise, $\mathcal{B}_1$ samples a random string $c \leftarrow\!\!\$\, \{0,1\}^\lambda$ and returns c. $\mathcal{B}_1$ sets $\mathsf{H}_{\mathsf{com}}(R) \leftarrow c$.
- $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$: If $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$ has been determined, i.e. it holds that $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \neq \perp$, then $\mathcal{B}$ returns $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$. Otherwise, $\mathcal{B}$ increments the counter $t \leftarrow t + 1$.
  Next, if $i^* \notin \mathcal{J}$, then, if $t = j^*$, then $\mathcal{B}$ aborts, otherwise, $\mathcal{B}$ samples a random element $h \leftarrow\!\!\$\, \mathbb{Z}_p$, sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$ and returns h to $\mathcal{A}$.
  If $i^* \in \mathcal{J}$, and if $t \neq j^*$, $\mathcal{B}$ iterates through all the $\mathsf{Sign}_2\mathsf{O}(\cdot, i^*, \cdot, \cdot)$ queries that $\mathcal{A}$ has made so far, i.e. for all $\mathsf{sid} \in \mathcal{S}'_{i^*}$, with $(m_{i^*}, \mathcal{J}_{i^*}, e) \leftarrow \mathcal{S}'_{i^*}(\mathsf{sid})$, if $m_{i^*} = m, \mathcal{J}_{i^*} = \mathcal{J}$, then, for each $j \in \mathcal{J} \setminus \{i^*\}$, $\mathcal{B}$ finds R_j s.t. $\mathsf{H}_{\mathsf{com}}(R_j) = \mathsf{msg}_{j,1}$, where $\mathsf{msg}_{j,1}$ is stored in $M_{i^*,1}(\mathsf{sid})$. If there's exactly one such R_j for all $j \in \mathcal{J} \setminus \{i^*\}$, then $\mathcal{B}$ computes $R^{i^*,\mathsf{sid}} = R_{i^*}(\mathsf{sid}) \prod_{j \in \mathcal{J} \setminus \{i^*\}} R_j$, and if $R^{i^*,\mathsf{sid}} = R$, then $\mathcal{B}$ sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$, using h stored in $S_{i^*}(\mathsf{sid})$.
  In all other cases (including the case $i^* \in \mathcal{J}$ and $t = j^*$), $\mathcal{B}$ samples a random message $m' \leftarrow\!\!\$\, \mathcal{M}$, queries its Schnorr challenger for $h \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(m', \mathsf{pk}^*, R^{\frac{1}{\lambda_{i^*}^{\mathcal{J}}}})$. $\mathcal{B}$ sets $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow h$, and returns h to $\mathcal{A}$. $\mathcal{B}$ also stores $Q_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R) \leftarrow m'$.

