Stronger Notions and a More Efficient Construction of Threshold Ring Signatures (Full Version)

Alexander Munch-Hansen[1], Claudio Orlandi[1], and Sophia Yakoubov[1]

Aarhus University, Aarhus, Denmark
{almun, orlandi, sophia.yakoubov}@cs.au.dk

Abstract. We consider *threshold ring signatures* (introduced by Bresson *et al.* [7]), where any t signers can sign a message while anonymizing themselves within a larger (size-n) group. The signature proves that t members of the group signed, without revealing anything else about their identities.
Our contributions in this paper are two-fold. First, we strengthen existing definitions of threshold ring signatures in a natural way; we demand that a signer cannot be de-anonymized even by their fellow signers. This is crucial, since in applications where a signer's anonymity is important, we do not want that anonymity to be compromised by a single insider. Our definitions demand non-interactive signing, which is important for anonymity, since truly anonymous interaction is difficult or impossible in many scenarios.
Second, we give the first rigorous construction of a threshold ring signature with size independent of n, the number of users in the larger group. Instead, our signatures have size linear in t, the number of signers. This is also a very important contribution; signers should not have to choose between achieving their desired degree of anonymity (possibly very large n) and their need for communication efficiency.

Keywords: Threshold ring signatures · Anonymity · Unique ring signatures · Compact signatures

1 Introduction

It is often desirable for parties to anonymously sign on behalf of a group. A *group signature scheme* [12] enables this; the signature proves that a member of the group signed, but does not reveal which one. However, the downside of group signatures is that the group must be set up and maintained by a trusted *group manager*.[1] *Threshold* (group) signatures similarly allow any t of the parties in

[1] *List signatures* [11] are a related primitive. Like group signatures, list signatures require a group manager to set up the the keys and parameters. However, in a list signature scheme, signers may only sign a certain amount of times before their anonymity is revoked.

a group to sign on behalf of the group together. The signature proves that t members of the group signed without revealing which ones. but, as in group signatures, trusted setup is required for each group.

A *ring signature scheme* (introduced by Rivest *et al.* [29]) enables signing on behalf of a group without the need for interactive or trusted setup. Instead, everyone independently generates a key pair, and publishes their public key. The signer chooses the group (or ring) to anonymize herself amongst at signing time, and does so using that ring's public keys. In this paper, we focus on *threshold ring signature scheme*s (introduced by Bresson *et al.* [7]), which are a natural extension of ring signature schemes. In a threshold ring signature scheme, any t signers can sign a message together while anonymizing themselves within a larger (size-n) group. Like a ring signature scheme, a threshold ring signature scheme allows the signers to pick the larger group they want to anonymize themselves amongst in an ad-hoc way at signing time.

We make two major contributions in this paper: a strengthening of threshold ring signature definitions, and a new construction with more compact signatures. Our new definition demands that a signer cannot be de-anonymized even by their fellow signers. In applications where a signer's anonymity is important, this protects their anonymity from insiders.

Our construction has signatures of size linear in t, the number of signers. All prior rigorous constructions have signatures with size dependent on n, the size of the larger group. Compact signatures are important; signers should not have to choose between achieving their desired degree of anonymity (possibly very large n) and their need for communication efficiency.

1.1 Application: Whistleblowing

We can imagine a set of people within a large corporation wanting to blow the whistle on some corrupt activity within that organization; however, they are afraid to come forward publicly because of the repercussions they might face. On the other hand, blowing the whistle anonymously may not be effective, since it is important that the public believe that the message came from within the organization, from a sufficient number of organization members (and that it thus has credibility). Threshold ring signatures are the perfect solution. The whistleblowers form a size-t sub-group, and anonymize themselves within the entire size-n organization. Anyone can then verify that t members of the organization all blew the whistle on the corrupt activity.

Small signature sizes are important here, since often the size n of an organization is unreasonably large. In this application, it also becomes especially important that each individual whistleblower retain anonymity, even against their fellow whistleblowers. Otherwise, in order to de-anonymize *all* of the whistleblowers, all the organization administration would have to do is get *one* of the whistleblowers' cooperation.

1.2 Our Contributions

As we mentioned earlier, we make two contributions: we give a stronger definition of threshold ring signatures, and a construction that meets those definitions while achieving signatures with size $O(t)$.

Stronger Definitions Our most significant definitional contribution is a strengthening of the anonymity property. We require that an adversary not be able to tell the difference between signatures produced by two different subsets of signers of the same size t (within the same group of size n), as long as the two subsets contain the same corrupt parties. All previous definitions of anonymity [21,27,28,32] do not allow the sets of signers to contain any corrupt parties at all; this is a dealbreaker in many applications, where one insider should not be able to bring down the entire group.

We use a strong syntax that fits well with our stronger notion of anonymity. We require that signers be able to produce *partial* signatures locally, without interacting with their fellow signers; the partial signatures should preserve the signers' anonymity, and should be combinable into a threshold signature by any third party. Having such a noninteractive structure is crucial for preserving anonymity against fellow signers; if signing were interactive, signers might learn their peers' identities via e.g. their IP addresses.

Construction with Succinct Signatures We build the first threshold ring signature scheme with signatures of size $O(t)$; all previous constructions have signatures with size dependent on n. For groups of signers of size t significantly smaller than the larger group of size n they wish to anonymize themselves amongst, this is crucial.

Naively, to produce a threshold ring signature, each of the t signers could produce a ring signature, and their threshold ring signature would simply be a concatenation of these. The issue here is that a verifier would need to be convinced that these ring signatures were produced by distinct signers. An immediate solution to this would be a zero-knowledge proof that each signature was generated using a different secret key; however, this proof would be large, inefficient, and producing it would require interaction between the signers.

Instead, we base our threshold ring signature scheme on a primitive called a unique ring signature scheme (URS), introduced by Franklin and Zhang [17][2]. A unique ring signature scheme is a ring signature scheme which allows the linking of two signatures produced by the same signer, if the two signatures contain the same unique identifier produced during signing using a nonce. By using the message and the public keys belonging to the size-n supergroup as the nonce, we can construct a threshold ring signature simply by concatenating t unique

[2] A similar approach to building a threshold ring signature scheme was mentioned by Yuen *et al.* [31] where they would instead use a traceable ring signature scheme [18]; however, it was not formalized or proven. As far as we can tell, the definition of security they use for a traceable ring signature scheme does not seem to allow such a proof.

ring signatures. (For definitional simplicity, we merge the notion of nonce and message, requiring signers to sign the message *and* the set of public keys, and guaranteeing linkability as long as two signatures verify for the same message.) A verifier can check that no two unique ring signatures were produced by the same signer, and so is convinced that t of the n users signed the message.

Any unique ring signature scheme (secure under our definitions, which are slightly modified from those of Franklin and Zhang) can be used to construct a threshold ring signature scheme in such a way. We present a new, intuitive unique ring signature scheme with signatures of size $O(1)$ which draws inspiration from the construction of Dodis *et al.* [14]. As the field of traceable ring signatures is larger than that of unique ring signatures, we briefly compare our URS to those. One could also use a slightly modified traceable ring signature scheme, such as proposed by Yuen *et al.*, but unlike the work of Yuen *et al.*, we leverage a random oracle, allowing us to get smaller unique signatures. We additionally use an RSA accumulator [4][3] and the generalized DDH assumption. These assumptions are more standard than the Link-Decisional RSA assumption used in some traceable ring signature constructions [1, 30].

At a high level, our unique ring signature scheme works as follows: each signer in the ring hashes the message (taken to be the original message together with the set of n public keys belonging to the super-set of users), and raises it to the power of their secret signing key. By the generalized DDH assumption, this does not reveal the signer's identity. Each signer then proves using non-interactive zero knowledge (NIZK) that they used a signing key corresponding to one of the public keys belonging to the ring.[4] It may seem that such a proof must be linear in the number n of public keys, but we get around that by using an *accumulator* [4] (a compact representation of an arbitrarily large set that supports efficient proofs of membershi) to represent the set of public keys.

As required by our definitions, our construction is completely non-interactive; each of the t signers produces a unique ring signature independently, and those signatures are then simply concatenated to produce the threshold ring signature. This concatenation can be done by any third party. An important consequence of this is that the scheme is *flexible*, meaning that a signer can contribute a partial signature at any point, resulting in a threshold signature with a threshold t that is larger by 1.

[3] We could instead use a bilinear map accumulator [9]; however, the use of such an accumulator would require an a-priori upper bound on the ring size.

[4] Our use of NIZK proofs requires the presence of a common reference string (CRS). At first glance, since a CRS is a form of setup, this might seem to make our construction a group signature scheme instead of a ring signature scheme. However, there is a qualitative difference between a CRS (which is a global and reusable trusted setup) and a per-user trusted setup (in group signatures, parties' secret keys need to be distributed by a trusted party). In particular, once the CRS is generated in a trusted way (perhaps using an MPC ceremony), the parties in our system can generate their own keys independently.

1.3 Fully Compact Threshold Ring Signatures

While our threshold ring signature scheme is the first scheme to give signatures of size independent of the ring size n, the signature size does still depend linearly on the threshold t. A natural question to ask is,

Is it possible to build a threshold ring signature scheme with signatures of constant size?

The answer is that it *is* possible; any threshold ring signature scheme can be altered to have constant-size signatures with the use of succinct non-interactive arguments of knowledge (SNARKs). This can be done simply by allowing any third party — or perhaps one of the signers — to take the produced signature (whose size might depend on n or t) and replace it with a SNARK of a verifying signature for the given ring. Since SNARK sizes do not depend on the statement being proven or the witness for that statement, this yields a constant-size signature.

While this transformation is optimal from an asymptotic point of view, the non-black box use of public-key cryptography inside a SNARK would make this construction prohibive in practice.[5]

1.4 Related Work

Work	Signature Size	Adversarial Keys?	Assumptions
Our work	$\mathbf{O(t)}$	Yes	**Generalized DDH, RSA, RO**
Bresson *et al.* [7]	$O(n \log n)$	No	RSA, RO
Petzoldt *et al.* [28]	$O(n)$	No	Quadratic MQ-problem, RO
Liu *et al.* [31]	$O(t\sqrt{n})$	No	Q-Strong DH, Subgroup Decision in $\mathcal{G}_q$, DDH-Inversion
Zhou *et al.* [33]	$O(n)$	No	Syndrome Decoding Problem, Indistinguishability of Goppa Codes, RO
Chen *et al.* [13]	$O(n)$	No	Ideal Lattice, Shortest Independent Vector Problem, RO
Okamoto *et al.* [27]	$O(tn)$	No	Discrete Log, RO, Trusted Dealer
Haque *et al.* [21]	$O(n)$	Yes	(Any) Trapdoor Commitments, QROM
Haque *et al.* [20]	$O(t)$	No	SPB hashing, NIWI

Fig. 1: Threshold Ring Signature Constructions

In Figure 1 we list some known threshold ring signature constructions, their signature sizes, whether they support adversarial key generation, and the assumptions they leverage. All prior constructions of threshold ring signatures have signatures whose size depends on the number n of users in the ring $\mathcal{R}$. This is not ideal, as the threshold t may be much smaller than n.

[5] Even the most basic public-key type operation, a scalar multiplication in an elliptic curve, requires billions of gates [22]. This needs to be multiplied by a function of n for any existing threshold ring signature, or t for our construction. While this is the state of the art, we cannot of course rule out that more efficient constructions might emerge in the future, and this could be an interesting venue for further research.

Concurrent Work Haque *et al.* [20], posted shortly after this paper, also construct threshold ring signatures of size $O(t)$. The advantage of their work is that their construction does not require a common reference string (CRS), which our construction uses for non-interactive zero knowledge (NIZK) proofs. They get around the need for a CRS by using NIWI (non-interactive witness-indistinguishable) proofs instead of NIZK proofs. However, the advantage of our work is that we support adversarially generated public keys. In the scheme of Haque *et al.*, an adversary who is able to generate and register keys himself is immediately able to break anonymity and unforgeability.[6]

Relying on honestly generated keys is significantly riskier than relying on an honestly generated CRS. CRS generation occurs once, and therefore efficiency is not too much of a concern: we can ensure security e.g. via secure multiparty computation (which can be slow), by involving a large number of parties all of whom are extremely unlikely to collude. However, taking such measures in the generation of every party's key pair, which can happen frequently, would be unreasonable.

