Count Me In!
Extendablity for Threshold Ring Signatures

Diego Aranha[1], Mathias Hall-Andersen[1], Anca Nitulescu[3],
Elena Pagnin[2⋆], and Sophia Yakoubov[1⋆⋆]

[1] Aarhus University, Aarhus, Denmark; {dfaranha, ma, sophia.yakoubov}@cs.au.dk
[2] Lund University, Lund, Sweden; elena.pagnin@eit.lth.se
[3] Protocol Labs

Abstract. Ring signatures enable a signer to sign a message on behalf of a group *anonymously*, without revealing her identity. Similarly, threshold ring signatures allow several signers to sign the same message on behalf of a group; while the combined signature reveals that some threshold t of the group members signed the message, it does not leak anything else about the signers' identities. Anonymity is a central feature in threshold ring signature applications, such as whistleblowing, e-voting and privacy-preserving cryptocurrencies: it is often crucial for signers to remain anonymous even from their fellow signers. When the generation of a signature requires interaction, this is difficult to achieve. There exist threshold ring signatures with non-interactive signing — where signers locally produce *partial* signatures which can then be aggregated — but a limitation of existing threshold ring signature constructions is that all of the signers must agree on the group on whose behalf they are signing, which implicitly assumes some coordination amongst them. The need to agree on a group before generating a signature also prevents others — from outside that group — from endorsing a message by adding their signature to the statement post-factum.

We overcome this limitation by introducing *extendability* for ring signatures, same-message linkable ring signatures, and threshold ring signatures. Extendability allows an untrusted third party to take a signature, and *extend* it by enlarging the anonymity set to a larger set. In the extendable threshold ring signature, two signatures on the same message which have been extended to the same anonymity set can then be combined into one signature with a higher threshold. This enhances signers' anonymity, and enables new signers to anonymously support a statement already made by others.

For each of those primitives, we formalize the syntax and provide a meaningful security model which includes different flavors of anonymous extendability. In addition, we present concrete realizations of each primitive and formally prove their security relying on signatures of knowledge and the hardness of the discrete logarithm problem. We also describe a generic transformation to obtain extendable threshold ring signatures from same-message-linkable extendable ring signatures. Finally, we implement and benchmark our constructions.

Keywords: Ring Signatures, Threshold Ring Signatures, Anonymity, Flexibility, Extendability

1 Introduction

Anonymity has become a requirement in many real-world implementations of cryptographic systems and privacy-enhancing technologies, including electronic voting [26], direct anonymous attestation [9], and private cryptocurrencies [29]. Another compelling scenario is whistleblowing of organizational wrongdoing. In this case, an insider publishes a *secret* in a manner that convinces the public of its authenticity, while having his/her identity protected [27]. In all of these applications, a large *anonymity set*, i.e., set of users who may have performed a certain action, is crucial in order to not reveal who exactly is behind it.

⋆ Funded in part by ELLIIT and the Swedish Foundation for Strategic Research grant RIT17-0035.
⋆⋆ Funded in part by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 803096 (SPEC).

Group signatures enable any member of a given group to sign a message, without revealing which member signed. However, group signatures suffer from the drawback that they require trusted setup for every group. Ring signatures are a manager-free variant of group signatures. They enable individual users to sign messages anonymously on behalf of a dynamically chosen group of users, while hiding the exact identity of the signer(s) [27]. Traditionally, this is enabled by including a "ring" $\mathcal{R}$ of public keys (belonging to all possible signers, including the actual signer) as an input to the signing algorithm; a ring signature does not reveal which of the corresponding secret keys was used to produce it. There are many ways to construct ring signatures using different building blocks: classic RSA [13], bilinear pairings [32,5,12], composite-order groups [28,7], non-interactive zero knowledge [6,21], and, most recently, quantum-safe isogenies and lattices [14,20,19,4].

Threshold ring signatures are a threshold variant of this primitive [8], which allow some t signers to sign a message on behalf of a ring $\mathcal{R}$ of size larger than t. The signature reveals that t members of the ring signed the message, but not the identities of those members. Some threshold ring signature schemes are *flexible* [24], meaning that even after the threshold ring signature has been produced for a given ring $\mathcal{R}$, another signer from that ring can participate, resulting in a threshold ring signature for the same ring $\mathcal{R}$ but with a threshold of $t + 1$. However, if a signer from *outside* the ring wants to participate, existing constructions do not support this. All existing constructions of ring and threshold ring signatures have a common limitation: the ring of potential signers is fixed at the time of signature generation. In particular, it is not possible to have the added flexibility of publicly "adjusting" the ring, i.e., to extend the initial ring to a larger one, increasing the anonymity set. Increasing the size of the set of potential signers not only increases the anonymity provided by the signature, but also makes threshold systems easier to realize in practice.

To work in practice, standard threshold ring signatures need all of the signers to independently sign the same message μ with the same ring $\mathcal{R}$, which must include the public keys of all t signers. We are interested in relaxing this implicit synchronization requirement.

1.1 Our Contributions

In this paper, we introduce a new property of (threshold) ring signatures which we call *extendability.* A (threshold) ring signature scheme is *extendable* if it allows anyone to enlarge the set of potential signers of a given signature. Extendable threshold ring signatures are fundamental for whistleblowing, where one party may want to "join the cause" after it becomes public. Extendability, together with flexibility, enables a signer A to join a threshold ring signature which was produced using an anonymity ring $\mathcal{R}$ that does not contain A. This can be done by first extending the existing signature to a new ring $\mathcal{R}' \supseteq \mathcal{R} \cup \{A\}$ which contains both the ring used by previous signers as well as the new signer. Then, thanks to flexibility, the new signer can add their own signature with respect to the new ring $\mathcal{R}'$ (using $\mathtt{sk}_A$). (Of course, an observer who has seen signatures under the old ring $\mathcal{R}$ and under the new ring $\mathcal{R}'$ will be able to determine $\mathcal{R}' \backslash \mathcal{R}$; this is inherent — since an observer can always tell which ring a signature is meant for by attempting verification — and can help that observer narrow down possibilities for the identity of A. However, an observer who has *not* seen a signature under the old ring $\mathcal{R}$ will learn nothing additional about the identity of A.)

In addition to drawing formal models, we give the first constructions of *extendable ring signatures*, *same-message linkable* extendable ring signatures and extendable *threshold* ring signatures. We provide a proof of concept implementation of our construction, benchmark the signing and verification running times as well as the signature size.

Constructions from Signatures of Knowledge and Discrete Log We build extendable ring signatures and same-message linkable extendable ring signatures using signatures of knowledge. Each signature will include several elements of a group, with the property that all of their discrete logs cannot be known. (This is because the product of the elements gives a discrete log challenge which is part of the public parameters.) A signer signs the message with a *signature of knowledge* that proves that she knows either her own secret key, or the discrete log of one of the elements. The signer uses her secret key for this (and so can use the element for which the discrete log is unknown), but for each of the other signers' public keys in the ring, she includes a signature of knowledge using the discrete log of one of the elements. Because all of the element discrete logs cannot be known, a verifier is convinced that at least one signature of knowledge is produced using a secret key, and that therefore the overall signature was produced by one of the members of the ring.

We build extendable threshold ring signatures similarly, but by choosing the elements in such a way that at least t of their discrete logs cannot be known without revealing the discrete log of a challenge element in the public parameters. We enforce this by placing the elements on a polynomial of appropriate degree.

A Generic Transformation One might hope to build extendable threshold ring signatures by concatenating t extendable ring signatures; however, we would need to additionally prove to the verifier that the t signatures were produced by t different signers. Building such a proof would require interaction between the signers, and it would be challenging to maintain the proof as the ring is expanded. Instead, we solve this problem using a primitive which we call a *same-message linkable extendable ring signatures*, where, given two signatures on the same message, it is immediately clear whether they were produced by the same signer. Our realizations of this primitive provide linkability without revealing the signer's identity or resorting to additional zero knowledge proofs and can be used to construct extendable threshold ring signatures in a generic way.

Implementation We provide an implementation that demonstrates the concrete efficiency of our schemes. The benchmarks place our constructions firmly within the realm of practicality: an extendable ring signature for a ring with 2048 members can be created in 0.24s.

1.2 Related Work

Ring signatures were first introduced by Rivest, Shamir, and Tauman in [27] as a mechanism to leak secrets anonymously. This initial construction was based on trapdoor permutations, but other schemes quickly followed. A threshold version of their scheme was proposed the following year by Bresson *et al.* [8], together with a revised security analysis for the original scheme. By using RSA accumulators and the Fiat-Shamir transform, a ring signature scheme with signature sizes independent of the ring size was later constructed by Dodis *et al.* [13]. (A similar scheme in the threshold setting was described by Munch-Hansen *et al.* [23].) In addition to the hardness of integer factorization, pairing groups were used in early constructions to obtain ring signatures in the conventional [5] and identity-based [32] settings.

The first ring signature constructions were all based on the random oracle model, but alternatives proven secure in the common reference string model were later proposed [12,28], including constructions with sublinear [10] and constant signature size [7]. In the standard model, early constructions were based on 2-round public coin witness-indistinguishable protocols [1], but more recent constructions rely on non-interactive zero-knowledge proofs [6,21]. The transition to post-quantum cryptography has also motivated revisiting many cryptographic primitives, and ring signatures are no different. As with other post-quantum signature schemes, the most popular underlying assumption is hard problems on lattices [14,20,19,4].

Threshold ring signature schemes come in many flavors, with many constructions based on RSA and bilinear maps and security based on number-theoretic assumptions [18,30,31]; and post-quantum schemes based both on lattices [3] and coding theory [22]. The post-quantum schemes have traditionally relied on the Fiat-Shamir transform, the quantum security of which is not fully determined. Recent work in threshold ring signatures has provided both improved security definitions [23] and constructions based on the quantum-safe Unruh's transform [16].

2 Background and Preliminaries

2.1 Notation

We denote the set of natural numbers by $\mathbb{N}$ and let the computational security parameter of our schemes to be $\lambda \in \mathbb{N}$. We say that a function is *negligible* (in λ), and we denote it by $\texttt{negl}$, if $\texttt{negl}(\lambda) = \Omega(\lambda^{-c})$ for any fixed constant $c > 1$. We also say that a probability is *overwhelming* (in λ) if it is greater than or equal to $1 - \texttt{negl}$. Given two values $a < b$, we denote the list of integer numbers between a and b as $[a, \ldots, b]$. For compactness, when $a = 1$, we simply write $[b]$ for $[1, \ldots, b]$. We denote empty strings as ϵ. Unless otherwise specified, all the algorithms defined throughout this work are assumed to be probabilistic Turing machines that run in polynomial time (abbreviated as PPT). When sampling the value a uniformly at random from a set X, we employ the notation $a \leftarrow_R X$. In our constructions, we denote by $\mathsf{GroupGen}(1^\lambda)$ the algorithm that, given in input the security parameter, outputs the tuple $(p, g, \mathbb{G})$, where p is a 2λ-bit prime; g is a group generator and $\mathbb{G}$ is a description of a group of order p, $\mathbb{G} = \langle g \rangle$. Through out the paper, we assume solving the Discrete Logarithm Problem in $\mathbb{G}$ is computationally hard.

2.2 Ring Signatures

Ring signatures come as a natural extension of group signature schemes. Group signatures have the drawback of requiring a trusted authority to act as a group manager. This group manager is responsible for defining the group of signers and distributing keys to them. (The group manager can then add and revoke signers over time.) The signers' keys can be used to anonymously sign messages on behalf of the entire group. Ring signatures improve on group signatures by not requiring a trusted group manager. Instead, they allow signers to generate their own key pairs, and to form groups in an ad-hoc way.

Syntax A ring signature scheme is defined as a tuple of four probabilistic polynomial time algorithms $\mathbf{RS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$, where the public parameters pp produced by Setup are implicitly available to KeyGen, Sign and Verify:

$\mathsf{Setup}(1^\lambda) \to \texttt{pp}$**:** Takes a security parameter λ and outputs a set of public parameters $\texttt{pp}$. The public parameters are implicitly input to all subsequent algorithms.

$\mathsf{KeyGen}() \to (\texttt{pk}, \texttt{sk})$**:** Produces a key pair $(\texttt{pk}, \texttt{sk})$.

$\mathsf{Sign}(\mu, \{\mathtt{pk}_j\}_{j\in\mathcal{R}}, \mathtt{sk}_i) \to \sigma$**:** Takes a message $\mu \in \{0,1\}^*$ to be signed, the set of public keys of the users within the ring of identifiers $\mathcal{R}$, and the secret key $\mathtt{sk}_i$ of the signer $i \in \mathcal{R}$ (i.e., the signer's public key must appear in the set $\{\mathtt{pk}_j\}_{j\in\mathcal{R}}$). Outputs a signature σ.

$\mathsf{Verify}(\mu, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \sigma) \to \mathtt{accept/reject}$**:** Takes a message, a set of public keys of the users within a ring, and a signature σ. Outputs `accept` or `reject`, reflecting the validity of the signature σ on the message μ with respect to the ring $\mathcal{R}$.

Naturally, a ring signature scheme should satisfy correctness, meaning that any signature generated by Sign should verify (against the signed message and the original ring). A *secure* ring signature scheme RS must additionally satisfy (a) unforgeability, meaning that no adversary should be able to produce a verifying signature without knowledge of at least one signing key corresponding to a public verification key in the ring, and (b) anonymity, meaning that no adversary should be able to tell from a signature which ring member produced it. We refer to prior work for the formal definitions of a ring signature scheme [8,13,17].

2.3 Threshold Ring Signatures

Threshold ring signatures are similar to ring signatures, but instead of allowing any one signer to anonymize themselves among a ring of signers, a threshold ring signature scheme allows any t signers to anonymize themselves among a ring of signers $\mathcal{R}$ where $t \leq |\mathcal{R}|$. A verifier can then check that at least t signers in the ring $\mathcal{R}$ signed the same message. Note that a ring signature scheme can be viewed as a threshold ring signature scheme with $t = 1$.

Syntax There are many different ways to formalize the threshold ring signature syntax, which force varying degrees of interaction between the t signers. A non-interactive threshold ring signature scheme is defined as a tuple of five probabilistic polynomial time algorithms (Setup, KeyGen, Sign, Combisign, Verify). The algorithms Setup, KeyGen, Sign and Verify are syntactically the same as in a ring signature scheme, with the exceptions that (1) Sign now outputs a *partial* signature σ_i for signer i, and (2) Verify now additionally takes the threshold t as input. The algorithm Combisign, described below, combines t partial signatures into a single threshold signature. It may be run by any third party, as it does not require any signers' secrets.

$\mathsf{Combisign}(\{\sigma_i\}_{i\in\mathcal{S}\subseteq\mathcal{R}}) \to \sigma$**:** Takes partial signatures $\{\sigma_i\}_{i\in\mathcal{S}}$ from $|\mathcal{S}| = t$ signers, and outputs a combined signature σ.

There are also interactive threshold ring signature schemes. In this case Sign (which in this case also subsumes Combisign) is an interactive protocol run between the signers, which implicitly requires the signers to be aware of one another's identities.

Finally, there is a solution in between, where one signer produces the initial signature, and then the remaining signers pass the signature around, and each "joins" the signature before passing it on. In such a syntax, each signer must only receive (at most) one message from one other signer, and send (at most) one message to one other signer. Instead of Combisign, in such a syntax we have a Join algorithm, described below.

