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 work we construct the first accountable threshold signature schemes that support a *proactive refresh*. 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. It is an important security mechanism that helps protect a secret key from a gradual exposure of key shares. However, until now, no ATS scheme supported a proactive refresh. We begin by giving several definitions for this new concept that achieve different levels of security. 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 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. The third is a hybrid construction that performs well for a large number of signers and satisfies our strongest security definition.

1 Introduction

A threshold signature scheme [21] 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 2 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 that the signing quorum $\mathcal{J}$ 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 finance and in blockchain applications. For example, Bitcoin multisig [2] is an ATS. Ethereum consensus [41] uses an ATS because accountability is needed for slashing a misbehaving attestor. Companies that provide custody services for digital assets often use an ATS to protect critical signing keys.

The most widely used t-out-of-n ATS scheme is called the *trivial ATS* and is built by concatenating t signatures. At setup, for $i = 1, \ldots, n$, signer i locally generates a public-private key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for a standard (non-threshold) signature scheme. The complete public key is the concatenation of all n local public keys, namely $\mathsf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. 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 with respect to some t of the n public keys in pk. The tracing algorithm can easily determine which parties participated in generating a given valid signature by checking which of the n public keys in pk were used. One downside of this scheme is that signature size and verification time are at least linear in $t\lambda$, where t is the threshold and λ is the security parameter. Although several ATS constructions achieve much lower signature size and verification time [35, 6, 37, 10, 13, 12], this trivial ATS is used widely, for example in Bitcoin multisig transactions [2].

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 [38] 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 [28, 27, 24, 23, 39, 4, 25, 18, 1, 30, 20, 32]. The use of proactive refresh could have made it harder to carry out some high profile hacks, such as the Ronin and Multichain bridge hacks, to name a few.

Until now, no ATS scheme supported a proactive refresh. This presents a difficulty for custody services who wish to periodically refresh their ATS secret key shares. Currently, they refresh by re-generating all the shares from scratch, and must re-register the new public key. Changing the public key is costly and this limits the frequency with which they refresh key shares.

Our results. In this paper we initiate the study of proactive refresh for accountable threshold signatures (ATS). An ATS with proactive refresh, or ATS-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 unforge-

able 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 counter-intuitive. Each party's secret key is used to hold that party accountable for a rogue signature. If we refresh the party's secret key, then 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 construct new ATS schemes where the tracing algorithm works correctly despite the fact that all the secret keys change at the beginning of every epoch.

In Section 2 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. [5] 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 in $\mathcal{J}$ 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' (recall that the public key remains unchanged, and hence the verification algorithm is oblivious to the epoch in which a signature was produced). 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 $\mathcal{J}$ 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 there is some party in $\mathcal{J}$ that is incorrectly blamed for signing m^*. The definition requires that no efficient adversary can satisfy this condition. We discuss this further in Section 2. 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 the full version [16, App. 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 2 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 the full version [16, App. 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, and

need not follow the update protocol honestly. 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 5 we discuss more efficient lifting techniques for our specific constructions.

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 [28, 27, 24, 23, 39], Schnorr [27, 31, 34, 32], and BLS [15, 10]. It satisfies $\mathsf{uf\text{-}1} \wedge \mathsf{acc\text{-}1}$ security, our strongest notion of security.

In Section 3 we present a 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 e 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_e; and (iii) sign the newly generated ATS public key pk_e using the scheme $\mathcal{S}_1$. A signature on a message m is a triple $(\mathsf{pk}_e, \sigma_1, \sigma_2)$, where σ_1 is the $\mathcal{S}_1$ signature on pk_e, 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 the full version [16] we construct a new factoring-based ATS that has both properties: a constant size public key (i.e., its size is independent of t and n) and a simple distributed key generation protocol. A recent work of [17] gives another example built from pairings.

Finally, in the full version [16] we construct very practical refreshable ATS schemes from standard signature schemes such as Schnorr [40] and BLS [15]. This leads to practical short ATS schemes that support proactive refresh. However, we describe an attack that shows that the schemes do not provide $\mathsf{acc\text{-}1}$ accountability. Instead, we prove that they provide $\mathsf{acc\text{-}0}$ accountability. Signature generation and verification in these schemes are essentially as efficient, and signatures are of the same length, as in non-refreshable versions of Threshold Schnorr and BLS.

Additional related work. The notion of an accountable threshold signature (ATS) is closely related to the concept of a multisignature defined in [35] and further developed in [6, 13, 37, 29, 3, 12]. 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 [12]).