1.5 Outline

In Section 2, we define ring and threshold ring signatures. In Section 3, we describe our threshold ring signature construction. Please refer to Section A of the Supplementary Materials for a description of the tools and assumptions necessary for our constructions, such as cryptographic accumulators and zero knowledge proofs.

2 (Threshold) Ring Signature Definitions

In this section, we recall the definitions of ring signatures and threshold ring signatures (focusing on the latter).

2.1 Ring Signature Definitions

Ring signatures were originally defined by Rivest *et al.* [29] as a natural extension of group signature schemes. Group signatures require some trusted authority to act as a group manager, predefining groups of signers and distributing keys to members of those groups. These keys can then be used to anonymously sign messages on behalf of the entire group. However, requiring a trusted authority that distributes — and knows — signers' keys can be a big drawback. Ring

[6] This is by design; in the proof of anonymity, the authors need to create simulated NIWI proofs that are independent of the identities of the signers. They do this by additionally allowing a witness to demonstrate a relationship between two keys in the ring, where this relationship never holds between keys that are honestly generated. If an adversary was able to register maliciously generated keys, she could register two keys that *do* have this relationship, and use this to forge signatures with arbitrarily high threhsolds, as long as those two corrupt keys are in the ring in question.

signatures instead allow signers to generate their own key pairs, and to form groups in an ad-hoc way.

Ring Signature Syntax A ring signature scheme is defined as a tuple of four algorithms (setup, keygen, sign, verify):

$\mathsf{setup}(1^\lambda) \to \mathtt{pp}$**:**
An algorithm that takes a security parameter λ and outputs a set of public parameters pp. These public parameters pp include the security parameter itself, and any global parameters which can be used within the other algorithms.

$\mathsf{keygen}(\mathtt{pp}) \to (pk, sk)$**:**
An algorithm that takes the public parameters pp and outputs a key pair (pk, sk).

$\mathsf{sign}(\mathtt{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, sk_i) \to \sigma$**:**
An algorithm that takes the public parameters, a message $msg \in \{0,1\}^*$ to be signed, the set of public keys of the users within the ring $\{pk_j\}_{j\in\mathcal{R}}$, and the secret key sk_i of the signer $i \in \mathcal{R}$ (which must correspond to a public key within the set of public keys $\{pk_j\}_{j\in\mathcal{R}}$). Outputs a signature σ on the message msg.

$\mathsf{verify}(\mathtt{pp}, msg, \{pk_i\}_{i\in\mathcal{R}}, \sigma) \to \mathtt{accept}/\mathtt{reject}$**:**
An algorithm that takes the public parameters, the message, the set of public keys of the users within the ring, and a signature σ. Outputs `accept` or `reject`, reflecting the validity of the signature σ on the message msg.

An important property of ring signatures is *setup freeness*, which requires that signers' keys be generated independently. (We note that most ring signature schemes do have a setup algorithm that is run by a trusted authority. However, this authority does not produce the secret keys for the signers; its only job is to produce the public parameters such as moduli and generators used throughout the scheme. The signers can then generate their keys independently using those public parameters.)

Ring Signature Security Definitions Informally, a ring signature scheme must satisfy the following properties [7, 14, 23]:

- *Correctness* requires that a correctly generated signature must verify.
- *Unforgeability* requires that an adversary should not be able to forge a signature on behalf of another user.
- *Anonymity* requires that a signature should completely hide the identity of the signer, even if the adversary has access to a signing oracle.
- *Unlinkability* requires that no adversary should be able to determine whether two signatures were produced by the same signer, even if the adversary has access to a signing oracle.

Remark 1. Note that anonymity implies unlinkability, and vice versa; however, when access to signing oracles is removed, this is no longer the case.

We omit the formal definitions of ring signatures from this paper, focusing instead on *threshold* ring signatures.

2.2 Threshold Ring Signature Definitions

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

Threshold Ring Signature Syntax A threshold ring signature scheme is usually defined as a tuple of four algorithms (setup, keygen, sign, verify), where sign is interactive and requires the secret keys of t of the signers. We instead choose to define a threshold ring signature scheme as a tuple of five algorithms, by adding combisign. We let sign be locally executed by each signer i (requiring only that signer's secret key sk_i), and produce *partial* signatures σ_i; combisign can then be run by any third party to combine those partial signatures into a threshold signature.

We describe the syntax of combisign below. Notice that it does not require the secret keys of any of the signers.

$\mathsf{combisign}(\mathsf{pp}, \{\sigma_i\}_{i\in\mathcal{S}}, t) \to \sigma$**:**
An algorithm that takes partial signatures $\{\sigma_i\}_{i\in\mathcal{S}}$ from t signers, and outputs a combined signature σ.

The syntax of of setup, keygen, sign and verify remain unchanged from those of a ring signature scheme, except that sign outputs partial signatures, and verify takes the threshold t as input.

This syntax specification is very strong. In particular, it demands the following desirable properties:

Setup Freeness:
Every signer can generate their own key pair. This is a feature of all ring signature schemes.

Dynamic Choice of Ring Size n:
Different sets of signers can choose rings of different sizes.

Dynamic Choice of Threshold t:
Arbitrarily many signers' partial signatures can be combined into a single threshold signature; the signers don't need to know t when they produce their partial signature. Verification takes a threshold t, and checks that at least that many signers have signed. The upside of this is what is called *flexibility* [27], meaning that signers can contribute their partial signatures after others have signed. Our syntax demands a weak notion of flexibility where signers can contribute their signatures before combination via combisign; if combisign

is as simple as e.g. concatenation of the partial signatures, the stronger notion of flexibility — where signers can contribute even after combination — follows.

The downside of this flexibility is that the number of signers cannot be hidden by a signature σ.

Non-Interactive Signing:

As per our syntax, parties generate partial signatures locally; those partial signatures can be combined into a threshold signature by any third party. Non-interactive signing is essential in ensuring the signers' privacy (even against their peers), since anonymous interactive signing would require anonymous communication, which is often difficult to achieve in practice.

Threshold Ring Signature Security Definitions We base our security definitions on Bresson *et al.* [7] and Haque *et al.* [21]. (In particular, we require security against an adversary who can generate and register public keys, as required by Haque *et al.*) We strengthen the definition of anonymity to require that signers remain anonymous even to their fellow signers.

Additionally, both of our security games are defined using *partial* signatures, where a complete signature will be formed by combining the partial signatures of all the signers. This allows for a simple statement of the games while still demanding security against fellow members of the signing rings $\mathcal{R}$. An adversary wins the unforgeability game if he is able to forge a partial signature, and he wins the anonymity game if he is able to distinguish between two partial signatures.

Definition 1 (TRS). *A threshold ring signature scheme is* secure *if it satisfies correctness (Definition 2), unforgeability (Definition 3), and anonymity (Definition 4).*

Definition 2 (Correctness for TRS). *Correctness requires that verification return* `accept` *on any honestly generated signature.*

More formally, let $\mathbf{TRS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{combisign}, \mathsf{verify})$ *be a TRS scheme. We say that* $\mathbf{TRS}$ *is* correct *if for all security parameters* $\lambda \in \mathbb{N}$*, for all messages* $msg \in \{0,1\}^*$*, all rings* $\mathcal{R}$*, and all signer sets* $\mathcal{S} \subseteq \mathcal{R}$*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathbf{TRS}.\mathsf{setup}(1^\lambda), \\ \{(pk_i, sk_i) \leftarrow \mathbf{TRS}.\mathsf{keygen}(\mathsf{pp})\}_{i\in\mathcal{R}}, \\ \{\sigma_i \leftarrow \mathbf{TRS}.\mathsf{sign}(\mathsf{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, sk_i)\}_{i\in\mathcal{S}}, \\ \sigma \leftarrow \mathbf{TRS}.\mathsf{combisign}(\mathsf{pp}, \{\sigma_i\}_{i\in\mathcal{S}}, t = |\mathcal{S}|) : \\ \mathbf{TRS}.\mathsf{verify}(\mathsf{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, \sigma, t = |\mathcal{S}|) = \mathtt{accept} \end{array}\right] = 1$$

Definition 3 (Unforgeability for TRS). *Unforgeability requires that no efficient adversary* $\mathcal{A}$ *is able to forge a valid signature* σ *for some ring* $\mathcal{R}$ *and message* msg^* *for which* $\mathcal{A}$ *has issued fewer than* t *corruption queries (on signers in* $\mathcal{R}$*) or signing queries (for ring* $\mathcal{R}$ *and message* msg^**), where* t *is the threshold.*

More formally, let $\mathbf{TRS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{combisign}, \mathsf{verify})$ *be a TRS scheme. Consider the game* $\mathsf{Game}^{Unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$ *in Figure 2 between a probabilistic polynomial-time adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

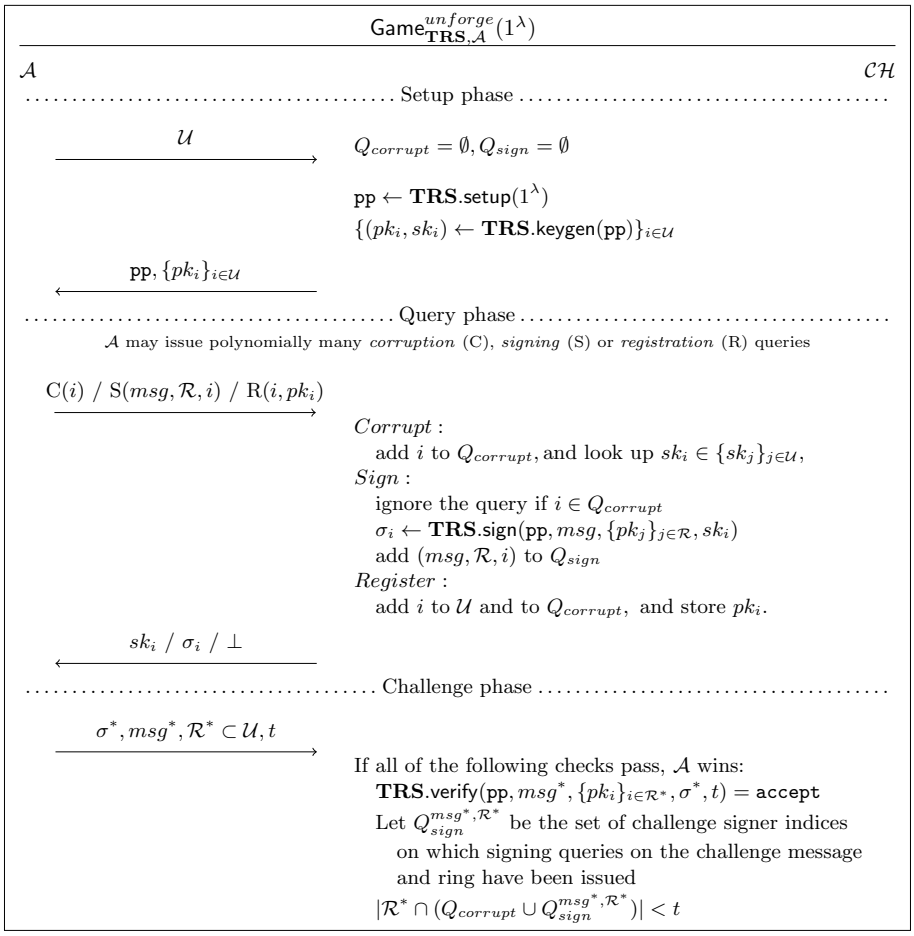


Fig. 2: The unforgeability game for TRS

We say that **TRS** *is* unforgeable *if for any efficient adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{Game}^{unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)] \leq \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$*.*

Remark 2. Note that in the unforgeability game, the challenger responds to signing queries with partial signatures. This is to capture that the adversary might know some of the secret keys (due to corruption queries), and is therefore only interested in seeing the partial signatures by the honest parties. The same holds true for the anonymity game.

