Analysis of "Count Me In! Extendability for Threshold Ring Signatures"

Dimitrios Vassiliou
el19830@mail.ntua.gr

Stylianos Zarifis
el20435@mail.ntua.gr

March 22, 2024

1 Introduction

Ring signatures enable anonymous message signing within a group, and *threshold ring signatures* extend this to multiple signers without revealing their identities. However, existing constructions lack flexibility and coordination, hindering post-factum endorsements. The "Count Me In! Extendability for Threshold Ring Signatures" paper [1] introduces *extendability* for these primitives, allowing third-party enlargement of the anonymity set. In *extendable threshold ring signatures*, signatures on the same message can be combined, enhancing anonymity. Formal models, security proofs, and concrete constructions are presented, relying on signatures of knowledge and discrete logarithm hardness. A transformation from same-message-linkable extendable ring signatures to extendable threshold ring signatures is outlined. Implementation benchmarks demonstrate practicality. **Keywords:** Threshold Ring Signatures, Anonymity, Extendability.

2 Key Concepts

2.1 Ring Signatures

Ring signatures are a type of digital signature that can be performed by any member of a group of users that each have keys. Therefore, a message signed with a ring signature is endorsed by someone in a particular set of people. One of the security properties of a ring signature is that it should be computationally infeasible to determine which of the group's members' keys was used to produce the signature. Ring signatures are an extension of group signatures, offering the advantage of not requiring a trusted group manager. Instead, signers generate their own key pairs and form groups in an ad-hoc manner.
A ring signature scheme, denoted as RS, is defined by four probabilistic polynomial time algorithms: Setup, KeyGen, Sign, and Verify.

- **Setup**$(1^\lambda) \rightarrow$ **pp:** Takes a security parameter λ and outputs a set of public parameters `pp`. The public parameters are implicitly input to all subsequent algorithms.
- **KeyGen()** $\rightarrow$ **(pk, sk):** Produces a key pair `(pk, sk)`.
- **Sign**$(\mu, \{\mathbf{pk}_j\}_{j\in\mathcal{R}}, \mathbf{sk}_i) \rightarrow \sigma$**:** 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 $\{\mathrm{pk}_j\}_{j\in R}$). Outputs a signature σ.
- **Verify**$(\mu, \{\mathbf{pk}_i\}_{i\in\mathcal{R}}, \sigma) \rightarrow$ `accept/reject`**:** Takes a message, a set of public keys of the users within a ring, and a signature σ. It outputs `accept` or `reject`, reflecting the validity of the signature σ on the message μ with respect to the ring $\mathcal{R}$.

A secure ring signature scheme should satisfy two properties: unforgeability and anonymity. Unforgeability ensures that no adversary can produce a verifying signature without knowledge of at least one signing key corresponding to a public verification key in the ring. Anonymity ensures that no adversary can tell from a signature which ring member produced it.

2.2 Threshold Ring Signatures

Threshold ring signatures extend the concept of ring signatures by allowing any t signers to anonymize themselves among a ring of signers R where $t \leq |R|$. A verifier can then check that at least t signers in the ring R signed the same message. A non-interactive threshold ring signature scheme is defined as a tuple of five probabilistic polynomial time algorithms (Setup, KeyGen, Sign, Combisign, Verify). The algorithm Sign now outputs a partial signature σ_i for signer i, and Verify

now additionally takes the threshold t as input. The algorithm Combisign combines t partial signatures into a single threshold signature.

The Combisign algorithm is used in non-interactive threshold ring signature schemes and can be run by any third party, as it does not require any signers' secrets. The Join algorithm is also used in interactive threshold ring signature schemes.

- **Combisign**$(\{\sigma_i\}_{i\in\mathcal{S}\subseteq\mathcal{R}}) \rightarrow \sigma$**:** Takes partial signatures $\{\sigma_i\}_{i\in\mathcal{S}}$ from $|S| = t$ signers, and outputs a combined signature σ.
- **Join**$(\mu,$ **{pk**$_j$**}**$_{j\in\mathcal{R}},$ **sk,** $\sigma) \rightarrow \sigma'$**:** 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 $t'+1$.

2.3 Signatures of Knowledge

Signatures of Knowledge (SoKs) are a form of digital signatures. Instead of using a public key, they use a statement from a specific language related to computational complexity. The key idea is that one may not create a new signature without knowing a certain piece of information, the witness, related to the statement.
In the context of SoKs, the instance (or statement) ϕ is a problem that we are trying to prove knowledge about. It is a publicly known claim that we are making. The witness w, on the other hand, is a piece of secret information that proves the statement is true. In the following definitions, $\mathscr{R}$ is a binary relation, $(\phi, w) \in \mathscr{R}$.
Setup$(1^\lambda, \mathscr{R}) \rightarrow pp$: Takes a security parameter λ and a binary relation $\mathscr{R}$ and returns public parameters pp. The input pp is implicit to all subsequent algorithms.
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.
Verify$(\mu, \phi, \sigma) \rightarrow$ {accept, reject}: Takes as input a message μ, a statement ϕ, and a signature σ. Outputs accept if the signature is valid, reject otherwise.
SimSetup$(1^\lambda, \mathscr{R}) \rightarrow (pp, td)$: A simulated setup which takes as input a relation $\mathscr{R}$ and returns public parameters pp and a trapdoor td.
SimSign$(td, \mu, \phi) \rightarrow \sigma'$: A simulated signing algorithm that takes as input a trapdoor td, a message μ, and a statement ϕ and returns a simulated signature σ'.

Example: Discrete Logarithm Problem Statement ϕ: "There exists an x such that $a = g^x$ in a given group G". Witness w: Is the x, the discrete logarithm of a to base g. Knowing x, one can prove they know the discrete logarithm of a to base g without revealing x (e.g. $\Sigma -$ Protocols).

3 Methodology

3.1 Extendable Ring Signatures

Ring signatures let a person sign a message anonymously within a group. Existing methods do not allow adding more members to the group after signing. Extendability in ring signatures means the ability to add members post-signature while preserving anonymity. The paper introduces and implements this concept. The Extend algorithm is used in Extendable Ring Signature (ERS) schemes.
Extend $(\mu, \{\mathrm{pk}_i\}_{i\in\mathcal{R}}, \sigma, \{\mathrm{pk}_j\}_{j\in\mathcal{R}'}) \rightarrow \sigma'$: 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 $\mathcal{R} \cup \mathcal{R}'$.

In an ERS scheme, if Extend can be iteratively applied to extend a signature, a simple instantiation is possible. The Sign operation produces a signature for the singleton ring {pk} with the signer's public key pk. Extend is applied only on singleton extension rings ($|\mathcal{R}'| = 1$), allowing the signer to iteratively apply Extend with a single additional public key, extending a signature for the singleton ring to any desired ring.

ERS Process Algorithm Ladders of rings are used, represented as tuples $\mathrm{lad} = \left(i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)}\right)$. Here, i is the signer identity, and $\mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)}$ are sets of signer identifiers. An algorithm is also employed, $\mathtt{Process}(\mu, \mathrm{L_{keys}}, \mathrm{lad})$, which processes a ladder on a given message μ using keys from $\mathrm{L_{keys}}$. The Process algorithm 1 signs μ using sk_i under the initial ring $\mathcal{R}^{(1)}$ and extends the signature to subsequent rings. The output is an extendable ring signature σ. For correctness, any possibly extended signature σ output by Process must verify for the given message under the final ring $\mathcal{R}^{(l)}$, which is the union of all initial, ladder's rings $\mathcal{R}^{(i)}$, as seen in the Process algorithm.

