Stronger Notions and a More Efficient Construction of Threshold Ring Signatures

Alexander Munch-Hansen, Claudio Orlandi, Sophia Yakoubov

June 6, 2020

Abstract

A *ring signature* (introduced by Rivest *et al.* [RST01]) allows a signer to sign a message without revealing their identity by anonymizing themselves within a group of users (chosen by the signer in an ad-hoc fashion at signing time). The signature proves that one member of the group is the signer, but does not reveal which one.

We consider *threshold ring signatures* (introduced by Bresson *et al.* [BSS02]), where any t signers can sign a message together 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.

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.

Contents

1 Introduction

A *ring signature* (introduced by Rivest *et al.* [RST01]) allows a user to sign a message without revealing their identity by anonymizing themselves within some group of users. The signature proves that one member of the group signed, but does not reveal which one. (A similar primitive is a *group signature*; however, in a group signature scheme one group is fixed during setup, while in a ring signature scheme the signer chooses members of the group at signing time.)

We consider *threshold ring signatures* (introduced by Bresson *et al.* [BSS02]), where any t signers can sign a message together while anonymizing themselves within a larger (size-n) group. The signature proves that t members of the group signed without revealing which ones. Like ring signatures, threshold ring signatures allow the signers to pick the larger group they want to anonymize themselves amongst in an ad-hoc way at signing time.

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. 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 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.

1.1 Applications

Cryptocurrency Wallets One possible application of threshold ring signatures is a shared cryptocurrency wallet. To use a shared wallet, some implementations require that a threshold t of the wallet owners sign a transaction; privacy demands that the owners not have to reveal their identities to do so. In this scenario, all of a wallet's owners would constitute the larger size-n group, while the ones signing are the size-t sub-group. Efficiency is of utmost importance due to the sheer amount of transactions performed in most cryptocurrencies; communication complexity should be kept to a minimum.

One might think to use regular threshold signatures in this setting, but this would have several downsides. First, it would require the wallet owners to run a setup protocol to arrive at their signing keys, which can be impractical (depending on the number of parties sharing the wallet). Second, a party sharing multiple wallets would need to store separate keys specific to each wallet. Threshold *ring* signatures have neither of these drawbacks — parties only need one signing key (regardless of how many wallets they share), and do not need to run any setup.

Whistleblowing Another possible application is 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 all n members of the 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 stronger definitions, and a more efficient construction, of threshold ring signatures.

Stronger Definitions In our definition of anonymity, 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 [YLA$^+$11, PBB12, OTYO18, HS20] 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.

Efficient Construction We build the first threshold ring signature scheme with signatures of size $\mathcal{O}(t)$; 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 however 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 traceable ring signature scheme (TrRS), introduced by Fujisaki and Suzuki [FS07].[1]. A traceable ring signature scheme is a ring signature scheme which allows the linking of two signatures produced by the same signer, if the same nonce was used during signing. 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 traceable ring signatures. (For compactness, the nonce is hashed, so its size need not affect the size of the signatures.) A verifier can check that no two traceable ring signatures were produced by the same signer, and so is convinced that t of n users signed the message.

Any traceable ring signature scheme (secure under our definitions) can be used to construct a threshold ring signature scheme in such a way. We present a new, intuitive traceable ring signature scheme construction with signatures of size $O(1)$. Unlike the work of Yuen *et al.*, we leverage a random oracle, allowing us to get smaller traceable signatures. We additionally use an RSA accumulator and the generalized DDH assumption. These assumptions are more standard than the Link-Decisional RSA assumption used in some other traceable ring signature constructions [TW05, ACST06].

[1]A similar approach to building a threshold ring signature scheme was mentioned by Yuen *et al.* [YLA$^+$13]; 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.

At a high level, our traceable ring signature scheme works as follows: a signer hashes the nonce (taken to be the 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. They then prove (in non-interactive zero knowledge) that they used a signing key corresponding to one of the public keys belonging to the super-set. 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 to represent the set of public keys. An accumulator is a compact representation of an arbitrarily large set that supports efficient proofs of membership.

Note that our construction is completely non-interactive; each of the t signers produces a traceable ring signature independently, and those signatures are then simply concatenated to produce the threshold ring signature. The concatenation can be done by any third party.

1.3 The Quest for Compact Signatures: Hardness of Lower Bounds

A very intriguing question is whether there is a fundamental lower bound for threshold ring signature size. Could we get signatures with size independent of t as well as n, or is that impossible?

Here, we give a somewhat dissatisfying answer: we sketch a construction with constant-size signatures that uses *obfuscation* — the transformation of a program in a way that hides its inner workings while preserving its behavior. (There are many flavors of obfuscation, ranging from virtual black box obfuscation which is known not to be possible for all programs, to indistinguishability obfuscation for which there exist candidate constructions.) Since obfuscation is a very strong assumption, and candidate constructions are very inefficient, such a threshold ring signature would not be useable in practice. However, the existence of such a construction implies that it would be hard to show a lower bound on signature size.

Our obfuscation-based construction uses a common reference string (CRS) which consists of a signature verification key, and an obfuscated program which holds the corresponding secret signing key. The program takes as input the public keys of a ring $\mathcal{S}$ of signers ($\{pk_i\}_{i\in\mathcal{S}}$), a message msg, and some number t of ring members' signatures (on $\{pk_i\}_{i\in\mathcal{S}}$ and msg). It checks the t signatures it receives, and if they all verify under different keys $pk \in \{pk_i\}_{i\in\mathcal{S}}$, it signs a statement of the form *"t members of the ring defined by public keys $\{pk_i\}_{i\in\mathcal{S}}$ signed the message msg"*.

The program's signature, which will have constant size, will constitute the threshold ring signature. Note that the statement itself need not count towards the threshold ring signature size, since it can be reconstructed from the other inputs to the threshold ring signature verification algorithm. It should be impossible to extract a signature from the program without actually having t ring members' signatures (unforgeability), and it is clearly impossible to deduce anything about the signers' identities from the program's signature (anonymity).

When using indistinguishability obfuscation to instantiate this, the program's signature can be a puncturable PRF output on the statement; the public verification key would then be a second obfuscated program. Using standard indistinguishability obfuscation techniques ([SW13]), this construction could be proven secure when leveraging constrained signatures ([BW13]) and similar primitives in the setting of static adversary corruptions, with program size dependent on some upper bound on the number of signing queries the adversary makes.

2 Preliminaries

In this section, we introduce some primitives that we leverage in our constructions. In Section 2.1, we describe cryptographic accumulators; in Section 2.2, we describe non-interactive zero knowledge arguments of knowledge.

2.1 Accumulators

At a high level, a cryptographic accumulator [Bd94] 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 2.1.3, we describe the RSA accumulator [Bd94], which requires only arithmetic operations and is thus more efficient to use within zero knowledge.

Baldimtsi *et al.* [BCY20] 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 accumulate algorithm.

2.1.1 Accumulator Syntax

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

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

$\mathsf{accumulate}(\texttt{params}, \mathcal{S}) \rightarrow a_\mathcal{S}$: An algorithm that, given the global parameters params 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.

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

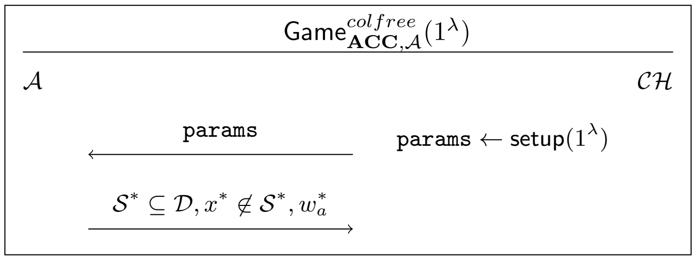
$\mathsf{verify}(\texttt{params}, x, a, w_a) \rightarrow \texttt{accept/reject}$: An algorithm that, given the global parameters params, an element x, an accumulator a and a witness w_a, checks whether w_a proves that x is in the underlying set a.

2.1.2 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 1 (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 challenger and a probabilistic polynomial-time adversary $\mathcal{A}$:*


$\mathsf{Game}^{colfree}_{\mathbf{ACC},\mathcal{A}}(1^\lambda)$
$\mathcal{A}$
$\mathcal{CH}$
$\mathtt{params}$
$\mathtt{params} \leftarrow \mathsf{setup}(1^\lambda)$
$\mathcal{S}^* \subseteq \mathcal{D}, x^* \notin \mathcal{S}^*, w_a^*$

Let $a \leftarrow \mathsf{accumulate}(\mathtt{params}, \mathcal{S}^*)$*. We say that $\mathcal{A}$ wins the game if* $\mathsf{verify}(\mathtt{params}, x^*, a, w_a^*) = \mathtt{accept}$*.*

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 $\nu(\lambda)$.*

2.1.3 The RSA Accumulator

The RSA accumulator, which was the original accumulator introduced by Benaloh and DeMare [Bd94], 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 in Construction 1.

