Concise Linkable Ring Signatures and Forgery Against Adversarial Keys

Brandon Goodell[1], Sarang Noether[1], and Arthur Blue[2]

[1] Monero Research Lab, {surae,sarang}@getmonero.org
[2] Independent researcher, randomrun@protonmail.com

Abstract. We demonstrate that a version of non-slanderability is a natural definition of unforgeability for linkable ring signatures. We present a linkable ring signature construction with concise signatures and multi-dimensional keys that is linkably anonymous if a variation of the decisional Diffie-Hellman problem with random oracles is hard, linkable if key aggregation is a one-way function, and non-slanderable if a one-more variation of the discrete logarithm problem is hard. We remark on some applications in signer-ambiguous confidential transaction models without trusted setup.

1 Introduction

First introduced in [19] in the RSA setting and in [13] in the discrete logarithm setting, ring signatures permit message signing on behalf of a set of public keys rather than a single public key. Ring signatures see myriad applications ranging from lightweight anonymous authentication in [24] to transaction protocols like Monero in [15] and CryptoNote in [23]. A verifier is assured that the signer knows the private key of at least one of these public keys, which are called ring members. Ring signatures are anonymous or signer-ambiguous in the sense that the verifier does not learn information from the signature about which key is the signer. We stress that methods of practical analysis such as those of [14, 18] can exploit metadata in real-life applications of anonymous authentication protocols to reduce anonymity.

Group signature constructions preceding [19, 13] require some degree of interactivity, a fixed set of participants, a trusted group manager or other trusted setup, or hardness assumptions not based on the discrete logarithm problem. Since [19], ring signatures have enjoyed many improvements, extensions, and modifications. For example, ring signatures are constructed in the bilinear pairing setting in [25], key structures are generalized in [1], security definitions are improved in [5], signature size is improved in [7, 11], and traceability is introduced in [8].

Linkable ring signature (LRS) constructions were first introduced in [13]; in the context of transaction protocols, linkable ring signatures are the basis for anonymous transaction authentication. Linkable ring signatures guarantee that two signatures with the same ring on arbitrary messages can be publicly linked if

signed using the same key. An implementation is presented in [13] in the discrete logarithm setting; that implementation functions for similar reasons as Schnorr signatures in [20].

The signature linking tags in [13] are unsuitable for applications where signatures must be linked key-by-key, not ring-by-ring (such as for "double-signing" protection in a setting where users generate new keys over time and select *ad hoc* ring members). Resistance to double-signing attempts is ensured using linking tags as described in [23] (therein called key images). More recent work of [15] extends the approach of [13] to enable an anonymous confidential transaction model with *ad hoc* ring member selection. In [15], transaction amounts are replaced with Pedersen commitments to amounts together with range proofs. Signatures are constructed from key vectors including differences of amount commitments as one of the keys. However, the proofs in [15] are informal and not based on rigorous security models.

Alternatives to ring signatures like more general zero-knowledge proving systems typically require a trusted party to honestly perform a setup process (as in [9, 4, 10]) or lack practical efficiency for large circuits (as in [6]), meaning that such systems may not be appropriate for distributed ledger applications. However, more recent approaches such as [12] show both improvements to the trust requirement as well as improvements in efficiency.

Definitions of existential unforgeability for ring signature schemes from [5] are generally inappropriate for LRS. These definitions ensure that forgeries appear to be computed from uncorrupted challenge keys by restricting choices of anonymity set members and oracle queries. We can do better with linkability.

The idea of existential unforgeability in usual digital signatures is to task an algorithm to produce any valid, non-oracle signature on any message such that the signature appears to be signed by an uncorrupted challenge key. The clear analogue for LRS schemes is to task an algorithm to produce any valid, non-oracle signature on any message such that the signature links to a challenge signature computed from an uncorrupted challenge key. Contrast this intuition with the definition of unforgeability for usual ring signatures with respect to insider corruption from [5], which does not count a valid signature σ with an anonymity set Q as a forgery if any key in Q is corrupt or not a challenge key, or a signing oracle has been queried with the same anonymity set Q and message. This leads to two problems.

The first problem is due to the disqualification of a purported forgery if there are any adversarially-selected keys or corrupted challenge keys in the anonymity set. Such a signature that links with an uncorrupted challenge key does not count as a forgery. In ring signature schemes without linkability, this restriction seems requisite to ensure that the signature appears to be signed by an uncorrupted challenge key. In applications like signer-ambiguous transaction protocols that use anonymity sets selected *ad hoc*, anonymity set members should be assumed to be adversarially generated. Definitions of unforgeability should take this into account.

The second problem is due to the signing oracle restriction in this definition. An adversary could generate a valid, non-oracle signature re-using the same message and anonymity set as some challenge signature generated by the oracle. A verifier who has found more than one valid non-oracle signature on the same message with the same anonymity set, under this definition, can only conclude that at least one anonymity set member signed at least one of the signatures. This allows a malicious Mallory-in-the-middle to collect outgoing honest signatures on some honest messages and attempt to construct clone signatures on these honest messages. A sender attempting to assert culpability over a signature could then be in danger of being unable to reconstruct a signature sent to some receiver.

These problems seem to interfere with the spirit behind existential unforgeability with insider corruption from [5]: the signature is valid, not produced by an oracle, appears to have been computed from an uncorrupted challenge key, and yet is not considered a forgery.

1.1 Our Contribution

We relax the notion of successful forgeries to require only linking with uncorrupted challenge keys, obtaining a notion of unforgeability for linkable ring signatures that is equivalent to the definition of non-slanderability in [2]. Non-slanderability allows for adversaries who can query a signature oracle with any anonymity set, even adversarially-selected anonymity sets. This models an extremely persuasive adversary who can convince users to sign messages with adversarially-selected anonymity sets. We show that non-slanderability implies unforgeability with respect to insider corruption from [5].