Definition 3.1 (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 $lad = \left(i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \cdots, \mathcal{R}^{(l)}\right)$ where $i \in \mathcal{R}^{(1)}$ and $l > 0$, it must hold that:*

$$\Pr\left[\begin{array}{l|l} \text{Verify}(\mu, \{pk_j\}_{j\in\mathcal{R}}, \sigma) & \mathcal{R} = \mathcal{R}^{(1)} \cup \ldots \cup \mathcal{R}^{(l)} \\ = \texttt{accept}\ OR\ \sigma = \perp & \boldsymbol{pp} \leftarrow \boldsymbol{ERS}.Setup(1^\lambda) \\ & \boldsymbol{L}_{keys} \leftarrow \{(\boldsymbol{pk}_j, \boldsymbol{sk}_j) \leftarrow \boldsymbol{ERS}.KeyGen()\}_{j\in\mathcal{R}} \\ & \sigma \leftarrow \boldsymbol{ERS}.Process(\mu, \boldsymbol{L}_{keys}, lad) \end{array}\right] = 1$$

Algorithm 1: ERS.Process($\mu, L_{\text{keys}}, lad$)

Input: Message μ, Key list L_{keys}, Ladder lad
Output: Extendable ring signature σ

Parse lad as $(i, \mathcal{R}^{(1)}, \mathcal{R}^{(2)}, \ldots, \mathcal{R}^{(l)})$
if $i \notin \mathcal{R}^{(1)}$ **then**
  return $\perp$ `// Make sure all public keys are in` L_{keys}
for $j \in \mathcal{R}^{(1)} \cup \cdots \cup \mathcal{R}^{(l)}$ **do**
  if $(j, \text{pk}_j, \cdot) \notin L_{keys}$ **then**
    return $\perp$ `// Make sure the signer's secret key is available in` L_{keys}
if $\text{sk}_i = \perp$ **then**
  return $\perp$ `// Make sure` sk_i `is not corrupted`
$\sigma^{(1)} \leftarrow \text{ERS.Sign}(\mu, \{\text{pk}_j\}_{j\in\mathcal{R}^{(1)}}, \text{sk}_i)$
for $l' \in [2, \ldots, l]$ **do**
  $\mathcal{R}^{(l')} \leftarrow \mathcal{R}^{(l')} \cup \mathcal{R}^{(l'-1)}$ `// Enforce rings form an increasing chain`
  $\sigma^{(l')} \leftarrow \text{ERS.Extend}(\mu, \{\text{pk}_j\}_{j\in\mathcal{R}^{(l'-1)}}, \sigma^{(l'-1)}, \{\text{pk}_j\}_{j\in\mathcal{R}^{(l')}})$
$\sigma \leftarrow \sigma^{(l)}$
return σ

Figure 1: The `Process` algorithm for extendable ring signatures.

Definition 3.2 (Secure ERS) *An extendable ring signature scheme is secure if it satisfies correctness (Definition 3.1), unforgeability (Definition 3.3), anonymity (Definition 3.4), and some notion of anonymous extendability (described below).*

Definition 3.3 (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.:*

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

OSign Oracle The OSign oracle is used to sign a message μ under a ring $\mathcal{R}$ using the secret key $\mathbf{sk}_i$ of a signer i. It checks that all keys in the query are initialized and that i is not in the list of corrupted keys and is in the ring $\mathcal{R}$. If these conditions are met, it signs the message and adds the tuple $(\mu, \mathcal{R}, i)$ to the list of signed messages.

$$\text{OSign}(\mu, \mathcal{R}, i) \rightarrow \sigma$$

OCorrupt Oracle The OCorrupt oracle is used to corrupt a signer i. If i has been initialized and its secret key $\mathbf{sk}_i$ is not $\perp$, it adds i to the list of corrupted keys and returns the public and secret keys of i. A signer i gets corrupted if $\mathcal{A}$ has used the OCorrupt oracle, learning their skecret key. They can also get corrupted if $\mathcal{A}$ gives a custom public key to them.

$$\text{OCorrupt}(i) \rightarrow (\text{pk}_i, \text{sk}_i) \text{ or } \perp$$

OKeyGen Oracle The OKeyGen oracle is used to generate a key for a signer i. If the public key pk is $\perp$, it generates a new key pair $(\text{pk}_i, \text{sk}_i)$ and adds the tuple $(i, \text{pk}_i, \mathbf{sk}_i)$ to the list of keys. If pk is not $\perp$, it adds i to the list of corrupted keys, sets pk_i to pk, and adds the tuple $(i, \text{pk}_i, \perp)$ to the list of keys.

$$\text{OKeyGen}(i, \text{pk}) \rightarrow \text{pk}_i$$

Algorithm 2: $\text{Exp}^{cmEUF}_{\mathcal{A},\mathbf{ERS}}(\lambda)$

$L_{\text{keys}}, L_{\text{corr}}, L_{\text{sign}} \leftarrow \varnothing$
$\text{pp} \leftarrow \text{ERS.Setup}(1^\lambda)$
$O \leftarrow \{O\text{ Sign}, O\text{ KeyGen}, O\text{ Corrupt}\}$
$(\mu^*, R^*, \sigma^*) \leftarrow \mathcal{A}^O(\text{pp})$ `// rule out trivial wins due to ring expansion`
if $\exists(\mu^*, R, \cdot) \in L_{sign}$ *s.t.* $\{pk_j\}_{j\in R} \subseteq \{pk_j\}_{j\in R^*}$ **then**
  return lose `// rule out trivial wins due to ring expansion`
end
if $\{pk_j\}_{j\in R^*} \cap \{pk_j\}_{j\in L_{corr}} \neq \varnothing$ **then**
  return lose `// rule out trivial wins due to key duplication`
end
if *Verify*$(\mu^*, \{pk_j\}_{j\in R^*}, \sigma^*)$ = *reject* **then**
  return lose
end
return win

Algorithm 3: $O\,\text{Corrupt}(i)$

if $(i, \text{pk}_i, \text{sk}_i) \in L_{keys}$ *and* $\text{sk}_i \neq \perp$ **then**
  $L_{\text{corr}} \leftarrow L_{\text{corr}} \cup \{i\}$
  return $(\text{pk}_i, \text{sk}_i)$
return $\perp$ `// if` i `has not been initialized`

Algorithm 4: $O\,\text{KeyGen}(i, \text{pk})$

if $\text{pk} = \perp$ **then**
  $(\text{pk}_i, \text{sk}_i) \leftarrow \text{ERS.KeyGen}()$
  $L_{\text{keys}} \leftarrow L_{\text{keys}} \cup \{(i, \text{pk}_i, \mathbf{sk}_i)\}$
else
  $L_{\text{corr}} \leftarrow L_{\text{corr}} \cup \{i\}$
  $\text{pk}_i \leftarrow \text{pk}$
  $L_{\text{keys}} \leftarrow L_{\text{keys}} \cup \{(i, \text{pk}_i, \perp)\}$
return pk_i

Algorithm 5: $O\,\text{Sign}(\mu, \mathcal{R}, i)$

if $(i \in L_{corr} \vee i \notin R)$ **then**
  return $\perp$ `// check that all keys in the query are initialized`
end
for *all* $j \in R$ **do**
  if $(j, \text{pk}_j, \cdot) \notin L_{keys}$ **then**
    return $\perp$ `// check that all keys in the query are initialized`
  end
end
$\sigma_i \leftarrow \text{ERS.Sign}(\mu, \{\text{pk}_j\}_{j\in\mathcal{R}}, \mathbf{sk}_i)$
$L_{\text{sign}} \leftarrow L_{\text{sign}} \cup \{\mu, \mathcal{R}, i\}$
return σ

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

Definition 3.4 (Anonymity for ERS) *An extendable ring signature scheme ERS 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 $lad_0^* = (i_0, R)$, $lad_1^* = (i_1, R)$, it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.:*

$$\Pr[Exp^{ANEXT}_{\mathcal{A},\boldsymbol{ERS}}(\lambda) = win_i] \leq \frac{1}{2} + negl.$$

Definition 3.5 (Weak Anonymous Extendability for ERS) *An extendable ring signature scheme ERS is said to be weakly anonymous extendable if for all PPT adversaries A taking part in the anonymous extendability experiment (ANEXT in Figure 3) and submitting to the challenger ladders of the type $lad_0^* = (i_0, R_0^{(1)}, R_0^{(2)})$, $lad_1^* = (i_1, R_1^{(1)}, R_1^{(2)})$, it holds that the success probability of A is negligibly close to random guessing, i.e.:*

$$\Pr[Exp^{ANEXT}_{\mathcal{A},ERS}(\lambda) = win_i] \leq \frac{1}{2} + negl.$$

We consider the game scenarios associated with the properties of anonymity and weak anonymous extendability in Extendable Ring Signature (ERS) schemes:

Anonymity Game : In the anonymity game, the adversary submits two ladders, `lad0` and `lad1`, each consisting of the same ring to the challenger. Subsequently, the challenger randomly selects one of these ladders and produces a challenge signature. The adversary's objective is to correctly discern the ladder used in generating the signature. The ERS scheme is deemed to provide anonymity if the adversary's success probability of correctly identifying the ladder significantly exceeds that of random guessing. Conversely, if the adversary's success probability closely aligns with random guessing, the ERS scheme maintains anonymity. The trivial win where the adversary uses the querried signers of the corrupt oracle is excluded, as they can distinguish with probability 1 the signer.