Construction 1 (RSA Accumulator)**.**

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

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 $\mathtt{params} = (m, g)$.*

$\mathsf{accumulate}(\mathtt{params} = (m, g), \mathcal{S})$**:** *Return $a = g^{\prod_{x \in \mathcal{S}} x} \bmod m$.*

$\mathsf{witcreate}(\mathtt{params} = (m, g), \mathcal{S}, x)$**:** *Return $w_a = g^{\prod_{y \in \mathcal{S}, y \neq x} y} \bmod m$.*

$\mathsf{verify}(\mathtt{params} = (m, g), x, a, w_a)$**:** *If $w_a^x \bmod m = a$, return* $\mathtt{accept}$*. Otherwise, return* $\mathtt{reject}$*.*

2.2 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 [BFM88, FS87, FLS90]. 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. Traditionally, zero-knowledge proofs and arguments of knowledge are interactive. Fiat and Shamir [FS87] give an easy transformation from any public coin interactive protocol to a non-interactive one, where the challenge from the verifier is replaced with a hash of all previous messages.

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 [GM17]. We also describe the concrete relation which we will need in Section 4.2.

2.2.1 NIZKAoK Syntax

A NIZKAoK scheme has the following algorithms, as described by Groth and Maller [GM17]:

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

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

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

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

2.2.2 NIZKAoK Security Definitions

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

Definition 2 (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 **NIZKAoK** $= (\mathsf{setup}, \mathsf{prove}, \mathsf{verify}, \mathsf{simprove})$ *be a NIZKAoK scheme. Consider the following game between a challenger and a probabilistic polynomial-time adversary $\mathcal{A}$:*

$\mathsf{Game}^{zk}_{\mathbf{NIZKAoK},\mathcal{A}}(1^\lambda)$

$\mathcal{A}$		$\mathcal{CH}$
	$\xleftarrow{\texttt{crs}}$	$(\texttt{crs}, \texttt{td}) \leftarrow \mathsf{setup}(1^\lambda)$
		$b \leftarrow_R \{0,1\}$
	Query / Challenge phase	
$\mathcal{A}$ may issue polynomially many prove queries	$\xrightarrow{\phi, w}$	

		if $b = 0 : \pi \leftarrow \mathsf{prove}(\texttt{crs}, \phi, w)$
		 if $b = 1 : \pi \leftarrow \mathsf{simprove}(\texttt{crs}, \texttt{td}, \phi)$

	$\xleftarrow{\pi}$	
	$\xrightarrow{b'}$	

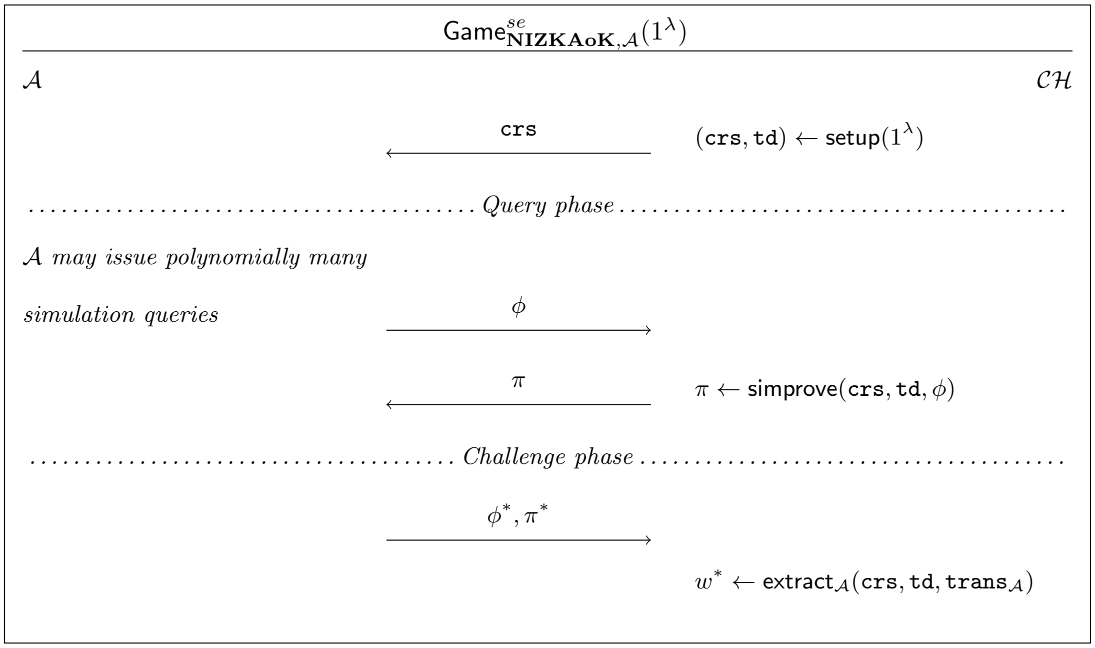
We say that $\mathcal{A}$ wins the game if $b = b'$.

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 3 (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 **NIZKAoK** $= (\mathsf{setup}, \mathsf{prove}, \mathsf{verify}, \mathsf{simprove})$ *be a NIZKAoK scheme. Consider the following game between a challenger and a probabilistic polynomial-time adversary $\mathcal{A}$, where $\texttt{trans}_{\mathcal{A}}$ denotes the adversary's inputs and outputs, including its randomness:*


$\mathsf{Game}^{se}_{\mathbf{NIZKAoK},\mathcal{A}}(1^\lambda)$
$\mathcal{A}$
$\mathcal{CH}$
crs
$(\texttt{crs}, \texttt{td}) \leftarrow \mathsf{setup}(1^\lambda)$
Query phase
$\mathcal{A}$ may issue polynomially many simulation queries
ϕ
π
$\pi \leftarrow \mathsf{simprove}(\texttt{crs}, \texttt{td}, \phi)$
Challenge phase
ϕ^*, π^*
$w^* \leftarrow \mathsf{extract}_{\mathcal{A}}(\texttt{crs}, \texttt{td}, \texttt{trans}_{\mathcal{A}})$

We say that $\mathcal{A}$ wins the game if $(\phi^*, w^*) \notin \mathcal{R}$, $\mathsf{verify}(\texttt{crs}, \phi^*, \pi^*) = \texttt{accept}$, and π was not returned by the challenger $\mathcal{CH}$ in answer to a simulation query.

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 $\nu(\lambda)$.*

2.2.3 NIZKAoK For Our Needs

Thanks to Goldreich [GMW91], we know that an interactive proof (specifically, a Σ-protocol) exists for any NP relation. Thanks to Fiat and Shamir [FS87], we know that any such proof can be turned non-interactive. However, these proofs are more compact and efficiently computable / verifiable for some relations than for others. Specifically, equality of exponents is known to be efficient. In Section 4.2, we will want to use the following relation $\mathcal{R}$:

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

Proofs for this relation are reasonably efficient to compute and verify, as they consist largely of simple statements about exponents. (Recall that verification in the RSA accumulator is also just a simple exponentiation.)

2.3 The Generalized Decisional Diffie-Hellman Problem

We leverage the Generalized Decisional Diffie-Hellman (Generalized DDH) Problem [BDZ03], described below.

Definition 4. *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 $\frac{1}{2}$.

3 (Threshold) Ring Signature Definitions

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

3.1 Ring Signature Definitions

Ring signatures were originally defined by Rivest *et al.* [RST01] 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 signatures instead allow signers to generate their own key pairs, and to form groups in an ad-hoc way. In Figure 1 we list some known ring signature constructions, their signature sizes, and the assumptions they leverage.

Work	**Signature Size**	**Building Blocks**	**Assumptions**
[RST01]	$O(n)$	Symmetric Encryption, Trapdoor Permutation	Ideal Cipher
[DKNS04]	$O(1)$	Sigma Protocols, Accumulators	Strong RSA, RO
[ACST06]	$O(1)$	Accumulators, SoKs	Link Decisional RSA
[BDR15]	$O(1)$	Any signature scheme, Groth-Sahai Commitments	Asymetric Pairing of Composite Order, q-SDH

Figure 1: Comparison of Select Ring Signature Schemes

3.1.1 Ring Signature Syntax

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

setup$(1^\lambda) \rightarrow$ `params`: An algorithm that takes a security parameter λ and outputs a set of public parameters `params`. These public parameters `params` include the security parameter itself, and any global parameters which can be used within the other algorithms.

keygen(`params`) $\rightarrow (pk, sk)$: An algorithm that takes the public parameters `params` and outputs a key pair (pk, sk).