Definition 4 (Anonymity for TRS). *Anonymity requires that no efficient adversary* $\mathcal{A}$ *be able to distinguish between partial signatures produced by two different signers in the same ring.*

More formally, let $\mathbf{TRS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{combisign}, \mathsf{verify})$ *be a TRS scheme. Consider the game* $\mathsf{Game}^{anon}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$ *in Figure 3 between a probabilistic polynomial-time adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

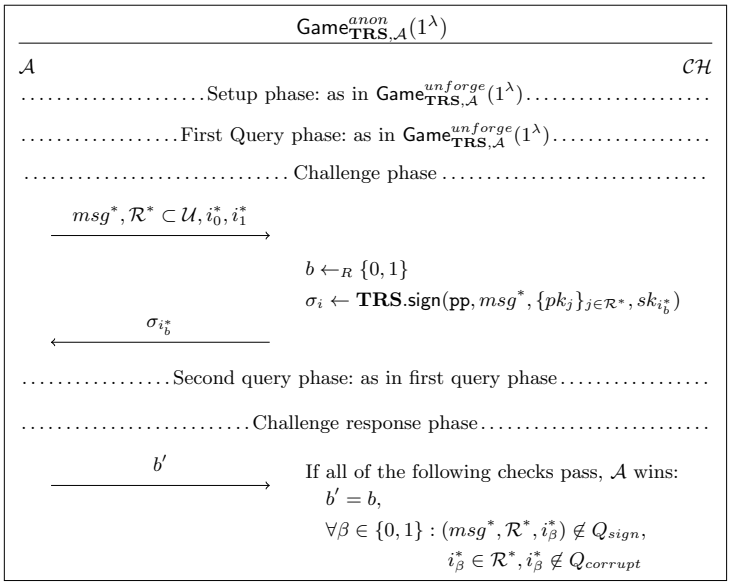


Fig. 3: The anonymity game for TRS

We say that $\mathbf{TRS}$ *is* anonymous *if for any efficient adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{Game}^{anon}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$*.*

3 Our Threshold Ring Signature Construction

A natural approach to building threshold ring signatures is having each of the t signers produce a ring signature, and then appending to the list of t signatures a zero knowledge proof that all of the signatures were produced using distinct signing keys. However, this approach has two downsides.

1. Producing the zero knowledge proof requires interaction among the signers.
2. The zero knowledge proof may be complex. (One way to do this is to commit to the secret keys used, order the commitments by secret key, prove that each key was used to produce the corresponding signature, and use t range proofs to prove that each committed key is strictly larger than the previous one - since we need to prove that the signatures were produced by t *distinct* signers)

In order to circumvent these two issues, we leverage *unique ring signatures (URS)* [17], which are a flavor of *linkable ring signatures (LRS)* [24]. Linkable ring signatures are ring signatures where a verifier can tell whether two signatures were produced by the same signer. If each of the t signers produces a *linkable* ring signature, there is no need to additionally prove that the signatures were produced using distinct signing keys, since this is immediately apparent.[7] If the underlying linkable ring signatures have size $O(1)$, then the threshold ring signatures will have size $O(t)$.

However, this construction is flawed, since the linkable ring signatures also allow linking across *different* threshold ring signatures, which violates the anonymity property we require of any threshold ring signature scheme (Definition 4).

To address this problem, we use *unique ring signatures* instead. Unique ring signatures satisfies the additional property that each signature contains a unique identifier. Traditionally, the signing algorithm produces an additional unique identifer and two signatures by the same signer are now only linkable if the two signatures contain the same identifier. This identifier can be produced in several ways, but a natural way would be by using a nonce for each signature. For definitional and notational simplicity however, we merge the nonce and the message: in our definition of unique ring signatures, we require linkability between any two signatures on the same message. In our construction of threshold ring signatures, each of our signers sign the original message *and* the set of public keys belonging to the signing ring using the unique ring signature scheme. By using the message and set of public keys belonging to the signing ring as the *nonce*, we ensure that the unique ring signatures used as components of two different threshold ring signatures remain unlinkable (under the assumption that the same ring never signs the same message more than once).

The rest of this section proceeds as follows:

1. In Section 3.1, we state a (somewhat modified) definition of a *unique ring signature scheme* (URS) [17].
2. In Section 3.2, we construct a URS scheme with signatures of size $O(1)$.
3. In Section 3.3, we use our URS scheme to construct a TRS scheme with signatures of size $O(t)$.

3.1 Unique Ring Signature Definitions

We leverage the notion of *unique ring signature (URS)* schemes, as defined by Franklin and Zhang [17].

We alter (and use) the definition of Franklin and Zhang in the following ways:

[7] A similar idea was mentioned by Yuen *et al.* [31]; however, it was not formalized or proven. In particular, a stronger linkability property — one that is similar to that of uniqueness — is needed from the underlying traceable ring signature scheme in order for the TRS construction to be secure. Additionally, since Yuen *et al.* focus on avoiding the random oracle assumption and we do not, we obtain a TRS construction with size $O(t)$ signatures, while they obtain a TRS construction with size $O(t\sqrt{n})$ signatures.)

1. Franklin and Zhang require that even if an adversary has access to some t secret keys belonging to a subset of ring members; then the adversary should be unable to produce $t+1$ unique signatures for the same message and ring.
2. We definitively merge the notion of message and unique identifier; If a signer can produce a correctly verifiable signature for some message with respect to some ring, then this is also a unique identifier for said signer. This yields a more efficient proof of correctness for the signature. This idea is also used by Franklin and Zhang.
3. Franklin and Zhang require that each signer only signs any message once. We require a stronger definition; we require linkability of any two signatures on the same message with respect to the same ring (resulting in a stronger definition of uniqueness, Definition 6)), and unlinkability across signatures of different messages (resulting in cross-message unlinkability, Definition 7).

Unique Ring Signature Syntax We define a unique ring signature scheme as a tuple of five algorithms (setup, keygen, sign, verify, link). The setup, keygen, sign and verify algorithms all have the same input and output behavior as the corresponding ring signature algorithms.

The link algorithm (described below) allows any verifier to determine whether two signatures were produced by the same signer (on the same message).

link(pp, $msg, (\sigma_0, \{pk_j\}_{j\in\mathcal{R}_0}), (\sigma_1, \{pk_j\}_{j\in\mathcal{R}_1})) \to \{\texttt{linked}, \texttt{unlinked}\}$:
An algorithm that takes a message msg, two signatures (σ_0, σ_1) and public keys belonging to members of rings $(\mathcal{R}_0, \mathcal{R}_1)$. Outputs `linked` or `unlinked`, depending on whether the two signatures were produced by the same signer.

Unique Ring Signature Security Definitions Informally, a unique ring signature scheme must satisfy the following properties:

- *Correctness* requires that a correctly generated signature must verify. (This is inherited from ring signatures.)
- *Unforgeability* requires that an adversary should not be able to forge a signature on behalf of another user. (This is inherited from ring signatures.)
- *Linkability*, adapted from linkable ring signatures [25], requires that no corrupt signer can produce two signatures that verify for the same message and appear unlinked.
- *Uniquenes*, which we strengthen, requires that no $t-1$ corrupt signers can produce t signatures that verify for the same message and appear unlinked. With $t = 2$, it implies linkability; with $t = 1$, it implies unforgeability.)
- *Cross-Message Unlinkability* requires that no adversary can determine whether two signatures that verify for different messages were produced by the same signer. (We present this property as Definition 7.)
- *Defamation Freeness* requires that no adversary can produce a signature that appears linked to an honest signer's signature. We do not require this property for our TRS construction, so we do not define it formally nor prove that our construction meets it. This is loosely defined as an adversary forging an identifier.

Definition 5 (URS). *A unique ring signature scheme is* secure *if it satisfies correctness, uniqueness (Definition 6, which implies unforgeability and linkability), and cross-message unlinkability (Definition 7).*

Definition 6 (Uniqueness for URS). *Let* $\mathbf{URS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{verify}, \mathsf{link})$ *be a URS scheme. Consider the game* $\mathsf{Game}^{unique}_{\mathbf{URS},\mathcal{A}}(1^\lambda)$ *in Figure 4 between a probabilistic polynomial-time adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

We say that $\mathbf{URS}$ *is* unique *if for any efficient adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{Game}^{unique}_{\mathbf{URS},\mathcal{A}}(1^\lambda)] \leq \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$*.*

Definition 7 (Cross-Message Unlinkability for URS). *Given two signatures for different messages it should be infeasible for an adversary to determine whether they were created by the same signer or not. More formally, let* $\mathbf{URS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{verify}, \mathsf{link})$ *be a URS scheme. Consider the game* $\mathsf{Game}^{cmunlink}_{\mathbf{URS},\mathcal{A}}(1^\lambda)$ *in Figure 5 between a probabilistic polynomial-time adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

We say that $\mathbf{URS}$ *is* cross-message unlinkable *if for any efficient adversary* $\mathcal{A}$*,*

$$\Pr[\mathcal{A} \textit{ wins } \mathsf{Game}^{cmunlink}_{\mathbf{URS},\mathcal{A}}(1^\lambda)] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$$

for some negligible function $\mathsf{negl}(\lambda)$*.*

3.2 A Unique Ring Signature Scheme

We describe a unique ring signature scheme in Construction 1 in terms of an underlying accumulator scheme $\mathbf{ACC}$, a non-interactive zero-knowledge argument of knowledge scheme $\mathbf{NIZKAoK}$, a group $\mathbb{G}$ (of order p, with generator g) in which the generalized DDH problem is hard, and a random oracle H which maps arbitrary strings to elements in $\mathbb{G}$. We refer to Section A of the Supplementary Materials for a description of these building blocks.

The non-interactive zero-knowledge argument of knowledge scheme $\mathbf{NIZKAoK}$ will be used for the relation $\mathcal{R}_{\mathsf{sig}}$, which is described below. In Section A.2 of the supplementary material we describe an instantiaion of this relation $\mathcal{R}_{\mathsf{sig}}$.

$$\mathcal{R}_{\mathsf{sig}}\begin{pmatrix} \phi = (\mathbb{G}, g, \mathbf{ACC}.\mathsf{pp}, \\ \quad a_{\mathcal{R}}, \sigma', h), \\ w = (pk, sk, w_a) \end{pmatrix} = \begin{pmatrix} (pk = g^{sk}) \\ \wedge \mathbf{ACC}.\mathsf{verify}(\mathbf{ACC}.\mathsf{pp}, a_{\mathcal{R}}, pk, w_a) \\ \wedge (\sigma' = h^{sk}) \end{pmatrix}$$

Construction 1.

$\mathsf{setup}(1^\lambda)$**:**

- *Run* $\mathbf{ACC}.\mathsf{pp} \leftarrow \mathbf{ACC}.\mathsf{setup}(1^\lambda)$.
- *Run* $(\mathbf{NIZKAoK}.\mathtt{crs}, \mathbf{NIZKAoK}.\mathtt{td}) \leftarrow \mathbf{NIZKAoK}.\mathsf{setup}(1^\lambda, \mathcal{R}_{\mathsf{sig}})$.
- *Set* $\mathsf{pp} = (\mathbf{ACC}.\mathsf{pp}, \mathbf{NIZKAoK}.\mathtt{crs})$.