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 [14] provides both properties: it is fully private to the public, but fully accountable to an authority that holds a secret tracing key.

2 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. To simplify the presentation, we focus on non-interactive schemes, and formally consider interactive schemes in the full version.

2.1 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 2.1 (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*

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

where the probability is over the random variables $(\mathsf{pk}, \mathsf{pkc}, \mathsf{sk}_1^1, \ldots, \mathsf{sk}_n^1) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{pp}, n, t)$, $(\mathsf{sk}_1^{i+1}, \ldots, \mathsf{sk}_n^{i+1}) \leftarrow\!\!\$\ \mathsf{Update}(\mathsf{pk}, \mathsf{sk}_1^i, \ldots, \mathsf{sk}_n^i)$ *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 2.2 (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*

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

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.

2.2 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 [5], 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. In the full version, we explore 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 [8]. 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 un-

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

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 : $(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{skO}(\cdot,\cdot),\mathsf{SignO}(\cdot,\cdot,\cdot)}(\mathsf{st}, \mathsf{pk}, \mathsf{pkc})$
7 : **if** $\mathsf{Vf}(\mathsf{pk}, m^*, \sigma^*) = 0$ **then**
8 : $\quad$**return** 0
9 : **if** $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e| < t \wedge |\mathcal{Q}^{\mathsf{sig}}_e(m^*)| = 0$ **then** $\mathsf{flag}_{\text{uf-0}} \leftarrow 1$
10 : **if** $\forall e \in [E], |\mathcal{Q}^{\mathsf{sk}}_e \cup \mathcal{Q}^{\mathsf{sig}}_e(m^*)| < t$ **then** $\mathsf{flag}_{\text{uf-1}} \leftarrow 1$
11 : **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** $\mathsf{flag}_{\text{acc-0}} \leftarrow 1$
12 : **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** $\mathsf{flag}_{\text{acc-1}} \leftarrow 1$
13 : Only game $\mathbf{G}^{\text{uf-}b}_{\mathsf{PRATS}[\mathsf{pp}]}$: **return** $\mathsf{flag}_{\text{uf-}b}$
14 : Only game $\mathbf{G}^{\text{acc-}b}_{\mathsf{PRATS}[\mathsf{pp}]}$: **return** $\mathsf{flag}_{\text{acc-}b}$
15 : Only game $\mathbf{G}^{\text{uf-}b\wedge\text{acc-}b'}_{\mathsf{PRATS}[\mathsf{pp}]}$: **return** $\mathsf{flag}_{\text{uf-}b} \vee \mathsf{flag}_{\text{acc-}b'}$

Oracle $\mathsf{skO}(e, i)$
1 : $\mathcal{Q}^{\mathsf{sk}}_e \leftarrow \mathcal{Q}^{\mathsf{sk}}_e \cup \{i\}$
2 : **return** sk^e_i

Oracle $\mathsf{SignO}(m, e, i)$
1 : $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}^e_i, m)$
2 : $\mathcal{Q}^{\mathsf{sig}}_e(m) \leftarrow \mathcal{Q}^{\mathsf{sig}}_e(m) \cup \{i\}$
3 : **return** σ_i

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 $\perp \not\subseteq \mathcal{Q}$ for any set $\mathcal{Q}$. Hence, if the adversary successfully outputs a valid signature σ^* on a message m^* such

that $\mathsf{Trace}(\mathsf{pk}, m^*, \sigma^*) = \perp$, 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 $\perp$ 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 the full version, 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 2.3 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 2.3. *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 2.3 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 2.3 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 [33]. 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 $\text{prop} \in \{\text{uf-}b, \text{acc-}b, \text{uf-}b \wedge \text{acc-}b'\}_{b,b' \in \{0,1\}}$, denote by $\mathbf{G}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ the semi-adaptive security game obtained from $\mathbf{G}^{\text{prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ 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 2.4. *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* $\text{prop} \in \{\text{uf-}b, \text{acc-}b, \text{uf-}b \wedge \text{acc-}b'\}_{b,b' \in \{0,1\}}$. *The advantage of an adversary* $\mathcal{A}$ *in* $\mathbf{G}^{\text{sa-prop}}_{\mathsf{PRATS}[\mathsf{pp}]}$ *is defined as*

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

Our combinatorial and two-tier constructions 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 the full version, we present direct constructions of ATS-PR schemes from BLS and Schnorr, 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 2.3), all probabilities are then also taken over the choice of H.