$\mathsf{sign}(\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk_i) \to \sigma$: 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 $\{pk_j\}_{j\in\mathcal{S}}$, and the secret key sk_i of the signer $i \in \mathcal{S}$ (which must correspond to a public key within the set of public keys $\{pk_j\}_{j\in\mathcal{S}}$). Outputs a signature σ on the message msg.

$\mathsf{verify}(\mathtt{params}, msg, \{pk_i\}_{i\in\mathcal{S}}, \sigma) \to \mathtt{accept}/\mathtt{reject}$: 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.)

3.1.2 Ring Signature Security Definitions

Informally, a ring signature scheme must satisfy the following properties [BSS02, DKNS04, Liu19]:

– *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.

3.2 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 of signers $\mathcal{S}$. A verifier can then check that at least t signers in the ring $\mathcal{S}$ signed the message. Note that a ring signature scheme can be viewed as a threshold ring signature scheme with $t = 1$.

In Figure 2 we list some known threshold ring signature constructions, their signature sizes, and the assumptions they leverage. Prior constructions of threshold ring signatures have signatures whose size depends on the number n of users in the group $\mathcal{S}$. This is not ideal, as the threshold t may be much smaller than n. Our focus in this paper is constructing a threshold ring signature scheme with signature size independent of n, while also satisfying a stronger and more practical notion of anonymity.

Work	**Signatue Size**	**Building Blocks**	**Assumptions**
Our work	$\mathcal{O}(\mathbf{t})$	**TrRS scheme**	**Generalized DDH, RSA, RO**
Petzoldt *et al.* [PBB12]	$\mathcal{O}(n)$	multivariate polynomials	Quadratic MQ-problem
Chen *et al.* [CHGL18]	$\mathcal{O}(n)$	lattices	Ideal Lattice
Bresson *et al.* [BSS02]	$\mathcal{O}(n \log n)$	One-way functions	RSA
Zhou *et al.* [ZZY+17]	$\mathcal{O}(n)$	Coding based	Syndrome Decoding Problem
Okamoto *et al.* [OTYO18]	$\mathcal{O}(tn)$	Short term keys from a TTP	Discrete Log
Haque *et al.* [HS20]	$\mathcal{O}(n)$	Trapdoor Commitments / Polynomial interpolation	Trapdoor commitments
Liu *et al.* [YLA+13]	$\mathcal{O}(t\sqrt{n})$	One-time signature scheme	Discrete Log

Figure 2: Threshold Ring Signature Constructions

3.2.1 Threshold Ring Signature Syntax

A threshold ring signature scheme is defined as a tuple of five algorithms (setup, keygen, sign, $\mathsf{combisign}$, verify). The algorithms setup, keygen, sign and verify are syntactically the same as in a ring signature scheme, with the exceptions that (1) sign now outputs a *partial* signature σ_i for signer i, and (2) verify now additionally takes the threshold t as input. The algorithm $\mathsf{combisign}$, described below, combines partial signatures into one signature. It may be run by any third party, as it does not require any signers' secrets.

$\mathsf{combisign}(\mathtt{params}, \{\sigma_i\}_{i \in \mathcal{S}'}) \to \sigma$: An algorithm that takes partial signatures $\{\sigma_i\}_{i \in \mathcal{S}'}$ from t signers, and outputs a combined signature σ.

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

Setup Freeness: Every signer can generate their own key pair.

Non-interactive Signing: As per the syntax described above, any third party can combine the partial signatures produced by sign into a single signature, without use of the signers' secret keys. Thus, there is no single point of failure in the signing process.

Dynamic Choice of n and of Threshold t: Arbitrarily many signers' partial signatures can be combined into a single threshold signature; verification takes a threshold t, and checks that at least that many signers have signed. The downside of this flexibility is that the number of signers cannot be hidden by a signature σ.

3.2.2 Threshold Ring Signature Security Definitions

We base our security definitions on Bresson *et al.* [BSS02] and Haque *et al.* [HS20]. We make the simplifying assumption that the same ring of signers never signs the same message twice; that is, properties such as unforgeability and anonymity only need to hold if the signing oracle was never queried on the challenge message msg^* and challenge signer ring $\mathcal{S}^*$. In practice, this is reasonable since it is easily circumvented by always appending a unique nonce or timestamp to every message.

Definition 5 (TRS)**.** *A threshold ring signature scheme is* secure *if it satisfies correctness (Definition 6), unforgeability (Definition 7), and anonymity (Definition 8).*

Definition 6 (Correctness for TRS)**.** *Correctness requires that verification return* $\mathtt{accept}$ *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 **TRS** *is* correct *if for all security parameters* $\lambda \in \mathbb{N}$*, for all messages* $msg \in \{0,1\}^*$*, for all signer universes* $\mathcal{U}$*, all rings* $\mathcal{S} \subseteq \mathcal{U}$*, and all signer sets* $\mathcal{S}' \in \mathcal{S}$*:*

$$\Pr\left[\begin{array}{l} \texttt{params} \leftarrow \textbf{TRS}.\mathsf{setup}(1^\lambda), \\ \{(pk_i, sk_i) \leftarrow \textbf{TRS}.\mathsf{keygen}(\texttt{params})\}_{i\in\mathcal{S}}, \\ \{\sigma_i \leftarrow \textbf{TRS}.\mathsf{sign}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk_i)\}_{i\in\mathcal{S}'}, \\ \sigma \leftarrow \textbf{TRS}.\mathsf{combisign}(\texttt{params}, \{\sigma_i\}_{i\in\mathcal{S}'}) : \\ \textbf{TRS}.\mathsf{verify}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, \sigma, t = |\mathcal{S}'|) = \texttt{accept} \end{array}\right] = 1$$

Definition 7 (Unforgeability for TRS)**.** *Unforgeability requires that no efficient adversary* $\mathcal{A}$ *is able to forge a valid signature* σ *for some ring* $\mathcal{S}$ *for which* $\mathcal{A}$ *knows fewer than t secret keys. More formally, let* **TRS** $=$ (setup, keygen, sign, combisign, verify) *be a TRS scheme. Consider the game* $\mathsf{Game}^{Unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$ *between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

$\mathsf{Game}^{unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{CH}$

Setup phase

$\xrightarrow{\mathcal{U}}$ $\quad Q_{corrupt} = \emptyset, Q_{sign} = \emptyset$

$\texttt{params} \leftarrow \textbf{TRS}.\mathsf{setup}(1^\lambda)$

$\{(pk_i, sk_i) \leftarrow \textbf{TRS}.\mathsf{keygen}(\texttt{params})\}_{i\in\mathcal{U}}$

$\xleftarrow{\texttt{params}, \{pk_i\}_{i\in\mathcal{U}}}$

Query phase

$\mathcal{A}$ *may issue polynomially many*
corruption *or* signing *queries*

$\xrightarrow{Corrupt(i)/Sign(msg, \mathcal{S}, \mathcal{S}')}$

Corrupt :
  Look up $sk_i \in \{sk_j\}_{j\in\mathcal{U}}$*, add pk to* $Q_{corrupt}$
Sign : for $i \in \mathcal{S}'$*,*
    $\sigma_i \leftarrow \textbf{TRS}.\mathsf{sign}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk_i)$
  $\sigma \leftarrow \textbf{TRS}.\mathsf{combisign}(\texttt{params}, \{\sigma_j\}_{j\in\mathcal{S}'})$
  add $(msg, \mathcal{S}, \mathcal{S}')$ *to* Q_{sign}

$\xleftarrow{sk_i/\sigma, \{\sigma_j\}_{j\in\mathcal{S}'}}$

Challenge phase

$\xrightarrow{\sigma^*, msg^*, \mathcal{S}^* \subset \mathcal{U}, t}$

If all of the following checks pass, $\mathcal{A}$ *wins:*
  $\textbf{TRS}.\mathsf{verify}(\texttt{params}, msg^*, \{pk_i\}_{i\in\mathcal{S}^*}, \sigma^*, t) = \texttt{accept}$
  $(msg^*, \mathcal{S}^*, \cdot) \notin Q_{sign}$
  $|\mathcal{S}^* \cap Q_{corrupt}| < t$

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

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

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

Remark 2. *Note that the challenger responds to signing queries not only with the combined signature, but also with all of the 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.*

Definition 8 (Anonymity for TRS)**.** *Anonymity requires that no efficient adversary $\mathcal{A}$ be able to distinguish between signatures produced by two different subsets of 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* $\mathsf{Game}^{anon}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$ *between an adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$.*

$\mathsf{Game}^{anon}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $\mathcal{CH}$

........................*Setup phase: as in* $\mathsf{Game}^{unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$........................