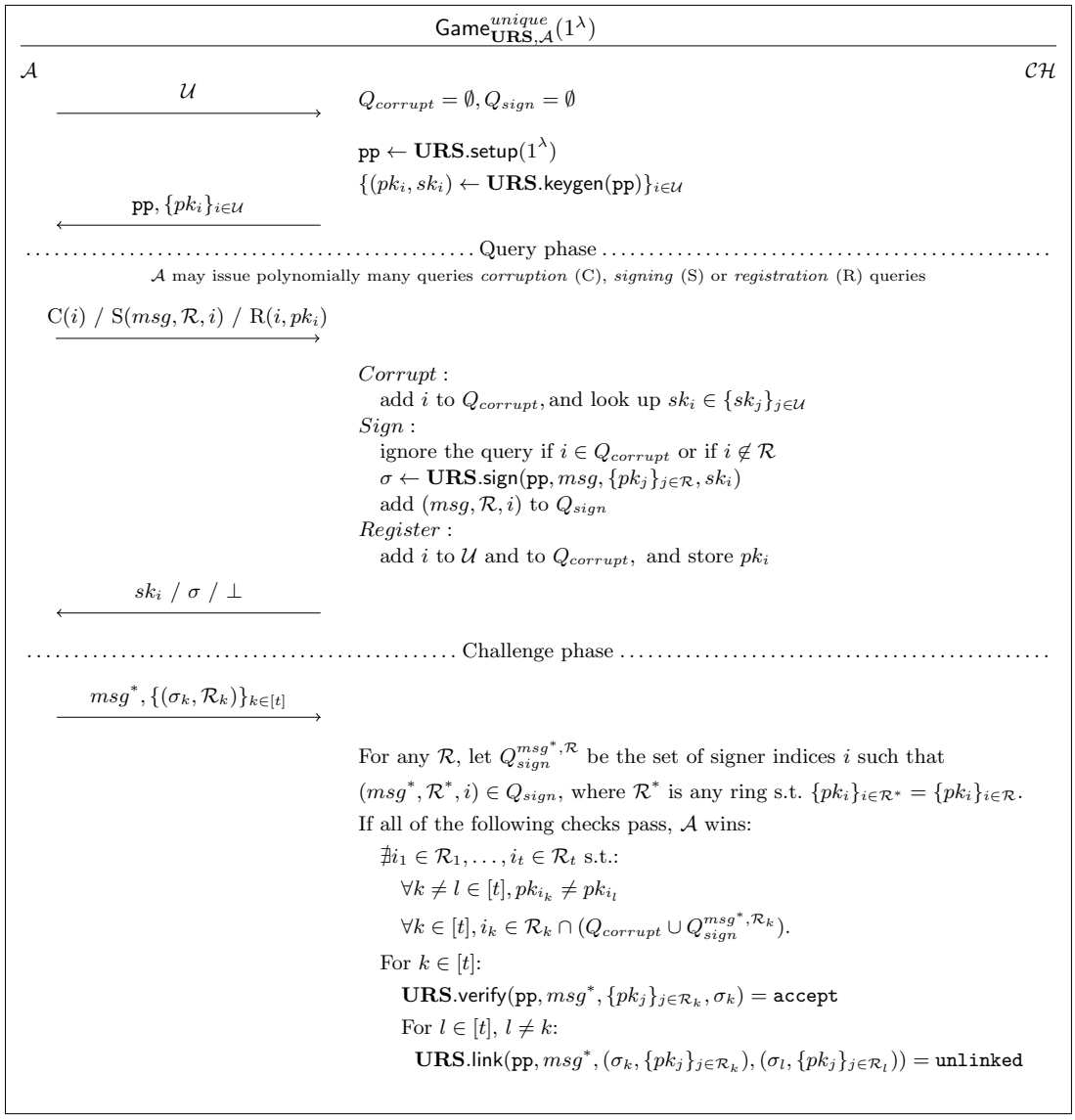


Fig. 4: The uniqueness game for URS.
Note that t verifying pairwise-unlinked signatures only count as a win for the adversary if the adversary has not corrupted (or queried the signing oracle on the appropriate message and ring for) t or more of the relevant parties. Furthermore, the adversary is not allowed to use the same public key for two of the signatures, but under different rings. Otherwise, the adversary could legitimately assemble the necessary unlinked signatures by using corrupt parties and signing oracle outputs, intuitively winning the game.

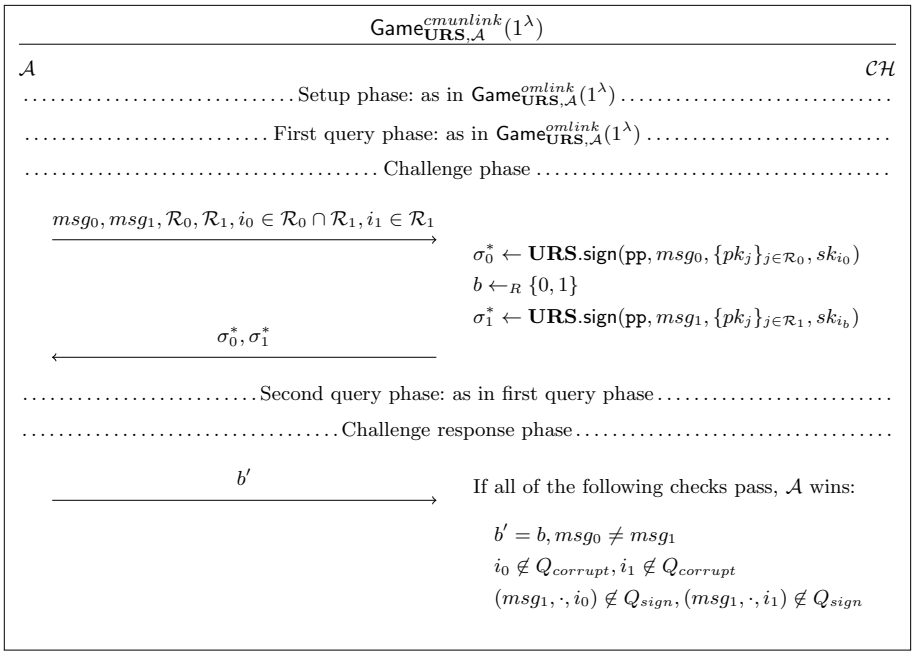


Fig. 5: The cross-message unlinkability game for URS.
Note that, if the adversary queried the signing oracle on either of the challenge signer identities and msg_1, he could legitimately link the output of the signing oracle to the second challenge signature, helping him determine whose secret key was used to produce it. So, if such a signing query was asked, we do not count the adversary's win.

$\mathsf{keygen}(\mathsf{pp})$**:**

- *Pick* $sk \leftarrow \mathbb{Z}_p$ *at random.*
- *Set* $pk = g^{sk}$.
- *If* pk *is not prime (when interpreted as an integer), redo the first two steps until it is. (We require the public keys to be prime so that they are within the domain of the RSA accumulator.)*

$\mathsf{sign}(\mathsf{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, sk)$**:**

- *Check that each* pk_j *is prime.*
- *Accumulate* $\{pk_j\}_{j\in\mathcal{R}}$ *as*

$$a_{\mathcal{R}} \leftarrow \mathbf{ACC}.\mathsf{accumulate}(\mathbf{ACC}.\mathsf{pp}, \{pk_j\}_{j\in\mathcal{R}}).$$

  (Note that this is publicly computable from the set of public keys, and thus does not need to be included in the threshold ring signature.)
- *Let* $pk \in \{pk_j\}_{j\in\mathcal{R}}$ *be the public key corresponding to the secret key* sk*. Compute an accumulator witness* $w_a \leftarrow \mathbf{ACC}.\mathsf{witcreate}(\mathbf{ACC}.\mathsf{pp}, \{pk_j\}_{j\in\mathcal{R}}, pk)$.
- *Compute* $\sigma' = \mathsf{H}(msg)^{sk}$.

– *Compute π proving that $\mathsf{H}(msg)$ was raised to the power of a secret key corresponding to a public key in the accumulator. In other words,*

$$\pi \leftarrow \mathbf{NIZKAoK}.\mathsf{prove}\begin{pmatrix}\mathbf{NIZKAoK}.\mathtt{crs},\\ \phi = (\mathbb{G}, g, \mathbf{ACC}.\mathsf{pp},\\ \quad a_{\mathcal{R}}, \sigma', \mathsf{H}(msg)),\\ w = (pk, sk, w_a)\end{pmatrix}$$

– *Return $\sigma = (\sigma', \pi)$.*

$\mathsf{verify}(\mathsf{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, \sigma = (\sigma', \pi))$**:**

– *Check that each pk_j is prime.*

– *Accumulate $\{pk_j\}_{j\in\mathcal{R}}$ as*

$$a_{\mathcal{R}} \leftarrow \mathbf{ACC}.\mathsf{accumulate}(ACC.\mathsf{pp}, \{pk_j\}_{j\in\mathcal{R}})$$

– *Verify the proof π; return*

$$NIZKAoK.\mathsf{verify}(\mathbf{NIZKAoK}.\mathtt{crs}, \phi = (\mathbb{G}, g, \mathbf{ACC}.\mathsf{pp}, a_{\mathcal{R}}, \sigma', \mathsf{H}(msg)), \pi).$$

$\mathsf{link}(\mathsf{pp}, msg, (msg_0, \sigma_0 = (\sigma'_0, \pi_0)),$
$(msg_1, \sigma_1 = (\sigma'_1, \pi_1)), \mathtt{nonce})$**:**
return `linked` *if $\sigma'_0 = \sigma'_1$, and* `unlinked` *otherwise.*

Theorem 1. *If* **NIZKAoK** *is a secure non-interactive zero knowledge argument of knowledge, if* **ACC** *is a secure accumulator, if* H *is a random oracle, and if the generalized DDH problem is hard in $\mathbb{G}$, then Construction 1 is a secure unique ring signature scheme (Definition 5).*

We prove Theorem 1 in Section B of the Supplementary Materials.

3.3 A Threshold Ring Signature Scheme

We build threshold ring signatures out of unique ring signatures in a generic way. If the underlying unique ring signatures have size $O(1)$, then the resulting threshold ring signatures have size $O(t)$, where t is the threshold. We require the additional assumption that no message msg is ever signed twice by the same ring $\mathcal{R}$. This is because we use the underlying unique ring signature scheme to sign the message together with the ring; if the same message is signed twice by the same ring, then the partial signatures will be linkable across the two threshold signature instances, and in this case we cannot guarantee anonymity.[8]

[8] One could think to use public keys belonging to the signing subset $\mathcal{S}$ as part of the message, instead of the keys belonging to the ring $\mathcal{R}$; however, this has two downsides. First, the signers must be aware of who their fellow signers are.

Second, the message must now be hidden from the adversary, as knowledge of the message would allow the adversary to de-anonymize the signing subset (by repeatedly deriving messages from the underlying message and any possible signing subset, and seeing which output matches the messages it knows).

Hiding the message from the adversary complicates zero knowledge proofs necessary in the underlying URS construction.

We describe our TRS construction formally below, in terms of the underlying URS. (We assume the public keys are always ordered in a canonical way (e.g. lexicographically), so that in the underlying URS, the same message and set of keys always hashes to the same value.)

Construction 2.

$\mathsf{setup}(1^\lambda)$**:** *Return* $\mathbf{URS}.\mathsf{pp} \leftarrow \mathbf{URS}.\mathsf{setup}(1^\lambda)$.
$\mathsf{keygen}(\mathsf{pp})$**:** *Return* $(sk, pk) \leftarrow \mathbf{URS}.\mathsf{keygen}(\mathbf{URS}.\mathsf{pp})$.
$\mathsf{sign}(\mathsf{pp}, msg, sk_i, \{pk_j\}_{j\in\mathcal{R}})$**:**
- *Set* $msg' = (msg, \{pk_j\}_{j\in\mathcal{R}})$.
- *Return* $\sigma_i \leftarrow \mathbf{URS}.\mathsf{sign}(\mathbf{URS}.\mathsf{pp}, msg', \{pk_j\}_{j\in\mathcal{R}}, sk_i)$.

$\mathsf{combisign}(\mathsf{pp}, \{\sigma_i\}_{i\in\mathcal{S}}, t = |\mathcal{S}|)$**:** *Return* $\sigma = \{\sigma_i\}_{i\in\mathcal{S}}$*. (So simple!)*[9]
$\mathsf{verify}(\mathsf{pp}, msg, \{pk_j\}_{j\in\mathcal{R}}, \sigma = \{\sigma_i\}_{i\in\mathcal{S}}, t)$**:**
- *If* $|\sigma| < t$*, return* `reject`.
- *Set* $msg' = (msg, \{pk_j\}_{j\in\mathcal{R}})$.
- *For* $\sigma_i \in \sigma$*, if* $\mathbf{URS}.\mathsf{verify}(\mathbf{URS}.\mathsf{pp}, msg', \{pk_j\}_{j\in\mathcal{R}}, \sigma_i) =$ `reject`*, return* `reject`.
- *For all pairs of different signatures* σ_i, σ_j *in* σ*, if* $\mathbf{URS}.\mathsf{link}(\mathbf{URS}.\mathsf{pp}, msg, \sigma_i, \sigma_j, \{pk_j\}_{j\in\mathcal{R}}) =$ `linked`*, return* `reject`.[10]
- *Return* `accept`.