$\mathsf{Join}(\mu, \{\mathtt{pk}_j\}_{j\in\mathcal{R}}, \mathtt{sk}, \sigma) \to \sigma'$**:** Takes a message μ, a set of public keys $\{\mathtt{pk}_j\}_{j\in\mathcal{R}}$, which includes the public key of the new signer, the new signer's secret key $\mathtt{sk}$, and a signature σ produced by a subset of $\mathcal{R}$ (with threshold level t'). Outputs a modified threshold ring signature σ' with threshold $t' + 1$.

2.4 Signatures of Knowledge

Signatures of Knowledge (SoKs) [11] generalise digital signatures by replacing the public key with an *instance*, or *statement*, in a NP language. The notion of SoKs mimics digital signatures with strong existential unforgeability: even if the adversary has seen many signatures on arbitrary messages under arbitrary statements, she cannot create a new signature (not seen before) without knowing the witness for the statement in question. Signatures of knowledge are closely related to simulation-extractable SNARKs.

We use the definitions of Signatures of Knowledge from a recent work [15] that implicitly considers only compact signatures. In the following, we will consider an efficiently decidable binary relation $\mathscr{R}$. For pairs $(\phi, w) \in \mathscr{R}$ we call ϕ the instance/statement and w the witness. Let $L_{\mathscr{R}}$ be the language consisting of statements ϕ for which there exist matching witnesses w such that $(\phi, w) \in \mathscr{R}$.

Syntax A SoK for an efficiently decidable binary relation $\mathscr{R}$ is defined as a tuple of PPT algorithms $\mathbf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SimSetup}, \mathsf{SimSign})$:

$\mathsf{Setup}(1^\lambda, \mathscr{R}) \rightarrow \mathtt{pp}$**:** Takes a security parameter λ and a binary relation $\mathscr{R}$ and returns public parameters $\mathtt{pp}$. The input $\mathtt{pp}$ is implicit to al subsequent algorithms.

$\mathsf{Sign}(\mu, \phi, w) \rightarrow \sigma$**:** Takes as input a message $\mu \in \{0,1\}^*$, a statement ϕ, and a witness w. Outputs a signature σ.

$\mathsf{Verify}(\mu, \phi, \sigma) \rightarrow \mathtt{accept/reject}$**:** Takes as input a message μ, a statement ϕ, and a signature σ. Outputs $\mathtt{accept}$ if the the signature is valid, $\mathtt{reject}$ otherwise.

$\mathsf{SimSetup}(1^\lambda, \mathscr{R}) \rightarrow (\mathtt{pp}, \mathtt{td})$**:** A simulated setup which takes as input a relation $\mathscr{R}$ and returns public parameters $\mathtt{pp}$ and a trapdoor $\mathtt{td}$.

$\mathsf{SimSign}(\mathtt{td}, \mu, \phi) \rightarrow \sigma'$**:** A simulated signing algorithm that takes as input a trapdoor $\mathtt{td}$, a message μ and a statement ϕ and returns a simulated signature σ'.

Security Model We require a scheme $\mathbf{SoK} = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{SimSetup}, \mathsf{SimSign})$ to have three properties: *correctness*, *simulatability* and *simulation extractability*. Below we give an intuition of what these properties offer. We refer the reader to Groth and Maller [15] for formal definitions.

Correctness: Informally, this implies that a signer holding a valid witness can always produce a signature that will convince the verifier.

Simulatability: This property essentially states that the verifier should not learn anything about the witness from the signature. The secrecy of the witness is modeled by the ability to simulate signatures without the witness. More precisely, we say a signature of knowledge is simulatable if an an adversary is unable to distinguish real public parameters and signatures from the ones generated by a simulator (that generates public parameters together with an associated trapdoor, and produces signatures using the trapdoor but without a witness).

Simulation Extractability: This notion guarantees that an adversary is not able to issue a new signature unless it knows a witness. This should hold even if the adversary gets to see signatures on arbitrary messages under arbitrary statements, which may include false statements. Even under this strong attack model, we require that whenever the adversary outputs a valid signature not queried before, it is possible to extract a witness for the signature.

3 Extendable Ring Signatures

Ring signatures enable a signer to generate a signature while hiding her identity within a ring of potential signers. Even though the ring of potential signers $\mathcal{R}$ can be arbitrary[4] — realizing ad-hoc anonymity sets — existing constructions do not let a third party increase the size of $\mathcal{R}$ after the signature is produced. Once a signature is generated, it is not possible to "extend" it to a larger anonymity set; in other words, ring signatures do not allow one to modify a signature and obtain a new signature for the same message but with a wider set of potential signers. Our notion of extendability aims to allow exactly this, while preserving signer anonymity.

We introduce the notion of extendability for ring signature schemes (3.1), a security model for anonymous extendability (Section 3.2) and present a realization based on standard assumptions (Section 3.3).

3.1 Syntax

An extendable ring signature scheme (ERS) is a ring signature scheme that has an additional algorithm, Extend, that allows any third party to enlarge the ring of potential signers of a given signature:

$\mathsf{Extend}(\mu, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \sigma, \{\mathtt{pk}_j\}_{j\in\mathcal{R}'}) \to \sigma'$**:** Takes a message, a set of public keys (indexed by the ring $\mathcal{R}$), a signature σ, and a second ring of public keys (indexed by $\mathcal{R}'$). It outputs a modified signature σ' which verifies under $\mathcal{R} \cup \mathcal{R}'$.

Remark 1. Consider an ERS scheme where Extend can be repeatedly applied to extend a signature a polynomial number of times. In this case, we can have a very simple instantiation where Sign always produces a signature for the singleton ring $\{\mathtt{pk}\}$ containing only the signer's public key $\mathtt{pk}$, and Extend is called only on singleton extension rings, i.e., $|\mathcal{R}'| = 1$. A signature for the singleton ring can be extended to any ring by having the signer iteratively apply Extend with a single additional public key.

For the following definitions, we use ladders of rings, i.e., tuples $\mathtt{lad} = (i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)})$, where i is a signer identity, and the rings $\mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)}$ are all sets of signer identifiers. In addition, we make use of an algorithm $\mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad})$, that we describe in Figure 1. As the name suggests, this algorithm processes a ladder $\mathtt{lad}$ on a given message μ using keys from $\mathsf{L}_{\mathsf{keys}}$ (the list of generated keys). $\mathsf{Process}$ signs μ using $\mathtt{sk}_i$ under the ring $\mathcal{R}^{(1)}$, and extends the signature to all the subsequent rings (using keys stored in the list $\mathsf{L}_{\mathsf{keys}}$). $\mathsf{Process}$ returns an extendable ring signature σ, which is the output of the last operation.
For correctness, we require that any — possibly extended — signature σ output by $\mathsf{Process}$ verifies for the given message, under the final ring $\mathcal{R}^{(l)}$.

Definition 1 (Correctness for ERS). *An extendable ring signature scheme ERS is said to be correct if, for all security parameters $\lambda \in \mathbb{N}$, for any message $\mu \in \{0,1\}^*$, for any ladder $\mathtt{lad} = (i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \cdots, \mathcal{R}^{(l)})$ where $i \in \mathcal{R}^{(1)}$ and $l > 0$, it must hold that:*

$$\Pr\left[\begin{array}{r|l} \mathbf{ERS}.\mathsf{Verify}(\mu, \{\mathtt{pk}_j\}_{j\in\mathcal{R}}, \sigma) & \mathcal{R} = \mathcal{R}^{(1)} \cup \cdots \cup \mathcal{R}^{(l)} \\ = \mathtt{accept}\ \mathit{OR}\ \sigma = \bot & \mathtt{pp} \leftarrow \mathbf{ERS}.\mathsf{Setup}(1^\lambda) \\ & \mathsf{L}_{\mathsf{keys}} \leftarrow \{(\mathtt{pk}_j, \mathtt{sk}_j) \leftarrow \mathbf{ERS}.\mathsf{KeyGen}()\}_{j\in\mathcal{R}} \\ & \sigma \leftarrow \mathbf{ERS}.\mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}) \end{array}\right] = 1$$

[4] The ring $\mathcal{R}$ should of course contain the signer's identity.

```
$\mathbf{ERS}.\mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad})$
1 : Parse $\mathtt{lad}$ as $(i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)})$
2 : if $i \notin \mathcal{R}^{(1)}$ return $\bot$    // make sure all public keys are in $\mathsf{L}_{\mathsf{keys}}$
3 : for $j \in \mathcal{R}^{(1)} \cup \cdots \cup \mathcal{R}^{(l)}$ : if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$ return $\bot$
// make sure the signer's secret key is available in $\mathsf{L}_{\mathsf{keys}}$
4 : if $\mathtt{sk}_i = \bot$ return $\bot$    // make sure $\mathtt{sk}_i$ is not corrupted
// process the instructions in the ladder
5 : $\sigma^{(1)} \leftarrow \mathbf{ERS}.\mathsf{Sign}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(1)}}, \mathtt{sk}_i)$
6 : for $l' \in [2, \ldots, l]$
7 :     $\sigma^{(l')} \leftarrow \mathbf{ERS}.\mathsf{Extend}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(l'-1)}}, \sigma^{(l'-1)}, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(l')}})$
8 : $\sigma \leftarrow \sigma^{(l)}$
9 : return $\sigma$
```

Fig. 1: The Process algorithm for extendable ring signatures.

3.2 Security Model

Definition 2 (ERS). *An extendable ring signature scheme is secure if it satisfies correctness (Definition 1), unforgeability (Definition 3), anonymity (Definition 4), and some notion of anonymous extendability (described below).*

Unforgeability Extendable ring signatures inherit their unforgeability requirement from regular ring signatures: no adversary should be able to produce a signature unless they know at least one secret key belonging to a party in the ring. Notably, the unforgeability experiment for ERS (cmEUF, detailed in Figure 2) needs to take into account that the adversary can arbitrarily expand the ring associated to a signature. To rule out trivial attacks derived with this strategy, the adversary does not break unforgeability if the candidate forgery could be generated by extending the outcome of a signing query (line 5 in $\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathbf{ERS}}(\lambda)$). Additionally, to account for the key duplication attack (where an adversary registers an existing public key to a new identity), instead of simply checking if the identities in the output ring are among the corrupted ones, the experiment checks if the *public keys* belonging to the parties involved in the adversary's output ring are among the corrupted ones (line 7, Figure 2).

Definition 3 (Unforgeability for ERS). *An extendable ring signature scheme ERS is said to be unforgeable if for all PPT adversaries $\mathcal{A}$ taking part in the unforgeability experiment (*cmEUF *in Figure 2), the success probability is negligible, i.e.:*

$$\Pr\left[\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \mathtt{negl}.$$

Anonymous Extendability For extendability, we consider security notions related to anonymity (thus the name anonymous extendability). We define an experiment that is general enough to support three different flavors of anonymous extendability: the standard *anonymity* notion, where no extension happens; *weak extendability*, where it is not possible to identify the original subring

```
$\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathbf{ERS}}(\lambda)$
1 :  $\mathsf{L}_{\mathsf{keys}}, \mathsf{L}_{\mathsf{corr}}, \mathsf{L}_{\mathsf{sign}} \leftarrow \emptyset$
2 :  $\mathtt{pp} \leftarrow \mathbf{ERS}.\mathsf{Setup}(1^\lambda)$
3 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
4 :  $(\mu^*, \mathcal{R}^*, \sigma^*) \leftarrow \mathcal{A}^O(\mathtt{pp})$
// rule out trivial wins due to ring expansion
5 :  if $\exists\ (\mu^*, \mathcal{R}, \cdot) \in \mathsf{L}_{\mathsf{sign}}$ $s.t.$
         $\{\mathtt{pk}_j\}_{j\in\mathcal{R}} \subseteq \{\mathtt{pk}_j\}_{j\in\mathcal{R}^*}$
6 :    return lose
// rule out trivial wins due to key duplication
7 :  if $\{\mathtt{pk}_j\}_{j\in\mathcal{R}^*} \cap \{\mathtt{pk}_j\}_{j\in\mathsf{L}_{\mathsf{corr}}} \neq \emptyset$
8 :    return lose
9 :  if $\mathsf{Verify}(\mu^*, \{\mathtt{pk}_j\}_{j\in\mathcal{R}^*}, \sigma^*) = \mathtt{reject}$
10 :   return lose
11 : return win
```

```
$O\mathsf{Corrupt}(i)$
1 :  if $(i, \mathtt{pk}_i, \mathtt{sk}_i) \in \mathsf{L}_{\mathsf{keys}}$ and $\mathtt{sk}_i \neq \bot$
2 :    $\mathsf{L}_{\mathsf{corr}} \leftarrow \mathsf{L}_{\mathsf{corr}} \cup \{i\}$
3 :    return $(\mathtt{pk}_i, \mathtt{sk}_i)$
4 :  return $\bot$  // if $i$ has not been initialized.
```

```
$O\mathsf{KeyGen}(i, \mathtt{pk})$
// standard key generation for a new identifier $i$
1 :  if $\mathtt{pk} = \bot$
2 :    $(\mathtt{pk}_i, \mathtt{sk}_i) \leftarrow \mathbf{ERS}.\mathsf{KeyGen}()$
3 :    $\mathsf{L}_{\mathsf{keys}} \leftarrow \mathsf{L}_{\mathsf{keys}} \cup \{(i, \mathtt{pk}_i, \mathtt{sk}_i)\}$
4 :  else
// $\mathcal{A}$ over-writes an identifier with malicious keys
5 :    $\mathsf{L}_{\mathsf{corr}} \leftarrow \mathsf{L}_{\mathsf{corr}} \cup \{i\}$
6 :    $\mathtt{pk}_i \leftarrow \mathtt{pk}$
7 :    $\mathsf{L}_{\mathsf{keys}} \leftarrow \mathsf{L}_{\mathsf{keys}} \cup \{(i, \mathtt{pk}_i, \bot)\}$
8 :  return $\mathtt{pk}_i$
```

```
$O\mathsf{Sign}(\mu, \mathcal{R}, i)$
1 :  if $(i \in \mathsf{L}_{\mathsf{corr}} \vee i \notin \mathcal{R})$ :   return $\bot$
// check that all keys in the query are initialized
2 :  for all $j \in \mathcal{R}$
3 :    if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$
4 :      return $\bot$
5 :  $\sigma \leftarrow \mathbf{ERS}.\mathsf{Sign}(\mu, \{\mathtt{pk}_j\}_{j\in\mathcal{R}}, \mathtt{sk}_i)$
6 :  $\mathsf{L}_{\mathsf{sign}} \leftarrow \mathsf{L}_{\mathsf{sign}} \cup \{(\mu, \mathcal{R}, i)\}$
7 :  return $\sigma$
```

Fig. 2: Existential Unforgeability under Chosen Message Attack for (Extendable) Ring Signatures (security experiment and oracles). Our key generation oracle allows $\mathcal{A}$ to register signers with arbitrary public keys (i.e., it also acts as a registration oracle).

of an extended signature; and *strong extendability*, where it is not possible to tell what sequence of extensions a signature has undergone.