........................*Query phase: as in* $\mathsf{Game}^{unforge}_{\mathbf{TRS},\mathcal{A}}(1^\lambda)$........................

................................*Challenge phase*................................

$\xrightarrow{msg^*, \mathcal{S}^* \subset \mathcal{U}, \mathcal{S'}^*_0, \mathcal{S'}^*_1}$

$b \leftarrow_R \{0,1\}$

for $i \in \mathcal{S'}^*_b$,

$\quad \sigma_i \leftarrow \mathbf{TRS}.\mathsf{sign}(\mathtt{params}, msg^*, \{pk_j\}_{j\in\mathcal{S}^*}, sk_i)$

$\xleftarrow{\sigma^*}$ $\qquad$ $\sigma^* \leftarrow \mathbf{TRS}.\mathsf{combisign}(\mathtt{params}, \{\sigma_j\}_{j\in\mathcal{S'}^*})$

$\xrightarrow{b'}$ $\qquad$ *If all of the following checks pass, $\mathcal{A}$ wins:*

$\quad b' = b,$

$\quad (msg^*, \mathcal{S}^*, \cdot) \notin Q_{sign},$

$\quad |\mathcal{S'}^*_0| = |\mathcal{S'}^*_1|, \mathcal{S'}^*_0 \subseteq \mathcal{S'}^*, \mathcal{S'}^*_1 \subseteq \mathcal{S'}^*,$

$\quad ((\mathcal{S'}^*_0 \cup \mathcal{S'}^*_1) \setminus (\mathcal{S'}^*_0 \cap \mathcal{S'}^*_1)) \cap Q_{corrupt} = \emptyset$

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

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

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

Remark 3. *Note that this is a stronger notion of anonymity than any that appear in prior literature, since anonymity must hold even if members of the ring are corrupt, as long as the corrupt members are present in either both or neither of the challenge signing sets* $\mathcal{S'}^*_0, \mathcal{S'}^*_1$*.*

4 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.)

In order to circumvent these two issues, we leverage *traceable ring signatures (TrRS)* [FS07], which are a flavor of *linkable ring signatures (LRS)*[LWW04]. 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.[2] If the underlying linkable ring signatures have size $\mathcal{O}(1)$, then the threshold ring signatures will have size $\mathcal{O}(t)$.

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 8).

To address this problem, we use *traceable ring signatures* instead. Traceable ring signatures are nonce-based: the signing algorithm additionally takes a nonce, and two signatures by the same signer are only linkable if the same nonce is used to produce them. By using the message and set of public keys belonging to the signing ring as the nonce, we ensure that the linkable 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 4.1, we state the definition of a *traceable ring signature scheme* (TrRS).
2. In Section 4.2, we construct a TrRS scheme with signatures of size $\mathcal{O}(1)$. In our construction, instead of allowing the nonce to take any value, we mandate that the nonce always contains the message. (In particular, this implies that two signatures on different messages are never linkable.) We make this restriction for the sake of concrete efficiency; without this restriction, our construction of traceable ring signatures would require an additional element.
3. In Section 4.3, we use our TrRS scheme to construct a TRS scheme with signatures of size $\mathcal{O}(t)$.

4.1 Traceable Ring Signature Definitions

We leverage the notion of *traceable ring signature (TrRS)* schemes, introduced by Fujisaki and Suzuki [FS07]. We alter the definition of Fujisaki and Suzuki in several ways:

1. Fujisaki and Suzuki require that if the same signer signed different messages with the same nonce twice, their identity be revealed. We do not reveal the signer's identity in such a case, as this is a stronger notion of traceability than we require.

[2] A similar idea was mentioned by Yuen *et al.* [YLA+13]; however, it was not formalized or proven. In particular, a stronger linkability property — *one-more linkability*, which we introduce in Definition 10 — is needed from the underlying linkable 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 $\mathcal{O}(t)$ signatures, while they obtain a TRS construction with size $\mathcal{O}(t\sqrt{n})$ signatures.)

2. Fujisaki and Suzuki require that the public keys belonging to all ring members always be an implicit part of the nonce. While we can certainly accomodate this, we do not make this restriction.

3. Fujisaki and Suzuki's definition of linkability assumes that the adversary has access to all n secret keys belonging to ring members; then, the adversary should be unable to produce $n+1$ unlinked signatures with the same nonce. However, we need a stronger definition of linkability. We allow the adversary access to any t secret keys belonging to a subset of ring members; then, the adversary should be unable to produce $t+1$ unlinked signatures with the same nonce.

4.1.1 Traceable Ring Signature Syntax

We define a traceable ring signature scheme as a tuple of five algorithms (setup, keygen, sign, verify, link). The setup and keygen algorithms have the same input and output behavior as the corresponding ring signature algorithms. We modify the sign and verify algorithms to take a nonce $\mathtt{nonce} \in \{0,1\}^*$ as an additional argument (described below). The link algorithm (also described below) allows any verifier to determine whether two signatures were produced by the same signer (using the same nonce).

$\mathsf{sign}(\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk, \mathtt{nonce}) \to \sigma$**:** An algorithm that takes the same arguments as RS.sign, with the addition of a nonce $\mathtt{nonce}$. It has the same output as RS.sign: a signature σ.

$\mathsf{verify}(\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, \sigma, \mathtt{nonce})$**:** An algorithm that takes the same arguments as RS.verify with the addition of a nonce $\mathtt{nonce}$. Outputs either $\mathtt{accept}$ or $\mathtt{reject}$.

$\mathsf{link}(\mathtt{params}, (msg_0, \sigma_0, \{pk_j\}_{j\in\mathcal{S}_0}), (msg_1, \sigma_1, \{pk_j\}_{j\in\mathcal{S}_1}), \mathtt{nonce}) \to \{\mathtt{linked}, \mathtt{unlinked}\}$**:** An algorithm that takes two messages (msg_0, msg_1), signatures (σ_0, σ_1) and public keys belonging to members of rings $(\mathcal{S}_0, \mathcal{S}_1)$, as well as a nonce $\mathtt{nonce}$. Outputs $\mathtt{linked}$ or $\mathtt{unlinked}$, depending on whether the two signatures were produced by the same signer.

The nonces allow a single signer to create two unlinkabe signatures (which is explicitly disallowed in an LRS scheme); for a TrRS, this is a desirable property.

4.1.2 Traceable Ring Signature Security Definitions

Informally, a traceable ring signature scheme must satisfy the following properties:

– *Correctness* requires that a correctly generated signature must verify. (This is inherited from ring signatures, with appropriate syntactic modifications.)

– *Linkability Correctness* requires that signatures correctly generated by the same signer with the same nonce appear linked. (This is inherited from linkable ring signatures, with appropriate syntactic modifications. This is a very natural property which we do not define formally in the interest of space.)

– *Unforgeability* requires that an adversary should not be able to forge a signature on behalf of another user. (This is inherited from ring signatures, with appropriate syntactic modifications. In particular, we allow the adversary to have asked signing queries for the challenge signer and challenge message with a nonce other than the challenge nonce.)

- *Linkability* requires that no corrupt signer can produce two signatures that verify under the same nonce and appear unlinked. (This is inherited from ring signatures, with appropriate syntactic modifications.)
- *One-More Linkability* requires that no $t-1$ corrupt signers can produce t signatures that verify under the same nonce and appear unlinked. (We present this property as Definition 10. With $t=2$, it implies linkability; with $t=1$, it implies unforgeability.)
- *Cross-Nonce Unlinkability* requires that no adversary can determine whether two signatures that verify under different nonces were produced by the same signer. (We present this property as Definition 11.)
- *Unframeability* 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 or prove that our construction meets it.

Definition 9 (TrRS)**.** *A traceable ring signature scheme is* secure *if it satisfies correctness, linkability correctness, one-more linkability (Definition 10, which implies unforgeability), and cross-nonce unlinkability (Definition 11).*

In order to use linkability to prove the security of our TRS scheme, we need a somewhat stronger linkability property than what appears in the literature. Linkability for a linkable ring signature scheme typically requires that an adversary who corrupted one signer in a ring cannot produce two signatures which appear unlinked; linkability for a traceable ring signature scheme requires that an adversary who corrupted all signers in a ring cannot produce $n+1$ signatures with the same nonce which appear unlinked. We generalize these properties to require that an adversary who corrupts some $t-1$ signers cannot produce t signatures with the same nonce which verify and appear unlinked. We call this property *one-more linkability.* For $t=2$, this implies linkability for a linkable ring signature scheme; for $t=n$, this implies linkability for a traceable ring signature scheme; for $t=1$, this implies unforgeability.