Algorithm 6: $\mathrm{Exp}^{\mathrm{ANEXT}}_{\mathcal{A},\mathbf{ERS}}(\lambda)$

$b \leftarrow_R \{0,1\}$
$\mathrm{L_{keys}}, \mathrm{L_{corr}}, \mathrm{L_{sign}} \leftarrow \varnothing$
$\mathrm{pp} \leftarrow \mathrm{ERS.Setup}\left(1^\lambda\right)$
$O \leftarrow \{O$ Sign, OKeyGen, O Corrupt $\}$
$(\mu^*, \mathrm{lad}_0^*, \mathrm{lad}_1^*) \leftarrow \mathcal{A}^O(\mathrm{pp})$
$\bar{\sigma} \leftarrow \mathrm{Chal}_b\left(\mu^*, \mathrm{lad}_0^*, \mathrm{lad}_1^*\right)$
$b^* \leftarrow \mathcal{A}^O(\bar{\sigma})$
if $i_0 \in \mathrm{L}_{corr} \vee i_1 \in \mathrm{L}_{corr}$ **then**
  return lose
if $b^* \neq b$ **then**
  return lose
return win

Algorithm 7: $\mathrm{Chal}_b\left(\mu^*, \mathrm{lad}_0^*, \mathrm{lad}_1^*\right)$

parse $\mathrm{lad}_0^* = \left(i_0, \mathcal{R}_0^{(1)}, \ldots, \mathcal{R}_0^{(l_0)}\right)$
parse $\mathrm{lad}_1^* = \left(i_1, \mathcal{R}_1^{(1)}, \ldots, \mathcal{R}_1^{(l_1)}\right)$
if $i_0, i_1 \in \mathrm{L}_{corr}$ **then**
  return $\perp$
$\sigma_0 \leftarrow \mathrm{Process}\left(\mu, \mathrm{L_{keys}}, \mathrm{lad}_0^*\right)$
$\sigma_1 \leftarrow \mathrm{Process}\left(\mu, \mathrm{L_{keys}}, \mathrm{lad}_1^*\right)$
if $\sigma_0 = \perp$ *or* $\sigma_1 = \perp$ **then**
  return $\perp$
if $\mathcal{R}_0^{(1)} \cup \cdots \cup \mathcal{R}_0^{(l_0)} \neq \mathcal{R}_1^{(1)} \cup \cdots \cup \mathcal{R}_1^{(l_1)}$ **then**
  return $\perp$
$\bar{\sigma} \leftarrow \sigma_b$
return $\bar{\sigma}$

Figure 3: Anonymity and Anonymous Extendability for Extendable Ring Signatures. The oracles OSign, OKeyGen and OCorrupt are defined in Figure 2.

Weak Anonymous Extendability Game : In the weak anonymous extendability game, the adversary submits two ladders, denoted as $(i_0, R_0^{(1)}, R_0^{(2)})$ and $(i_1, R_1^{(1)}, R_1^{(2)})$, to the challenger. The condition is that the union of $R_i^{(j)}$ yields the same ring R. Subsequently, the challenger randomly selects one of these ladders and generates an extended ring signature. The adversary's task is to correctly identify the ladder used in generating the extended signature. Analogous to the anonymity game, if the adversary's success probability of correctly identifying the ladder significantly surpasses random guessing, the adversary wins the game. Conversely, if the adversary's success probability closely approximates random guessing, the ERS scheme is deemed to possess weak anonymous extendability.

Unforgeability Game : $\mathcal{A}$ win if they create a signature σ^* for the message μ^* that verifies under ring R^* having access to the three oracles. Some trivial wins must be excluded. A trivial win is when $\mathcal{A}$ has queried the oracle and has retrieved a signature for μ^* under the ring R where $R \subseteq R^*$. $\mathcal{A}$ can then use the Extend algorithm to make the signature valid under R^*. Another trivial win is if there exist public keys in R^* that are corrupted. This means that $\mathcal{A}$ knows secret keys in R^* or has given custom public keys for identifiers in R^* so they gain an "illegal" advantage.

Definition 3.6 (Strong Anonymous Extendability for ERS) *An extendable ring signature scheme is said to be strongly anonymous extendable if for all PPT adversaries A taking part in the anonymous extendability experiment (Figure 3), it holds that:*

$$\Pr[Exp^{ANEXT}_{\mathcal{A},ERS}(\lambda) = win_i] \leq \frac{1}{2} + negl.$$

Strong extendability implies that extending a ring signature is indistinguishable from obtaining a fresh ring signature, preventing adversaries from distinguishing between the two in the strong extendability game when extending once ($l_0 = 1$) or more ($l_1 > 1$).

Extendable Ring Signatures and SoK Signing works as the algorithm states. We explain Verify and Extend:
Extend: The extender picks a trapdoor td', computes $h' = g^{td'}$, and a knowledge signature π' for the key pk' that it needs to add to the ring. The tuple (h', pk', p') is added to the ring, and td becomes $td - td'$.
Verify: It is assumed that a signature is generated as follows: Initially, the ring contains a pk and an sk used for the signature. Then, `Extend` is called multiple times to add more pks to the ring. For verification to occur, it must first be true:

$$H = g^{td} \prod_{i \in \mathcal{R}'} h_i$$

For example, assume that the total number of identifiers in the ring is 4. Assume that the first identifier (0) initially signs the message by calling `ERS.Sign` and computing $h_0 = Hg^{-td_0}$. The remaining identifiers sequentially call `Extend` and compute $h_i = g^{td_i}, i \in \{1,2,3\}$. The final td will have become equal to $td = td_0 - td_1 - td_2 - td_3$ due to the fact that it is updated as $td \leftarrow td - td'$. For Verification, we have:

$$g^{td} \prod_{i \in \mathcal{R}'} h_i = g^{td_0 - td_1 - td_2 - td_3} H g^{-td_0 + td_1 + td_2 + td_3} = H$$

Also, for the signature to be verified, all SoKs produced by the participants must be verified using the `Sok.Verify` algorithm.