Remark 3. Note that, since $\mathsf{combisign}$ simply takes a concatenation of the partial signatures, our construction satisfies flexibility [27]. Flexibility requires that a signer $i \in \mathcal{R}$ can take an existing threshold signature σ on message msg using the ring $\mathcal{R}$ that verifies with threshold t, and create a signature σ^* on the same msg and $\mathcal{R}$, that verifies with threshold $t + 1$. This is trivially achieved in our construction; signer i simply produces his own partial signature σ_i, and appends it to the existing signature.

Remark 4. Note that there is an immediate transformation from this construction to a *linkable threshold ring signature scheme*. Our threshold ring signature scheme uses a unique ring signature scheme as a primitive, providing a way of using the signatures to verify the distinctness of the t signers while disallowing linking across signatures. If one instead uses a regular linkable ring signature scheme (where signatures from the same signer are linkable across messages and rings), our TRS construction (Construction 2) would also be linkable across multiple signatures. See Munch-Hansen [26] for details.

Theorem 2. *If URS is a secure unique ring signature scheme (Definition 5), then Construction 2 is a secure threshold ring signature scheme (Definition 1).*

We prove Theorem 2 in Section C of the Supplementary Materials.

[9] The signing set $\mathcal{S}$ is only mentioned here for the sake of clarity. The set of signers is never leaked to the party who performs the combining of the signatures, as each signature is anonymous and does not leak the individual signers.

[10] Recall that the link algorithm simply checks equality of two sub-strings in σ_i, σ_j. Thus the running time of verify can be made $O(t \log(t))$ by sorting these strings and checking for repetead entries.

References

1. Au, M.H., Chow, S.S.M., Susilo, W., Tsang, P.P.: Short linkable ring signatures revisited. In: Atzeni, A.S., Lioy, A. (eds.) Public Key Infrastructure. pp. 101–115. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
2. Baldimtsi, F., Canetti, R., Yakoubov, S.: Universally composable accumulators. In: Topics in Cryptology – CT-RSA 2020. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Feb 24–28, 2020)
3. Bao, F., Deng, R.H., Zhu, H.: Variations of Diffie-Hellman problem. In: ICICS 03: 5th International Conference on Information and Communication Security. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Oct 10–13, 2003)
4. Benaloh, J.C., de Mare, M.: One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In: Advances in Cryptology – EUROCRYPT'93. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (May 23–27, 1994)
5. Benarroch, D., Campanelli, M., Fiore, D., Kolonelos, D.: Zero-knowledge proofs for set membership: Efficient, succinct, modular. Cryptology ePrint Archive, Report 2019/1255 (2019), https://eprint.iacr.org/2019/1255
6. Blum, M., Feldman, P., Micali, S.: Non-interactive zero-knowledge and its applications. In: Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing. STOC 88, Association for Computing Machinery (1988), https://doi.org/10.1145/62212.62222
7. Bresson, E., Stern, J., Szydlo, M.: Threshold ring signatures and applications to ad-hoc groups. In: Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Aug 18–22, 2002)
8. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Report 2017/1066 (2017), https://eprint.iacr.org/2017/1066
9. Camenisch, J., Kohlweiss, M., Soriente, C.: An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In: PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Mar 18–20, 2009)
10. Campanelli, M., Fiore, D., Querol, A.: LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. Cryptology ePrint Archive, Report 2019/142 (2019), https://eprint.iacr.org/2019/142
11. Canard, S., Schoenmakers, B., Stam, M., Traor, J.: List signature schemes. Discrete Applied Mathematics **154**(2), 189–201 (2006). https://doi.org/https://doi.org/10.1016/j.dam.2005.08.003, https://www.sciencedirect.com/science/article/pii/S0166218X05002283, coding and Cryptography
12. Chaum, D., van Heyst, E.: Group signatures. In: Advances in Cryptology – EUROCRYPT'91. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Apr 8–11, 1991)
13. Chen, J., Hu, Y., Gao, W., Liang, H.: Lattice-based threshold ring signature with message block sharing. KSII Transactions on Internet and Information Systems (02 2018)
14. Dodis, Y., Kiayias, A., Nicolosi, A., Shoup, V.: Anonymous identification in ad hoc groups. In: Advances in Cryptology – EUROCRYPT 2004. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (May 2–6, 2004)

15. Feige, U., Lapidot, D., Shamir, A.: Multiple non-interactive zero knowledge proofs based on a single random string. Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science pp. 308–317 vol.1 (1990)
16. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology — CRYPTO' 86. pp. 186–194. Springer Berlin Heidelberg, Berlin, Heidelberg (1987)
17. Franklin, M., Zhang, H.: A framework for unique ring signatures. Cryptology ePrint Archive, Report 2012/577 (2012), https://eprint.iacr.org/2012/577
18. Fujisaki, E., Suzuki, K.: Traceable ring signature. Cryptology ePrint Archive, Report 2006/389 (2006), https://eprint.iacr.org/2006/389
19. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Aug 20–24, 2017)
20. Haque, A., Krenn, S., Slamanig, D., Striecks, C.: Logarithmic-size (linkable) threshold ring signatures in the plain model. Cryptology ePrint Archive, Report 2020/683 (2020), https://eprint.iacr.org/2020/683
21. Haque, A., Scafuro, A.: Threshold ring signatures: New definitions and post-quantum security. In: PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (May 4–7, 2020)
22. Jayaraman, B., Li, H., Evans, D.: Decentralized certificate authorities. CoRR **abs/1706.03370** (2017), http://arxiv.org/abs/1706.03370
23. Liu, J.K.: Ring Signature, pp. 93–114. Springer Singapore, Singapore (2019). https://doi.org/10.1007/978-981-13-1483-4_5, https://doi.org/10.1007/978-981-13-1483-4_5
24. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) Information Security and Privacy. pp. 325–335. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
25. Liu, J.K., Wong, D.S.: Linkable ring signatures: Security models and new schemes. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) Computational Science and Its Applications – ICCSA 2005. pp. 614–623. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
26. Munch-Hansen, A.: Stronger notions and a more efficient construction of threshold ring signatures (06 2020)
27. Okamoto, T., Tso, R., Yamaguchi, M., Okamoto, E.: A k-out-of-n ring signature with flexible participation for signers. IACR Cryptology ePrint Archive **2018**, 728 (2018)
28. Petzoldt, A., Bulygin, S., Buchmann, J.: A multivariate based threshold ring signature scheme. Cryptology ePrint Archive, Report 2012/194 (2012), https://eprint.iacr.org/2012/194
29. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: Advances in Cryptology – ASIACRYPT 2001. Lecture Notes in Computer Science, Springer, Heidelberg, Germany (Dec 9–13, 2001)
30. Tsang, P.P., Wei, V.K.: Short linkable ring signatures for e-voting, e-cash and attestation. In: Deng, R.H., Bao, F., Pang, H., Zhou, J. (eds.) Information Security Practice and Experience. pp. 48–60. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)

31. Yuen, T.H., Liu, J.K., Au, M.H., Susilo, W., Zhou, J.: Efficient linkable and/or threshold ring signature without random oracles. The Computer Journal **56**(4), 407–421 (2013)
32. Yuen, T., Liu, J., Au, M.H., Susilo, W., Zhou, J.: Threshold ring signature without random oracles (01 2011)
33. Zhou, G., Zeng, P., Yuan, X., Chen, S., Choo, K.K.: An efficient code-based threshold ring signature scheme with a leader-participant model. Security and Communication Networks (08 2017)

Supplementary Materials

A Preliminaries

In this section, we introduce some primitives that we leverage in our constructions. In Section A.1, we describe cryptographic accumulators; in Section A.2, we describe non-interactive zero knowledge arguments of knowledge.

A.1 Accumulators

At a high level, a cryptographic accumulator [4] is defined as a compact representation of a set $\mathcal{S} = \{x_1, \ldots, x_n\}$ that supports proofs of membership in the underlying set. One natural example of a cryptographic accumulator is a Merkle hash tree; the root of the tree is the accumulator value corresponding to the set $\mathcal{S}$ of leaf elements, and the authenticating path of a leaf element is its membership witness. However, the disadvantage of Merkle hash trees is that they are inefficient to use within zero knowledge proofs. Instead, in Section A.1, we describe the RSA accumulator [4], which requires only arithmetic operations and is thus more efficient to use within zero knowledge.

Baldimtsi *et al.* [2] give a thorough guide to accumulators and all of their various flavors. In this paper, we only need a limited subset of accumulator functionality, and we present simplified definitions of accumulators accordingly (pared down from Baldimtsi *et al.* and the work cited therein). In particular, we do not address dynamic changes to the accumulated sets (that is, we only consider *static* accumulators). We also split the algorithm that was called gen in previous work into two: a setup algorithm, and an $\mathsf{accumulate}$ algorithm. This allows us to include the parameters produced by gen that are independent of the accumulated set in the public parameters of our threshold ring signature scheme.

Accumulator Syntax An accumulator parameterized by a domain $\mathcal{D}$ has the following algorithms:

$\mathsf{setup}(1^\lambda) \to \mathsf{pp}$**:**
: An algorithm that, given the security parameter, sets up the global public parameters for the accumulator system.

$\mathsf{accumulate}(\mathsf{pp}, \mathcal{S}) \to a_{\mathcal{S}}$**:**
: An algorithm that, given the global public parameters pp and a set $\mathcal{S} \subseteq \mathcal{D}$, returns an accumulator $a_{\mathcal{S}}$ representing the set $\mathcal{S}$. In this paper, we require this algorithm to be deterministic.

$\mathsf{witcreate}(\mathsf{pp}, \mathcal{S}, x) \to w_a$**:**
: An algorithm that, given the parameters pp, a set $\mathcal{S} \subseteq \mathcal{D}$ and an element $x \in \mathcal{S}$, returns a membership witness w_a for the element x.

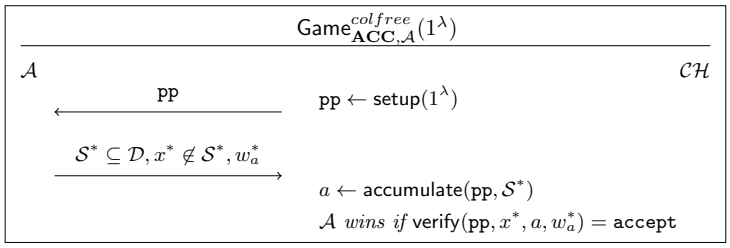
$\mathsf{verify}(\mathsf{pp}, x, a, w_a) \to \mathtt{accept/reject}$**:**
: An algorithm that, given the parameters pp, an element x, an accumulator a and a witness w_a, checks whether w_a proves that x is in the underlying set a.

Accumulator Security Definitions Of course, an accumulator must be *correct* (that is, verification using an honestly produced witness must return `accept`). The important security property of an accumulator is *collision freeness* as defined below.

Definition 8 (Collision Freeness for Accumulators).

Informally, an accumulator is collision-free *if it is hard to fabricate a membership witness w_a for a value x that is not in some accumulated set a.*

More formally, let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathbf{ACC} = (\mathsf{setup}, \mathsf{accumulate}, \mathsf{witcreate}, \mathsf{verify})$ *be an accumulator scheme. Consider the following game between a probabilistic polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$:*



ACC is collision-free *for the domain $\mathcal{D}$ of elements if for any sufficiently large security parameter λ, for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function ν in the security parameter λ such that the probability that $\mathcal{A}$ wins the game is less than $\nu(\lambda)$.*

The RSA Accumulator The RSA accumulator, which was the original accumulator introduced by Benaloh and DeMare [4], is the one most suitable for our needs. The domain $\mathcal{D}$ for the RSA accumulator is the set of prime integers. We describe the RSA accumulator below.

$\mathsf{setup}(1^\lambda)$**:**

1. Select two 1^λ-bit safe primes $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are also prime, and let $m = pq$.
2. Select a random integer $g' \leftarrow \mathbb{Z}_m^*$.
3. Let $g = (g')^2 \bmod m$.
4. Return $\mathsf{pp} = (m, g)$.

$\mathsf{accumulate}(\mathsf{pp} = (m, g), \mathcal{S})$**:**

Return $a = g^{\prod_{x \in \mathcal{S}} x} \bmod m$.