3 An Efficient Construction with Strong Security Guarantees

In this section, we present a generic two-tier approach for obtaining ATS-PR schemes efficiently and with strong security guarantees: the construction satisfies the stronger acc-1 accountability notion. 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.

Although this basic idea might seem simple at first blush, it is unclear altogether that it can be instantiated efficiently. In fact, instantiating it using existing ATS schemes will result in an inefficient ATS-PR scheme. Making it efficient requires coming up with an entirely new ATS scheme that fits surprisingly well with our two-tier approach. We discuss the reasons for this later in this section, and we present our new ATS scheme in Appendix 4.

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 $\mathsf{PRTS} = (\mathsf{PRTS.KGen}, \mathsf{PRTS.Sign}, \mathsf{PRTS.Combine}, \mathsf{PRTS.Vf}, \mathsf{PRTS.Update})$.[1]
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 PRTS.Sign, where the i-th signer runs on local input (sk_i, m) and σ is the result of applying 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 the full version for a formal definition of interactive ATS-PR schemes).

Our ATS-PR scheme, called PRATS, is then defined as follows.[2]

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

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

1. Sample $(\mathsf{PRTS.pk}, \mathsf{PRTS.pkc}, (\mathsf{PRTS.sk}_1, \ldots, \mathsf{PRTS.sk}_n)) \leftarrow\!\!\$\ \mathsf{PRTS.KGen}(\mathsf{pp}, n, n)$.
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 $\mathsf{sk}_i \leftarrow (\mathsf{PRTS.sk}_i, \mathsf{ATS.sk}_i, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}})$.

[1] We only need PRTS to support n-out-of-n signing, which may be easier to instantiate than general threshold signature schemes.

[2] 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.

5. Output $(\mathsf{pk} = (n, t, \mathsf{PRTS.pk}),\ \mathsf{pkc} = \mathsf{ATS.pkc},\ (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

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

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 $s_i = (\mathcal{J}, \mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, s_m)$.

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

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 $\sigma = (\mathsf{ATS.pk}, \sigma_{\mathsf{pk}}, \sigma_m)$.

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

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.

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

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 $\mathcal{J} = \mathsf{ATS.Trace}(\mathsf{ATS.pk}, m, \sigma_m)$.

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

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 $\mathsf{PRTS.Sign}(\mathsf{PRTS.pk}.\mathsf{PRTS.sk}'_i, \mathsf{ATS.pk})$ 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 [7]), their key generation procedures rely on a trapdoor, and hence do not admit efficient distributed analogs. To fill in this gap, in Section 4 we present a new construction of an ATS with short public keys and an efficient key distribution protocol.

Security. Theorem 3.1 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 3.1. *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*

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

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

The proof of Theorem 3.1 is presented in the full version.

4 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 Appendix 3, this yields a concretely-efficient ATS-PR construction.

Our scheme relies on the strong RSA assumption. Previous constructions of ATS schemes from RSA-like assumptions in hidden-order groups (e.g., [7]) require knowledge of the group's order to compute parties' secret keys. Hence, they inherently require either trusted key generation or a heavy MPC for distributed key generation. Additionally, they can only be instantiated in groups that admit a trapdoor, like RSA groups or subgroups thereof, and cannot be instantiated in trapdoorless groups like class groups imaginary quadratic fields.

Our construction. To address the aforesaid issues, we adopt a different approach than previous constructions and draw inspiration from cryptographic accumulators (see [19] as well as [9, 11] and the many references therein). In a nutshell, 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 deterministically derived from the index i. We carefully generalize the GQ protocol [26] to allow 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 via the Fiat-Shamir transform [22]. 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 $i \in [n]$. 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.

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

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

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 $(\mathsf{pk} = (n, t, Y), \mathsf{pkc} = \perp, (\mathsf{sk}_1, \ldots, \mathsf{sk}_n))$.

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

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).

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

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 $\sigma = (\mathcal{J}, h, S)$.

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

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.

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

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 public 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 3. 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 public 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 4.1.

Definition 4.1. *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*

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

Theorem 4.2 below, whose proof can be found in the full version, reduces the security of RSAATS to the hardness of the strong RSA problem.

Theorem 4.2. *For any adversary $\mathcal{A}$ there exists an algorithm $\mathcal{B}$ such that*

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

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.

5 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 the full version.

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 signers is known a-priori, this step can be replaced with a distributed key-generation protocol. In our BLS- and Schnorr-based constructions, the (honestly-generated) secret keys are sampled independently of each other. Hence, these constructions further support signers that (i) join at any time, and (ii) locally generate their key material.