For standard anonymity we consider adversaries that output ladders ($\mathtt{lad}_0^*, \mathtt{lad}_1^*$ in line 5 of $\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}$ in Figure 3) each containing only one ring. To avoid making the game trivial to win, the two rings need to be identical (line 7 of Chal_b). Moreover since the extension algorithm is never called ($l_0 = l_1 = 1$ in this case), it is clear that — with this restriction on the adversary's input to the challenger — our ANEXT experiment is the same as the standard anonymity one.

Definition 4 (Anonymity for ERS). *An extendable ring signature scheme is said to be anonymous if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (*ANEXT *in Figure 3) and submitting to the challenger ladders of the type $\mathtt{lad}_0^* = (i_0, \mathcal{R}), \mathtt{lad}_1^* = (i_1, \mathcal{R})$, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing. i.e.,:*

$$\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \tfrac{1}{2} + \mathtt{negl}.$$

For weak anonymous extendability we require the adversary to submit ladders of rings to the challenger such that each ladder only contains two rings. In other words, the adversary chooses two (possibly distinct) two-ring ladders $(i_0, \mathcal{R}_0^{(1)}, \mathcal{R}_0^{(2)})$ and $(i_1, \mathcal{R}_1^{(1)}, \mathcal{R}_1^{(2)})$ such that $\mathcal{R}_0^{(1)} \cup \mathcal{R}_0^{(2)} = \mathcal{R}_1^{(1)} \cup$

```
$\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda)$
1:  $b \leftarrow_R \{0,1\}$
2:  $\mathsf{L}_{\mathsf{keys}}, \mathsf{L}_{\mathsf{corr}}, \mathsf{L}_{\mathsf{sign}} \leftarrow \varnothing$
3:  $\mathsf{pp} \leftarrow \mathbf{ERS}.\mathsf{Setup}(1^\lambda)$
    // handle of oracles, for compact notation
4:  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
5:  $(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*) \leftarrow \mathcal{A}^O(\mathsf{pp})$
6:  $\bar{\sigma} \leftarrow \mathsf{Chal}_b(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*)$
7:  $b^* \leftarrow \mathcal{A}^O(\bar{\sigma})$
    // make sure $\mathcal{A}$ did not corrupt the challenge
    // keys during the second query phase
8:  if $i_0 \in \mathsf{L}_{\mathsf{corr}} \vee i_1 \in \mathsf{L}_{\mathsf{corr}}$
9:      return $\mathtt{lose}$
10: if $b^* \neq b$
11:     return $\mathtt{lose}$
12: return $\mathtt{win}$
```

```
$\mathsf{Chal}_b(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*)$
1:  parse $\mathtt{lad}_0^* = (i_0, \mathcal{R}_0^{(1)}, \dots, \mathcal{R}_0^{(l_0)})$
2:  parse $\mathtt{lad}_1^* = (i_1, \mathcal{R}_1^{(1)}, \dots, \mathcal{R}_1^{(l_1)})$
    // challenge signing keys should not be corrupted
3:  if $i_0, i_1 \in \mathsf{L}_{\mathsf{corr}}$ return $\perp$
    // sign and extend following the instructions
    //  in both ladders
4:  $\sigma_0 \leftarrow \mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}_0^*)$
5:  $\sigma_1 \leftarrow \mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}_1^*)$
6:  if $\sigma_0 = \perp$ or $\sigma_1 = \perp$ return $\perp$
    // check that ladders end with the same ring
7:  if $\mathcal{R}_0^{(1)} \cup \dots \cup \mathcal{R}_0^{(l_0)} \neq \mathcal{R}_1^{(1)} \cup \dots \cup \mathcal{R}_1^{(l_1)}$
8:      return $\perp$
    // set the challenge signature according to $b$
9:  $\bar{\sigma} \leftarrow \sigma_b$
10: return $\bar{\sigma}$
```

Fig. 3: Anonymity and Anonymous Extendability for Extendable Ring Signatures. The oracles $O\mathsf{Sign}$, $O\mathsf{KeyGen}$ and $O\mathsf{Corrupt}$ are defined in Figure 2.

$\mathcal{R}_1^{(2)} = \mathcal{R}$. The adversary breaks weak anonymous extendability if, given an extended signature for the super-ring $\mathcal{R}$, it is able to identify (with better accuracy than random guessing) which ladder was used.

Definition 5 (Weak Anonymous Extendability for ERS). *An extendable ring signature scheme ERS is said to be weakly anonymous extendable if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (*ANEXT *in Figure 3) and submitting to the challenger ladders of the type* $\mathtt{lad}_0^* = (i_0, \mathcal{R}_0^{(1)}, \mathcal{R}_0^{(2)}), \mathtt{lad}_1^* = (i_1, \mathcal{R}_1^{(1)}, \mathcal{R}_1^{(2)})$*, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.:*

$$\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \tfrac{1}{2} + \mathtt{negl}.$$

Remark 2. Weak anonymous extendability requires that an adversary not be able to distinguish between two extended signatures, as long as (a) they were extended to the same ring, and (b) they were both only extended once. One might also consider a form of weak extendability where, instead of limiting the signatures to a single extension, we let them be extended any number of times, as long as their numbers of extensions are the same.

Finally, for strong anonymous extendability we consider adversaries that output any other type of ladders that culminate in the same ring. In particular, we could have $l_0 \neq l_1$. Notice that strong anonymous extendability implies both weak anonymous extendability and anonymity.

Definition 6 (Strong Anonymous Extendability for ERS). *An extendable ring signature scheme is said to be strongly anonymous extendable if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (Figure 3), it holds that:*

$$\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \ \frac{1}{2} + \mathtt{negl}.$$

We remark that strong extendability implies that the act of extending a ring signature is seamless, i.e., an adversary is not able to distinguish between a fresh ring signature (returned by Sign), and an extension of it (returned by Extend). This is covered in the strong extendability game for $l_0 = 1$ and $l_1 > 1$.

3.3 ERS from Signatures of Knowledge and Discrete Log

In what follows, we exhibit an efficient realization of extendable ring signature scheme from prime order groups and signatures of knowledge.

Our Construction in a Nutshell The setup generates a prime-order group $\mathbb{G} = \langle g \rangle$, a random group element $H \leftarrow_R \mathbb{G}$ and public parameters for a SoK scheme for the relation

$$\mathscr{R}_{\mathbb{G}}(\phi = (h, \mathtt{pk}), w = x) = \{g^x = h \vee g^x = \mathtt{pk}\}.$$

Intuitively, $\mathscr{R}_{\mathbb{G}}$ requires that the witness be either the discrete log of $\mathtt{pk}$ (which is the corresponding secret key), or the element h. The signing procedure simply samples a random value $\mathtt{td} \leftarrow_R \mathbb{Z}_p$, creates an element $h := H \cdot g^{-\mathtt{td}}$ (which implies that $h \cdot g^{\mathtt{td}} = H$), and computes a signature of knowledge π for $(h, \mathtt{pk})$ using her secret key $\mathtt{sk}$. The signature σ contains $\mathtt{td}$, and a set $P = \{(h, \mathtt{pk}, \pi)\}$. Extending works essentially like signing, except that the extender uses the other kind of witness. Concretely, the extender samples a new $\mathtt{td}'$, computes $h' = g^{\mathtt{td}'}$ and a signature of knowledge π' for the $\mathtt{pk}'$ she wishes to add to the ring, using $\mathtt{td}'$ as the witness. The tuple $(h', \mathtt{pk}', \pi')$ is added to P, and $\mathtt{td}$ is replaced by $\mathtt{td} - \mathtt{td}'$. The verification checks that $H = g^{\mathtt{td}} \cdot \prod h_i$ for all h_i present in P, and that all π_i verify. This ensures that at least one of the π_i was produced using $\mathtt{sk}_i$ as a witness (otherwise we would be able to extract $dlog(H)$). A formal description of this construction is given in Figure 4.

Theorem 1. *Assuming that* **SoK** *is a secure signature of knowledge scheme, and that the discrete log problem is hard in the group* $\mathbb{G}$*, then the scheme* **ERS** $=$ (Setup, KeyGen, Sign, Verify, Extend) *described in Figure 4 is an extendable ring signature scheme that satisfies* correctness *(Definition 1),* unforgeability *(Definition 3), and* strong anonymous extendability *(Definition 6).*

Proof. The correctness of the construction follows by inspection.

Unforgeability To prove unforgeability, we present a sequence of hybrid games at the end of which the reduction is able to extract a solution to a discrete logarithm challenge from $\mathcal{A}$'s forgery with high-enough probability. Essentially this involves: embedding a discrete logarithm into H; moving to the simulatable setup for the SoK; replacing all signatures of knowledge with simulated ones; and using the witness extracted from π^* to learn $dlog(H)$.

In more detail, following the statement of Definition 3, we want to prove that an adversary $\mathcal{A}$ can successfully forge a signature only with negligible probability. For the sake of contradiction, assume that $\mathcal{A}$ wins the unforgeability game with non-negligible probability. We want to exhibit a reduction $\mathcal{B}$ that interacts with $\mathcal{A}$ — playing the role of the unforgeability challenger — and extracts from $\mathcal{A}$'s forgery a solution to a discrete log challenge. We describe a sequence of hybrids in each of which the behavior of $\mathcal{B}$ changes in ways that $\mathcal{A}$ should not be able to detect. In the final hybrid, $\mathcal{B}$ can use $\mathcal{A}$ to solve a discrete log challenge.

```
$\mathbf{ERS}.\mathsf{Setup}(1^\lambda) \mapsto \mathtt{pp}$
1 : $(p, g, \mathbb{G}) \leftarrow \mathsf{GroupGen}(1^\lambda)$
2 : $\mathbf{SoK}.\mathtt{pp} \leftarrow \mathbf{SoK}.\mathsf{Setup}(1^\lambda, \mathscr{R}_\mathbb{G})$
3 : $H \leftarrow_R \mathbb{G}$
4 : $\mathbf{return}\ \mathtt{pp} := (\mathbf{SoK}.\mathtt{pp}, g, H)$

$\mathbf{ERS}.\mathsf{KeyGen}() \mapsto (\mathtt{pk}, \mathtt{sk})$
1 : $\mathtt{sk} \leftarrow \mathbb{Z}_p$
2 : $\mathtt{pk} := g^{\mathtt{sk}}$
3 : $\mathbf{return}\ (\mathtt{sk}, \mathtt{pk})$

$\mathbf{ERS}.\mathsf{Sign}(\mu, \mathtt{sk}) \mapsto \sigma$
1 : $\mathtt{td} \leftarrow_R \mathbb{Z}_p$
2 : $h := H \cdot g^{-\mathtt{td}}$
// signer does not know $dlog(h)$
// compute the signature
3 : $\mathtt{nonce} \leftarrow_R \{0,1\}^\lambda$
4 : $\phi := (h, \mathtt{pk})$
5 : $w := \mathtt{sk}$
6 : $\pi \leftarrow \mathbf{SoK}.\mathsf{Sign}((\mathtt{nonce}, \mu), \mathscr{R}, \phi, w)$
7 : $P := \{(h, \mathtt{pk}, \pi)\}$
8 : $\mathbf{return}\ \sigma := (\mathtt{nonce}, \mathtt{td}, P)$
```

```
$\mathbf{ERS}.\mathsf{Verify}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}}, \sigma) \mapsto \mathtt{accept/reject}$
1 : $\mathbf{parse}\ \sigma = (\mathtt{nonce}, \mathtt{td}, P = \{(h_i, \mathtt{pk}_i, \pi_i)\}_{i \in \mathcal{R}'})$
2 : $\mathbf{if}\ H \neq g^{\mathtt{td}} \cdot \prod_{i \in \mathcal{R}'} h_i : \mathbf{return}\ \mathtt{reject}$
3 : $\mathbf{if}\ \{\mathtt{pk}_j\}_{j \in \mathcal{R}} \neq \{\mathtt{pk}_i\}_{i \in \mathcal{R}'} :$
        $\mathbf{return}\ \mathtt{reject}$
4 : $\mathbf{for}\ i \in \mathcal{R}' :$
        $\phi_i := (h_i, \mathtt{pk}_i)$
        $\mathbf{if}\ \mathbf{SoK}.\mathsf{Verify}((\mathtt{nonce}, \mu), \mathscr{R}, \phi_i, \pi_i) = \mathtt{reject} :$
            $\mathbf{return}\ \mathtt{reject}$
5 : $\mathbf{return}\ \mathtt{accept}$

$\mathbf{ERS}.\mathsf{Extend}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}}, \sigma, \mathtt{pk}) \mapsto \sigma'$
1 : $\mathbf{if}\ \mathtt{pk} \in \{\mathtt{pk}_j\}_{j \in \mathcal{R}} : \mathbf{return}\ \bot$
2 : $\mathbf{parse}\ \sigma = (\mathtt{nonce}, \mathtt{td}, P = \{(h_i, \mathtt{pk}_i, \pi_i)\}_{i \in \mathcal{R}'})$
3 : $\mathtt{td}' \leftarrow_R \mathbb{Z}_p$ // pick a trapdoor
4 : $\mathtt{td} \leftarrow \mathtt{td} - \mathtt{td}' \pmod p$
5 : $h := g^{\mathtt{td}'}$ // to simulate using the trapdoor
6 : $\phi := (h, \mathtt{pk}), w := \mathtt{td}'$
7 : $\pi \leftarrow \mathbf{SoK}.\mathsf{Sign}((\mathtt{nonce}, \mu), \mathscr{R}, \phi, w)$
8 : add $(h, \mathtt{pk}, \pi)$ to $P$ // update the signature
9 : $\mathbf{return}\ \sigma' := (\mathtt{nonce}, \mathtt{td}, P)$
```

Fig. 4: Extendable Ring Signatures from Signature of Knowledge and Discrete Log. The relation used by the SoK scheme is $\mathscr{R}_\mathbb{G} = \{(\phi, w) = (h, \mathtt{pk}, x) \in \mathbb{G} \times \mathbb{G} \times \mathbb{Z}_p : g^x = h \vee g^x = \mathtt{pk}\}$.

$\mathcal{G}_0$**:** This is precisely the unforgeability game described in Figure 2.

$\mathcal{G}_1$**:** This is the same as $\mathcal{G}_0$ except that H is set to be the challenge $\mathcal{B}$ receives from its dlog challenger. $\mathcal{A}$ cannot distinguish this hybrid from the previous one, since its views in the two games are identically distributed.

$\mathcal{G}_2$**:** This is the same as $\mathcal{G}_1$ except that the real setup for the signature of knowledge, Σ.Setup, is replaced by the simulated setup. Notably, the simulated setup gives $\mathcal{B}$ a trapdoor that allows it to (a) simulate signatures without knowledge of a witness, and (b) extract a witness from any signature produced by the adversary. In this game, $\mathcal{B}$ additionally replaces all signatures of knowledge with simulated signatures. In particular, this means that none of the signatures of knowledge depend on any honest party secret keys.

If $\mathcal{A}$ can distinguish this game from the previous one, $\mathcal{B}$ can use $\mathcal{A}$ to break the simulatability property of the signature of knowledge.

$\mathcal{G}_3$**:** This is the same as $\mathcal{G}_2$ except that, when receiving a signing oracle query $(\mu, i, \mathcal{R})$, $\mathcal{B}$ randomly picks the index $j \in \mathcal{R}$ such that it doesn't know the discrete log of h_j (instead of using i as instructed by $\mathcal{A}$ in the signing oracle query). Since all signatures of knowledge are already simulated (and thus no witnesses — in the form of either discrete logs or secret keys — are used), $\mathcal{A}$ cannot distinguish this hybrid from the previous one since its views in the two games are identically distributed.