$\mathsf{witcreate}(\mathsf{pp} = (m, g), \mathcal{S}, x)$**:**

Return $w_a = g^{\prod_{y \in \mathcal{S}, y \neq x} y} \bmod m$.

$\mathsf{verify}(\mathsf{pp} = (m, g), x, a, w_a)$**:**

If x is a prime and $w_a^x \bmod m = a$, return `accept`. Otherwise, return `reject`.

The RSA accumulator is collision-free under the strong RSA assumption.

A.2 Non-Interactive Zero Knowledge Arguments of Knowledge (NIZKAoK)

Non-Interactive Zero-Knowledge (NIZK) proof and argument systems are a well studied area and have been so for over 30 years [6, 15, 16]. Informally, a zero-knowledge proof of knowledge allows a prover to convince a verifier that the prover knows a witness w for a statement ϕ such that (ϕ, w) satisfy some relation $\mathcal{R}$. The difference between a proof and an argument is in the soundness requirement; a proof guarantees that even an all-powerful prover cannot break soundness, while an argument only guarantees soundness against *efficient* (computationally bounded) provers. Generally, for practical purposes, an argument is enough.

In this section we present the definition of a non-interactive zero knowledge argument of knowledge (NIZKAoK), taken from the work of Groth and Maller [19]. We also describe the concrete relation which we will need in Section 3.2.

NIZKAoK Syntax A NIZKAoK scheme has the following algorithms, as described by Groth and Maller [19]:

$\mathsf{setup}(1^\lambda, \mathcal{R}) \to (\mathtt{crs}, \mathtt{td})$**:**
An algorithm that, given the security parameter, sets up the common reference string `crs` and the trapdoor `td` for the NIZKAoK system.

$\mathsf{prove}(\mathtt{crs}, \phi, w) \to \pi$**:**
An algorithm that, given the common reference string `crs` for a relation $\mathcal{R}$, a statement ϕ and a witness w, returns a proof π that $(\phi, w) \in \mathcal{R}$.

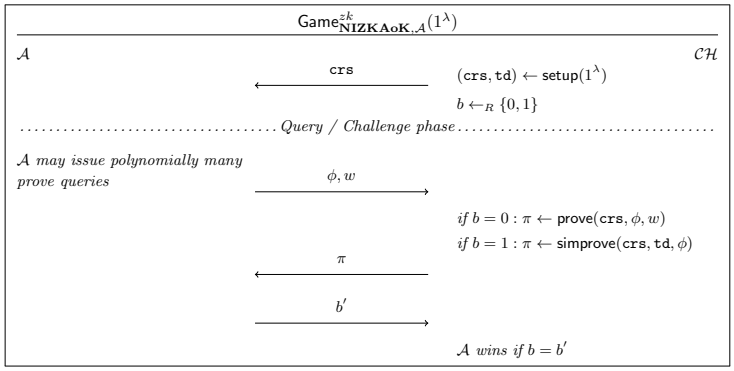
$\mathsf{verify}(\mathtt{crs}, \phi, \pi) \to \mathtt{accept/reject}$**:**
An algorithm that, given the common reference string `crs` for a relation $\mathcal{R}$, a statement ϕ and a proof π, checks whether π proves the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.

$\mathsf{simprove}(\mathtt{crs}, \mathtt{td}, \phi) \to \pi$**:**
An algorithm that, given the common reference string `crs` for a relation $\mathcal{R}$, the trapdoor `td` and a statement ϕ, simulates a proof of the existence of a witness w such that $(\phi, w) \in \mathcal{R}$.

NIZKAoK Security Definitions Of course, a NIZKAoK scheme must be *correct* (that is, verification using an honestly produced proof must return `accept`). The important security properties of a NIZKAoK scheme are *zero knowledge*, *knowledge soundness*, and *simulation extractability*, described below.

Definition 9 (Zero Knowledge for NIZKAoK). *Informally, a NIZKAoK scheme has* zero knowledge *if a proof does not leak any more information than the truth of the statement.*

More formally, let $\lambda \in \mathbb{N}$ *be the security parameter, and let* $\mathbf{NIZKAoK} = (\mathsf{setup}, \mathsf{prove}, \mathsf{verify}, \mathsf{simprove})$ *be a NIZKAoK scheme. Consider the following game between a probabilistic polynomial-time adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*:*



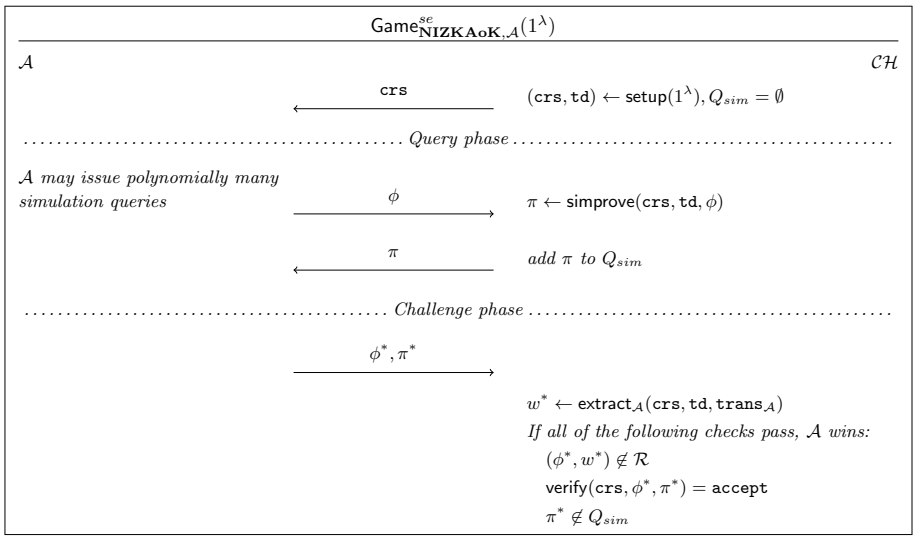
NIZKAoK *has* zero knowledge *if for any sufficiently large security parameter λ, for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function ν in the security parameter λ such that the probability that $\mathcal{A}$ wins the game is less than $\frac{1}{2} + \nu(\lambda)$.*

Informally, *knowledge soundness* is the property that guarantees that it is always possible to extract a valid witness from a proof that verifies. *Simulation extractability* is a stronger version of knowledge soundness that it is always possible to extract a valid witness from a proof that verifies even if the adversary has access to a simulation oracle. This is a flavor of non-malleability; an adversary should not even be able to modify a simulated proof in order to forge a proof.

Definition 10 (Simulation Extractability for NIZKAoK). *Informally, a NIZKAoK scheme has* simulation extractability *if it is always possible to extract a valid witness from a proof that verifies.*

More formally, let $\lambda \in \mathbb{N}$ be the security parameter, and let $\mathbf{NIZKAoK} = (\mathsf{setup}, \mathsf{prove}, \mathsf{verify}, \mathsf{simprove})$ *be a NIZKAoK scheme. Consider the following game between a probabilistic polynomial-time adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$, where $\mathtt{trans}_{\mathcal{A}}$ denotes the adversary's inputs and outputs, including its randomness*[11]*:*

[11] In the standard simulation-extractability for NIZKs the extractor extracts the witness from the proof only. The definition of Groth and Maller which we use here is more general and also captures non-black box extractions which is used e.g., in SNARKS.



NIZKAoK *has* simulation extractability *if for any sufficiently large security parameter λ, for any probabilistic polynomial-time adversary $\mathcal{A}$, there exists an extraction algorithm $\mathsf{extract}_{\mathcal{A}}$ and a negligible function ν in the security parameter λ such that the probability that $\mathcal{A}$ wins the game is less than $\nu(\lambda)$.*

NIZKAoK For Our Needs In Section 3.2 (and subsequently Section 3.3) we use a NIZKAoK for a relation essentially proving that a signer knows some secret key for a public key $pk_i = g^{sk_i}$ that is within a set of signers $\mathcal{R}$ accumulated as $a_{\mathcal{R}}$ and this secret key is used to compute $\sigma = \mathsf{H}(msg)^{sk_i}$. Throughout Section 3.2 and Section 3.3 we are agnostic to how this specific proof can be instantiated, but we now describe an instantiation of this proof that uses commit-and-prove NIZKs.

Combining NIZKs using commit-and-prove We use the framework for black-box modular composition of commit-and-prove NIZKs (CP-NIZKs) [5, 10]. A CP-NIZK is, informally, a NIZK that can efficiently prove properties of committed inputs through some commitment scheme. Let x be a public input and c a commitment. A CP-NIZK is then a NIZK proving knowledge of (u, w, r) such that u, r open c and the relation $\mathcal{R}(x; u, w)$ is true. In this general description, w is some non-committed part of the witness (potentially nothing). Now, through the commitments, we may compose several CP-NIZKs together in an efficient manner. Given two CP-NIZKs for two relations $\mathcal{R}_0$ and $\mathcal{R}_1$, respectively, we can prove their conjunction $\mathcal{R}(x_0, x_1, u, w_0, w_1) = \mathcal{R}_0(x_0, u, w_0) \wedge \mathcal{R}_1(x_1, u, w_1)$, where u is a shared part of the witness. The prover commits to u as c_u, generates proofs π_0, π_1 from the respective schemes and outputs $\pi = (c_u, \pi_0, \pi_1)$. The verifier then checks each proof of the respective inputs (x_0, c_u) and (x_1, c_u). The following theorem follows directly from [10].

Theorem 3 (Black-Box Composition of CP-NIZKs). *The construction above is a secure NIZK for the relation $\mathcal{R}$.*

In Section 3.2, we will want to use the following relation $\mathcal{R}_{\mathsf{sig}}$:

$$\mathcal{R}_{\mathsf{sig}}\begin{pmatrix}\phi = (\mathbb{G}, g, \mathbf{RSAACC}.\mathsf{pp},\\ \quad a_{\mathcal{R}}, \sigma', h),\\ w = (pk, sk, w_a)\end{pmatrix} = \begin{pmatrix}(pk = g^{sk})\\ \wedge \mathbf{RSAACC}.\mathsf{verify}(\\ \qquad \mathbf{RSAACC}.\mathsf{pp}, pk, a_{\mathcal{R}}, w_a)\\ \wedge(\sigma' = h^{sk})\end{pmatrix}$$

A proof for this relation, however, is complicated due to the nature of the construction in which we use it (a unique ring signature scheme), as we cannot leak the public key pk used within the proof; this would naturally leak the signer's identity. So, we instead decompose $\mathcal{R}_{\mathsf{sig}}$ into three minor commit-and-prove schemes, and then combine these using Theorem 3. We decompose $\mathcal{R}_{\mathsf{sig}}$ into the three relations $\mathcal{R}_{pk}$, $\mathcal{R}_{\mathbf{RSAACC}}$ and $\mathcal{R}_{\sigma}$. Given commitments c_{pk} and c_{sk} to the public and secret keys, respectively, $\mathcal{R}_{pk}$ proves that sk corresponds to pk; $\mathcal{R}_{\mathbf{RSAACC}}$ proves that pk belongs to the accumulator; and $\mathcal{R}_{\sigma}$ proves that sk was indeed used to compute $\sigma' = \mathsf{H}(msg)^{sk}$. By combining these, we prove the complete relation $\mathcal{R}_{\mathsf{sig}}$. Formally we define the three relations as follows:

$$\mathcal{R}_{pk}\begin{pmatrix}\phi = (\mathsf{pp}, c_{pk}, c_{sk}, g, g_1, h_1, g_2, h_2),\\ w = (pk, sk, r_1, r_2))\end{pmatrix} = \begin{pmatrix}\wedge c_{pk} = g_1^{pk} h_1^{r_1}\\ \wedge c_{sk} = g_2^{sk} h_2^{r_2}\\ \wedge pk = g^{sk}\end{pmatrix}$$

$$\mathcal{R}_{\mathbf{RSAACC}}\begin{pmatrix}\phi = (\mathbb{G}, g, \mathbf{RSAACC}.\mathsf{pp}, a_{\mathcal{R}}, g_1, h_1)\\ w = (pk, w_a, r)\end{pmatrix}$$
$$= \begin{pmatrix}\mathbf{RSAACC}.\mathsf{verify}(\mathbf{RSAACC}.\mathsf{pp}, pk, a_{\mathcal{R}}, w_a)\wedge\\ c_{pk} = g_1^{pk} h_1^{r}\end{pmatrix}$$

$$\mathcal{R}_{\sigma}\begin{pmatrix}\phi = (\mathsf{pp}, c_{sk}, \sigma', h, g_1, h_1)\\ w = (sk, r)\end{pmatrix} = \begin{pmatrix}\sigma' = h^{sk}\\ \wedge c_{sk} = g_1^{sk} h_1^{r}\end{pmatrix}$$

We can now obtain a proof for $\mathcal{R}_{\mathsf{sig}}$ by composing proofs for $\mathcal{R}_{pk}$, $\mathcal{R}_{\mathbf{RSAACC}}$ and $\mathcal{R}_{\sigma}$ and applying Theorem 3.