Definition 10 (One-More Linkability for TrRS)**.** *let* $\mathbf{TrRS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{verify}, \mathsf{link})$ *be a TrRS scheme. Consider the game* $\mathsf{Game}^{omlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$ *between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{CH}$*.*

$\mathsf{Game}^{omlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ $\qquad\qquad$ $\mathcal{CH}$

$\xrightarrow{\mathcal{U}}$ $\quad Q_{corrupt} = \emptyset, Q_{sign} = \emptyset$

$\quad \texttt{params} \leftarrow \mathbf{TrRS}.\mathsf{setup}(1^\lambda)$
$\quad \{(pk_i, sk_i) \leftarrow \mathbf{TrRS}.\mathsf{keygen}(\texttt{params})\}_{i\in\mathcal{U}}$

$\xleftarrow{\texttt{params}, \{pk_i\}_{i\in\mathcal{U}}}$

. . . . *Query phase* . . . .

$\mathcal{A}$ may issue polynomially many
corruption *or* signing *queries*

$\xrightarrow{Corrupt(i)/Sign(msg, \mathcal{S}, i, \texttt{nonce})}$

Corrupt :
$\quad$ *Look up $sk_i \in \{sk_j\}_{j\in\mathcal{U}}$, add pk to $Q_{corrupt}$*
Sign :
$\quad \sigma \leftarrow \mathbf{TrRS}.\mathsf{sign}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk_i, \texttt{nonce})$
$\quad$ *add $(msg, \mathcal{S}, i, \texttt{nonce})$ to Q_{sign}*

$\xleftarrow{sk_i/\sigma}$

. . . . *Challenge phase* . . . .

$\xrightarrow{\{(msg_k, \sigma_k, \mathcal{S}_k)\}_{k\in[t]}, \texttt{nonce}}$

If all of the following checks pass, $\mathcal{A}$ wins:
$\quad |(\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_t) \cap Q_{corrupt}| \leq t - 1$
$\quad$ *For $k \in [t]$:*
$\quad\quad \mathbf{TrRS}.\mathsf{verify}(\texttt{params}, msg_k, \{pk_j\}_{j\in\mathcal{S}_k}, \sigma_k, \texttt{nonce}) = \texttt{accept}$
$\quad\quad (msg_k, \mathcal{S}_k, \cdot, \texttt{nonce}) \notin Q_{sign}$
$\quad\quad$ *For $l \in [t]$, $l \neq k$:*
$\quad\quad\quad \mathbf{TrRS}.\mathsf{link}(\texttt{params}, (msg_k, \sigma_k, \{pk_j\}_{j\in\mathcal{S}_k}),$
$\quad\quad\quad\quad (msg_l, \sigma_l, \{pk_j\}_{j\in\mathcal{S}_l}), \texttt{nonce}) = \texttt{unlinked}$

We say that **TrRS** *is* one-more linkable *if for any efficient adversary $\mathcal{A}$,*

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

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

Definition 11 (Cross-Nonce Unlinkability for TrRS)**.** *Given two signatures using different nonces, it should be infeasible for an adversary to determine whether they were created by the same signer or not. More formally, let $\mathbf{TrRS} = (\mathsf{setup}, \mathsf{keygen}, \mathsf{sign}, \mathsf{verify}, \mathsf{link})$ be a TrRS scheme. Consider the game $\mathsf{Game}^{cnunlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{CH}$.*

$\mathsf{Game}^{cnunlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$

$\mathcal{A}$ $\mathcal{CH}$

.... *Setup phase: as in* $\mathsf{Game}^{omlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$ ....

.... *Query phase: as in* $\mathsf{Game}^{omlink}_{\mathbf{TrRS},\mathcal{A}}(1^\lambda)$ ....

.... *Challenge phase* ....

$\xrightarrow{msg_0, msg_1, \mathcal{S}_0, \mathcal{S}_1, i_0 \in \mathcal{S}_0 \cap \mathcal{S}_1, i_1 \in \mathcal{S}_1, \texttt{nonce}_0, \texttt{nonce}_1}$

$\sigma_0^* \leftarrow \mathbf{TrRS}.\mathsf{sign}(\texttt{params}, msg_0, \{pk_j\}_{j\in\mathcal{S}_0}, sk_{i_0}, \texttt{nonce}_0)$
$b \leftarrow_R \{0,1\}$
$\sigma_1^* \leftarrow \mathbf{TrRS}.\mathsf{sign}(\texttt{params}, msg_1, \{pk_j\}_{j\in\mathcal{S}_1}, sk_{i_b}, \texttt{nonce}_1)$

$\xleftarrow{\sigma_0^*, \sigma_1^*}$

$\xrightarrow{b'}$ *If all of the following checks pass,* $\mathcal{A}$ *wins:*

$b' = b, \texttt{nonce}_0 \neq \texttt{nonce}_1$
$i_0 \notin Q_{corrupt}, i_1 \notin Q_{corrupt}$
$(\cdot,\cdot,i_0,\texttt{nonce}_0) \notin Q_{sign}, (\cdot,\cdot,i_0,\texttt{nonce}_1) \notin Q_{sign}$
$(\cdot,\cdot,i_1,\texttt{nonce}_0) \notin Q_{sign}, (\cdot,\cdot,i_1,\texttt{nonce}_1) \notin Q_{sign}$

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

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

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

4.2 A Traceable Ring Signature Scheme

We describe a traceable ring signature scheme in Construction 2 in terms of an underlying accumulator scheme **RSAACC**, a non-interactive zero-knowledge argument of knowledge scheme **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 $\mathbb{G}$.

NIZKAoK will be used for the relation $\mathcal{R}$, which is described below.

$$\mathcal{R}\left(\begin{array}{l} \phi = (\texttt{params} = \\ \quad (\mathbf{RSAACC}.\texttt{params}, p, g \in \mathbb{G}), \\ \quad a_{\mathcal{S}}, h, \sigma'), \\ w = (pk, sk, w_a) \end{array}\right) = \left(\begin{array}{l} (pk = g^{sk}) \\ \wedge \mathbf{RSAACC}.\mathsf{verify}(\\ \qquad \mathbf{RSAACC}.\texttt{params}, a_{\mathcal{S}}, pk, w_a) \\ \wedge(\sigma' = h^{sk}) \end{array}\right)$$

Construction 2.

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

- *Run* $\mathbf{RSAACC}.\texttt{params} \leftarrow \mathbf{RSAACC}.\mathsf{setup}(1^\lambda)$.
- *Run* $(\mathbf{NIZKAoK}.\texttt{crs}, \mathbf{NIZKAoK}.\texttt{td}) \leftarrow \mathbf{NIZKAoK}.\mathsf{setup}(1^\lambda, \mathcal{R})$.

- *Set* $\texttt{params} = (\mathbf{RSAACC}.\texttt{params}, \mathbf{NIZKAoK}.\texttt{crs})$.

$\mathsf{keygen}(\texttt{params})$:

- *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.*

$\mathsf{sign}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk, \texttt{nonce})$:

- *Check that for each* $pk \in \mathcal{S}$, pk *is prime.*
- *Accumulate* $\{pk_j\}_{j\in\mathcal{S}}$ *as* $a_{\mathcal{S}} \leftarrow \mathbf{RSAACC}.\mathsf{accumulate}(\mathbf{RSAACC}.\texttt{params}, \{pk_j\}_{j\in\mathcal{S}})$. *(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* $pk \in \{pk_j\}_{j\in\mathcal{S}}$ *be the public key corresponding to the secret key* sk. *Compute an accumulator witness* $w_a \leftarrow \mathbf{RSAACC}.\mathsf{witcreate}(\mathbf{RSAACC}.\texttt{params}, \{pk_j\}_{j\in\mathcal{S}}, pk)$.
- *Compute* $\sigma' = \mathsf{H}(msg, \texttt{nonce})^{sk}$.
- *Compute* π *proving that* $\mathsf{H}(msg, \texttt{nonce})$ *was raised to the power of a secret key corresponding to a public key in the accumulator. In other words,*

  $$\pi \leftarrow \mathbf{NIZKAoK}.\mathsf{prove}(\mathbf{NIZKAoK}.\texttt{crs}, \phi = (\texttt{params}, a_{\mathcal{S}}, \mathsf{H}(msg, \texttt{nonce}), \sigma'), w = (pk, sk, w_a)).$$

- *Return* $\sigma = (\sigma', \pi)$.