Algorithm 8: ERS . Setup $\left(1^{\lambda}\right) \mapsto$ pp

$(p, g, \mathbb{G}) \leftarrow \text{GroupGen}(1^{\lambda})$;
$\text{SoK.pp} \leftarrow \text{SoK.Setup}\left(1^{\lambda}, \mathscr{R}_{\mathbb{G}}\right)$;
$H \leftarrow_R \mathbb{G}$;
return pp := (SoK.pp, g, H);

Algorithm 9: ERS . KeyGen() $\mapsto$ (pk, sk)

$\text{sk} \leftarrow \mathbb{Z}_p$;
$\text{pk} := g^{\text{sk}}$;
return (sk, pk);

Algorithm 10: ERS . Sign(μ, sk) $\mapsto \sigma$

$\text{td} \leftarrow_R \mathbb{Z}_p$;
$h := H \cdot g^{-\text{td}}$;
$\text{nonce} \leftarrow_R \{0,1\}^{\lambda}$;
$\phi := (h, \text{pk})$;
$w := \text{sk}$;
$\pi \leftarrow \text{SoK} \cdot \text{Sign}((\text{ nonce},\mu), \mathscr{R}, \phi, w)$;
$P := \{(h, \text{pk}, \pi)\}$;
return $\sigma :=$ (nonce, td, P);

Algorithm 11: ERS . Verify $\mu, \left(\left\{\text{pk}_j\right\}_{j \in \mathcal{R}}, \sigma\right) \mapsto$ accept / reject

parse $\sigma = (\text{ nonce, td, } P = \{(h_i, \text{pk}_i, \pi_i)\}_{i \in \mathcal{R}'})$;
if $H \neq g^{\text{td}} \cdot \prod_{i \in \mathcal{R}'} h_i$ **then**
  return reject;
if $\left\{\text{pk}_j\right\}_{j \in \mathcal{R}} \neq \{\text{pk}_i\}_{i \in \mathcal{R}'}$ **then**
  return reject;
for $i \in \mathcal{R}'$ **do**
  $\phi_i := (h_i, \text{pk}_i)$;
  if *SoK.Verify* $((\text{ nonce, } \mu), \mathscr{R}, \phi_i, \pi_i) =$ *reject* **then**
    return reject;
return accept;

Algorithm 12: ERS . Extend $\left(\mu, \left\{\text{pk}_j\right\}_{j \in \mathcal{R}}, \sigma, \text{pk}\right) \mapsto \sigma'$

if $\text{pk} \in \left\{\text{pk}_j\right\}_{j \in \mathcal{R}}$ **then**
  return $\perp$;
parse $\sigma = (\text{ nonce, td, } P = \{(h_i, \text{pk}_i, \pi_i)\}_{i \in \mathcal{R}'})$;
$\text{td}' \leftarrow_R \mathbb{Z}_p$// pick a trapdoor;
$\text{td} \leftarrow \text{td} - \text{td}' \pmod{p}$;
$h := g^{\text{td}'}$// to simulate using the trapdoor;
$\phi := (h, \text{pk}), w := \text{td}'$;
$\pi \leftarrow \text{SoK} \cdot \text{Sign}((\text{ nonce},\mu), \mathscr{R}, \phi, w)$;
add (h, pk, π) to P// update the signature;
return $\sigma' :=$ (nonce, td, P);

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

3.2 Same-Message Linkable Extendable Ring Signatures

A Same-Message Linkable Ring Signature Scheme (SMLRS) is a type of ring signature scheme that allows anyone to determine if two signatures on the same message were created by the same signer. This is useful because it allows the creation of threshold ring signatures by simply concatenating multiple same-message linkable ring signatures. Furthermore, SMLRS can be used to create other types of signature schemes with varying degrees of linkability.
We then introduce the concept of Extendable Same-Message Linkable Ring Signatures (SMLERS), which is a new primitive that builds on our previous ERS construction. A SMLERS is defined by six algorithms: Setup, KeyGen, Sign, Verify, Extend, and Link. The first five are inherited from extendable ring signatures, and the Link algorithm allows any verifier to determine whether two signatures on a particular message were produced by the same signer.

For the Link algorithm, we have:
Link(μ, (σ_0, $\{pk_j\}_{j \in \mathcal{R}_0}$), ($\sigma_1$, $\{pk_j\}_{j \in \mathcal{R}_1}$)) $\rightarrow$ {linked, unlinked}: Takes a message μ, two signatures (σ_0, σ_1) and two sets of public keys belonging to members of the rings R_0, R_1. It outputs linked if σ_0 and σ_1 were produced by the same signer, and unlinked otherwise.

Definition 3.7 (Cross-Signer Correctness for SMLERS) *For all security parameters $\lambda \in \mathbb{N}$, for any message $\mu \in \{0,1\}^*$, and for any two ladders $lad_0 = (i_0, R_0^{(1)}, \ldots, R_0^{(l_0)})$, $lad_1 = (i_1, R_1^{(1)}, \ldots, R_1^{(l_0)})$ where $i_0 \in R_0^{(1)}$, $i_1 \in R_1^{(1)}$, $l_0 > 0$,*

Algorithm 13: $\mathrm{Exp}^{\mathrm{omlink}}_{\mathcal{A},\mathrm{SMLERS}}(\lambda)$

$\mathrm{pp} \leftarrow \mathrm{Setup}\left(1^\lambda\right)$;
$\mathrm{L}_{\mathrm{keys}}, \mathrm{L}_{\mathrm{corr}}, \mathrm{L}_{\mathrm{sign}} \leftarrow \varnothing$;
$O \leftarrow \{O$ Sign, O KeyGen, O Corrupt $\}$;
$\left(\mu^*, \{(\sigma_k^*, \mathcal{R}_k^*)\}_{k\in[1\ldots,t]}\right) \leftarrow \mathcal{A}^O(\mathrm{pp})$;
// $\mathcal{A}$ has never seen a signature for the message and a subring of the forgery rings
if $\exists\,(\mu^*, \mathcal{R}, \cdot) \in \mathrm{L}_{sign}$ *s.t.* $\mathcal{R} \subseteq \mathcal{R}_k^*$ *for some* $k \in [1,\ldots,t]$ **then**
  return lose
// $\mathcal{A}$ holds at most $t-1$ secret keys, among the keys identified by the forgery rings
if $|(\mathcal{R}_1^* \cup \cdots \cup \mathcal{R}_t^*) \cap \mathrm{L}_{corr}| \geq t$ **then**
  return lose
// All the signatures in the forgery verify (for the same message)
if $\exists k \in [1\ldots,t]$ *s.t. Verify* $\left(\mu^*, \left\{\mathrm{pk}_j\right\}_{j\in\mathcal{R}_k^*}, \sigma_k^*\right) =$ *reject* **then**
  return lose
// All signatures in the forgery are unlinked (here $k,l \in [1,\ldots,t]$)
if $\exists k \neq l$ *s.t.* $\mathrm{Link}\left(\mu^*, \left(\sigma_k^*, \left\{\mathrm{pk}_j\right\}_{j\in\mathcal{R}_k^*}\right), \left(\sigma_l^*, \left\{\mathrm{pk}_j\right\}_{j\in\mathcal{R}_l^*}\right)\right) =$ ***linked*** **then**
  return lose
return win;

Figure 5: Security experiment for same-message one-more linkability. The signing, key generation and corruption oracles are as defined in Figure 3, except that the algorithms for ERS are replaced with the corresponding algorithms for SMLERS. We recall that the list $\mathbf{L}_{\mathrm{sign}}$ of sign-queries contains elements of the form $(\mu, \mathcal{R}, i)$.

$l_1 > 0$, and $i_0 \neq i_1$, it must hold that:

$$\Pr\left[\begin{array}{l|l} \mathrm{Link}(\mu, (\sigma_0, \{pk_j\}_{j\in\mathcal{R}_0}), & \mathcal{R}_0 = \mathcal{R}_0^{(1)} \cup \ldots \cup \mathcal{R}_0^{(l_0)} \\ (\sigma_1, \{pk_j\}_{j\in\mathcal{R}_1})) \rightarrow \mathrm{unlinked} & \mathcal{R}_1 = \mathcal{R}_1^{(1)} \cup \ldots \cup \mathcal{R}_1^{(l_1)} \\ & \boldsymbol{pp} \leftarrow \mathit{Setup}(1^\lambda) \\ & \boldsymbol{L}_{keys} \leftarrow \{(\boldsymbol{pk}_j, \boldsymbol{sk}_j) \leftarrow \mathit{KeyGen}()\}_{j\in R_0\cup R_1} \\ & \sigma_0 \leftarrow \boldsymbol{ERS}.\mathit{Process}(\mu, \boldsymbol{L}_{keys}, lad_0) \\ & \sigma_1 \leftarrow \boldsymbol{ERS}.\mathit{Process}(\mu, \boldsymbol{L}_{keys}, lad_1) \end{array}\right] = 1 - negl$$

Definition 3.8 (Secure SMLERS) *A same-message linkable extendable ring signature scheme (SMLERS) is secure if it satisfies correctness, same-message one-more linkability (Definition 3.9, which implies unforgeability), and cross-message unlinkability (Definition 3.10).*

Definition 3.9 (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 ($Exp^{omlink}_{\mathcal{A},SMLERS}(\lambda)$ depicted in Figure 5), it holds that:* $\Pr[Exp^{omlink}_{\mathcal{A},SMLERS}(\lambda) = win] \leq negl$.