$\mathcal{G}_4$**:** This is the same as $\mathcal{G}_3$ except that, if the candidate forgery returned by the adversary contains any signatures of knowledge on statements already contained in simulated signatures (returned in response to signing oracle queries) *where $\mathcal{B}$ does not know the discrete log of h_i*, $\mathcal{B}$ aborts.
With polynomial probability, $\mathcal{B}$ does not abort in this game. This is true since, for any signing oracle query $(\mu, i, \mathcal{R})$, the candidate forgery cannot be on (a) the same message and (b) on a superset of $\mathcal{R}$; so, from any simulated signature, only a subset of statements can be included. Since every simulated signature will be on a different nonce with overwhelming probability, only statements from a single simulated signature will be included. Exactly one random h_i in a given simulated signature has a discrete log not known to $\mathcal{B}$, and with probability at least $\frac{1}{|\mathcal{R}|}$, that one is not included in the candidate forgery.

$\mathcal{G}_5$**:** This is the same as $\mathcal{G}_4$ except that, if the candidate forgery returned by the adversary contains only signatures of knowledge on statements already contained in simulated signatures, $\mathcal{B}$ aborts.
If $\mathcal{B}$ aborts in this game but not in the previous one, $\mathcal{A}$ can be used to solve the discrete log problem. Since $\mathcal{B}$ knows the discrete log of each h_i used in the candidate forgery, if the product of the h_is times $g^{\mathtt{td}}$ (for a known $\mathtt{td}$) gives H, $\mathcal{B}$ has learned the discrete log of H.

$\mathcal{G}_6$**:** This is the same as $\mathcal{G}_5$ except that, for a randomly-chosen i^*, when prompted to generate a key pair for i^*, $\mathcal{B}$ returns $\mathtt{pk}_{i^*} := H^e$ for a random exponent $e \leftarrow \mathbb{Z}_p$ (no matching secret key is generated). If $\mathcal{A}$ submits a corruption query for i^*, $\mathcal{B}$ aborts. When $\mathcal{A}$ outputs a valid forgery $(\mu^*, \mathcal{R}^*, \sigma^*)$, $\mathcal{B}$ checks that $i^* \in \mathcal{R}$; otherwise it aborts.
In order for $\mathcal{A}$ to win, there must exist at least one uncorrupted party in $\mathcal{R}^*$; so, $\mathcal{B}$ does not abort in this game with probability at least $1/q_{\mathsf{KG}}$, where q_{KG} is the number of $\mathcal{A}$'s queries to the key generation oracle.

$\mathcal{G}_7$**:** This is the same as $\mathcal{G}_5$ except that $\mathcal{B}$ calls the extractor $\varepsilon_{\mathcal{A}}$ for SoK to extract witnesses w_i from each signature of knowledge π_i in σ^* (which are not simulated signatures).
If extraction fails from any non-simulated signature, $\mathcal{B}$ aborts. If $\mathcal{B}$ aborts with non-negligible probability here, $\mathcal{A}$ can be used to break the simulation extractability of the signature of knowledge scheme.
If π_{i^*} is simulated, $\mathcal{B}$ aborts. $\mathcal{B}$ aborts here with probability at most $\frac{|\mathcal{R}|-1}{\mathcal{R}}$, since — after $\mathcal{G}_4$ — we are guaranteed that one of the signatures will not be simulated.
If all the extracted witnesses w_i are such that $g^{w_i} = h_i$, $\mathcal{B}$ computes the discrete log of H as the sum of these witnesses (as well as the discrete logs of the h_i's from simulated signatures). Otherwise, if the witness w_{i^*} is such that $\mathtt{pk}_{i^*} = g^{w_{i^*}}$, $\mathcal{B}$ computes the discrete log of H as $\frac{w_{i^*}}{e}$ mod p (e is the random exponent generated for i^*).
Otherwise, $\mathcal{B}$ aborts. $\mathcal{B}$ aborts here with probability at most $\frac{|\mathcal{R}|-1}{\mathcal{R}}$, since at least one of the witnesses w_i is such that $\mathtt{pk}_i = g^{w_i}$ (if not all of them are of the other form).

Anonymous Extendability To prove the strong anonymous extendability of our construction it suffices to show that if an adversary $\mathcal{A}$ can successfully break anonymous extendability, we can build a reduction $\mathcal{B}$ that breaks the security of the signature of knowledge. Imagine that $\mathcal{B}$, playing the role of the challenger, runs the simulated setup for the signature of knowledge, instead of the real setup. This gives $\mathcal{B}$ a trapdoor that allows it to simulate signatures without knowledge of a witness. $\mathcal{B}$ uses this trapdoor to simulate all signatures of knowledge in response to signing queries from $\mathcal{A}$. $\mathcal{B}$ generates the challenge signature with no reference to the ladders. It simply chooses $\mathtt{td}$

at random, generates the h_i's as random values such that $g^{\mathtt{td}} \cdot \prod h_i = H$, and uses the trapdoor to simulate all signatures of knowledge. If $\mathcal{A}$ can distinguish $\mathcal{B}$ from an honest challenger, $\mathcal{B}$ can use $\mathcal{A}$ to break the simulatability property of the signature of knowledge. If $\mathcal{A}$ cannot distinguish $\mathcal{B}$ from an honest challenger, since $\mathcal{B}$'s behavior does not depend on choice of b, $\mathcal{A}$ cannot possibly win the anonymous extendability game with probability non-negligibly more than half. $\square$

4 Same-Message Linkable Extendable Ring Signatures

A same-message linkable ring signature scheme (SMLRS) is a ring signature scheme that additionally allows any third party to publicly identify (link) whether two signatures were generated by the same signer for the same message. This means that if the same party signs the same message twice, even for different rings, the two signatures can be linked by any third party.

The main motivation for same-message linkable ring signatures is that they leads to a very intuitive and natural construction of *threshold* ring signatures: a threshold ring signature can be built as a simple concatenation of t same-message linkable ring signatures. Furthermore, SMLRS can be used to build other schemes with varying degrees of linkability, in generic ways. For example, *un*linkable (ring) signatures can be realized from a SMLRS scheme simply by requiring each signer to include a random nonce in their message. Standard *linkable* (ring) signatures — where two signatures from the same signer are linkable no matter which message they sign — can be instantiated from a SMLRS scheme by requiring the signer to always sign a fixed value (e.g. $\perp$) in addition to the message μ.

In what follows, we introduce the notion of *extendable* same-message linkable ring signatures (SMLERS). We give a security model for this new primitive, and describe an instantiation that builds on our ERS construction from Section 3.3.

4.1 Syntax

A same-message linkable extendable ring signature scheme is a tuple of six algorithms **SMLERS** = $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Extend}, \mathsf{Link})$. The first five algorithms are inherited from extendable ring signatures. The Link algorithm (described below) allows any verifier to determine whether two signatures on a particular message were produced by the same signer.

$\mathsf{Link}(\mu, (\sigma_0, \{\mathtt{pk}_j\}_{j\in\mathcal{R}_0}), (\sigma_1, \{\mathtt{pk}_j\}_{j\in\mathcal{R}_1})) \to \{\mathtt{linked}, \mathtt{unlinked}\}$: An algorithm that takes a message μ, two signatures (σ_0, σ_1) and two sets of public keys belonging to members of the rings $\mathcal{R}_0, \mathcal{R}_1$. It outputs $\mathtt{linked}$ if σ_0 and σ_1 were produced by the same signer, and $\mathtt{unlinked}$ otherwise.

We remark that Link does not necessarily reveal the identity of the common signer if signatures are linked. Next we discuss correctness for extendable same-message linkable ring signature schemes, which encompasses two statements: *extended signatures verify*, which is inherited from correctness for extendable ring signatures (Definition 1); and *extended signatures from different signers are unlinked*, which we formalize in the following definition.

Definition 7 (Cross-Signer Correctness for SMLERS). *For all security parameters $\lambda \in \mathbb{N}$, for any message $\mu \in \{0,1\}^*$, for any two ladders* $\mathtt{lad}_0 = (i_0, \mathcal{R}_0^{(1)}, \ldots, \mathcal{R}_0^{(l_0)})$, $\mathtt{lad}_1 = (i_1, \mathcal{R}_1^{(1)}, \ldots, \mathcal{R}_1^{(l_0)})$

where $i_0 \in \mathcal{R}_0^{(1)}$, $i_1 \in \mathcal{R}_1^{(1)}$, $l_0 > 0$, $l_1 > 0$ and $i_0 \neq i_1$, it must hold that:

$$\Pr\left[\begin{array}{c|l} \mathsf{Link}(\mu, (\sigma_0, \{\mathtt{pk}_j\}_{j\in\mathcal{R}_0}), & \mathcal{R}_0 = \mathcal{R}_0^{(1)} \cup \cdots \cup \mathcal{R}_0^{(l_0)} \\ (\sigma_1, \{\mathtt{pk}_j\}_{j\in\mathcal{R}_1})) \to \mathtt{unlinked} & \mathcal{R}_1 = \mathcal{R}_0^{(1)} \cup \cdots \cup \mathcal{R}_1^{(l_1)} \\ & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ & \mathsf{L}_{\mathsf{keys}} \leftarrow \{\mathsf{KeyGen}()\}_{j\in\mathcal{R}_0\cup\mathcal{R}_1} \\ & \sigma_0 \leftarrow \mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}_0) \\ & \sigma_1 \leftarrow \mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}_1) \end{array}\right] = 1 - \mathtt{negl}$$

where Process *is the algorithm described in Figure 1 except that the ERS algorithms are replaced with the corresponding SMLERS ones.*

Remark 3. To build some intuition that may come in handy for understanding the security model, the reader might consider the following natural strategy for constructing an extendable same-message linkable ring signature scheme: ensuring that (part of) the signature is unique for every public key and message pair. In other words, the signer's public key and the signed message uniquely determine a part of the ring signature; we will refer to this part as the *linkability tag*. This tag is not modified by ring extensions and can be used to identify if two ring signatures, on the same message, were produced by the same signer simply by checking whether they share the same tag.

4.2 Security Model

Informally, a same-message linkable extendable ring signature scheme is an extendable ring signature that additionally satisfies the following properties:

Same-Message One-More Linkability: no set of $(t-1)$ corrupt signers can produce t signatures for the same message which appear pairwise unlinked. (We present this property in Definition 9).

Cross-Message Unlinkability: no adversary can determine whether two signatures for different messages were produced by the same signer. (We present this property in Definition 10).

Unframeability (optional): no adversary can produce a signature that appears linked to an honest signer's signature. (We do not require unframeability for our extendable threshold ring signature scheme, so we do not define it formally or prove that our construction meets it.) This property can be thought of as a strengthening of cross-signer correctness to account for malicious signers.

Definition 8 (Secure SMLERS). *A same-message linkable extendable ring signature scheme (SMLERS) is* secure *if it satisfies correctness, same-message one-more linkability (Definition 9, which implies unforgeability), and cross-message unlinkability (Definition 10).*

Definition 9 (Same-Message One-more Linkability for SMLERS). *A same-message linkable extendable ring signature scheme SMLERS is said to be one-more linkable if for all PPT adversaries $\mathcal{A}$ taking part in the same-message one-more linkability experiment ($\mathsf{Exp}^{\text{omlink}}_{\mathcal{A},\mathbf{SMLERS}}(\lambda)$ depicted in Figure 5), it holds that:* $\Pr[\mathsf{Exp}^{\text{omlink}}_{\mathcal{A},\mathbf{SMLERS}}(\lambda) = \mathtt{win}] \leq \mathtt{negl}$.

Definition 10 (Cross-Message Unlinkability for SMLERS). *A same-message linkable extendable ring signature scheme SMLERS is said to be cross-message unlinkable if for all PPT adversaries $\mathcal{A}$ taking part in the cross-message unlinkability experiment ($\mathsf{Exp}^{\text{cmunlink}}_{\mathcal{A},\mathbf{SMLERS}}(\lambda)$ depicted in Figure 6), it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.,:* $\Pr[\mathsf{Exp}^{\text{cmunlink}}_{\mathcal{A},\mathbf{SMLERS}}(\lambda) = \mathtt{win}] \leq \frac{1}{2} + \mathtt{negl}$.

```
$\mathsf{Exp}^{\mathrm{omlink}}_{\mathcal{A},\mathbf{SMLERS}}(\lambda)$
1 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
2 :  $\mathsf{L_{keys}}, \mathsf{L_{corr}}, \mathsf{L_{sign}} \leftarrow \varnothing$
3 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
4 :  $(\mu^*, \{(\sigma^*_k, \mathcal{R}^*_k)\}_{k\in[1\ldots,t]}) \leftarrow \mathcal{A}^O(\mathtt{pp})$
     // $\mathcal{A}$ has never seen a signature for the message and a subring of the forgery rings
5 :  if $\exists\ (\mu^*, \mathcal{R}, \cdot) \in \mathsf{L_{sign}}\ s.t.\ \mathcal{R} \subseteq \mathcal{R}^*_k$ for some $k \in [1, \ldots, t]$   return lose
     // $\mathcal{A}$ holds at most $t-1$ secret keys, among the keys identified by the forgery rings
6 :  if $|(\mathcal{R}^*_1 \cup \cdots \cup \mathcal{R}^*_t) \cap \mathsf{L_{corr}}| \geq t$   return lose
     // all the signatures in the forgery verify (for the same message)
7 :  if $\exists\ k \in [1\ldots,t]$ s.t. $\mathsf{Verify}(\mu^*, \{\mathtt{pk}_j\}_{j\in\mathcal{R}^*_k}, \sigma^*_k) = \mathtt{reject}$   return lose
     // all signatures in the forgery are unlinked (here $k, l \in [1,\ldots,t]$)
8 :  if $\exists\ k \neq l$ s.t. $\mathsf{Link}(\mu^*, (\sigma^*_k, \{\mathtt{pk}_j\}_{j\in\mathcal{R}^*_k}), (\sigma^*_l, \{\mathtt{pk}_j\}_{j\in\mathcal{R}^*_l})) = \mathtt{linked}$
9 :    return lose
10 : return win
```

Fig. 5: Security experiment for same-message one-more linkability. The signing, key generation and corruption oracles are as defined in Figure 2, except that the algorithms for ERS are replaced with the corresponding algorithms for SMLERS. We recall that the list $\mathsf{L_{sign}}$ of sign-queries contains elements of the form $(\mu, \mathcal{R}, i)$.

4.3 SMLERS from Signatures of Knowledge and Discrete Log

Our SMLERS construction builds on the ERS construction in Figure 4. Since the nuance is limited, we only briefly describe the tweaks needed to transform our ERS into an SMLERS.

First, we adopt a slightly different relation $\mathscr{R}_{\mathbf{SMLERS}}$:

$$\mathscr{R}_{\mathbf{SMLERS}}\left(\phi = (h, \mathtt{pk}, g', \tau), w = x\right) = \left\{g^x = h \vee \left(g^x = \mathtt{pk} \wedge (g')^x = \tau\right)\right\}$$

Notably, the last *AND* not only requires a signer to prove knowledge of the secret key, but it also enforces that the same secret key is used to generate the linkability tag τ. The signatures of knowledge for SMLERS are with respect to the new relation $\mathscr{R}_{\mathsf{SML}}$.