$\mathsf{verify}(\texttt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, \sigma = (\sigma', \pi))$:

- *Check that for each* $pk \in \mathcal{S}$, pk *is prime.*
- *Accumulate* $\{pk_j\}_{j\in\mathcal{S}}$ *as* $a_{\mathcal{S}} \leftarrow \mathbf{RSAACC}.\mathsf{accumulate}(\mathbf{RSAACC}.\texttt{params}, \{pk_j\}_{j\in\mathcal{S}})$.
- *Verify the proof* π*; return*

  $$\mathbf{NIZKAoK}.\mathsf{verify}(\mathbf{NIZKAoK}.\texttt{crs}, \phi = (\texttt{params}, a_{\mathcal{S}}, \mathsf{H}(msg, \texttt{nonce}), \sigma'), \pi)$$

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

4.2.1 Proof of Security

Theorem 1. *If* **NIZKAoK** *is a secure non-interactive zero knowledge argument of knowledge, if* **RSAACC** *is a secure accumulator, if* H *is a random oracle, and if the generalized DDH problem is hard in* $\mathbb{G}$*, then Construction 2 is a secure traceable ring signature scheme (Definition 9) as long as each nonce always includes the message.*

We prove Theorem 1 in several steps. First, correctness and linkability correctness are apparent on inspection. Second, in Lemma 1 we address one-more linkability (Definition 10). Last, in Lemma 2 we address cross-nonce unlinkability (Definition 11).

Lemma 1. *If* **NIZKAoK** *is a secure non-interactive zero knowledge argument of knowledge, if* **RSAACC** *is a secure accumulator, if* H *is a random oracle, and if the generalized DDH problem is hard in* $\mathbb{G}$*, then Construction 2 is one-more-linkable.*

Proof. We will construct an algorithm $\mathcal{B}$ which will use an $\mathcal{A}$ who can break the one-more-linkability of the TrRS scheme in Construction 2 to break the discrete logarithm problem with non-negligible probability if **NIZKAoK** and **RSAACC** 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 3).

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**, **RSAACC**, 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 non-negligible 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 one-more-linkability game described in Definition 10.

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 2). 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 2, 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 and nonces $\mathtt{nonce}$ it is asked signing queries on, or which it is given forgeries for. If it sees $(msg_0, \mathtt{nonce}_0) \neq (msg_1, \mathtt{nonce}_1)$ such such that $\mathsf{H}(msg_0, \mathtt{nonce}_0) = \mathsf{H}(msg_1, \mathtt{nonce}_1)$, 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 $\mathcal{S}$ it is asked signing queries on behalf of, or which it is given forgeries on behalf of. If it ever sees two signer sets $\mathcal{S} \neq \mathcal{S}'$ such that $a_{\mathcal{S}} = a_{\mathcal{S}'}$ (where $a_{\mathcal{S}} =$ **RSAACC**.$\mathsf{accumulate}($**RSAACC**.$\mathtt{params}$, $\{pk_i\}_{i \in \mathcal{S}})$), 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 **RSAACC** (Definition 1).

Game $\mathcal{G}_4$: This is the same as the previous game, but when the adversary returns its unlinked signatures $(\{(msg_k, \sigma_k = (\sigma'_k, \pi_k), \mathcal{S}_k)\}_{k\in[t]}, \texttt{nonce})$, $\mathcal{B}$ extracts the witness $w_k = (pk_{i_k}, sk_{i_k}, w_{a,k}) \leftarrow \mathsf{extract}_{\mathcal{A}}(\mathbf{NIZKAoK}.\texttt{crs}, \mathbf{NIZKAoK}.\texttt{td}, \texttt{trans}_{\mathcal{A}})$.

If it holds that $(\phi_k = (\texttt{params}, a_{\mathcal{S}_k}, \mathsf{H}(msg_k, \texttt{nonce}), \sigma'_k), w_k) \notin \mathcal{R}$ and $\mathbf{NIZKAoK}.\mathsf{verify}(\mathbf{NIZKAoK}.\texttt{crs}, \phi_k, \pi_k) = \texttt{accept}$, $\mathcal{B}$ aborts.

Note that for $\mathcal{A}$ to have succeeded in breaking one-more linkability, it must be that $\mathcal{A}$ never asked $\mathcal{B}$ a signing query on msg_k on behalf of $\mathcal{S}_k$ with $\texttt{nonce}$. Since in the previous two games $\mathcal{B}$ aborted if it ever saw two message, nonce pairs hash to the same value or two signer sets accumulate to the same value, it must be that the statement ϕ 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 3). (Just imagine that $\mathcal{B}$ interacts with a simulation extractability challenger to obtain $\mathbf{NIZKAoK}.\texttt{crs}$ and the proofs π.)

If $\mathcal{A}$ succeeds in winning the one-more linkability game and if $\mathcal{B}$ does not abort at this point, $\mathcal{B}$ has successfully extracted witnesses $\{w_k = (pk_{i_k}, sk_{i_k}, w_{a,k})\}_{k\in[t]}$ from the unlinked signatures $(\{(msg_k, \sigma_k = (\sigma'_k, \pi_k), \mathcal{S}_k)\}_{k\in[t]}, \texttt{nonce})$ such that $(pk_{i_k} = g^{sk_{i_k}}) \wedge \mathbf{RSAACC}.\mathsf{verify}(\mathbf{RSAACC}.\texttt{params}, a_{\mathcal{S}_k}, pk_{i_k}, w_{a,k}) \wedge (\sigma'_k = \mathsf{H}(msg_k, \texttt{nonce})^{sk_{i_k}})$. For $\mathcal{A}$ to have won, it must also be true that $\mathcal{B}$ never signed msg_k on behalf of $\mathcal{S}_k$ with nonce $\texttt{nonce}$ in response to a signing query.

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 **RSAACC** (Definition 1). Recall that $\mathcal{B}$ computes $a_{\mathcal{S}_k}$ as $a_{\mathcal{S}_k} \leftarrow \mathbf{RSAACC}.\mathsf{accumulate}(\mathbf{RSAACC}.\texttt{params}, \{pk_i\}_{i\in\mathcal{S}_k})$. $\mathcal{B}$ aborts if $\mathbf{RSAACC}.\mathsf{verify}(\mathbf{RSAACC}.\texttt{params}, pk, a_{\mathcal{S}_k}, w_{a,k}) = \texttt{accept}$, and $pk_{i_k} \notin \{pk_i\}_{i\in\mathcal{S}_k}$.

$\mathcal{B}$ only aborts with negligible probability, since if $\mathcal{A}$ finds $pk_{i_k}, w_{a,k}, \mathcal{S}_k$ that make $\mathcal{B}$ abort, $\mathcal{A}$ can trivially be used to break the collision freeness property of **RSAACC**. (Just imagine that $\mathcal{B}$ interacts with a collision freeness challenger to obtain $\mathbf{RSAACC}.\texttt{params}$.)

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

Game $\mathcal{G}_6$: If $\mathcal{A}$ successfully breaks the one-more-linkability property, we know that the following conditions hold:

1. None of the t signatures are on message — signer set combinations for which $\mathcal{B}$ answered signing queries for nonce $\texttt{nonce}$.
2. All t signatures verify (so $\mathcal{B}$ is able to extract the witnesses $w_1 = (pk_{i_1}, sk_{i_1}, w_{a,1}), \ldots, w_t = (pk_{i_t}, sk_{i_t}, w_{a,t})$).
3. At most $t-1$ signers in the union of the signing sets are corrupt.
4. Each pair of signatures appears unlinked.

Since in our scheme we assume the message is always part of the nonce, we also have $msg_1 = \cdots = msg_t = msg$.