Same-Message One-more Linkability Game Same-Message One-more Linkability is a property that ensures that an adversary cannot create multiple unlinked signatures for the same message. The Same-Message One-More Linkability Game works as follows: The adversary queries the three divinations and generates one message (μ^*) and t signature-ring pairs $(\sigma^*, \mathcal{R}^*)$. It is also assumed that the adversary has at most $t-1$ secret keys by having them corrupted. If even two of these signatures are linked, then the adversary loses the game; otherwise, they win. It is desired, that the adversary's probability of success is negligible. Some trivial wins are excluded. For example, if a signature is generated for a message that has already been queried in the oracle, we have a failure by default. If at least t signatures are corrupted, it means that the adversary knows all secret keys, so they can construct t unlinked signatures.

Definition 3.10 (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 ($Exp^{cmulink}_{\mathcal{A},SMLERS}(\lambda)$ depicted in Figure 6), it holds that the success probability of $\mathcal{A}$ is negligibly close to random guessing, i.e.,:* $\Pr[Exp^{omlink}_{\mathcal{A},SMLERS}(\lambda) = win] \leq \frac{1}{2} + negl$.

Algorithm 14: $Exp^{\text{cmunlink}}_{\mathcal{A},\mathbf{LRS}}(\lambda)$

$b \leftarrow_R \{0,1\}, \mathrm{L}_{\text{keys}}, \mathrm{L}_{\text{corr}}, \mathrm{L}_{\text{sign}} \leftarrow \varnothing$;
$\text{pp} \leftarrow \text{Setup}\left(1^\lambda\right)$;
$O \leftarrow \{O$ Sign, O KeyGen, O Corrupt $\}$;
$\left(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\}\right) \leftarrow \mathcal{A}^O(\text{pp})$;
$(\bar{\sigma}_0, \bar{\sigma}_1) \leftarrow \text{Chal}_b\left(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\}\right)$;
$b^* \leftarrow \mathcal{A}^O\left(\bar{\sigma}_0, \bar{\sigma}_1\right)$;
`// Rule out corruption of challenge identities`
if $i_0 \in \mathrm{L}_{corr} \vee i_1 \in \mathrm{L}_{corr}$ **then**
  return lose
`// Rule out trivial attacks using Link`
if $\mu_0 = \mu_1$ **then**
  return lose
`// Rule out trivial attacks using Link`
if $(\mu_0, \cdot, i_0) \in \mathrm{L}_{sign} \vee (\mu_0, \cdot, i_1) \in \mathrm{L}_{sign} \vee (\mu_1, \cdot, i_0) \in \mathrm{L}_{sign} \vee (\mu_1, \cdot, i_1) \in \mathrm{L}_{sign}$ **then**
  return lose
if $b^* \neq b$ **then**
  return lose
return win;

Algorithm 15: $\text{Chal}_b\left(\{\mu_0, \mathcal{R}_0, i_0\}, \{\mu_1, \mathcal{R}_1, i_1\}\right)$

`// The challenge identities must be uncorrupted`
if $i_0 \in \mathrm{L}_{corr} \vee i_1 \in \mathrm{L}_{corr}$ **then**
  return $\perp$
`// One identity needs to be in both rings`
if $i_0 \notin \mathcal{R}_0 \cap \mathcal{R}_1 \vee i_1 \notin \mathcal{R}_1$ **then**
  return $\perp$
`// Signing keys must exist`
if $\nexists \left(i_0, \text{pk}_{i_0}, \text{sk}_{i_0}\right) \in \mathrm{L}_{keys}$ **then**
  return $\perp$
if $\nexists \left(i_1, \text{pk}_{i_1}, \text{sk}_{i_1}\right) \in \mathrm{L}_{keys}$ **then**
  return $\perp$
`// Generate a signature`
$\bar{\sigma}_0 \leftarrow \mathbf{LRS}.\text{Sign}\left(\mu_0, \{\text{pk}_i\}_{i \in \mathcal{R}_0}, \mathbf{sk}_{i_0}\right)$;
`// Generate the second signature according to the experiment's bit` b
$\bar{\sigma}_1 \leftarrow \text{LRS.Sign}\left(\mu_1, \{\text{pk}_i\}_{i \in \mathcal{R}_1}, \mathbf{sk}_{i_b}\right)$;
return $(\bar{\sigma}_0, \bar{\sigma}_1)$;

Figure 6: Cross-message unlinkability. The signing, key generation and corruption oracles are as defined in Figure 3, except that the ERS algorithms are substituted with the respective **SMLERS** variants.

Cross-Message Unlinkability Game In the Cross-Message Unlinkability game, the adversary generates two tuples: $(\mu_0, \mathcal{R}_0, i_0)$ and $(\mu_1, \mathcal{R}_1, i_1)$. The challenger then creates a signature σ_0 using the first tuple. For the second signature, the challenger randomly decides to use either σ_0 or σ_1, but in both cases, the message μ_1 and the ring $\mathcal{R}_1$ are used. The random choice lies in which secret key is used to sign the message. The adversary's task is to determine who signed the second signature. The SMLERS scheme is considered to satisfy the property of `Cross-Message Unlinkability` if the adversary's success probability is negligibly close to that of random guessing. This ensures that it is computationally infeasible for the adversary to link signatures across different messages. Trivial wins of $\mathcal{A}$ are excluded. A trivial win is when the identifiers i_0 and i_1 are corrupted so $\mathcal{A}$ knows their secret keys and can generate σ_0 and σ_1. Another trvial win is if $\mu_0 = \mu_1$. In this case, $\mathcal{A}$ could easily use the Link algorithm to simply check if σ_0 and σ_1 are produced by the same signer. Finally, we exclude the case in which $\mathcal{A}$ has generated signatures in μ_0 or μ_1 using i_0 or i_1.

3.3 Extendable Threshold Ring Signatures

An extendable threshold ring signature scheme allows parties to collectively produce a signature on a message μ for a ring $\mathcal{R}$, revealing that at least t out of $\mathcal{R}$ potential signers in the ring participated, while maintaining their anonymity. Additionally, it offers flexibility, allowing non-interactive combination of two threshold signatures for the same message and ring, resulting in a new threshold signature with a threshold equal to the total number of unique contributors and extendability, enabling the transformation of a signature for a given ring into a signature for a larger ring while maintaining the same threshold.

A non-interactive extendable threshold ring signature scheme (ETRS) is defined as a tuple of six PPT algorithms $ETRS = (Setup, KeyGen, Sign, Verify, Combine, Extend)$, where the public parameters pp produced by $Setup$ are implicitly available to all other algorithms:
Setup$(1^\lambda) \rightarrow pp$: Takes a security parameter λ and outputs a set of public parameters pp.
KeyGen$() \rightarrow (pk, sk)$: Generates a new public and secret key pair.
Sign$(\mu, \{pk_i\}_{i \in R}, sk) \rightarrow \sigma$: Returns a signature with threshold $t = 1$ using the secret key sk corresponding to a public key pk_i with $i \in R$.
Verify$(t, \mu, \{pk_i\}_{i \in R}, \sigma) \rightarrow$ `accept/reject`: Verifies a signature σ for the message μ against the public keys $\{pk_i\}_{i \in R}$ with threshold t.

Combine$(\mu, \sigma_0, \sigma_1, \{pk_i\}_{i\in R}) \to \sigma'$: Combines two signatures σ_0, σ_1 for the same ring 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.
Extend$(\mu, \sigma, \{pk_i\}_{i\in R}, \{pk_i\}_{i\in R'}) \to \sigma'$: Extends the signature σ with threshold t for the ring R into a new signature σ' with threshold t for the larger ring $R \cup R'$.