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. This will require employing a mechanism to defend against rogue key attacks, in which the adversary maliciously chooses the keys of corrupted parties based on what it knows about the keys of honest parties [13]. In the simpler case where the identity of potential signers is known in advance, such mechanisms can be incorporated in the distributed key generation protocol. In the setting where keys are locally and non-interactively generated by the parties, the signing protocol needs to be extended to disallow such attacks. For more information on rogue key attacks and how to protect against them, see [36, 13, 10, 6, 12, 37] and the references therein.

In the full version, we outline a mechanism to securely allow local key generation in our BLS-based ATS-PR. This mechanism is based on the fact that all the secret key shares in that scheme are chosen independently of one another.

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.

Key updates with partial participation. Similarly to the standard notion of proactive refresh in (non-accountable) threshold signatures, our definitions and constructions assume that all parties are online during key updates. An interesting open question is to construct ATS schemes with proactive refresh in which key updates can be performed when some of the parties are offline. Offline parties can then refresh their keys when they go back online. This task was recently considered in the non-accountable setting [32].

However, the problem becomes much more technically involved when requiring accountability. In non-accountable refreshable signatures, when an offline signer goes back online, she can engage in a multiparty computation protocol with t previously-online parties to learn her updated secret key. The reason is that in such signature schemes, a quorum of t signers with up-to-date secret keys can compute the up-to-date secret keys of all other $n-t$ signers. This provably cannot be the case in the accountable setting (since otherwise, the t online parties could frame the offline party). Interestingly, our two-tier scheme (Section 3) seems to be the best-suited for an extension to this offline/online setting. At the beginning of each epoch, we can generate keys for the underlying ATS scheme exclusively for the parties who are currently online. Since the underlying refreshable scheme does not need to be accountable, it can also be refreshed, as discussed above. The issue that remains is how to sign the new ATS public key using the refreshable scheme when not all parties are online. We leave resolving this as an interesting open question.

References

1. 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)
2. Andresen, G.: Bitcoin m-of-n standard transactions (2011), BIP-0011
3. 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
4. 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
5. 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)
6. 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)
7. Bellare, M., Neven, G.: Identity-based multi-signatures from RSA. In: Topics in Cryptology – CT-RSA 2007. pp. 145–162 (2007)
8. 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)
9. Benaloh, J., de Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Advances in Cryptology – EUROCRYPT '93. pp. 274–285 (1993)
10. 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)
11. 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)
12. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: ASIACRYPT'18. pp. 435–463 (2018)
13. 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)
14. Boneh, D., Komlo, C.: Threshold signatures with private accountability. In: CRYPTO'22. LNCS, vol. 13509. Springer (2022)
15. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Advances in Cryptology — ASIACRYPT 2001. p. 514–532 (2001)
16. Boneh, D., Partap, A., Rotem, L.: Proactive refresh for accountable threshold signatures. Cryptology ePrint Archive, Paper 2022/1656 (2022), link
17. Boneh, D., Partap, A., Waters, B.: Accountable multi-signatures with constant size public keys. Cryptology ePrint Archive, Paper 2023/1793 (2023), link
18. 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
19. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: Advances in Cryptology – CRYPTO '02. pp. 61–76 (2002)

20. 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
21. 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
22. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO'86. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (Aug 1987). https://doi.org/10.1007/3-540-47721-7_12
23. 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
24. 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
25. 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
26. 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)
27. 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
28. 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
29. Itakura, K., Nakamura, K.: A public-key cryptosystem suitable for digital multisignatures. NEC research and development (1983)
30. 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)
31. Komlo, C., Goldberg, I.: FROST: flexible round-optimized schnorr threshold signatures. In: Selected Areas in Cryptography. p. 34–65 (2020)
32. 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
33. 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)
34. Lindell, Y.: Simple three-round multiparty schnorr signing with full simulatability. IACR Cryptol. ePrint Arch. (2022), https://eprint.iacr.org/2022/374
35. Micali, S., Ohta, K., Reyzin, L.: Accountable-subgroup multisignatures: Extended abstract. In: CCS'01. pp. 245–254. ACM (2001)
36. 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

37. Nick, J., Ruffing, T., Seurin, Y.: Musig2: Simple two-round schnorr multi-signatures. In: Advances in Cryptology – CRYPTO' 21. pp. 189–221 (2021)
38. 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
39. 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
40. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Advances in Cryptology – CRYPTO '89. pp. 239–252 (1989)
41. Smith, C.: Ethereum proof of stake (2022), link