If each pair of signatures appears unlinked, then we know that the signatures $\sigma'_1 = \mathsf{H}(msg, \texttt{nonce})^{sk_{i_1}}, \ldots, \sigma'_t = \mathsf{H}(msg, \texttt{nonce})^{sk_{i_t}}$ are all distinct, so $sk_{i_1}, \ldots, sk_{i_t}$ must all be distinct. Since at most $t-1$ signers in the union of the signing sets are corrupt, at least one of those secret keys was never given to $\mathcal{A}$ by $\mathcal{B}$. Let i^* be the identity of the signer whose secret key was never given to $\mathcal{A}$, but whose signature was one of the t unlinked signatures produced by $\mathcal{A}$ (and whose secret key sk_{i^*} was thus 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 $\sigma' = \mathsf{H}(msg, \texttt{nonce})^{sk_{i^*}}$ for signing queries on msg and $\texttt{nonce}$ 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 and nonce $\texttt{nonce}$ 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 $(u = \mathsf{H}(msg, \texttt{nonce}), v = \sigma')$ pairs. $\mathcal{B}$ will store the (u, v) tuples, and set $\mathsf{H}(msg, \texttt{nonce}) = u, \sigma = v$ 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, 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. *If* **NIZKAoK** *is a secure non-interactive zero knowledge argument of knowledge, if* **RSAACC** *is a secure accumulator, if* H *is a random oracle, and if the generalized DDH problem is hard in* $\mathbb{G}$*, then Construction 2 is cross-nonce unlinkable.*

Proof. **Game $\mathcal{G}_0$:** $\mathcal{B}$ honestly executes the role of the challenger in the strong anonymity game described in Definition 11.

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**.`td` to simulate the proofs using the **NIZKAoK**.simprove algorithm.

This game is indistinguishable from $\mathcal{G}_0$ by the zero knowledge property of **NIZKAoK** (Definition 2) (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 and nonce $\texttt{nonce}_0$ ("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 $\mathsf{H}(msg_0, \texttt{nonce}_0)$ 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 $\sigma' = \mathsf{H}(msg, \texttt{nonce})^{sk_{i^*}}$ for signing queries (/ challenges) on msg and $\texttt{nonce}$ 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 and nonce $\texttt{nonce}$ 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 and nonce $\texttt{nonce}_1$ ("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 ciphertext is independent of b, so the adversary cannot win with probability greater than $\frac{1}{2}$.

□

4.3 A Threshold Ring Signature Scheme

We build threshold ring signatures out of traceable ring signatures in a generic way. If the underlying traceable ring signatures have size $\mathcal{O}(1)$, then the resulting threshold ring signatures have size $\mathcal{O}(t)$, where t is the threshold. We require the additional assumption that no message msg is ever signed twice by the same ring $\mathcal{S}$. This is because we set our nonces to be the message together with the public keys of the ring members; if the same message is signed twice by the same ring, then the same nonce will be used across those two signatures, and in this case we cannot guarantee anonymity.[3]

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

Construction 3.

[3] One could think to use public keys belonging to the signing subset $\mathcal{S}'$ to generate the nonce, instead of the keys belonging to the ring $\mathcal{S}$; however, this has two downsides. First, the signers must be aware of who their fellow signers are. Second, the nonce must now be hidden from the adversary, as knowledge of the nonce would allow the adversary to de-anonymize the signing subset (by repeatedly deriving nonces from the message any any possible signing subset, and seeing which output matches the nonce it knows). Hiding the nonce from the adversary complicates zero knowledge proofs necessary in the underlying TrRS construction.

$\mathsf{setup}(1^\lambda)$**:** *Return* $\mathbf{TrRS}.\mathtt{params} \leftarrow \mathbf{TrRS}.\mathsf{setup}(1^\lambda)$.

$\mathsf{keygen}(\mathtt{params})$**:** *Return* $(sk, pk) \leftarrow \mathbf{TrRS}.\mathsf{keygen}(\mathtt{params})$.

$\mathsf{sign}(\mathtt{params}, msg, sk_i, \{pk_j\}_{j\in\mathcal{S}})$**:**

- *Set* $\mathtt{nonce} = (msg, \{pk_j\}_{j\in\mathcal{S}})$.
- *Return* $\sigma_i \leftarrow \mathbf{TrRS}.\mathsf{sign}(\mathbf{TrRS}.\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, sk_i, \mathtt{nonce})$.

$\mathsf{combisign}(\mathtt{params}, \{\sigma_i\}_{i\in\mathcal{S}'})$**:** *Return* $\sigma = \{\sigma_i\}_{i\in\mathcal{S}'}$. *(So simple!)*

$\mathsf{verify}(\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, \sigma = \{\sigma_i\}_{i\in\mathcal{S}'}, t)$**:**

- *If* $|\sigma| < t$, *return* $\mathtt{reject}$.
- *Set* $\mathtt{nonce} = (msg, \{pk_j\}_{j\in\mathcal{S}})$.
- *For* $\sigma_i \in \sigma$, *if* $\mathbf{TrRS}.\mathsf{verify}(\mathbf{TrRS}.\mathtt{params}, msg, \{pk_j\}_{j\in\mathcal{S}}, \sigma_i, \mathtt{nonce}) = \mathtt{reject}$, *return* $\mathtt{reject}$.
- *For all pairs of different signatures* σ_i, σ_j *in* σ, *if* $\mathbf{TrRS}.\mathsf{link}(\mathbf{TrRS}.\mathtt{params}, (msg, \sigma_i), (msg, \sigma_j), \{pk_j\}_{j\in\mathcal{S}}, \mathtt{nonce}) = \mathtt{linked}$, *return* $\mathtt{reject}$.
- *Return* $\mathtt{accept}$.

4.3.1 Proof of Security

Theorem 2. *If TrRS is a secure traceable ring signature scheme (Definition 9), then Construction 3 is a secure threshold ring signature scheme (Definition 5).*

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 TrRS satisfies* cross-nonce unlinkability *(Definition 11) then Construction 3 satisfies* anonymity *(Definition 8).*

Proof. We will construct an algorithm $\mathcal{B}$ which will break the *cross-nonce unlinkability* of the underlying TrRS scheme against a TrRS challenger $\mathcal{CH}$, by assuming we have an attacker $\mathcal{A}$ who can break the *anonymity* of the TRS scheme in Construction 3. Let $\mathcal{G}_0$ represent the TRS anonymity game where the challenger always uses $\mathcal{S}'_0$, and $\mathcal{G}_d$ represent the TRS anonymity game where the challenger always use $\mathcal{S}'_1$ (we will define d later). We introduce a number of games $(\mathcal{G}_1, \ldots, \mathcal{G}_{d-1})$ in between, such that $\mathcal{G}_j$ is designed to be indistinguishable from $\mathcal{G}_{j-1}$ for $1 \le j \le d$. If $\mathcal{A}$ can can break the *anonymity* of the TRS scheme, it can tell the difference between $\mathcal{G}_0$ and $\mathcal{G}_d$; thus, for some j, it can tell the difference between $\mathcal{G}_j$ and $\mathcal{G}_{j-1}$. If it does, we leverage $\mathcal{A}$ to break the cross-nonce unlinkability of TrRS.

Below, we use indexing notation $L[j]$ to denote an element within some list L. $L[1]$ refers to the first element of L, and $L[|L|]$ (where $|\cdot|$ is the cardinality function) refers to the last element of L. $L[j,\ldots,k]$ denotes a range of the list L; $L[j,\ldots,k] = L[j],\ldots,L[k]$. If $j = k$, we get $L[j,\ldots,j] = L[j]$; if $k > j$, we get an empty list.

Game $\mathcal{G}_j$, $j \in [0,\ldots,d]$:

Setup $\mathcal{B}$ recieves 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 TrRS challenger $\mathcal{CH}$, receiving the public parameters $\texttt{params}$ as well as public keys for each user $i \in \mathcal{U}$. It forwards this information to the TRS adversary $\mathcal{A}$.

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

Corruption of 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** $\mathcal{S}' \subseteq \mathcal{S}$**:** $\mathcal{B}$ sets $\texttt{nonce} = (msg, \{pk_i\}_{i\in\mathcal{S}})$, and issues $t = |\mathcal{S}'|$ signing queries to $\mathcal{CH}$, one for each $i \in \mathcal{S}'$ (with the same msg and $\mathcal{S}$, and with nonce $\texttt{nonce}$). $\mathcal{CH}$ returns $\{\sigma_i\}_{i\in\mathcal{S}'}$. $\mathcal{B}$ then runs $\sigma \leftarrow \textbf{TRS}.\mathsf{combisign}(\texttt{params}, \{\sigma_i\}_{i\in\mathcal{S}'})$, and sends σ to $\mathcal{A}$.