Definition 3.11 (Correctness for ETRS) *For correctness, we require that for all ladders* `lad`*, the signature returned by* `Process(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 $R :=$* `lad.sr` *of public-key identifiers, and for any chosen threshold value $1 \le t \le |\mathcal{R}|$, it holds:*

$$\Pr\left[\begin{array}{l|l} \text{Verify}(t, \mu, \{pk_i\}_{i\in\mathcal{R}}, \sigma) & \boldsymbol{pp} \leftarrow \mathit{Setup}(1^\lambda) \\ = \texttt{accept}\ \mathit{OR}\ \sigma = \perp & \boldsymbol{L}_{keys} \leftarrow \{\mathit{KeyGen}()\ \}_{j\in \texttt{lad.sr}} \\ & (\sigma, t, \mathcal{R} \leftarrow \texttt{Process}(\mu, \boldsymbol{L}_{keys}, \texttt{lad})) \end{array}\right] = 1$$

Generalized Ladders The concept of ladders is generalized to support sequences of various actions leading to a valid threshold ring signature. These actions include Sign (producing a signature with a specific ring and signer), Combine (combining two signatures under the same ring) and Extend (extending an existing signature to a larger ring). The Process algorithm is introduced to handle these operations, producing the final signature, corresponding threshold, and the validating ring. The `lad.sr` represents the union of all identities and rings in `lad`. What the Process algorithm achieves is that the final signature, which results from successive applications of the Sign, Combine and Extend actions and is ETRS, must be verified with probability 1.

Definition 3.12 (Secure ETRS) *An extendable threshold ring signature scheme is secure if it satisfies correctness (Definition 3.11), unforgeability (Definition 3.13), anonymity (Definition 3.14), and some notion of anonymous extendability.*

Definition 3.13 (Unforgeability for ETRS) *An extendable threshold ring signature scheme ETRS is said to be unforgeable if, for all thresholds t and all PPT adversaries A, the success probability in the cmEUF experiment in Figure 8 is $\Pr[Exp^{cmEUF}_{\mathcal{A},ETRS}(\lambda) = win] \le negl$.*

Unforgeability Game In Figure 8, the unforgeability game is defined as follows: The adversary interacts with the oracles to generate a threshold tuple consisting of the parameters (t^*, m^*, R^*, p^*), where t^* represents the threshold, μ^* denotes the message, $\mathcal{R}^*$ represents the ring, and σ^* is the signature. Let q be the number of signatures that the adversary has honestly generated for message μ^* by querying the oracles, and let L_{corr} be the set of secret keys learned by the adversary through corruption. The game checks whether the sum of the known secret keys and q exceeds or equals the threshold, t. If this condition is met, it implies that the adversary knows at least q honest signatures and at least $t - q$ secret keys, obtained via oracle queries, so at least tq valid signatures, and as a result they may create the final valid signature via `Combine`, from the t valid partial signatures, which is a trivial win that must be excluded. The adversary wins the game if the tuple is valid; otherwise, they lose.
Two edge cases can be obtained in the first if-statement of the algorithm:

- $q = t$: In this case, the adversary can query the oracle with the form $(m, R_i, \cdot)$ where R_i are elements of a partition of R into t independent sets. Thus, the adversary can make t partial signatures that are valid, combine them, and the final signature σ^* is valid.
- $|\mathcal{R}^* \cap \mathrm{L}_{\text{corr}}| = t$: In this case, the adversary queries the oracle corrupting t secret keys so they create t valid partial signatures, combine them, and the final signature σ^* is valid.

Definition 3.14 (Anonymity for ETRS) *An extendable threshold ring signature scheme is anonymous if, for all PPT adversaries A taking part in the anonymous extendability experiment (ANEXT in Figure 9) and submitting to the challenger two ladders with the specified structure, it holds that the success probability of A is negligibly close to random guessing, i.e.:*

$$\Pr[Exp^{ANEXT}_{A,ETRS}(\lambda) = win] \le \frac{1}{2} + negl.$$

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 (Sign, (R, i)), where 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 (Combine, $(l_1, l_2, \mathcal{R})$), where R is the same for all instructions in both ladders, $l_1 = 1, 2, \ldots, t-1$, and $l_2 = t, t+1, \ldots, 2t-2$.

Algorithm 16: ETRS.Process(μ, $\mathrm{L_{keys}}$, lad)

Parse lad as $\Big((\text{action}^{(1)}, \text{input}^{(1)}), \ldots, (\text{action}^{(l)}, \text{input}^{(l)})\Big)$;
for $l' \in [1, \ldots, l]$ **do**
  if $action^{(l')} = Sign$ **then**
    Parse $\text{input}^{(l')}$ as $(\mathcal{R}^{(l')}, i^{(l')})$;
    for $j \in \mathcal{R}^{(l')}$ **do**
      if $\big(j, \text{pk}_j, \cdot\big) \notin \mathrm{L}_{keys}$ **then**
        return $\perp$; // Public keys are initialized
    if $\mathbf{sk}_i = \perp$ **then**
      return $\perp$; // Signer's secret key was generated honestly
    if $i \notin \mathcal{R}^{(l')}$ **then**
      return $\perp$; // Signer is not in the ring
    $\mathcal{S}^{(l')} = \{i^{(l')}\}$;
    $\sigma^{(l')} \leftarrow \text{Sign}(\mu, \{\text{pk}_j\}_{j \in \mathcal{R}^{(l')}}, \mathbf{sk}_{i^{(l')}})$;
  else if $action^{(l')} = Combine$ **then**
    Parse $\text{input}^{(l')}$ 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 $j \in \mathcal{R}_1^{(l')}$ **do**
      if $\big(j, \text{pk}_j, \cdot\big) \notin \mathrm{L}_{keys}$ **then**
        return $\perp$; // Public keys are initialized
    if $\mathcal{R}^{(l_1^{(l')})} \neq \mathcal{R}^{(l')}$ *or* $\mathcal{R}^{(l_1^{(l')})} \neq \mathcal{R}^{(l')}$ **then**
      return $\perp$; // Combined signatures use the same ring
    $\mathcal{S}^{(l')} = \mathcal{S}^{(l_1^{(l')})} \cup \mathcal{S}^{(l_2^{(l')})}$;
    $\sigma^{(l')} \leftarrow \text{Combine}(\mu, \sigma^{(l_1^{(l')})}, \sigma^{(l_2^{(l')})}, \{\text{pk}_j\}_{j \in \mathcal{R}^{(l')}})$;
  else if $action^{(l')} = Extend$ **then**
    Parse $\text{input}^{(l')}$ as $(l_1^{(l')}, \mathcal{R}^{(l')})$;
    Retrieve $(\mathcal{R}^{(l_1^{(l')})}, \mathcal{S}^{(l_1^{(l')})}, \sigma^{(l_1^{(l')})})$;
    for $j \in \mathcal{R}^{(l')}$ **do**
      if $\big(j, \text{pk}_j, \cdot\big) \notin \mathrm{L}_{keys}$ **then**
        return $\perp$; // Public keys are initialized
    if $\mathcal{R}^{(l_1^{(l')})} \not\subseteq \mathcal{R}^{(l')}$ **then**
      return $\perp$; // The extended ring is a superset
    $\mathcal{S}^{(l')} = \mathcal{S}^{(l_1^{(l')})}$;
    $\sigma^{(l')} \leftarrow \text{Extend}(\mu, \sigma^{(l_1^{(l')})}, \{\text{pk}_j\}_{j \in \mathcal{R}_1^{(l_1^{(l')})}}, \{\text{pk}_j\}_{j \in \mathcal{R}^{(l')}})$;
return $(\sigma^{(l)}, t^{(l)}, \mathcal{R}^{(l)})$; // Signature returned by the last operation of lad, threshold, and the ring