Instantiating Relations We instantiate $\mathcal{R}_{\mathbf{RSAACC}}$ using the set membership proof described in Section 4 of [5] while $\mathcal{R}_{\sigma}$ is a Schnorr-like proof. The protocol for $\mathcal{R}_{\sigma}$ is described in Figure 6, and can be made non-interactive using the Fiat-Shamir transformation. The relation $\mathcal{R}_{pk}$ can be instantiated using Bulletproofs [8], a zero-knowledge scheme with short proofs that are compatible with [5]. Lastly, we note that if we use the transformation of Fiat and Shamir [16] to make our proof non-interactive, we don't automatically get simulation-extractability (Definition 10). Therefore, we must augment the resulting non-interactive proof by encrypting the witness to a public key pk embedded in the common reference string. The corresponding secret key would be the trapdoor `td`, and in a real execution, would not be known to anyone. We would also require a proof that the encryption is correct.

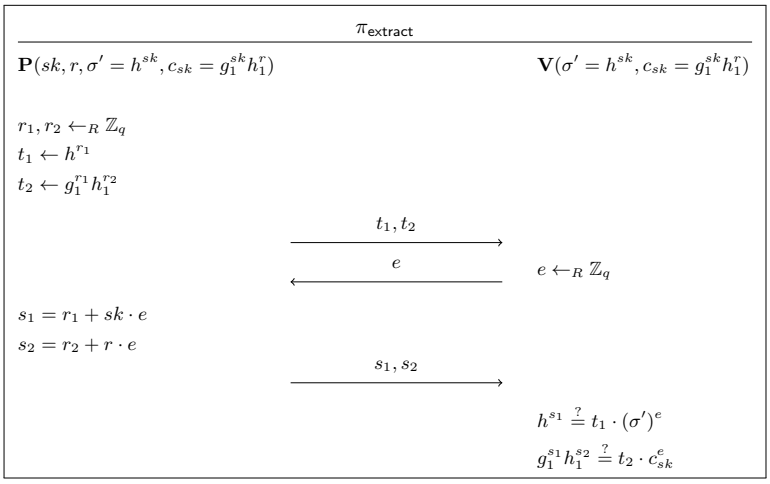


Fig. 6: Proof for Relation $\mathcal{R}_\sigma$

A.3 The Generalized Decisional Diffie-Hellman Problem

We leverage the Generalized Decisional Diffie-Hellman (Generalized DDH) Problem [3], described below.

Definition 11. *The* Generalized DDH Problem *in group* $\mathbb{G}$ *asks that, given a polynomial-length list* L *of tuples* (u, v) *of elements in a group* $\mathbb{G}$*, an adversary* $\mathcal{A}$ *determines whether there exists a fixed* r *such that for all* $(u, v) \in L$ u *is a random element of* $\mathbb{G}$ *and* $v = u^r$*, or* v *and* u *are independent random elements of* $\mathbb{G}$*.*

The generalized DDH problem is considered to be hard in group $\mathbb{G}$ if for all efficient adversaries $\mathcal{A}$, the probability that $\mathcal{A}$ solves a random instance of the generalized DDH problem correctly is only negligibly greater than $\frac{1}{2}$. (We define a random instance of the generalized DDH problem as an L contains independent random elements with probability $\frac{1}{2}$, and elements $v = u^r$ for a random r and independent random values u otherwise.)

B Proof of Security of Construction 1

We prove Theorem 1 in several steps. First, correctness is apparent on inspection. Second, in Lemma 1 we address uniqueness (Definition 6). Last, in Lemma 2 we address cross-message unlinkability (Definition 7).

Lemma 1. *Construction 1 is unique under the assumptions listed in Theorem 1.*

Proof. We will construct an algorithm $\mathcal{B}$ which will use an adversary $\mathcal{A}$ who can break the uniqueness of the URS scheme in Construction 1 to break the discrete

logarithm problem with non-negligible probability if **NIZKAoK** and **ACC** are both secure, if the generalized DDH assumption holds, and if H is a random oracle.

We augment our algorithm $\mathcal{B}$ with the following powers:

Programmable Random Oracle: We allow $\mathcal{B}$ to program the random oracle H.

Simulation Extractor $\mathsf{extract}_{\mathcal{A}}$: We give $\mathcal{B}$ access to the **NIZKAoK** simulation extractor $\mathsf{extract}_{\mathcal{A}}$ corresponding to the adversary $\mathcal{A}$. Such an efficient extractor is guaranteed to exist, by the simulation extractability of **NIZKAoK** (Definition 10).

Inputs and Outputs of $\mathcal{A}$: We give $\mathcal{B}$ access to the inputs and outputs of $\mathcal{A}$, including its randomness tape. We denote this transcript as $\mathtt{trans}_{\mathcal{A}}$.

We build $\mathcal{B}$ in a sequence of games. The final game — $\mathcal{G}_6$ — describes the full behavior of $\mathcal{B}$. If the adversary $\mathcal{A}$ can distinguish interacting with $\mathcal{B}$ from interacting with an honest challenger, it will have broken **NIZKAoK**, **ACC**, or the generalized DDH assumption. If it cannot distinguish between the two, then it must supply $\mathcal{B}$ with sufficiently many unlinked signatures with non-negligible probability, which $\mathcal{B}$ can then use to solve an instance of the discrete logarithm problem. (Note that $\mathcal{B}$ only solves the discrete logarithm problem with respect to prime challenges; however, since there is a noticeable probability that a random input to the discrete logarithm problem will be prime, this is sufficient.)

Game $\mathcal{G}_0$: $\mathcal{B}$ honestly executes the role of the challenger in the uniqueness game described in Definition 6.

Game $\mathcal{G}_1$: This is the same as the previous game, but instead of computing the proofs π honestly in response to signing queries, $\mathcal{B}$ uses the trapdoor **NIZKAoK**.$\mathtt{td}$ to simulate the proofs using the **NIZKAoK**.$\mathsf{simprove}$ algorithm.

This game is indistinguishable from $\mathcal{G}_0$ by the zero knowledge property of **NIZKAoK** (Definition 9). Imagine that $\mathcal{B}$ interacts with a zero knowledge challenger to obtain **NIZKAoK**.$\mathtt{crs}$ and the proofs π. If, in the game described in Definition 9, the challenger chooses $b = 0$, the view of the adversary will be as in the previous game; if instead the challenger chooses $b = 1$, the view of the adversary will be as in this game. If it can guess b with non-negligible probability, it will have broken zero-knowledge.

Game $\mathcal{G}_2$: This is the same as the previous game, but $\mathcal{B}$ keeps track of all of the messages msg it is asked signing queries on, or which it is given forgeries for. If it sees $msg_0 \neq msg_1$ such that $\mathsf{H}(msg_0) = \mathsf{H}(msg_1)$, it aborts.

$\mathcal{B}$ only aborts with negligible probability, since if $\mathcal{A}$ can find two messages that hash to the same thing, it can be used to break the collision-resistance of H.

Game $\mathcal{G}_3$: This is the same as the previous game, but $\mathcal{B}$ keeps track of all of the signing sets $\mathcal{R}$ it is asked signing queries on behalf of, or which it is given forgeries on behalf of. If it ever sees two signer sets $\mathcal{R}, \mathcal{R}'$ such that $\{pk_i\}_{i \in \mathcal{R}} \neq$

$\{pk_i\}_{i\in\mathcal{R}'}$ and $a_{\mathcal{R}} = a_{\mathcal{R}'}$ (where $a_{\mathcal{R}} = \mathbf{ACC}.\mathsf{accumulate}(\mathbf{ACC}.\mathsf{pp}, \{pk_i\}_{i\in\mathcal{R}})$), it aborts.

$\mathcal{B}$ only aborts with negligible probability, since if $\mathcal{A}$ can find two signer sets that accumulate to the same value, it can be used to break the collision freeness of **ACC** (Definition 8).

Game $\mathcal{G}_4$: This is the same as the previous game, but when the adversary returns its unlinked signatures, $\mathcal{B}$ extracts the witnesses from those signatures that were not previously returned in response to a signing oracle query. Let $L \subset [t]$ be the indices of such signatures. For $(msg^*, \{(\sigma_k = (\sigma'_k, \pi_{S,k}), \mathcal{R}_k)\}_{k\in L})$, $\mathcal{B}$ extracts the witnesses $w_k = (pk_{i_k}, sk_{i_k}, w_{a,k}) \leftarrow \mathsf{extract}_{\mathcal{A}}(\mathbf{NIZKAoK}.\mathtt{crs}, \mathbf{NIZKAoK}.\mathtt{td}, \mathtt{trans}_{\mathcal{A}})$.

If it holds that $(\phi_k = (\mathbb{G}, g, \mathbf{ACC}.\mathsf{pp}, a_{\mathcal{R}_k}, \sigma'_k, \mathsf{H}(msg^*)), w_k) \notin \mathcal{R}$ and $\mathbf{NIZKAoK}.\mathsf{verify}(\mathbf{NIZKAoK}.\mathtt{crs}, \phi_k, \pi_k) = \mathtt{accept}$, $\mathcal{B}$ aborts.

Since in the previous two games $\mathcal{B}$ aborted if it ever saw two messages hash to the same value or two signer sets accumulate to the same value, it must be that the statement ϕ_k is one it has never returned a proof for.

$\mathcal{B}$ only aborts with negligible probability, since if $\mathcal{A}$ can find such a statement ϕ and witness w that cause $\mathcal{B}$ to abort, $\mathcal{A}$ can trivially be used to break the simulation extractability of **NIZKAoK** (Definition 10). (Just imagine that $\mathcal{B}$ interacts with a simulation extractability challenger to obtain $\mathbf{NIZKAoK}.\mathtt{crs}$ and the simulated proofs π for signing query responses, and forwards the proofs to $\mathcal{A}$. It then forwards the proofs supplied by $\mathcal{A}$ to the challenger. Of course, in the simulation extractability game, the adversary gives only one proof π from which extraction should succeed; however, if $\mathcal{B}$ picks a proof from $\{\pi_{S,k}\}_{k\in L}$ at random to forward to the challenger, if extraction fails for any of the proofs, $\mathcal{B}$ breaks simulation extractability with non-negligible probability.)

If $\mathcal{A}$ succeeds in winning the uniqueness game and if $\mathcal{B}$ does not abort at this point, $\mathcal{B}$ has successfully extracted witnesses $\{w_k = (pk_{i_k}, sk_{i_k}, w_{a,k})\}_{k\in L}$ from the unique signatures $(msg^*, \{(\sigma_k = (\sigma'_k, \pi_{S,k}), \mathcal{R}_k)\}_{k\in[t]})$ such that $(pk_{i_k} = g^{sk_{i_k}}) \wedge \mathbf{ACC}.\mathsf{verify}(\mathbf{ACC}.\mathsf{pp}, a_{\mathcal{R}_k}, pk_{i_k}, w_{a,k}) \wedge (\sigma'_k = \mathsf{H}(msg^*)^{sk_{i_k}})$. For $\mathcal{A}$ to have won, it must also be true that there exists a $k^* \in L$ such that i_{k^*} is not corrupt.