Second, we modify the Sign algorithm of our ERS in Figure 4 so that it additionally computes $g' := \mathsf{H}(\mu)$ and $\tau := (g')^{\mathsf{sk}}$ for some hash function H, and it includes the linkability tag τ as part of the signature. Finally, the algorithm Link simply compares the linkability tags in the two signatures. It returns `linked` if they are equal, and `unlinked` otherwise.

This scheme can be shown to be same-message one-more linkable (resp. cross-message unlinkable) with only minor modifications to the proof of unforgeability (resp. anonymous extendability) of the extendable ring signature scheme.

5 Extendable Threshold Ring Signatures

Like a traditional threshold ring signature scheme, an *extendable* threshold ring signature scheme enables parties to produce a signature on a message μ for a ring $\mathcal{R}$ showing that at least t of the $|\mathcal{R}|$ potential signers in the ring participated, without revealing which. An extendable threshold ring signature scheme additionally has the following properties:

```
$\mathsf{Exp}\ ^{\mathrm{cmunlink}}_{\mathcal{A},\mathbf{LRS}}(\lambda)$
1 :  $b \leftarrow_R \{0,1\}, \mathsf{L}_{\mathsf{keys}}, \mathsf{L}_{\mathsf{corr}}, \mathsf{L}_{\mathsf{sign}} \leftarrow \varnothing$
2 :  $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3 :  $O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
4 :  $(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\}) \leftarrow \mathcal{A}^O(\mathsf{pp})$
5 :  $(\bar{\sigma}_0, \bar{\sigma}_1) \leftarrow \mathsf{Chal}_b(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\})$
6 :  $b^* \leftarrow \mathcal{A}^O(\bar{\sigma}_0, \bar{\sigma}_1)$
// Rule out corruption of challenge identities
7 :  if $i_0 \in \mathsf{L}_{\mathsf{corr}} \lor i_1 \in \mathsf{L}_{\mathsf{corr}}$ return lose
// Rule out trivial attacks using Link
8 :  if $\mu_0 = \mu_1$ return lose
// Rule out trivial attacks using Link
9 :  if $(\mu_0, \cdot, i_0) \in \mathsf{L}_{\mathsf{sign}} \lor (\mu_0, \cdot, i_1) \in \mathsf{L}_{\mathsf{sign}}$
         $\lor (\mu_1, \cdot, i_0) \in \mathsf{L}_{\mathsf{sign}} \lor (\mu_1, \cdot, i_1) \in \mathsf{L}_{\mathsf{sign}}$
10 :   return lose
11 : if $b^* \neq b$   return lose
12 : return win
```

```
$\mathsf{Chal}_b(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\})$
// the challenge identities must be uncorrupted
1 :  if $i_0 \in \mathsf{L}_{\mathsf{corr}} \lor i_1 \in \mathsf{L}_{\mathsf{corr}}$
2 :    return $\perp$
// one identity needs to be in both rings
3 :  if $i_0 \notin \mathcal{R}_0 \cap \mathcal{R}_1 \lor i_1 \notin \mathcal{R}_1$
4 :    return $\perp$
// signing keys must exist
5 :  if $\nexists\ (i_0, \mathtt{pk}_{i_0}, \mathtt{sk}_{i_0}) \in \mathsf{L}_{\mathsf{keys}}$ return $\perp$
6 :  if $\nexists\ (i_1, \mathtt{pk}_{i_1}, \mathtt{sk}_{i_1}) \in \mathsf{L}_{\mathsf{keys}}$ return $\perp$
// generate a signature
7 :  $\bar{\sigma}_0 \leftarrow \mathbf{LRS}.\mathsf{Sign}(\mu_0, \{\mathtt{pk}_i\}_{i\in\mathcal{R}_0}, \mathtt{sk}_{i_0})$
// generate the second signature according to
// the experiment's bit $b$
8 :  $\bar{\sigma}_1 \leftarrow \mathbf{LRS}.\mathsf{Sign}(\mu_1, \{\mathtt{pk}_i\}_{i\in\mathcal{R}_1}, \mathtt{sk}_{i_b})$
9 :  return $(\bar{\sigma}_0, \bar{\sigma}_1)$
```

Fig. 6: Cross-message unlinkability. The signing, key generation and corruption oracles are as defined in Figure 2, except that the ERS algorithms are substituted with the respective **SMLERS** variants.

Flexibility: Given any two threshold signatures σ_0 and σ_1 that verify for the same message μ and for the same ring $\mathcal{R}$, anyone can *non-interactively* combine the signatures to obtain σ. The new signature σ is also a threshold ring signature and its threshold is equal to the total number of unique signers who contributed to at least one of the two signatures. This functionality is provided by the Combine algorithm (below).

Extendability: Given a signature σ on a message μ for the ring $\mathcal{R}$ with threshold t, anyone can *non-interactively* transform σ into a signature σ' on the same message μ with the same threshold$t$, but for a larger ring $\mathcal{R}' \supseteq \mathcal{R}$. This functionality is provided by the Extend algorithm (below).

5.1 Syntax

A non-interactive extendable threshold ring signature scheme (ETRS) is defined as a tuple of six PPT algorithms $\mathbf{ETRS} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Combine}, \mathsf{Extend})$, where the public parameters pp produced by Setup are implicitly available to all other algorithms:

$\mathsf{Setup}(1^\lambda) \rightarrow \mathsf{pp}$**:** Takes a security parameter λ and outputs a set of public parameters pp.

$\mathsf{KeyGen}() \rightarrow (\mathtt{pk}, \mathtt{sk})$**:** Generates a new public and secret key pair.

$\mathsf{Sign}(\mu, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \mathtt{sk}) \rightarrow \sigma$**:** Returns a signature with threshold $t = 1$ using the secret key sk corresponding to a public key $\mathtt{pk}_i$ with $i \in \mathcal{R}$.

$\mathsf{Verify}(t, \mu, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \sigma) \rightarrow \mathtt{accept/reject}$**:** Verifies a signature σ for the message μ against the public keys $\{\mathtt{pk}_i\}_{i\in\mathcal{R}}$ with threshold t.

$\mathsf{Combine}(\mu, \sigma_0, \sigma_1, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}) \mapsto \sigma'$**:** Combines two signatures σ_0, σ_1 for the same ring $\mathcal{R}$ into a signature σ' with threshold $t = |S_0 \cup S_1|$ where S_0, S_1 is the set of (hidden) signers for σ_0 and σ_1 respectively.

$\mathsf{Extend}(\mu, \sigma, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \{\mathtt{pk}_i\}_{i\in\mathcal{R}'}) \mapsto \sigma'$**:** Extends the signature σ with threshold t for the ring $\mathcal{R}$ into a new signature σ' with threshold t for the larger ring $\mathcal{R} \cup \mathcal{R}'$.

For a somewhat more interactive syntax, we can replace 'Sign&Combine' executions with a Join operation (described in Section 2.3). For the sake of formalism, we present our security model only for schemes with Combine and defer the discussion on how to handle Join operations to the Section 5.4, where we present a construction that uses the Join operation from signatures of knowledge and the discrete log problem.

For the following definitions, we use ladders `lad` in a slightly different way than we did in the context of extendable ring signatures (Section 3). Previously, `lad` contained a sequence of rings, which were used in repeated invocations of Extend. Now, we generalize `lad` to support arbitrary sequences of actions that could lead to a valid threshold ring signature (on some fixed message). `lad` will contain a sequence of tuples of the form $(\mathtt{action}, \mathtt{input})$. The first component, `action`, can take on the values $\mathsf{Sign}, \mathsf{Combine}$, or Extend. If $\mathtt{action} = \mathsf{Sign}$, we expect $\mathtt{input} = (\mathcal{R}, i)$, where $\mathcal{R}$ and i are the ring and signer identity with which the signature should be produced. If $\mathtt{action} = \mathsf{Combine}$, we expect $\mathtt{input} = (l_1, l_2, \mathcal{R})$, where l_1 and l_2 are indices of two signatures under the same ring $\mathcal{R}$. If $\mathtt{action} = \mathsf{Extend}$, we expect $\mathtt{input} = (l', \mathcal{R})$, where l' is the index of an existing signature which we will extended to $\mathcal{R}$.

For use in our definitions, we define an algorithm $\mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad})$, which processes all of the operations in `lad` on the message μ (using keys stored in the list $\mathsf{L}_{\mathsf{keys}}$) and returns $(\sigma, t, \mathcal{R})$: the signature returned by the last operation of `lad`, the corresponding threshold, and the ring that σ verifies under. We define `lad.sr` to be the union of all identities and rings in `lad`. (`sr` stands for super-ring.)

Definition 11 (Correctness for ETRS). *For correctness, we require that for all ladders* `lad`*, the signature returned by* $\mathsf{Process}(\mathtt{lad})$ *verifies. Formally: for all security parameters* $\lambda \in \mathbb{N}$*, for any message* $\mu \in \{0,1\}^*$*, for any ladder* `lad` *of polynomial size identifying a ring* $\mathcal{R} := \mathtt{lad.sr}$ *of public-key identifiers, for any chosen threshold value* $1 \le t \le |\mathcal{R}|$*, it holds:*

$$\Pr\left[\begin{array}{r} \mathsf{Verify}(t, \mu, \{\mathtt{pk}_i\}_{i\in\mathcal{R}}, \sigma) \\ = \mathtt{accept}\ \mathit{OR}\ \sigma = \bot \end{array} \middle| \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \mathsf{L}_{\mathsf{keys}} \leftarrow \{\mathsf{KeyGen}()\}_{j\in\mathtt{lad.sr}} \\ (\sigma, t, \mathcal{R}) \leftarrow \mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad}) \end{array}\right] = 1.$$

5.2 Security Model

Our security definitions are loosely based on the ones given for threshold ring signatures by Munch-Hansen *et al.* [23].

Definition 12 (Secure ETRS). *An extendable threshold ring signature scheme is secure if it satisfies correctness (Definition 11), unforgeability (Definition 13), anonymity (Definition 14), and some notion of anonymous extendability.*

```
$\mathbf{ETRS}.\mathsf{Process}(\mu, \mathsf{L}_{\mathsf{keys}}, \mathtt{lad})$
Parse $\mathtt{lad}$ as $((\mathtt{action}^{(1)}, \mathtt{input}^{(1)}), \ldots, (\mathtt{action}^{(l)}, \mathtt{input}^{(l)}))$
for $l' \in [1, \ldots, l]$
  if $\mathtt{action}^{(l')} = \mathsf{Sign}$   parse $\mathtt{input}^{(l')}$ as $(\mathcal{R}^{(l')}, i^{(l')})$
    for all $j \in \mathcal{R}^{(l')}$ if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$ return $\bot$  // public keys are initialized
    if $\mathtt{sk}_i = \bot$ return $\bot$  // signer's secret key was generated honestly
    if $i \notin \mathcal{R}^{(l')}$ return $\bot$  // signer is in the ring
    $\mathcal{S}^{(l')} = \{i^{(l')}\}$  // generate the signer set
    $\sigma^{(l')} \leftarrow \mathsf{Sign}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(l')}}, \mathtt{sk}_{i^{(l')}})$
  if $\mathtt{action}^{(l')} = \mathsf{Combine}$   parse $\mathtt{input}^{(l')}$ as $(l_1^{(l')}, l_2^{(l')}, \mathcal{R}^{(l')})$
    retrieve $(\mathcal{R}^{(l_1^{(l')})}, \mathcal{S}^{(l_1^{(l')})}, \sigma^{(l_1^{(l')})})$ and $(\mathcal{R}^{(l_2^{(l')})}, \mathcal{S}^{(l_2^{(l')})}, \sigma^{(l_2^{(l')})})$
    for all $j \in \mathcal{R}_1^{(l')}$ if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$ return $\bot$  // public keys are initialized
    if $\mathcal{R}^{(l_1^{(l')})} \neq \mathcal{R}^{(l')}$ or $\mathcal{R}^{(l_1^{(l')})} \neq \mathcal{R}^{(l')}$ return $\bot$  // comb. signatures use same ring
    $\mathcal{S}^{(l')} = \mathcal{S}^{(l_1^{(l')})} \cup \mathcal{S}^{(l_2^{(l')})}$  // generate the signers' set
    $\sigma^{(l')} \leftarrow \mathsf{Combine}(\mu, \sigma^{(l_1^{(l')})}, \sigma^{(l_2^{(l')})}, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(l')}})$
  if $\mathtt{action}^{(l')} = \mathsf{Extend}$   parse $\mathtt{input}^{(l')}$ as $(l_1^{(l')}, \mathcal{R}^{(l')})$
    retrieve $(\mathcal{R}^{l_1^{(l')}}, \mathcal{S}^{l_1^{(l')}}, \sigma^{l_1^{(l')}})$
    for all $j \in \mathcal{R}^{(l')}$ if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$ return $\bot$  // public keys are initialized
    if $\mathcal{R}^{l_1^{(l')}} \not\subset \mathcal{R}^{(l')}$ return $\bot$  // the extended ring is a superset
    $\mathcal{S}^{(l')} = \mathcal{S}^{l_1^{(l')}}$  // generate the signers' set
    $\sigma^{(l')} \leftarrow \mathsf{Extend}(\mu, \sigma^{l_1^{(l')}}, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{l_1^{(l')}}}, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^{(l')}})$
```

Fig. 7: The Process algorithm for extendable threshold ring signatures.

```
$\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathbf{ETRS}}(\lambda)$
1 : $\mathsf{L_{keys}}, \mathsf{L_{corr}}, \mathsf{L_{sign}} \leftarrow \varnothing$
2 : $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
3 : $O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
4 : $(t^*, \mu^*, \mathcal{R}^*, \sigma^*) \leftarrow \mathcal{A}^O(\mathsf{pp})$
5 : $q \leftarrow |\{(\mu^*, \mathcal{R}, \cdot) \in \mathsf{L_{sign}}\ s.t.\ \mathcal{R} \subseteq \mathcal{R}^*)\}|$
// rule out attacks if $\mathcal{A}$ knows too many sk:s or honestly generated signatures for $\mu^*$
6 : if $|\mathcal{R}^* \cap \mathsf{L_{corr}}| + q \geq t$   return lose
// rule out outputs that do not verify
7 : if $\mathsf{Verify}(t, \mu^*, \{\mathtt{pk}_j\}_{j \in \mathcal{R}^*}, \sigma^*) = \mathtt{reject}$   return lose
8 : return win
```

Fig. 8: Existential Unforgeability under Chosen Message Attack for (Extendable) Threshold Ring Signatures . The key generation, corruption and signing oracles are as in Figure 2, with the difference that the ERS algorithms are substituted with the ETRS variants, and the signing oracle now returns partial signatures.

Definition 13 (Unforgeability for ETRS). *An extendable threshold ring signature scheme* **ETRS** *is said to be unforgeable if for all thresholds t, for all PPT adversaries $\mathcal{A}$ the success probability in the* cmEUF *experiment in Figure 8 is* $\Pr\left[\mathsf{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathbf{ETRS}}(\lambda) = \mathtt{win}\right] \leq \mathtt{negl}$.

Just like for extendable ring signatures, the notion of anonymity for extendable threshold ring signatures captures scenarios where the adversary distinguishes fresh (not-extended) signatures, i.e., the challenge will be a threshold ring signature which has not be extended.