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

Anonymity Game The adversary $\mathcal{A}$ queries the oracles and produces a message m^* and two ladders lad_0^* lad_1^*. This is the challenge, and $\mathcal{A}$ sends it to the challenger $Chal_b$. The challenger has to create two ETRS (σ_0, t_0, R_0) and (σ_1, t_1, R_1) and send back to $\mathcal{A}$ one signature randomly. The challenger win if they can understand which siganture was send back to them, with probability higher than random guessing. Trivial wins are achieved when lad_0 or lad_1 contain corrupted signers, giving $\mathcal{A}$ advantage by knowing identities that have signed the message. Another trivial win is when $\mathcal{A}$ has queried OSign oracle creating a signature signed by signer in lad_0 or in lad_1. This means that $\mathcal{A}$ could gain information about which ladder was used since he knows identifiers that have signed the message. Moreover, if for one of the signers in a ladder $\mathcal{A}$ has already produced a signature for message μ^*, they can identify all partial signatures and combine them for both ladders by discerning which signature was sent to them. Essentially, $\mathcal{A}$ initiates the `Process` algorithm independently to distinguish the signatures. In this case $\mathcal{A}$ loses the game because they try to create the signatures by accessing the

Algorithm 17: $\mathrm{Exp}^{\mathrm{cmEUF}}_{\mathcal{A},\mathrm{ETRS}}(\lambda)$

$\mathrm{L}_{\mathrm{keys}}, \mathrm{L}_{\mathrm{corr}}, \mathrm{L}_{\mathrm{sign}} \leftarrow \varnothing$;
$\mathrm{pp} \leftarrow \mathrm{Setup}\left(1^{\lambda}\right)$;
$O \leftarrow \{O$ Sign, O KeyGen, O Corrupt $\}$;
$(t^*, \mu^*, \mathcal{R}^*, \sigma^*) \leftarrow \mathcal{A}^{O}(\mathrm{pp})$;
$q \leftarrow\mid \{(\mu^*, \mathcal{R}, \cdot) \in \mathrm{L}_{\mathrm{sign}} \text{ s.t. } \mathcal{R} \subseteq \mathcal{R}^*)\} \mid$;
`// rule out attacks if` $\mathcal{A}$ `knows too many sk:s or honestly generated signatures for` μ^*
if $|\mathcal{R}^* \cap \mathrm{L}_{corr}| + q \geq t$ **then**
  return lose;
`// rule out outputs that do not verify`
if $\mathrm{Verify}\left(t, \mu^*, \left\{\mathrm{pk}_j\right\}_{j \in \mathcal{R}^*}, \sigma^*\right) = \mathit{reject}$ **then**
  return lose;
return win;

Figure 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.

oracles without any restrictions.

Definition 3.15 (Weak Anonymous Extendability for ETRS) *An extendable threshold ring signature scheme ETRS is said to be weakly anonymous extendable if, for all PPT adversaries A taking part in the anonymous extendability experiment (ANEXT in Figure 9) and submitting ladders with the specified structure, it holds that the success probability of A is negligibly close to random guessing, i.e.:*

$$\Pr[Exp^{ANEXT}_{A,ETRS}(\lambda) = win] \leq \frac{1}{2} + negl.$$

For weak anonymous extendability the adversary submits ladders with the following structure: the first t instructions are of the type $(Sign, (i, \mathcal{R}))$, where the signer identities are pairwise distinct 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 $(Combine, (l_1, l_2, \mathcal{R}))$ where the indices l_1, l_2 progressively combine (threshold) signatures on the same ring $\mathcal{R}$; the final ladder instruction is $(Extend, (l', \mathcal{R}'))$ and extends the latest threshold ring signature to a wider ring, $\mathcal{R}' \supseteq \mathcal{R}$.

Weak Anonymous Extendability Game This game is the same as Anonymity Game with an `Extend` action at the end of each ladder.

Definition 3.16 (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.:*

$$\Pr\left[ExpANEXT^{StrongAnon}_{\mathcal{A},ETRS}(\lambda) = win\right] \leq \frac{1}{2} + negl.$$

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 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.

A Compiler for ETRS It is possible to build an ETRS from any SMLERS using a compiler, which is omitted here for simplicity. The following theorem establishes the security properties of the derived ETRS scheme:

Theorem 1 *Assuming that SMLERS is a secure same-message linkable extendable ring signature scheme, the scheme ETRS = (Setup, KeyGen, Sign, Verify, Extend, Combine) is an extendable threshold ring signature scheme that satisfies the properties Correctness, Unforgeability, and Anonymity.*

Algorithm 18: $\text{Exp}^{\text{ANEXT}}_{\mathcal{A},\text{ETRS}}(\lambda)$

$b \leftarrow_R \{0,1\}$;
$\text{L}_{\text{keys}}, \text{L}_{\text{corr}}, \text{L}_{\text{sign}} \leftarrow \varnothing$;
$\text{pp} \leftarrow \text{ETRS.Setup}(1^\lambda)$;
$O \leftarrow \{O \text{ Sign}, O \text{ KeyGen}, O \text{ Corrupt }\}$;
$(\mu^*, \text{lad}_0^*, \text{lad}_1^*) \leftarrow \mathcal{A}^O(\text{pp})$;
$\bar{\sigma} \leftarrow \text{Chal}_b(\mu^*, \text{lad}_0^*, \text{lad}_1^*)$;
$b^* \leftarrow \mathcal{A}^O(\bar{\sigma})$;
if $\exists i \in \text{lad}_0^*.\textit{signers such that } i \in \text{L}_{corr}$ **then**
  return lose;
if $\exists i \in \text{lad}_1^*.\textit{signers such that } i \in \text{L}_{corr}$ **then**
  return lose;
if $\exists(\mu^*, \cdot, i) \in \text{L}_{sign}$ *for* $i \in \text{lad}_0^*.\textit{signers}$ **then**
  return lose;
if $\exists(\mu^*, \cdot, i) \in \text{L}_{sign}$ *for* $i \in \text{lad}_1^*.\textit{signers}$ **then**
  return lose;
if $b^* \neq b$ **then**
  return lose;
return win;

Algorithm 19: $\text{Chal}_b(\mu^*, \text{lad}_0^*, \text{lad}_1^*)$

if lad_0^* *or* lad_1^* *is not well-formed* **then**
  return $\perp$;
if $\exists i \in \text{lad}_0^*.\textit{signers such that } i \in \text{L}_{corr}$ **then**
  return $\perp$;
if $\exists i \in \text{lad}_1^*.\textit{signers such that } i \in \text{L}_{corr}$ **then**
  return $\perp$;
if $\exists i \in \text{lad}_0^*.sr \textit{ such that } (\text{pk}_i, \cdot) \notin \text{L}_{keys}$ **then**
  return $\perp$;
if $\exists i \in \text{lad}_1^*.sr \textit{ such that } (\text{pk}_i, \cdot) \notin \text{L}_{keys}$ **then**
  return $\perp$;
$(\sigma_0, t_0, \mathcal{R}_0) \leftarrow \text{Process}(\mu^*, \text{L}_{\text{keys}}, \text{lad}_0)$;
$(\sigma_1, t_1, \mathcal{R}_1) \leftarrow \text{Process}(\mu^*, \text{L}_{\text{keys}}, \text{lad}_1)$;
if $\mathcal{R}_0 \neq \mathcal{R}_1$ *or* $t_0 \neq t_1$ **then**
  return $\perp$;
$\bar{\sigma} \leftarrow \sigma_b$;
return $\bar{\sigma}$;

Figure 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).