Challenge Once $\mathcal{A}$ is done issuing queries, $\mathcal{A}$ sends $\mathcal{B}$ a challenge message msg^*, a ring $\mathcal{S}^*$, and two signing sets — $\mathcal{S}'_0$ and $\mathcal{S}'_1$, such that $|\mathcal{S}'_0| = |\mathcal{S}'_1| = t$. $\mathcal{B}$ sets $\texttt{nonce} = (msg^*, \{pk_i\}_{i\in\mathcal{S}^*})$. Let L_0 be a list of signer indices present in $\mathcal{S}'_0$ but not in $\mathcal{S}'_1$ ($L_0 = \mathcal{S}'_0 \backslash \mathcal{S}'_1$), and L_1 be a list of signer indices present in $\mathcal{S}'_1$ but not in $\mathcal{S}'_0$ ($L_1 = \mathcal{S}'_1 \backslash \mathcal{S}'_0$). Note that because $|\mathcal{S}'_0| = |\mathcal{S}'_1|$, we have $|L_0| = |L_1|$. We define $d = |L_0| = |L_1|$. Let $L = [\mathcal{S}'_0 \cap \mathcal{S}'_1] + L_0[1,\dots,j] + L_1[j+1,\dots,d]$ such that L contains the users i which are both in $\mathcal{S}'_0$ and $\mathcal{S}'_1$ and then *some* users from $\mathcal{S}'_0$ and others from $\mathcal{S}'_1$. This corresponds to L slowly evolving from containing only users in $\mathcal{S}'_0$ to containing only users in $\mathcal{S}'_1$. $\mathcal{B}$ issues signing queries to $\mathcal{CH}$ for $i \in L$ (all with msg^*, $\mathcal{S}^*$, $\texttt{nonce}$). $\mathcal{CH}$ returns $\{\sigma_i^*\}_{i\in L}$. $\mathcal{B}$ then runs $\sigma^* \leftarrow \textbf{TRS}.\mathsf{combisign}(\texttt{params}, \{\sigma_i^*\}_{i\in\mathcal{S}'})$, and sends σ^* to $\mathcal{A}$. $\mathcal{A}$ returns a bit b.

Note that the probability that $\mathcal{A}$ returns $b = 1$ in any $\mathcal{G}_j, j \in [1,\dots,d]$ can only be negligibly different than the probability that he returns $b = 1$ in $\mathcal{G}_{j-1}$. Otherwise, $\mathcal{B}$ can use $\mathcal{A}$ to break TrRS cross-nonce unlinkability. $\mathcal{B}$ would send the $\mathcal{CH}$ a challenge request on $i_0 = L_0[j+1]$ with $\texttt{nonce}_0 = \perp$ (or some arbitrary different value), and $i_1 = L_1[j+1]$ with $\texttt{nonce}_1 = \texttt{nonce}$. $\mathcal{B}$ uses the values σ_1^* returned by $\mathcal{CH}$ in $\mathsf{combisign}$. If $\mathcal{CH}$ picks $b = 0$ (and uses $L_0[j+1]$ in the computation of σ_1^*, $\mathcal{A}$'s view is identical to that of $\mathcal{G}_{j+1}$), and if $\mathcal{CH}$ picks $b = 1$ (and uses $L_1[j+1]$ in the computation of σ_1^*, $\mathcal{A}$'s view is identical to that of $\mathcal{G}_j$). When $\mathcal{B}$ receives b from $\mathcal{A}$, it passes that bit on to $\mathcal{CH}$.

$\square$

Lemma 4. *If TrRS satisfies* one-more linkability *(Definition 10) then Construction 3 satisfies* unforgeability *(Definition 7).*

Proof. We will construct an algorithm $\mathcal{B}$ which will break the *one-more linkability* of the underlying TrRS scheme against a TrRS challenger $\mathcal{CH}$, by assuming we have an attacker $\mathcal{A}$ who can break the *unforgeability* of the TRS scheme in Construction 3.

Setup $\mathcal{B}$ recieves 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 TrRS challenger $\mathcal{CH}$, receiving the public parameters $\texttt{params}$ 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 now issue **corruption** and **signing** queries to $\mathcal{B}$, which $\mathcal{B}$ handles in the same fashion as for the anonymity game Lemma 3, by forwarding to the TrRS challenger $\mathcal{CH}$ with the appropriate nonce $\texttt{nonce} = (msg, \{pk_i\}_{i \in \mathcal{S}})$, and returns the challenger's response to the adversary.

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

- $\mathcal{B}$ parses $\sigma^* = \{\sigma_1^*, \ldots, \sigma_t^*\}$
- $\mathcal{B}$ sends $\{(msg_k, \sigma_k^*, \mathcal{S})\}_{k \in [t]}, \texttt{nonce} = (msg, \{pk_i\}_{i \in \mathcal{S}})$ 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 TrRS one-more linkability game against the challenger $\mathcal{CH}$.

□

5 Conclusion

In this paper, we made two contributions to the field of threshold ring signature schemes. First, we gave a stronger security definition for TRS anonymity which is more natural in practice. In particular, our definition demands that signers cannot be de-anonymized even by their fellow signers and ring-members. This is crucial, as in many applications, it is unrealistic to assume that there are no insiders in the ring.

Secondly, we construct the first TRS scheme with signatures of size $\mathcal{O}(t)$ (where t is the number of signers), independent of the number n of parties in the ring $\mathcal{S}$. We achieve this by using a TrRS scheme as a building block. To this end, we strengthen the definitions of TrRS, and propose a new TrRS construction from standard assumptions which produces signatures of constant size.

References

[ACST06] Man Ho Au, Sherman S. M. Chow, Willy Susilo, and Patrick P. Tsang. Short linkable ring signatures revisited. In Andrea S. Atzeni and Antonio Lioy, editors, *Public Key Infrastructure*, pages 101–115, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BCY20] Foteini Baldimtsi, Ran Canetti, and Sophia Yakoubov. Universally composable accumulators. pages 638–666, 2020.

[Bd94] Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). pages 274–285, 1994.

[BDR15] Priyanka Bose, Dipanjan Das, and C. Pandu Rangan. Constant size ring signature without random oracle. Cryptology ePrint Archive, Report 2015/164, 2015. `http://eprint.iacr.org/2015/164`.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of Diffie-Hellman problem. pages 301–312, 2003.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC 88, page 103112, New York, NY, USA, 1988. Association for Computing Machinery.

[BSS02] Emmanuel Bresson, Jacques Stern, and Michael Szydlo. Threshold ring signatures and applications to ad-hoc groups. pages 465–480, 2002.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. pages 280–300, 2013.

[CHGL18] J. Chen, Y. Hu, W. Gao, and H. Liang. Lattice-based threshold ring signature with message block sharing. *KSII Transactions on Internet and Information Systems*, 13:1003–1019, 02 2018.

[DKNS04] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. Anonymous identification in ad hoc groups. pages 609–626, 2004.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string. *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 308–317 vol.1, 1990.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[FS07] Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. pages 181–200, 2007.

[GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. pages 581–612, 2017.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *J. ACM*, 38:691–729, 1991.

[HS20] Abida Haque and Alessandra Scafuro. Threshold ring signatures: New definitions and post-quantum security. pages 423–452, 2020.

[Liu19] Joseph K. Liu. *Ring Signature*, pages 93–114. Springer Singapore, Singapore, 2019.

[LWW04] Joseph K. Liu, Victor K. Wei, and Duncan S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan, editors, *Information Security and Privacy*, pages 325–335, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[OTYO18] Takeshi Okamoto, Raylin Tso, Michitomo Yamaguchi, and Eiji Okamoto. A k-out-of-n ring signature with flexible participation for signers. *IACR Cryptology ePrint Archive*, 2018:728, 2018.

[PBB12] Albrecht Petzoldt, Stanislav Bulygin, and Johannes Buchmann. A multivariate based threshold ring signature scheme. Cryptology ePrint Archive, Report 2012/194, 2012. `http://eprint.iacr.org/2012/194`.

[RST01] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. pages 552–565, 2001.

[SW13] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. Cryptology ePrint Archive, Report 2013/454, 2013. `http://eprint.iacr.org/2013/454`.

[TW05] Patrick P. Tsang and Victor K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. In Robert H. Deng, Feng Bao, HweeHwa Pang, and Jianying Zhou, editors, *Information Security Practice and Experience*, pages 48–60, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[YLA$^+$11] Tsz Yuen, Joseph Liu, Man Ho Au, Willy Susilo, and Jianying Zhou. Threshold ring signature without random oracles. pages 261–267, 01 2011.

[YLA$^+$13] T. H. Yuen, J. K. Liu, M. H. Au, W. Susilo, and J. Zhou. Efficient linkable and/or threshold ring signature without random oracles. *The Computer Journal*, 56(4):407–421, 2013.

[ZZY$^+$17] Guomin Zhou, Peng Zeng, Xiaohui Yuan, Siyuan Chen, and Kim-Kwang Choo. An efficient code-based threshold ring signature scheme with a leader-participant model. *Security and Communication Networks*, 2017:1–7, 08 2017.