– $\mathsf{skO}(k,i)$: if $i = i^*$, $\mathcal{B}$ aborts. Otherwise, if $k = 1$, $\mathcal{B}$ returns α_i, and if $k > 1$, $\mathcal{B}$ returns $\alpha_i + \sum_{j=2}^{k} \delta_i^j$.
– $\mathsf{Sign_1O}(e,i,\mathcal{J},m)$: $\mathcal{B}$ first sets $\mathsf{sid}_i \leftarrow \mathsf{sid}_i + 1$, $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \cup \{\mathsf{sid}_i\}$ and $\mathcal{S}'_i(\mathsf{sid}_i) \leftarrow (m,\mathcal{J},e)$. If $i \neq i^*$, then $\mathcal{B}$ samples random $r \leftarrow\$ \mathbb{Z}_p$ and computes $R_i(\mathsf{sid}_i) \leftarrow g^r$. Also, $\mathcal{B}_1$ sets $S_i(\mathsf{sid}_i) \leftarrow r$. If $\mathsf{H_{com}}(R_i(\mathsf{sid}_i)) \neq \perp$, then $\mathcal{B}_1$ returns $\mathsf{msg}_{i,1} \leftarrow \mathsf{H_{com}}(R_i(\mathsf{sid}_i))$. Otherwise, $\mathcal{B}$ samples a random string $c \leftarrow\$ \{0,1\}^\lambda$, sets $\mathsf{H_{com}}(R_i(\mathsf{sid}_i)) \leftarrow c$, and returns $\mathsf{msg}_{i,1} \leftarrow c$ to $\mathcal{A}$.
If $i = i^*$, $\mathcal{B}$ samples a random $c_{i^*} \leftarrow\$ \{0,1\}^\lambda$, stores $S_{i^*}(\mathsf{sid}_{i^*}) \leftarrow c_{i^*}$ and returns $\mathsf{msg}_{i^*,1} \leftarrow c_{i^*}$.
In all cases, $\mathcal{B}$ also sends sid_i to $\mathcal{A}$.
– $\mathsf{Sign_2O}(e,i,\mathsf{sid}_i,\{\mathsf{msg}_{j,1}\})$: $\mathcal{B}$ first checks if $\mathsf{sid}_i \in \mathcal{S}_{i,1}$ and the epoch number in $\mathcal{S}'_i(\mathsf{sid}_i)$ is e, if not, it returns $\perp$. Otherwise, denoting $(m,\mathcal{J},e) \leftarrow \mathcal{S}'_i(\mathsf{sid}_i)$, for each $j \in \mathcal{J} \setminus \{i\}$, $\mathcal{B}$ finds the element R_j such that $\mathsf{H_{com}}(R_j) = \mathsf{msg}_{j,1}$. If for some j there is more than one such, then $\mathcal{B}$ aborts. If for some $j \neq i^*$ this value is undefined, $\mathcal{B}$ sets a flag $\mathsf{flag} \leftarrow 1$, returns $R_i(\mathsf{sid}_i)$, and stores all the messages received, $M_{i,1}(\mathsf{sid}_i) \leftarrow \{\mathsf{msg}_{j,1}\}_{j\in\mathcal{J}\setminus\{i\}}$. Also, for $i \neq i^*, j = i^*$, if there is no $\mathsf{sid}_{i^*,i}$ s.t. $S_{i^*}(\mathsf{sid}_{i^*,i}) = \mathsf{msg}_{i^*,1}$, then $\mathcal{B}_1$ sets $\mathsf{flag} \leftarrow 1$.
Otherwise, if $i \neq i^*$, $\mathcal{B}$ simply returns $R_i(\mathsf{sid}_i)$ and stores all the messages received, $M_{i,1}(\mathsf{sid}_i) \leftarrow \{\mathsf{msg}_{j,1}\}_{j\in\mathcal{J}\setminus\{i\}}$. But if $i = i^*$, $\mathcal{B}$ first samples $s_{i^*} \leftarrow\$ \mathbb{Z}_p$, and a random $h \leftarrow\$ \mathbb{Z}_p$. Then, $\mathcal{B}$ computes $\mathsf{pk}^e_{i^*} \leftarrow \mathsf{pk}^* \cdot g^{\sum_{k=2}^{e} \delta_{i^*}^k}$ and sets $R_{i^*}(\mathsf{sid}_{i^*}) \leftarrow \frac{g^{s_{i^*}}}{(\mathsf{pk}^e_{i^*})^{h\cdot\lambda_{i^*}^{\mathcal{J}}}}$, and sets $\mathsf{H_{com}}(R_{i^*}(\mathsf{sid}_{i^*})) \leftarrow c_{i^*}$, where c_{i^*} is stored in $S_{i^*}(\mathsf{sid}_{i^*})$. $\mathcal{B}$ additionally sets $S_{i^*}(\mathsf{sid}_{i^*}) \leftarrow (S_{i^*}(\mathsf{sid}_{i^*}), s_{i^*}, h)$.
Next, $\mathcal{B}$ computes $R = R_{i^*}(\mathsf{sid}_{i^*}) \prod_{j\in\mathcal{J}\setminus\{i^*\}} R_j$, wherein $\mathsf{H_{com}}(R_j) = \mathsf{msg}_{j,1} \forall j \in \mathcal{J} \setminus \{i^*\}$.
  - If $\mathsf{H_{chal}}(m,\mathsf{pk},\mathcal{J},R)$ was ever queried by $\mathcal{A}$ before, or was programmed by $\mathcal{B}$ while responding to another $\mathsf{Sign_2O}(m,e',i',\mathcal{J},\{\mathsf{msg}_{j,1}\})$ query in a different epoch or signing session, then $\mathcal{B}$ aborts.