4 Results and Discussion

The following figure illustrates the clock time required for the `Sign` operation in the ERS, SMLERS, and ETRS schemes, considering various thresholds. The depicted time includes both the initial signature generation and subsequent joinings or extensions. The time complexity of generating a signature of the ETRS scheme is higher than those of SMLERS and ERS. The figure below presents the clock time required for the `Verify` operation in the ERS, SMLERS, and ETRS

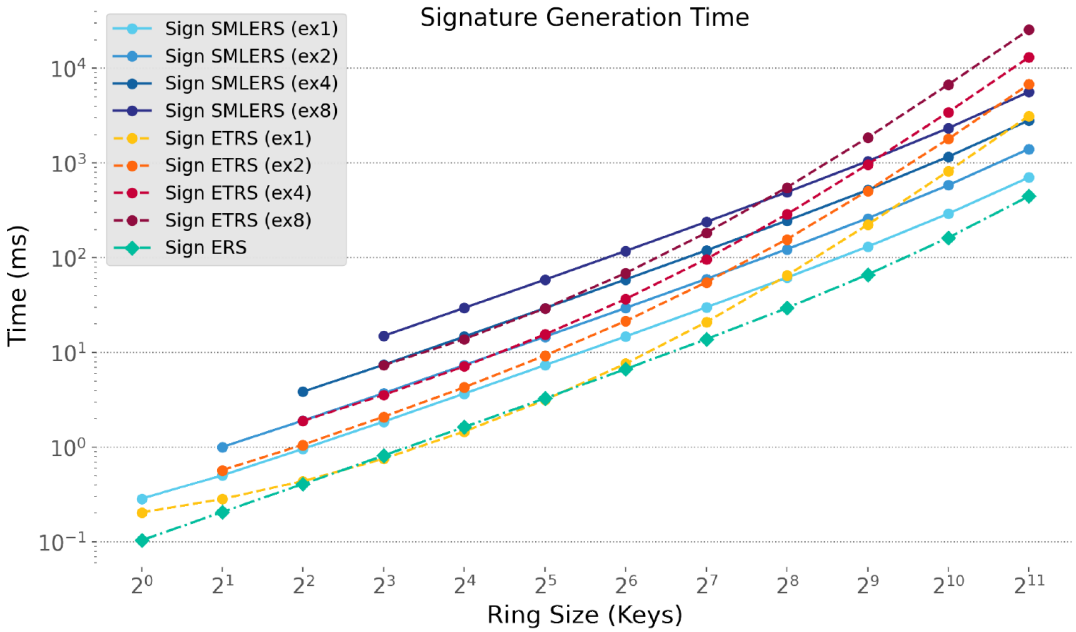

Figure 10: Clock time for `Sign` in the three implemented schemes for different thresholds. The signature generation time includes the initial signature generation and subsequent joinings/extensions.

schemes across different thresholds. The measured time corresponds to the verification of the final extended signature. One can see that the time complexity of verifying a signature of the ETRS scheme is also higher than those of SMLERS

and ERS. This figure showcases the signature sizes for the three implemented schemes – ERS, SMLERS, and ETRS –

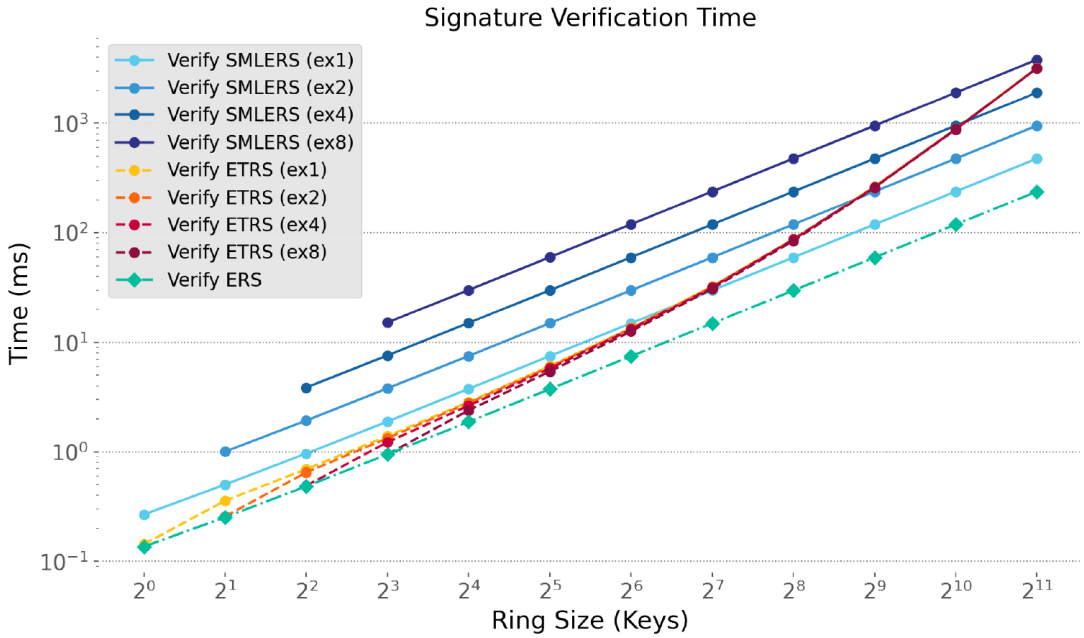

Figure 11: Clock time for `Verify` in the three implemented schemes for different thresholds. The verification time is that of verifying the final extended signature.

with varying thresholds. Notably, in the ETRS scheme, the signature size remains independent of the threshold, whereas SMLERS exhibits a linear dependence. We observe that the spatial complexity of a signature of the schemes is the same for all schemes. We note that the scales of the graphs are log - log with different bases, thus straight lines correspond to

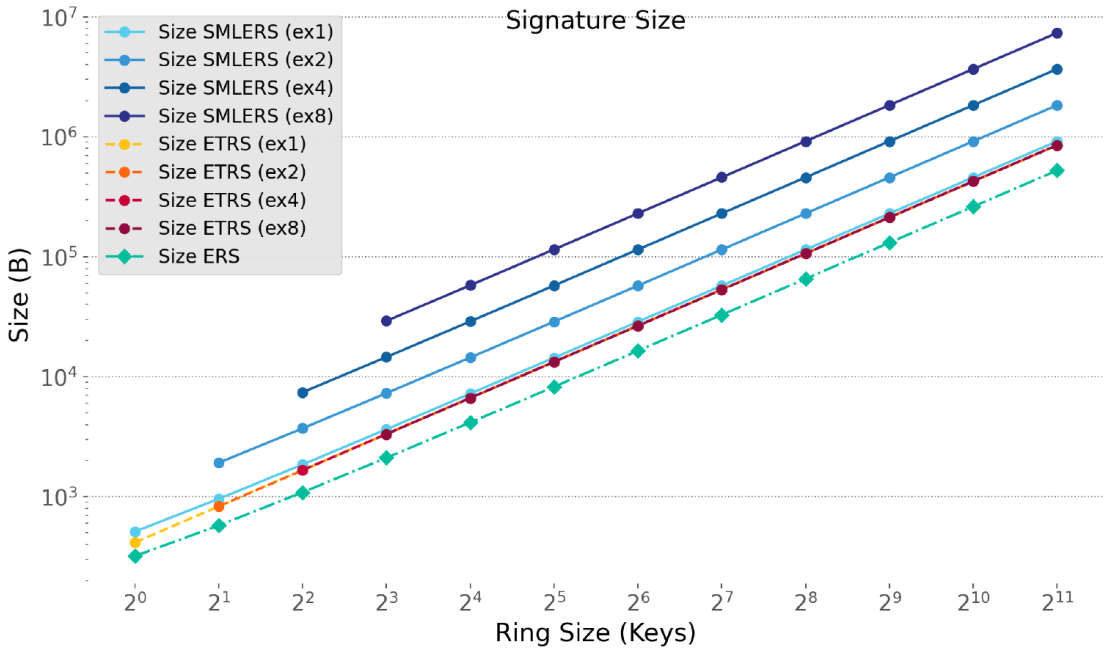

Figure 12: Signature sizes for all the three implemented schemes, with varying thresholds. In the ETRS scheme, the signature size is independent of the threshold, while in SMLERS there is a linear dependence.

exponential complexities while exponential curves correspond to doubly exponential complexities (!).

5 Conclusion

In conclusion, our project provides an overview of the paper "Count Me In! Extendability for Threshold Ring Signatures". It summarizes the key contributions, including the extendability for ring signatures, same-message linkable ring signatures, and extendable threshold ring signatures. Furthermore, it outlines the formal models, security proofs and algorithms presented in the paper. Also it demonstrates and analyzes implementation results of the proposed schemes. Overall, our project serves as a comprehensive summary and explanation of the essential concepts and findings of the original research.

References

[1] Diego F. Aranha, Mathias Hall-Andersen, Anca Nitulescu, Elena Pagnin, and Sophia Yakoubov. Count me in! extendability for threshold ring signatures. Cryptology ePrint Archive, Paper 2021/1240, 2021. https://eprint.iacr.org/2021/1240.