Definition 14 (Anonymity for ETRS). *An extendable threshold ring signature scheme is said to be anonymous if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (*ANEXT *in Figure 9) and submitting to the challenger two ladders with the structure explained below, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.:* $\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ETRS}}(\lambda) = \mathtt{win}\right] \leq \frac{1}{2} + \mathtt{negl}$.

For anonymity, the ladders submitted by the adversary to the challenger have the following structure (here t denotes the threshold of the scheme): the first t instructions are of the type $(\mathsf{Sign}, (\mathcal{R}, i))$*, where $\mathcal{R}$ is the same for all instructions in both ladders, and the signer indexes i are all distinct within the same ladder; the last $(t-1)$ instructions are of the type* $(\mathsf{Combine}, (l_1, l_2, \mathcal{R}))$*, where $\mathcal{R}$ is the same for all instructions in both ladders, $l_1 = 1, 2, \ldots, t-1$, and $l_2 = t, t+1, \ldots, 2t-2$.*

The notion of anonymous extendability is modelled by the following two definitions (which are essentially adaptations of the ones given in Section 3.2 for extendable ring signatures, to the threshold setting).

Definition 15 (Weak Anonymous Extendability for ETRS). *An extendable threshold ring signature scheme ETRS is said to be weakly anonymous extendable if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (*ANEXT *in Figure Figure 9) and submitting to the challenger ladders with the structure specified below, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.:* $\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \frac{1}{2} + \mathtt{negl}$.

For weak anonymous extendability the adversary submits ladders with the following structure: the first t instructions are of the type $(\mathsf{Sign}, (i, \mathcal{R}))$*, where the signer identities are pairwise distinct*

```
Exp $^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ETRS}}(\lambda)$
$b \leftarrow_R \{0,1\}$
$\mathsf{L_{keys}}, \mathsf{L_{corr}}, \mathsf{L_{sign}} \leftarrow \varnothing$
$\mathsf{pp} \leftarrow \mathbf{ETRS}.\mathsf{Setup}(1^\lambda)$
$O \leftarrow \{O\mathsf{Sign}, O\mathsf{KeyGen}, O\mathsf{Corrupt}\}$
$(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*) \leftarrow \mathcal{A}^O(\mathsf{pp})$
$\bar{\sigma} \leftarrow \mathsf{Chal}_b(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*)$
$b^* \leftarrow \mathcal{A}^O(\bar{\sigma})$
if $\exists i \in \mathtt{lad}_0^*.\mathtt{signers}$ $s.t.$ $i \in \mathsf{L_{corr}}$
  return $\mathtt{lose}$
if $\exists i \in \mathtt{lad}_1^*.\mathtt{signers}$ $s.t.$ $i \in \mathsf{L_{corr}}$
  return $\mathtt{lose}$
if $\exists(\mu^*, \cdot, i) \in \mathsf{L_{sign}}$ for $i \in \mathtt{lad}_0^*.\mathtt{signers}$
  return $\mathtt{lose}$
if $\exists(\mu^*, \cdot, i) \in \mathsf{L_{sign}}$ for $i \in \mathtt{lad}_1^*.\mathtt{signers}$
  return $\mathtt{lose}$
if $b^* \neq b$
  return $\mathtt{lose}$
return $\mathtt{win}$
```

```
$\mathsf{Chal}_b(\mu^*, \mathtt{lad}_0^*, \mathtt{lad}_1^*)$
if $\mathtt{lad}_0^*$ or $\mathtt{lad}_1^*$ is not well-formed
  return $\perp$
if $\exists i \in \mathtt{lad}_0^*.\mathtt{signers}$ $s.t.$ $i \in \mathsf{L_{corr}}$
  return $\perp$
if $\exists i \in \mathtt{lad}_1^*.\mathtt{signers}$ $s.t.$ $i \in \mathsf{L_{corr}}$
  return $\perp$
// make sure the public keys are known / initialized
if $\exists i \in \mathtt{lad}_0^*.\mathtt{sr}$ $s.t.$ $(\mathtt{pk}_i, \cdot) \notin \mathsf{L_{keys}}$
  return $\perp$
if $\exists i \in \mathtt{lad}_1^*.\mathtt{sr}$ $s.t.$ $(\mathtt{pk}_i, \cdot) \notin \mathsf{L_{keys}}$
  return $\perp$
$(\sigma_0, t_0, \mathcal{R}_0) \leftarrow \mathsf{Process}(\mu^*, \mathsf{L_{keys}}, \mathtt{lad}_0)$
$(\sigma_1, t_1, \mathcal{R}_1) \leftarrow \mathsf{Process}(\mu^*, \mathsf{L_{keys}}, \mathtt{lad}_1)$
// rule out trivial attacks
if $\mathcal{R}_0 \neq \mathcal{R}_1$ or $t_0 \neq t_1$
  return $\perp$
$\bar{\sigma} \leftarrow \sigma_b$
return $\bar{\sigma}$
```

Fig. 9: Anonymity and Anonymous Extendability for Extendable Threshold Ring Signatures. The key generation, corruption and signing oracles are exactly as described in the unforgeability experiment (Figure 8).

within a ladder, and the ring $\mathcal{R}$ is the same within the ladder (but possibly different for each ladder); the subsequent $t-1$ instructions are of the form $(\mathsf{Combine}, (l_1, l_2, \mathcal{R}))$ *where the indexes l_1, l_2 progressively combine (threshold) signatures on the same ring $\mathcal{R}$; the final ladder instruction is* $(\mathsf{Extend}, (l', \mathcal{R}'))$ *and extends the latest threshold ring signature to a wider ring, $\mathcal{R}' \supseteq \mathcal{R}$.*

Definition 16 (Strong Anonymous Extendability for ETRS). *An extendable threshold ring signature scheme ETRS is said to be strongly anonymous extendable if for all PPT adversaries $\mathcal{A}$ taking part in the anonymous extendability experiment (*ANEXT *in Figure 9) and submitting to the challenger ladders with the structure specified below, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.:*

$$\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A}_{\mathrm{sAnon}},\mathbf{ERS}}(\lambda) = \mathtt{win}\right] \leq \frac{1}{2} + \mathtt{negl}.$$

For strong anonymous extendability the adversary submits ladders that have the same structure as for weak anonymous extendability, except for the final Extend *instruction. While in weak anonymous extendability we allow a single extension (to the same ring $\mathcal{R}'$), in strong anonymous extendability each ladder may contain an arbitrary (polynomial, and possibly different for each ladder) number of subsequent* Extend *instructions, so long the final one of each ladder culminates in the same ring.*

5.3 A Generic Compiler for ETRS from SMLERS

In what follows, we formalize the intuition given in Remark 3 (Section 4.1) on how to generically derive an extendable threshold ring signature scheme from any given same-message linkable extendable ring signature scheme. The compiler is detailed in Figure 10.

```
$\mathsf{Setup}(1^\lambda) \mapsto \mathtt{pp}$
    return $\mathbf{SMLERS}.\mathsf{Setup}(1^\lambda)$

$\mathsf{KeyGen}() \mapsto (\mathtt{pk}, \mathtt{sk})$
    return $\mathbf{SMLERS}.\mathsf{KeyGen}()$

$\mathsf{Sign}(\mu, \mathtt{sk}, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}) \mapsto \sigma$
    return $\mathbf{SMLERS}.\mathsf{Sign}(\mu, \mathtt{sk}, \{\mathtt{pk}_i\}_{i \in \mathcal{R}})$

$\mathsf{Extend}(\mu, \sigma, \{\mathtt{pk}_i\}_{i \in \mathcal{R}_0}, \{\mathtt{pk}_i\}_{i \in \mathcal{R}_1}, ) \mapsto \sigma'$
    return $\{\mathbf{SMLERS}.\mathsf{Extend}(\{\mathtt{pk}_i\}_{i \in \mathcal{R}_1}, \{\mathtt{pk}_j\}_{j \in \mathcal{R}_2}, \sigma_i)\}_{\sigma_i \in \sigma}$

$\mathsf{Combine}(\mu, \sigma_0, \sigma_1, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}) \mapsto \sigma'$
  1 : $\sigma_0' \leftarrow \{s_0 \in \sigma_0 \mid \forall s_1 \in \sigma_1 : \mathsf{Link}(\mu, (s_0, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}), (s_1, \{\mathtt{pk}_i\}_{i \in \mathcal{R}})) = \mathtt{unlinked}\}$
  2 : return $\sigma_0' \cup \sigma_1$

$\mathsf{Verify}(t, \mu, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}, \sigma) \mapsto \mathtt{accept}/\mathtt{reject}$
  1 : Parse $\sigma = \{s_0, \ldots, s_\ell\}$ as a set of signatures  // removing duplicates
  2 : if $|\sigma| < t$  return reject
  3 : if $\exists\ s_i \in \sigma : \mathsf{Verify}(\mu, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}, s_i) = \mathtt{reject}$  return reject
  4 : if $\exists\ (s_i, s_j) \in \sigma \times \sigma : s_i \neq s_j\ \wedge$
          $\mathsf{Link}(\mu, (s_i, \{\mathtt{pk}_i\}_{i \in \mathcal{R}}), (s_j, \{\mathtt{pk}_i\}_{i \in \mathcal{R}})) = \mathtt{linked}$  return reject
  5 : return accept
```

Fig. 10: Generic Compiler for Extendable Threshold Ring Signatures from Extendable Same-Message Linkable Ring Signatures.

Theorem 2. *Assuming that* **SMLERS** *is a secure same-message linkable extendable ring signature scheme, then the scheme* **ETRS** = (Setup, KeyGen, Sign, Verify, Extend, Combine) *described in Figure 10 is an extendable threshold ring signature scheme that satisfies* correctness *(Definition 11),* unforgeability *(Definition 13), and* anonymity *(Definition 14).*

Proof. The correctness of the construction follows by inspection.

Unforgeability The unforgeability of the ETRS scheme reduces directly to the one-more linkability of the underlying SMLERS scheme. The reduction is straightforward and tight, because a forgery for the ETRS scheme corresponds to t unlinked SMLERS signatures.

Anonymity The anonymity of the ETRS construction reduces to the anonymity of the underlying SMLERS. The proof goes though a sequence of hybrid games. In $\mathcal{H}_0$, the challenger always picks the anonymity bit $b = 0$ (deterministically). In the final hybrid $\mathcal{H}_t$, the challenger always picks the anonymity bit $b = 1$. For simplicity, assume that the two sets of signers selected by the adversary for the challenge are distinct (i.e., $\texttt{lad}_0^*.\texttt{signers} \cap \texttt{lad}_1^*.\texttt{signers} = \varnothing$). The intermediate hybrids progressively change the signer set of the challenge signature from $\texttt{lad}_0^*.\texttt{signers}$ to $\texttt{lad}_1^*.\texttt{signers}$. In detail, for $j = 1, \ldots, t$, in hybrid $\mathcal{H}_j$ the challenger returns a signature $\bar{\sigma}$ obtained combining the signatures of the first j signer identities from $\texttt{lad}_1^*.\texttt{signers}$, and the last $t-j$ identities from $\texttt{lad}_0^*.\texttt{signers}$. Let E_j denote the event where $\mathcal{A}$ guesses $b^* = 1$ at the end of $\mathcal{H}_j$. Clearly, $|\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A}_{\mathrm{Anon}},\mathbf{ETRS}}(\lambda) = \texttt{win}\right] - \frac{1}{2}| = |\Pr[E_0] - \Pr[E_t]|$. We can bound this probability by: $|\Pr[E_0] - \Pr[E_t]| = |\sum_{j=1}^{t} \Pr[E_{j-1}] - \Pr[E_j]| \leq t \cdot |\Pr\left[\mathsf{Exp}^{\mathrm{ANEXT}}_{\mathcal{A}_{\mathrm{Anon}},\mathbf{ERS}}(\lambda) = \texttt{win}\right] - \frac{1}{2}|$, where t is the threshold. The last inequality follows from the triangular inequality and the following reduction. For every j, define $\mathcal{B}$ to forward all of $\mathcal{A}$'s queries to its ANEXT-ERS challenger. Upon receiving $(\mu^*, \texttt{lad}_0^*, \texttt{lad}_1^*)$ from $\mathcal{A}$ (line 6 in the ANEXT-ETRS experiment of Figure 9), the reduction submits signing queries of the form $(\mu^*, \mathcal{R}^*, i)$, where i ranges over all of the first $(j-1)$ identities in $\texttt{lad}_1^*.\texttt{signers}$ and the last $t-j$ identities in $\texttt{lad}_0^*.\texttt{signers}$. Let σ_i denote the obtained responses. Third, $\mathcal{B}$ submits to its ANEXT-ERS challenger the tuple $(\mu^*, \texttt{lad}_0, \texttt{lad}_1)$ where $\texttt{lad}_0 = (i_j^{(0)}, \mathcal{R}^*)$ and $\texttt{lad}_1 = (i_j^{(1)}, \mathcal{R}^*)$. Here, $i_j^{(b)}$ denotes the identity of the j-th signer in ladder $\texttt{lad}_b$. Let $\bar{\sigma}$ be the signature returned by the challenger. The reduction combines the t same-message-linkable extendable ring signatures $\{\sigma_i\}_{i \in [1,\ldots,j-1,j+1,\ldots,n]} \cup \{\bar{\sigma}\}$ to create $\bar{\sigma}^*$, the challenge extendable threshold ring signature for $\mathcal{A}$. Clearly, the reduction is simulating $\mathcal{H}_j$ when $b = 0$, and $\mathcal{H}_{j+1}$ when $b = 1$. Thus $\mathcal{B}$ can exploit an adversary that can distinguish between a pair of consecutive hybrids to break the anonymity of the underlying SMLERS.

□

5.4 ETRS from Signatures of Knowledge and Discrete Log

In what follows we present a somewhat more interactive Extendable Threshold Ring Signature Scheme that supports Join operations and enjoys more compact signatures. Concretely, the size of extended threshold signatures is independent of the threshold t, instead it grows linearly with n' (an upper bound on the ring size). This is an improvement compared to the compiler presented in Figure 10, which if instantiated using our SMLERS from Signatures of Knowledge and Discrete Log of Section 4.3, returns signatures of size linear in $t \cdot |\mathcal{R}|$.

Our Construction in a Nutshell Similarly to the ERS construction of Figure 4, we work with a prime order group $\mathbb{G}$, with two public elements $g, H \in \mathbb{G}$ and a signature of knowledge for a relation $\mathscr{R}_{\mathbb{G}}$ for knowledge of the discrete logarithm either of a given value h or of a $\texttt{pk}$.