Game $\mathcal{G}_5$: This is the same as the previous game, but $\mathcal{B}$ now aborts if $\mathcal{A}$ can be used to break the collision freeness property of **ACC** (Definition 8). Recall that $\mathcal{B}$ computes $a_{\mathcal{R}_{k^*}}$ as $a_{\mathcal{R}_{k^*}} \leftarrow \mathbf{ACC}.\mathsf{accumulate}(\mathbf{ACC}.\mathsf{pp}, \{pk_i\}_{i\in\mathcal{R}_{k^*}})$. $\mathcal{B}$ aborts if $\mathbf{ACC}.\mathsf{verify}(\mathbf{ACC}.\mathsf{pp}, pk_{i_k}, a_{\mathcal{R}_{k^*}}, w_{a,k^*}) = \mathtt{accept}$, and $pk_{i_{k^*}} \notin \{pk_i\}_{i\in\mathcal{R}_{k^*}}$.

$\mathcal{B}$ only aborts with negligible probability, since if $\mathcal{A}$ finds $pk_{i_{k^*}}, w_{a,k^*}, \mathcal{R}_{k^*}$ that make $\mathcal{B}$ abort, $\mathcal{A}$ can trivially be used to break the collision freeness property of **ACC**. (Just imagine that $\mathcal{B}$ interacts with a collision freeness challenger to obtain $\mathbf{ACC}.\mathsf{pp}$.)

If $\mathcal{A}$ succeeds in winning the uniqueness game and if $\mathcal{B}$ does not abort at this point, it must be that $pk_{i_k} \in \{pk_i\}_{i\in\mathcal{R}_k}$.

Game $\mathcal{G}_6$: If $\mathcal{A}$ successfully breaks the uniqueness property, we know that each pair of signatures it returned appears unique. So, $\sigma'_k = \mathsf{H}(msg)^{sk_{i_k}}$ for $k \in L$

are all distinct, and sk_{i_k} for $k \in L$ must all be distinct as well. At least one of those secret keys belongs to an honest party. Let $i^* = i_{k^*}$ be the identity of that honest signer (whose secret key sk_{i^*} was extracted by $\mathcal{B}$).

In this game, $\mathcal{B}$ guesses i^* at the beginning of the game. If $\mathcal{B}$ does not abort in the previous game, pk_{i^*} is guaranteed to be an actual public key corresponding to one of the signers i^* in the system (of which there are polynomially many), so $\mathcal{B}$ has a non-negligible (one-in-polynomial) chance of guessing correctly. At the beginning, when it is generating public-private key pairs, it generates all the others honestly, but sets pk_{i^*} to a random prime element of $\mathbb{G}$ (for which it does not know the corresponding secret key). (Note that pk_{i^*} is still identically distributed to an honestly generated public key.)

Now that $\mathcal{B}$ does not know sk_{i^*}, it will have trouble coming up with $\sigma' = \mathsf{H}(msg)^{sk_{i^*}}$ for signing queries on msg on behalf of signer i^*. (Note that the proof π in the signature is already being simulated, and so not knowing sk_{i^*} does not pose an obstacle to producing π.) Instead of computing them honestly, $\mathcal{B}$ will now pick σ' to be a random element of $\mathbb{G}$ (consistently returning the same element per message msg that $\mathcal{A}$ asks for a signature from signer i^* on).

This game is indistinguishable from the previous game by the hardness of the generalized decisional Diffie-Hellman problem, thanks to the use of the programmable random oracle H. Just imagine that $\mathcal{B}$ interacts with a generalized DDH challenger at the beginning of the game to obtain $(u_1 = g, v_1 = pk_{i^*})$ (aborting if pk_{i^*} isn't prime) and all $(u = \mathsf{H}(msg), v = \sigma')$ pairs. $\mathcal{B}$ will store the (u, v) tuples, and set $\mathsf{H}(msg) = u, \sigma = v$ as needed.

Finally, if $\mathcal{B}$ is correct in its guess of i^*, then it will have been able to use $\mathcal{A}$ to compute the discrete log of pk_{i^*}, since if $\mathcal{A}$ succeeds in winning the unforgeability game, $\mathcal{B}$ can extract sk_{i^*} such that $pk_{i^*} = g^{sk_{i^*}}$ from $\mathcal{A}$'s forgery. (Just imagine that, instead of picking pk_{i^*} randomly, $\mathcal{B}$ gets pk_{i^*} as a discrete log challenge.)

□

Lemma 2. *Construction 1 is cross-message unlinkable under the assumptions listed in Theorem 1 (however, it does not require the security of the RSA accumulator).*

Proof. **Game $\mathcal{G}_0$:** $\mathcal{B}$ honestly executes the role of the challenger in the cross-message unlinkability game described in Definition 7.

Game $\mathcal{G}_1$: This is the same as the previous game, but instead of computing the proofs π honestly in response to signing queries, $\mathcal{B}$ uses the trapdoor **NIZKAoK**.$\mathtt{td}$ to simulate the proofs using the **NIZKAoK**.$\mathsf{simprove}$ algorithm.

This game is indistinguishable from $\mathcal{G}_0$ by the zero knowledge property of **NIZKAoK** (Definition 9) (as in the proof of Lemma 1).

Game $\mathcal{G}_2$: At the beginning of this game, $\mathcal{B}$ guesses the signer index i_0 that $\mathcal{A}$ will ask for a challenge on. It also guesses when $\mathcal{A}$ will ask the first hash query

on the challenge message msg_0 ("never" being a valid guess). $\mathcal{B}$ has a non-negligible (one-in-polynomial) chance of guessing both those things correctly. It sets pk_{i_0} to be a random prime element of $\mathbb{G}$ (such that the corresponding secret key is not known) and $\mathsf{H}(msg_0)$ to be a random element of $\mathbb{G}$.
Now that $\mathcal{B}$ does not know sk_{i_0}, it will have trouble coming up with $\sigma' = \mathsf{H}(msg)^{sk_{i^*}}$ for signing queries (/ challenges) on msg on behalf of signer i_0. (Note that the proof π in the signature is already being simulated, and so not knowing sk_{i_0} does not pose an obstacle to producing π; the only remaining challenge is in producing σ'.) Instead of computing σ' honestly, $\mathcal{B}$ will now pick σ' to be a random element of $\mathbb{G}$ (consistently returning the same element per message msg that $\mathcal{A}$ asks for a signature from signer i_0 on).
If $\mathcal{B}$ is incorrect in its guesses, it aborts.
Just like in the last game of the proof of Lemma 1, if $\mathcal{B}$ does not abort, this game is indistinguishable from the previous game by the generalized DDH assumption, thanks to the powers of the programmable random oracle.

Game $\mathcal{G}_3$: At the beginning of this game, $\mathcal{B}$ additionally guesses the signer index i_1 that $\mathcal{A}$ will ask for a challenge on. It also guesses when $\mathcal{A}$ will ask the first hash query on the challenge message msg_1 ("never" being a valid guess). If $\mathcal{B}$ is incorrect in its guesses, it aborts. It handles signing queries / challenges for i_1 just like it does for i_0.
If $\mathcal{B}$ does not abort, this game is indistinguishable from the previous game, for the same reasons as above.
Note that now, the distribution of the challenge is independent of b, so the adversary cannot win with probability greater than $\frac{1}{2}$.

□

C Proof of Security of Construction 2

We prove Theorem 2 in several steps. First, correctness is apparent on inspection. Second, in Lemma 3 we address anonymity. Last, in Lemma 4 we address unforgeability.

Lemma 3. *If URS satisfies* cross-message unlinkability *(Definition 7) then Construction 2 satisfies* anonymity *(Definition 4).*

Proof. We will construct an algoritm $\mathcal{B}$ which will break the *cross-message unlinkability* of the underlying URS scheme against a URS challenger $\mathcal{CH}$, by assuming we have an attacker $\mathcal{A}$ who can break the *anonymity* of the TRS scheme in Construction 2.

Setup $\mathcal{B}$ receives from $\mathcal{A}$ the set of users $\mathcal{U}$ on which $\mathcal{A}$ wants to play the game. $\mathcal{B}$ then sets up the game with the URS challenger $\mathcal{CH}$, receiving the public parameters $\mathtt{pp}$ as well as public keys for each user $i \in \mathcal{U}$. It forwards this information to the TRS adversary $\mathcal{A}$.

First Query Phase $\mathcal{A}$ may issue **corruption**, **signing** and **registration** queries to $\mathcal{B}$, which are handled as follows:

Corruption or Registration for Party i: $\mathcal{B}$ forwards this query to $\mathcal{CH}$, and the answer from $\mathcal{CH}$ is then forwarded back to $\mathcal{A}$.

Signing message msg by $i \in \mathcal{R}$: $\mathcal{B}$ sets $msg' = (msg, \{pk_j\}_{j \in \mathcal{R}})$, and issues a signing query to $\mathcal{CH}$ for $(msg', \mathcal{R}, i)$, getting σ_i. $\mathcal{B}$ then returns σ_i to $\mathcal{A}$.

Challenge Once $\mathcal{A}$ is done issuing queries, $\mathcal{A}$ sends $\mathcal{B}$ a challenge message msg^*, a ring $\mathcal{R}^*$, and two users i_0^* and i_1^*. $\mathcal{B}$ then sets $msg_0 = \perp$ (or any other arbitrary value) and $msg_1 = (msg^*, \{pk_i\}_{i \in \mathcal{R}^*})$. $\mathcal{B}$ then sends $(msg_0, msg_1, \mathcal{R}^*, \mathcal{R}^*, i_0^*, i_1^*)$ to $\mathcal{CH}$ and gets back (σ_0^*, σ_1^*). $\mathcal{B}$ forwards σ_1^* to $\mathcal{A}$.

Second Query Phase $\mathcal{A}$ is allowed to issue additional **signing**, **registration** and **corruption** queries to $\mathcal{B}$. These are handled in similar fashion to the first query phase.

Challenge Response $\mathcal{A}$ returns a bit b' to $\mathcal{B}$. This bit b' is simply fowarded to $\mathcal{CH}$.

If $\mathcal{A}$ has a non-negligible probability of winning the TRS *anonymity* game then $\mathcal{B}$ has non-negligible probability of winning the URS *cross-message unlinkability* game. This follows from σ_1^* being a partial signature for msg^* using the ring $\mathcal{R}^*$ signed by either i_0^* or i_1^*. If $\mathcal{A}$ can then with non-negligible probability guess which case we are in, with $b = 0$ implying that i_0^* signed the message and $b = 1$ that i_1^* signed the message, then $\mathcal{B}$ will with non-negligible probability guess correctly as well. □

Lemma 4. *If URS satisfies* uniqueness *(Definition 6) then Construction 2 satisfies* unforgeability *(Definition 3).*

Proof. We will construct an algorithm $\mathcal{B}$ which will break the *uniqueness* of the underlying URS scheme against a URS challenger $\mathcal{CH}$, by assuming we have an attacker $\mathcal{A}$ who can break the *unforgeability* of the TRS scheme in Construction 2.

Setup $\mathcal{B}$ receives from $\mathcal{A}$ the set of users $\mathcal{U}$ on which $\mathcal{A}$ wants to play the game. $\mathcal{B}$ then sets up the game with the URS challenger $\mathcal{CH}$, receiving the public parameters $\mathtt{pp}$ as well as public keys for each user $i \in \mathcal{U}$. $\mathcal{B}$ forwards this information to the TRS adversary $\mathcal{A}$.

Query $\mathcal{A}$ may issue **corruption**, **signing** and **registration** queries to $\mathcal{B}$, which $\mathcal{B}$ handles as in the proof of Lemma 3, by forwarding to the URS challenger $\mathcal{CH}$ (with the appropriate modifications to the messages) and returning the challenger's response to the adversary.

Challenge $\mathcal{A}$ produces a signature σ^* on some message msg^* and under some ring $\mathcal{R}^*$ such that fewer than t members of $\mathcal{R}^*$ are corrupt.

- $\mathcal{B}$ parses $\sigma^* = \{\sigma_1^*, \ldots, \sigma_t^*\}$.
- Let $msg' = (msg^*, \{pk_i\}_{i \in \mathcal{R}^*})$.
- $\mathcal{B}$ sends $\{(msg', \sigma_k^*, \mathcal{R}^*)\}_{k \in [t]}$ to $\mathcal{CH}$.

If $\mathcal{A}$ has a non-negligible probability of winning the TRS unforgeability game, then $\mathcal{B}$ has non-negligible probability of winning the URS uniqueness game against the challenger $\mathcal{CH}$. □

Table of Contents