$\mathcal{B}$ returns $R_{i^*}(\mathsf{sid}_{i^*})$ and stores all the messages received, $M_{i^*,1}(\mathsf{sid}_{i^*}) \leftarrow \{\mathsf{msg}_{j,1}\}_{j\in\mathcal{J}\setminus\{i^*\}}$.
In all cases where $\mathcal{B}$ doesn't abort or send $\perp$, $\mathcal{B}$ sets $\mathcal{S}_{i,1} \leftarrow \mathcal{S}_{i,1} \setminus \{\mathsf{sid}_i\}, \mathcal{S}_{i,2} \leftarrow \mathcal{S}_{i,2} \cup \{\mathsf{sid}_i\}$.
– $\mathsf{Sign_3O}(e,i,\mathsf{sid}_i,\{\mathsf{msg}_{j,2}\})$: $\mathcal{B}$ first checks that all the commitments sent by $\mathcal{A}$ are valid. We denote $(m,\mathcal{J},e) \leftarrow \mathcal{S}'_i(\mathsf{sid}_i)$. If, for some $j \in \mathcal{J} \setminus \{i\}, \mathsf{H_{com}}(\mathsf{msg}_{j,2}) = \perp$, then $\mathcal{B}$ samples a random number $c \leftarrow\$ \{0,1\}^\lambda$ and sets $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \leftarrow c$.
Next, if for some $j \in \mathcal{J} \setminus \{i\}$, $\mathsf{H_{com}}(\mathsf{msg}_{j,2}) \neq \mathsf{msg}_{j,1}$ (using $\mathsf{msg}_{j,1}$ stored in $M_{i,1}(\mathsf{sid}_i)$), then $\mathcal{B}$ returns $\perp$. Also, if $\mathsf{flag} = 1$, then $\mathcal{B}$ aborts. Note that this perfectly simulates the game $\mathbf{G}^{\text{uf-}\mathbf{0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}$, since the protocol returns null if the messages received are malformed.
If not aborted or returned $\perp$ yet, $\mathcal{B}$ computes $R = R_i(\mathsf{sid}_i) \prod_{j\in\mathcal{J}\setminus\{i\}} \mathsf{msg}_{j,2}$.
  - If $i = i^*$ and if the j^*th $\mathsf{H_{chal}}$ query has already been made, then $\mathcal{B}$ checks if the j^*th H query had inputs $(m,\mathsf{pk},\mathcal{J},R)$, if so, then $\mathcal{B}$ aborts since for $\mathcal{A}$ to return a valid acc-0 forgery on $(m,\mathsf{pk},\mathcal{J},R)$, $\mathcal{A}$ is not allowed to query partial signature of the i^* signer on the forgery message.
  - Note that $\mathsf{H_{chal}}() \neq \perp$ because of our assumption that $\mathcal{A}$ always queries $\mathsf{H_{chal}}$ before calling the corresponding $\mathsf{Sign_3O}$ query.

Then, if $i \neq i^*$, $\mathcal{B}$ returns $s_i \leftarrow r_i + \lambda_i^{\mathcal{J}}(\alpha_i + \sum_{k=2}^{e} \delta_i^k) \cdot \mathsf{H_{chal}}(m,\mathsf{pk},\mathcal{J},R)$, where $r_i \leftarrow S_i(\mathsf{sid}_i)$.
If $i = i^*$, $\mathcal{B}$ returns s_{i^*} from $S_{i^*}(\mathsf{sid}_{i^*})$.

It can be seen that $\mathcal{B}$ is able to perfectly simulate the game $\mathbf{G}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}$.

Eventually, $\mathcal{A}$ outputs a forgery $(m^*,\mathcal{J}^*,z^*,R^*)$. If $i^* \notin \mathcal{J}^*$ or if $m^*,\mathsf{pk},\mathcal{J}^*,R^*$ are not the j^*th $\mathsf{H_{chal}}$ query, then $\mathcal{B}$ aborts. Otherwise, We assume w.l.o.g. that $\mathsf{H_{chal}}(m^*,\mathsf{pk},\mathcal{J}^*,R^*) \neq \perp$ and denote it as $h^* \leftarrow \mathsf{H_{chal}}(m^*,\mathsf{pk},\mathcal{J}^*,R^*)$.

$\mathcal{B}$ finds $(m'^*) \leftarrow Q_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, R^*)$ which was used to program $\mathsf{H}_{\mathsf{chal}}(m^*, \mathsf{pk}, \mathcal{J}^*, R^*) \leftarrow \hat{\mathsf{H}}_{\mathsf{sig}}(m'^*, \mathsf{pk}^*, (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}})$.