Let $n' \in \mathbb{N}$ be an upper bound on the ring size. We achieve the threshold functionality by leveraging features of polynomials in a similar way to Shamir secret sharing. Intuitively, the signer samples $n' > 0$ pairs of values $(x_i, \texttt{td}_i) \in \mathbb{Z}_p \times \mathbb{G}$. These pairs of values define a unique polynomial $f(x)$ of degree n' such that $f(0) = \mathrm{dlog}_g(H)$ and $f(x_i) = \texttt{td}_i$ for every $i \in [n']$. Of course, since $\mathrm{dlog}_g(H)$ is unknown, our signers don't know the coefficients of this polynomial. However, since polynomial interpolation involves only linear operations (when the x-coordinates are fixed and known), the signers can interpolate this polynomial *in the exponent* to learn additional points $(\hat{x}, y = g^{f(\hat{x})})$ for any given $\hat{x}$. In order to sign, and later to endorse a statement (Join a signature), the signer is required to produce a signature of knowledge for $\mathscr{R}_{\mathbb{G}}$ for a random point $(\hat{x}, y = g^{f(\hat{x})})$ on the polynomial such that $\hat{x} \notin \{x_i\}_{i \in [n']}$. Crucially, the signer does not know the discrete log of y

```
$\mathsf{PolySign}(P, T, \hat{x}, w, \mathtt{pk}, \mu) \rightarrow (y_{\hat{x}}, \pi)$

// compute values for the Lagrange interpolation
1 :  $\mathcal{Z} := \{(0, H)\} \cup \{(x, g^{\mathtt{td}})\}_{(x,\mathtt{td})\in T} \cup \{(x, y)\}_{(x,y,\mathtt{pk},c,\pi)\in P}$
// compute evaluation points for the Lagrange interpolation
2 :  $\mathcal{X} := \{x\}_{(x,y)\in\mathcal{Z}}$
// evaluate the polynomial on the input point $\hat{x}$
3 :  $y_{\hat{x}} \leftarrow \prod_{(x,y)\in\mathcal{Z}} y^{L_{(\mathcal{X},x)}(\hat{x})}$  // note: $dlog(y_{\hat{x}})$ is unknown because $dlog(H)$ is
4 :  $\phi := (\hat{y}, \mathtt{pk}_s)$  // include new poly. value in the statement
// Lagrange interpolation in the exponent over the standard set $\{1, ..., n'\}$
5 :  for $i \in [n']$ :   $V_i \leftarrow \prod_{(x,y)\in\mathcal{Z}} y^{L_{(\mathcal{X},x)}(i)}$
6 :  $\hat{\mu} := (\mu, \{V_i\}_{i\in[n']})$  // include a 'commitment' to the polynomial in the message
7 :  $\pi \leftarrow \mathbf{SoK}.\mathsf{Sign}(\hat{\mu}, \mathscr{R}_{\mathbb{G}}, \phi, w)$
// note: $w$ is given in input to the algorithm, $w = \mathtt{sk}$ for Sign & Join, otherwise $w = \mathtt{td}$
8 :  return $(\hat{y}, \pi)$
```

Fig. 11: Subroutine used in our ETRS construction depicted in Figure 12.

(i.e., $(\hat{x}, y)$ is not among the 'trapdoored' values $(x_i, g^{\mathtt{td}_i})$), and thus must satisfy the second clause of the relation (proving knowledge of their secret key). On the other hand, to extend a signature, anyone can pick one of the (remaining) 'trapdoored' points $(x_i, \mathtt{td}_i)$, and generate a proof for $\mathscr{R}_{\mathbb{G}}$ by satisfying the first clause (proving knowledge of $\mathtt{td}_i$), to include any $\mathtt{pk}$ in the ring. The pair $(x_i, \mathtt{td}_i)$ is then removed from the list of trapdoors. (In case the owner of $\mathtt{pk}$ later wants to join the signature, the Extend algorithm encrypts $\mathtt{td}_i$ to $\mathtt{pk}$; later, the owner of $\mathtt{pk}$ can recover $\mathtt{td}_i$ and return it to the list of trapdoors before producing a fresh signature of knowledge using her secret key.)

The key idea of our construction is detailed in Figure 11 (the PolySign subroutine employed in Signand Join–where this is called using the signer's secret key as w and on a random value $\hat{x}$– and in Extend–where an evaluation point and its corresponding trapdoor are used as $\hat{x}$and w respectively).

For any field $\mathbb{F}$ (often implicit) and $\mathcal{X} \subseteq \mathbb{F}$, $j \in \mathcal{X}$, define the degree $|\mathcal{X}|-1$ Lagrange polynomial $L_{(\mathcal{X},j)}(X) := \prod_{m\in\mathcal{X}\setminus\{j\}} \frac{X-m}{j-m} \in \mathbb{F}[X]$.

Remarks on Modeling Join Operations Intuitively, our construction replaces Combine with Join, where Join essentially performs Sign& Combine in a more efficient manner. To formally support Join, we need to make a few tweaks to the security definitions introduced in Section 5.2.

First, we define how to process join actions in a ladder. If $\mathtt{action} = \mathsf{Join}$, we expect $\mathtt{input} = (l', \mathcal{R}, i)$, where l' is the index of an existing signature in the ladder which is to be joined by signer i. Concretely, ETRS.Process retrieves $(\mathcal{R}^{(l')}, \mathcal{S}^{(l')}, \sigma^{(l')})$; checks that all of the keys in $\mathcal{R}$ have been initialized, i.e., appear in $\mathsf{L}_{\mathsf{keys}}$; checks that $\mathcal{R}^{(l')} \cup \{i\} = \mathcal{R}$, and checks that the index of the new signer appears in $\mathcal{R}$ but not among the singers set $\mathcal{S}^{(l')}$. If all these checks pass, the signer is added to $\mathcal{S}$; that is, we set $\mathcal{S} := \mathcal{S}^{(l')} \cup \{i\}$. We then process the join action as $\sigma \leftarrow \mathsf{Join}(\mu, \{\mathtt{pk}_j\}_{j\in\mathcal{R}^{(l')}}, \mathtt{sk}_i, \sigma^{(l')})$. Finally, we store $(\mathcal{R}, \mathcal{S}, \sigma)$ and return $(\mathcal{R}, t = |\mathcal{S}|, \sigma)$.

```
KeyGen() ↦ (pk, sk)
1 : $(\mathtt{pk}_s, \mathtt{sk}_s) \leftarrow \mathbf{ERS}.\mathsf{KeyGen}()$
2 : $(\mathtt{pk}_e, \mathtt{sk}_e) \leftarrow \mathbf{PKE}.\mathsf{KeyGen}()$
3 : return $(\mathtt{pk} = (\mathtt{pk}_s, \mathtt{pk}_e), \mathtt{sk} = (\mathtt{sk}_s, \mathtt{sk}_e))$

Sign($\mu$, sk) ↦ $\sigma$
1 : $X \leftarrow_R \binom{\mathbb{Z}_p^*}{n'}$  // pick $n'$ distinct evaluation points
2 : $T := \varnothing$;  $P := \varnothing$
3 : for $x \in X$
4 :     $\mathtt{td} \leftarrow_R \mathbb{Z}_p$  // generate trapdoors for poly. values
5 :     $T \leftarrow T \cup \{(x, \mathtt{td})\}$  // populate trapdoor set
6 : $c \leftarrow \mathsf{Enc}(\mathtt{pk}_e, \perp)$  // no info to pass on
7 : $\hat{x} \leftarrow_R \mathbb{Z}_p^* \setminus X$  // pick a new evaluation point
8 : $(y, \pi) \leftarrow \mathsf{PolySign}(P, T, \hat{x}, w := \mathtt{sk}, \mathtt{pk}, \mu)$
9 : $P := \{(\hat{x}, y, \mathtt{pk}_s, \pi, c)\}$
10 : return $\sigma := (T, P)$

Join($\mu$, $\{\mathtt{pk}_j\}_{j\in\mathcal{R}}$, sk, $\sigma$) ↦ $\sigma'$
  // check if current signer's $\mathtt{pk}_s$ is in $P$
1 : if $\exists\ (x, y, \mathtt{pk}, \pi, c) \in P$ s.t. $\mathtt{pk} = \mathtt{pk}_s$
  // remove simulated proof for the signer who wants to join
2 :     $P \leftarrow P \setminus \{(x, y, \mathtt{pk}_s, \pi, c)\}$
  // retrieve trapdoor value
3 :     $\mathtt{td} \leftarrow \mathsf{Dec}(\mathtt{sk}_e, c)$
  // add eval. point and td to the set of available trapdoors
4 :     $T \leftarrow T \cup \{(x, \mathtt{td})\}$
5 : $c' \leftarrow \mathsf{Enc}(\mathtt{pk}_e, \perp)$  // no info to pass on
6 : $\hat{x} \leftarrow_R \mathbb{Z}_p^* \setminus X$  // pick a new evaluation point
  // interpolate a unique representation of the polynomial
7 : $(y', \pi') \leftarrow \mathsf{PolySign}(P, T, \hat{x}, w := \mathtt{sk}, \mathtt{pk}, \mu)$
8 : $P \leftarrow P \cup \{(\hat{x}, y', \mathtt{pk}_s, \pi', c')\}$
9 : Randomly permute $P$
10 : return $\sigma := (T, P)$

Extend($\mu$, $\{\mathtt{pk}_j\}_{j\in\mathcal{R}}$, $\sigma$, pk) ↦ $\sigma'$
1 : if $\mathtt{pk} \in \{\mathtt{pk}_j\}_{j\in\mathcal{R}}$ :   return $\perp$
2 : $(\hat{x}, \hat{\mathtt{td}}) \leftarrow_R T$  // Pick eval-point and trapdoor
3 : $c' \leftarrow \mathsf{Enc}(\mathtt{pk}_e, \hat{\mathtt{td}})$  // enable future endorsing
  // interpolate a unique representation of the polynomial
4 : $(y', \pi') \leftarrow \mathsf{PolySign}(P, T, \hat{x}, w := \hat{x}, \mathtt{pk}, \mu)$
5 : $T \leftarrow T \setminus \{(\hat{x}, \hat{\mathtt{td}})\}$  // erase used trapdoor
  // Add simulated signature to the set of proofs
6 : $P \leftarrow P \cup \{(\hat{x}, y', \mathtt{pk}_s, \pi', c')\}$
7 : Randomly permute $P$
8 : return $\sigma' := (T, P)$

Verify($t$, $\mu$, $\{\mathtt{pk}_j\}_{j\in\mathcal{R}}$, $\sigma$) ↦ accept/reject
1 : if $\{\mathtt{pk}_j\}_{j\in\mathcal{R}} \neq \{\mathtt{pk}_i\}_{(\cdot,\cdot,\mathtt{pk}_i,\cdot,\cdot)\in P}$ :
2 :     return reject
  // check $y$'s are consistent with a degree $n'$ polynomial
3 : $\mathcal{Z} := \{(0, H)\} \cup \{(x, g^{\mathtt{td}})\}_{(x,\mathtt{td})\in T}$
4 : $\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(x, y)\}_{(x,y,\mathtt{pk},c,\pi)\in P}$
5 : Pick $\hat{\mathcal{Z}} \subseteq \mathcal{Z}$ s.t. $|\hat{\mathcal{Z}}| = n' + 1$
6 : $\mathcal{X} := \{x\}_{(x,y)\in\mathcal{Z}}$; $\hat{\mathcal{X}} := \{x\}_{(x,y)\in\hat{\mathcal{Z}}}$
7 : for $(x, y) \in \mathcal{Z}$ :
8 :     if $y \neq \prod_{(\hat{x},\hat{y})\in\hat{\mathcal{Z}}} \hat{y}^{L_{(\hat{\mathcal{X}},\hat{x})}(x)}$ : return reject
  // Interpolation over the standard set $\{1, \ldots, n'\}$
9 : for $i \in [n']$ :   $V_i \leftarrow \prod_{(x,y)\in\mathcal{Z}} y^{L_{(\mathcal{X},x)}(i)}$
10 : $\hat{\mu} := (\mu, \{V_i\}_{i\in[n']})$
11 : for $(x, y, \mathtt{pk}_s, \pi, c) \in P$  // check proofs individually
12 :     $\phi := (y, \mathtt{pk}_s)$
13 :     if $\mathbf{SoK}.\mathsf{Verify}(\hat{\mu}, \mathscr{R}_{\mathbb{G}}, \phi, \pi) = $ reject
14 :         return reject
15 : if $|T| + |P| \geq t + n'$ return accept
16 : else return reject
```

Fig. 12: Extendable Threshold Ring Signatures from Signature of Knowledge and Hardness of Discrete Log. The Setup algorithm is the same as in the ERS construction of Figure 4 (with $\mathscr{R}_{\mathbb{G}} = \{(\phi, w) = (h, \mathtt{pk}, x) \in \mathbb{G} \times \mathbb{G} \times \mathbb{Z}_p : g^x = h \vee g^x = \mathtt{pk}\}$). In the description, $n' > 0$ denotes the maximum amount of times a signature can be extended (it can be set in pp, or chosen upon signing). We always let pk denote the public key corresponding to sk; any algorithm that is given sk as input implicitly has access to pk. The parsing of pk into $(\mathtt{pk}_s, \mathtt{pk}_e)$ (or of $\mathtt{pk}_i$ into $(\mathtt{pk}_{s,i}, \mathtt{pk}_{e,i})$), of sk into $(\mathtt{sk}_s, \mathtt{sk}_e)$ and of σ into (T, P) is done implicitly.

```
$O\mathsf{Join}(\mu, \mathcal{R}, i, \sigma)$
1 : if $i \in \mathsf{L}_{\mathsf{corr}}$ return $\perp$
2 : for all $j \in \mathcal{R}$
3 :     if $(j, \mathtt{pk}_j, \cdot) \notin \mathsf{L}_{\mathsf{keys}}$ return $\perp$
4 : $\sigma' \leftarrow \mathsf{Join}(\mu, \{\mathtt{pk}_j\}_{j \in \mathcal{R}}, \mathtt{sk}_i, \sigma)$
5 : $\mathsf{L}_{\mathsf{join}} \leftarrow \mathsf{L}_{\mathsf{join}} \cup \{(\mu, i, \sigma)\}$
6 : return $\sigma'$
```

Second, for unforgeability we add $O\mathsf{Join}$ (described to the left) to the oracle handle O available to the adversary (ln 3 in Figure 8). The join oracle also manages a list $\mathsf{L}_{\mathsf{join}}$ of join queries, initialized as empty at the beginning of the experiment.

In addition, we need to keep track of signatures $\mathcal{A}$ obtains through $O\mathsf{Join}$ among the list of signatures that trivially lead to a forgery. To do so, we add to the count on line 5 of Figure 8 the number of relevant join operations $|\{(\mu^*, i, \cdot) \in \mathsf{L}_{\mathsf{join}}\}|$ where $\mathsf{L}_{\mathsf{join}}$ denotes the list of join queries, μ^* is the message specified in the forgery and $i \in \mathcal{R}^*$ is among the signer identities specified in the forgery ring.

Theorem 3. *Assuming that* **SoK** *is a secure signature of knowledge scheme, and that the discrete log problem is hard in the group* $\mathbb{G}$*, then the scheme* **ETRS** $= (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Extend}, \mathsf{Join})$ *described in Figure 12 is an extendable threshold ring signature scheme that satisfies* correctness *(Definition 11),* unforgeability *(Definition 13), and* strong anonymous extendability *(Definition 14).*

Proof. Correctness follows by inspection. The proof for anonymous extendability is almost identical to that in the proof of Theorem 1.

Unforgeability The proof of unforgeability closely follows the proof of unforgeability of the ERS scheme described in Figure 4 (Theorem 1), but is a bit more involved. Here, we describe how each hybrid is different.

$\mathcal{G}_0$**:** Same as in the proof of Theorem 1. This is the unforgeability game.
$\mathcal{G}_1$**:** Same as in the proof of Theorem 1. H is set to be the challenge $\mathcal{B}$ receives from its dlog challenger.
$\mathcal{G}_2$**:** Same as in the proof of Theorem 1. All signatures of knowledge are now simulated.
$\mathcal{G}_{2a}$**:** This is the same as $\mathcal{G}_0$, except that now, in response to sign queries, all encryptions to honest parties are encryptions of $\perp$. The reduction remembers which trapdoor should have been encrypted, for future use.
This is indistinguishable from the previous game by CCA security of the encryption scheme.
$\mathcal{G}_3$**:** Same as in the proof of Theorem 1. In response to a sign query, the reduction randomizes which h_i it does not know the discrete log of. (Note that we needed $\mathcal{G}_{2a}$ before this game, since otherwise, this change would affect the distribution of the generated signatures.)
$\mathcal{G}_{4a}$**:** This is the same as $\mathcal{G}_3$, except now, if the candidate forgery contains *a subset* of a response to a sign query and this subset contains any signatures of knowledge on statements *where* $\mathcal{B}$ *does not know the discrete log of* h_i, $\mathcal{B}$ aborts.
With polynomial probability, $\mathcal{B}$ does not abort in this game, by the same reasons given in $\mathcal{G}_4$ in the proof of Theorem 1.
$\mathcal{G}_5$**:** Same as in the proof of Theorem 1. $\mathcal{B}$ now aborts if the candidate forgery contains only simulated signatures of knowledge (returned by either the sign or join oracles). In order to be a good candidate forgery, a signature cannot consist of a response to a sign query together with

responses to join queries; so, it must contain only a subset of the simulated signatures returned in response to the relevant sign query. $\mathcal{B}$ can now obtain the discrete log of H via interpolation instead of via addition.

$\mathcal{G}_6$: Same as in the proof of Theorem 1. When prompted to generate a key pair for i^*, $\mathcal{B}$ returns $\mathtt{pk}_{i^*} := H^e$ for a random exponent $e \leftarrow \mathbb{Z}_p$ (no matching secret key is generated). If $\mathcal{A}$ submits a corruption query for i^*, $\mathcal{B}$ aborts. When $\mathcal{A}$ outputs a valid forgery $(\mu^*, \mathcal{R}^*, \sigma^*)$, $\mathcal{B}$ checks that $i^* \in \mathcal{R}$; otherwise it aborts.

$\mathcal{G}_7$: Same as in the proof of Theorem 1.

Note that at this point, honest party i^* is in the challenge ring. Note also that either

(a) the polynomial was generated by $\mathcal{B}$ in response to a sign query — in which case $\mathcal{B}$ already knows n' points on the polynomial, and simulated signatures might be a part of the candidate forgery — or

(b) the polynomial was generated by the adversary — in which case at least $t-q$ signatures are not simulated, where q is the number of join queries the adversary asked with the challenge message and polynomial on behalf of honest parties in the challenge ring.

$\mathcal{B}$ now calls the extractor $\varepsilon_{\mathcal{A}}$ for SoK to extract witnesses w_i from each signature of knowledge π_i in σ^* (which are not simulated signatures).

Like in the proof of Theorem 1, if extraction fails from any non-simulated signature, $\mathcal{B}$ aborts. If π_{i^*} is simulated, $\mathcal{B}$ aborts.

If any of the witnesses are secret keys, and if w_{i^*} is not a secret key, $\mathcal{B}$ aborts. (This happens with at most polynomial probability.) If w_{i^*} is the secret key of party i^*, $\mathcal{B}$ can find the discrete log of H from this (by dividing the secret key by e).

Otherwise, if all the extracted witnesses are points on the polynomial, $\mathcal{B}$ then extracts at least $n-q$ points on the polynomial; an additional $n'-(n-t)$ are available as trapdoors. If $q \geq t$ the candidate forgery is invalid; so, $n-q > n-t$, and $\mathcal{B}$ ends up knowing at least $n'+1$ points on the polynomial. $\mathcal{B}$ can now find the discrete log of H by interpolating the polynomial.

□

Remark 4. Note that a malicious extender can prevent the newly added members of the ring from later joining a signature, simply by not encrypting the correct trapdoor under that new member's public key. This is not captured by our security definitions, but precluding such attacks would be an interesting and valuable extension. We can modify our construction to disallow this by adding a zero knowledge proof that the encrypted value is in fact the discrete log of the h in question.

6 Implementation Results

We have implemented the ERS and ETRS constructions from Section 3 and 5 with two different choices of groups at the 128-bit security level within the RELIC [5] library. The first parameter choice is the conservative `edwards25519` curve used in the Ed25519 signature scheme [2], and the second is the record-setting `GLS254` binary curve [25]. The latter provides an efficient endomorphism that accelerates scalar multiplication by breaking scalars into smaller subscalars that can be handled in an interleaved manner. In the ERS construction, the performance depends on the ring size only, so the number of extensions is always the number of keys. For the ETRS construction, the quadratic cost of interpolation clearly dominates the signing, joining and verification steps. The forked library is available at `https://github.com/relic-toolkit/relic`.

[5] `https://github.com/relic-toolkit/relic`

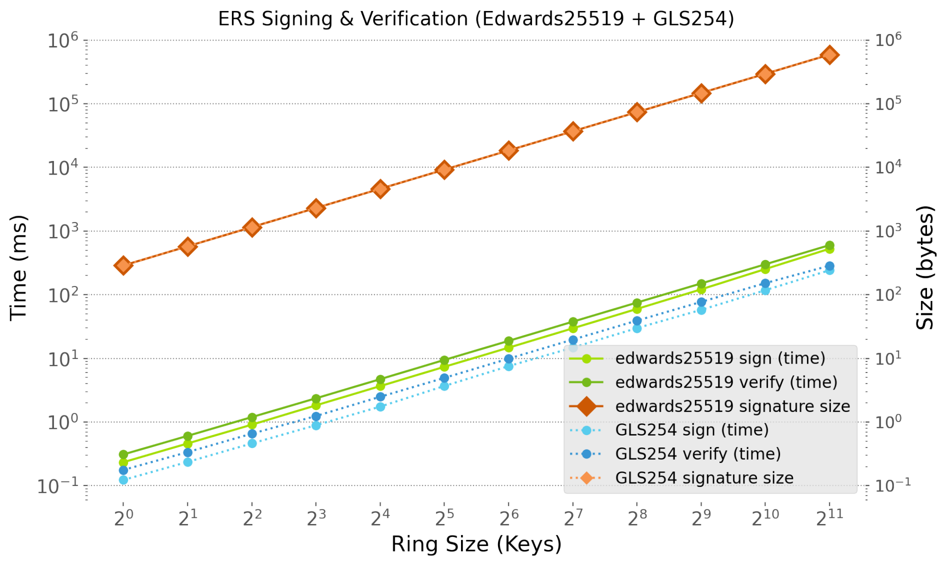

Fig. 13: Clock time for Sign, Verify and signature sizes for extendable ring signatures (Figure 4). The signature size is the same for both `Ed25519` and `GLS254`.

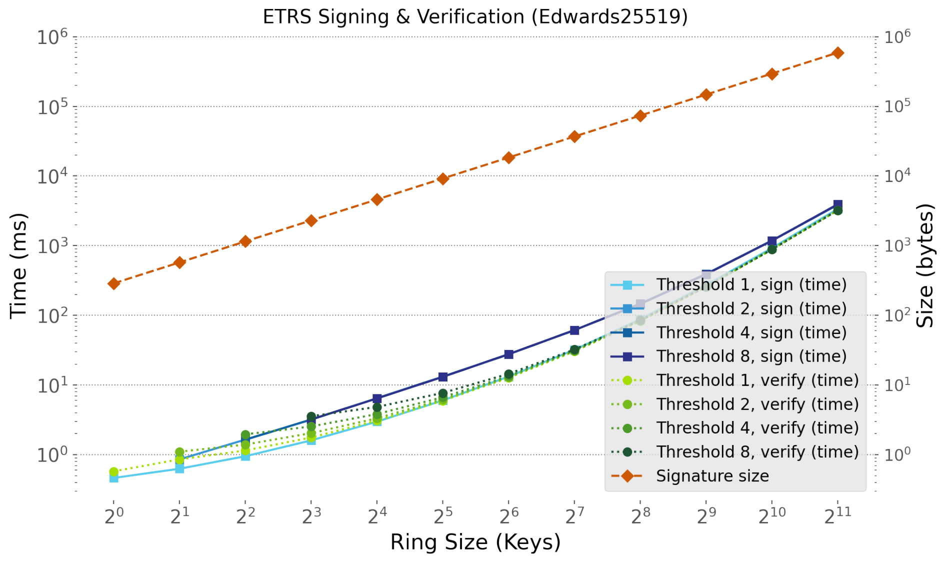

Fig. 14: Clock time and signature sizes for our ETRS scheme (Figure 12), for different thresholds. The signature generation time includes the initial signature generation and subsequent extensions. The signature size is independent of the threshold.

ERS Benchmark We benchmark our ERS implementation for ring sizes of 1 to 2^{11} on an Intel Core i7-6700K Skylake running at 4 Ghz, with HyperThreading and TurboBoost disabled. The signature generation time, verification times and signature sizes (without point compression) are shown in Figures 13.

ETRS Benchmark We benchmark our ETRS implementation for thresholds of $1, 2, 4, 8$ and ring sizes of 1 to 2^{11} over `edwards25519` on an Intel Core i7-6700K Skylake running at 4 Ghz, with HyperThreading and TurboBoost disabled. For ease of exposition, we combined the wall time for the initial signature generation and subsequent extensions in the plot (Figure 14). The verification time is that of verifying the final extended signature. Compared to the ERS scheme, the additional computational overhead is dominated by the polynomial interpolation. Finally, we also implemented and benchmarked the less efficient generic ETRS from SMLERS, with the data plotted in Figure 15.

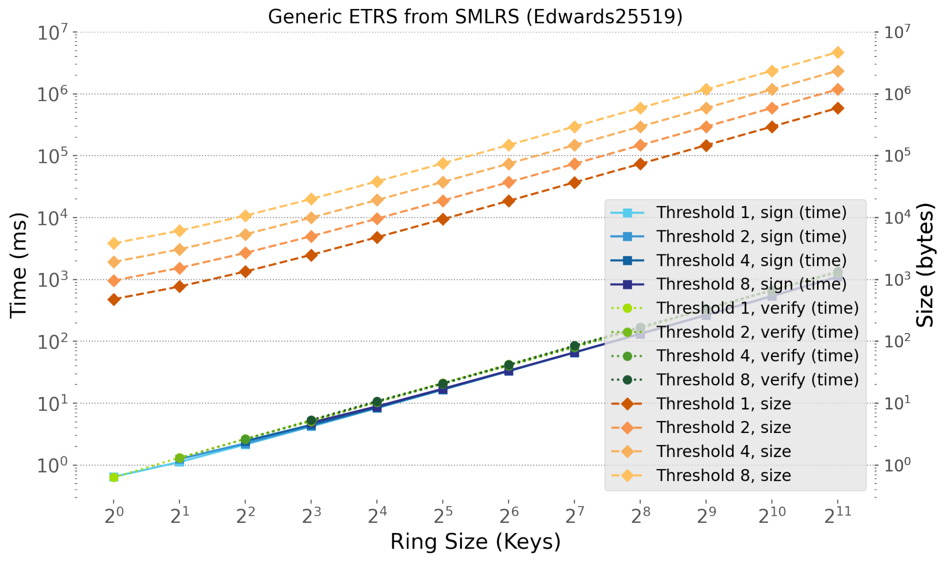

Fig. 15: Clock time and signature sizes for our generic ETRS scheme (Figure 10) from our SMLRS, for different thresholds. The signature generation time includes the initial signature generation and subsequent extensions. The signature size depends linearly on the threshold.

References

1. Bender, A., Katz, J., Morselli, R.: Ring signatures: Stronger definitions, and constructions without random oracles. In: TCC 2006
2. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering
3. Bettaieb, S., Schrek, J.: Improved lattice-based threshold ring signature scheme. In: Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013
4. Beullens, W., Katsumata, S., Pintore, F.: Calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. Cryptology ePrint Archive, Report 2020/646
5. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: EUROCRYPT 2003
6. Bootle, J., Cerulli, A., Chaidos, P., Ghadafi, E., Groth, J., Petit, C.: Short accountable ring signatures based on DDH. In: ESORICS 2015, Part I
7. Bose, P., Das, D., Rangan, C.P.: Constant size ring signature without random oracle. In: ACISP 15
8. Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: CRYPTO 2002
9. Brickell, E.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: ACM CCS 2004
10. Chandran, N., Groth, J., Sahai, A.: Ring signatures of sub-linear size without random oracles. In: ICALP 2007
11. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: CRYPTO 2006
12. Chow, S.S.M., Wei, V.K.W., Liu, J.K., Yuen, T.H.: Ring signatures without random oracles. In: ASIACCS 06

13. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: EUROCRYPT 2004
14. Esgin, M.F., Steinfeld, R., Sakzad, A., Liu, J.K., Liu, D.: Short lattice-based one-out-of-many proofs and applications to ring signatures. In: ACNS 19
15. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: CRYPTO 2017, Part II
16. Haque, A., Scafuro, A.: Threshold ring signatures: New definitions and post-quantum security. In: PKC 2020, Part II
17. Liu, J.K.: Ring signature. In: Advances in Cyber Security: Principles, Techniques, and Applications
18. Liu, J.K., Wong, D.S.: On the security models of (threshold) ring signature schemes. In: ICISC 04
19. Liu, Z., Nguyen, K., Yang, G., Wang, H., Wong, D.S.: A lattice-based linkable ring signature supporting stealth addresses. In: ESORICS 2019, Part I
20. Lu, X., Au, M.H., Zhang, Z.: Raptor: A practical lattice-based (linkable) ring signature. In: ACNS 19
21. Malavolta, G., Schröder, D.: Efficient ring signatures in the standard model. In: ASIACRYPT 2017, Part II
22. Melchor, C.A., Cayrel, P.L., Gaborit, P., Laguillaumie, F.: A new efficient threshold ring signature scheme based on coding theory. IEEE Transactions on Information Theory
23. Munch-Hansen, A., Orlandi, C., Yakoubov, S.: Stronger notions and a more efficient construction of threshold ring signatures. Tech. rep., Cryptology ePrint Archive, Report 2020/678
24. Okamoto, T., Tso, R., Yamaguchi, M., Okamoto, E.: A k-out-of-n ring signature with flexible participation for signers. Cryptology ePrint Archive, Report 2018/728
25. Oliveira, T., López-Hernández, J.C., Aranha, D.F., Rodríguez-Henríquez, F.: Two is the fastest prime: lambda coordinates for binary elliptic curves. Journal of Cryptographic Engineering
26. Patachi, S., Schürmann, C.: Eos a universal verifiable and coercion resistant voting protocol. In: E-VOTE-ID
27. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: ASIACRYPT 2001
28. Shacham, H., Waters, B.: Efficient ring signatures without random oracles. In: PKC 2007
29. Sun, S.F., Au, M.H., Liu, J.K., Yuen, T.H.: RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: ESORICS 2017, Part II
30. Tsang, P.P., Wei, V.K., Chan, T.K., Au, M.H., Liu, J.K., Wong, D.S.: Separable linkable threshold ring signatures. In: INDOCRYPT 2004
31. Yuen, T.H., Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Threshold ring signature without random oracles. In: ASIACCS 11
32. Zhang, F., Kim, K.: ID-based blind signature and ring signature from pairings. In: ASIACRYPT 2002