$\mathcal{B}$ returns $(m'^*, (R^*)^{\frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}}, \frac{1}{\lambda_{i^*}^{\mathcal{J}^*}}(z^* - h^* \cdot \sum_{j \in \mathcal{J}^* \setminus \{i^*\}} \lambda_j^{\mathcal{J}^*} \alpha_j))$ as a Schnorr signature forgery.

Similar to the proof of Lemma 7, it can be seen that this is a valid forgery. Hence, $\mathcal{B}$ is able to produce a valid forgery using $\mathcal{A}$ whenever it doesn't abort. Let Abort denote the event in which $\mathcal{B}$ prematurely aborts the simulation. This means that $\Pr\left[\mathbf{G}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]} = 1 \mid \overline{\mathsf{Abort}}\right] = \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR\text{-}1}[\mathbb{G}]}(\mathcal{A})$. Hence we get (similar to D.7),

$$\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}) = \Pr\left[\overline{\mathsf{Abort}}\right] \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A})$$

Let us use E_{1a} to denote the event that $\mathcal{B}$ guesses i^* correctly (i.e. $i^* \in \mathcal{J}^*$ and $i^* \notin Q_e^{SK} \cup Q_e^{Sig}(m^*)$ for all epochs e) and E_{1b} denotes the event that $\mathcal{B}$ guesses j^* correctly. Also, let E_{2a} to denote the event that for all q_S signing queries, $\mathcal{A}$ had not queried $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$, before $\mathcal{A}$ calls $\mathsf{Sign_2O}(m, e, i, \mathcal{J})$ for some e, i, and $\mathsf{E_{2b}}$ to denote the event that, for all signing queries, $\mathcal{B}$ had not already programmed $\mathsf{H}_{\mathsf{chal}}(m, \mathsf{pk}, \mathcal{J}, R)$, before $\mathcal{A}$ calls $\mathsf{Sign_2O}(m, e, i, \mathcal{J})$ for some e, i. Let E_3 be the event that $\mathcal{B}_2$ doesn't abort due to a collision in $\mathsf{H}_{\mathsf{com}}$ (which is checked in $\mathsf{Sign}_2O(\cdot)$ queries), and E_4 be the event that $\mathcal{B}_2$ doesn't abort because $\mathsf{flag} = 1$.

We have $\Pr[\mathsf{E}_{1a}] \geq \frac{1}{n}$, $\Pr[\mathsf{E}_{1b}] \geq \frac{1}{q_H + q_S}$ (similar to D.7) and, $\Pr\left[\overline{\mathsf{E}_{2b}}\right] \leq \frac{q_S^2}{p}$, $\Pr\left[\overline{\mathsf{E}_{2a}}\right] \leq \frac{q_S q_H}{p}$, $\Pr[\overline{\mathsf{E}_3}] \leq q_C^2/2^\lambda$ and $\Pr[\overline{\mathsf{E}_4}] \leq q_C/2^\lambda \leq q_C^2/2^\lambda$, similar to D.7.

Then, similar to the analysis in D.7, we get,

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{uf}}_{\mathsf{Schnorr}[\mathbb{G}]}(\mathcal{B}) &= \Pr\left[\overline{\mathsf{Abort}}\right] \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \\
&= (\Pr[\mathsf{E}_{1a} \wedge \mathsf{E}_{1b} \wedge \mathsf{E}_{2a} \wedge \mathsf{E}_{2b} \wedge \mathsf{E}_3 \wedge \mathsf{E}_4]) \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \\
&\geq (\Pr[\mathsf{E}_{1a} \wedge \mathsf{E}_{1b}] - \Pr\left[\overline{\mathsf{E}_{2a}}\right] - \Pr\left[\overline{\mathsf{E}_{2b}}\right] - \Pr[\overline{\mathsf{E}_3}] - \Pr[\overline{\mathsf{E}_4}]) \\
&\quad \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \\
&\geq \left(\frac{1}{n(q_S + q_H)} - \frac{q_S^2}{p} - \frac{q_S q_H}{p} - \frac{q_C^2}{2^\lambda} - \frac{q_C^2}{2^\lambda}\right) \cdot \mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A}) \\
&\geq \frac{\mathsf{Adv}^{\text{sa-acc-0}}_{\mathsf{Schnorr3\text{-}PR}[\mathbb{G}]}(\mathcal{A})}{n(q_S + q_H)} - \frac{q_S(q_S + q_H)}{p} - \frac{2q_C^2}{2^\lambda}
\end{aligned}$$

This completes the proof.