We also present a Schnorr-like linkable ring signature scheme we call d-CLSAG that exploits key aggregation techniques to improve signature size. We describe an application for the d-CLSAG scheme in ring confidential transaction protocols, allowing transactions consisting of $d-1$ distinct assets simultaneously. We prove that d-CLSAG is linkable assuming collision resistance in key aggregation and linkably anonymous assuming the decisional Diffie-Hellman game is hard. Recall that Schnorr-like signature schemes do not cleanly reduce to the discrete logarithm assumption (*c.f.* [16]), generally are only secure up to the (relatively flimsier) one-more hardness assumptions, and proof techniques typically exploit programmable random oracles with forking lemmata, leading to weak bounds. We use the general forking lemma to prove that d-CLSAG is non-slanderable assuming a κ-one-more variation of the discrete logarithm game is hard.

2 Preliminaries

Let λ be a security parameter and $1 \leq q, \eta, \kappa \in poly(\lambda)$. Let $\mathbb{G} = \langle G \rangle$ denote a group over a field $\mathbb{F}$ with generator G. Let $\mathcal{H}^s : \{0,1\}^* \to \mathbb{F}$ and $\mathcal{H}^p : \{0,1\}^* \to \mathbb{G}$ be two independent cryptographic hash functions under the random oracle

model, *i.e.* sampled uniformly from the set of all functions $\{0,1\}^* \to \mathbb{F}$ and $\{0,1\}^* \to \mathbb{G}$, respectively.

If there exists a function f that is negligible in some parameter λ, and there exists λ-dependent events A, B such that $|\mathbb{P}[A] - \mathbb{P}[B]| \leq f(\lambda)$, we denote this $\mathbb{P}[A] \approx \mathbb{P}[B]$. An event A such that $\mathbb{P}[A] \approx 0$ is said to be negligible in λ (or to occur with probability at most negligible in λ). The complement of an event A is denoted $\overline{A}$. For a field $\mathbb{F}$, we denote the non-zero elements $\mathbb{F}^* := \mathbb{F} \setminus \{0\}$. For a finite set S, we define S^* by interpreting S as a list of letters that generate the free monoid S^*. For example, $\{0,1\}^*$ consists of all finite-length bitstrings.

For a set S, we define $\mathcal{P}(S)$ as the power set of S, i.e. the set of all subsets of S.

We denote vectors with bold, i.e. for a sequence $x_i \in \mathbb{F}$ for $i = 0, \ldots, n-1$, we denote the tuple $(x_0, \ldots, x_{n-1})$ with $\mathbf{x}$. We denote the Hadamard product between vectors with $\mathbf{x} \circ \mathbf{y} = (x_i y_i)_i$. We begin all indices at 0 to avoid all ambiguity.

3 Definitions

3.1 Hardness Assumptions

We begin by noting that if $\{\mu_i\}_{i=0}^{d-1} \subseteq \mathbb{F}$ are selected uniformly at random, then the function $\mu : \mathbb{F}^d \to \mathbb{F}$ defined by mapping $\mathbf{x} \mapsto \sum_i \mu_i x_i$ has uniformly distributed output and is collision resistant. We use this property when considering linkability and linkable anonymity.

The following game is equivalent[3] to the usual κ-one-more discrete logarithm game under the random oracle model.

Definition 1 (κ-One-More Linear Combination Discrete Logarithm). *Let $\kappa \geq 1$. We say any PPT algorithm $\mathtt{A}$ that can succeed at the following game in time at most t and with success probability at least ϵ is a (t, ϵ, q)-solver of the κ-one-more discrete logarithm of linear combinations problem in $\mathbb{G}$.*

1. *Challenger draws $\{sk_i\}_{i=0}^{q-1} \subseteq \mathbb{F}$ at random, computes $pk_i := sk_i \cdot G$, sends $S := \{pk_i\}_{i=0}^{q-1}$ to $\mathtt{A}$.*
2. *$\mathtt{A}$ is granted access to a corruption oracle $\mathtt{CO}$ which takes as input a public key $pk_i \in S$ and produces as output the corresponding private key sk_i. Corrupted keys are recorded in a table C.*
3. *$\mathtt{A}$ outputs some $w \in \mathbb{F} \setminus \{0\}$, a sequence of field elements $\{h_j\}_{j=0}^{\kappa} \subseteq \mathbb{F} \setminus \{0\}$, and a subset of challenge keys $\{pk_j^*\}_{j=0}^{\kappa} \subseteq \{pk_i\}_{i=0}^{q-1}$, succeeding if and only if $w \cdot G = \sum_{j=0}^{\kappa} h_j \cdot pk_j^*$ and $\mathtt{A}$ queried $\mathtt{CO}$ no more than κ times.*

[3] A usual κ-one-more discrete logarithm solver can succeed at the game in Definition 1 with the same probability and some extra time by merely summing solutions. A solver of the game in Definition 1 can use their discrete logarithm knowledge to solve for an otherwise uncorrupted challenge key with the same probability and some extra time.

We say that the discrete logarithms of linear combinations are computationally hard to compute in $\mathbb{G}$ if any (t, ϵ, q)-solver of this game has an advantage that is ϵ negligible in q.

For an algorithm $\mathtt{A}$ that takes as input a random tape ρ, a sequence of random oracle queries $\mathbf{h}$ and some input inp and produces $(out, idx) \leftarrow \mathtt{A}(\rho, \mathbf{h}, inp)$ as output, the general forking algorithm works as follows.

(1) Pick a random tape ρ for $\mathtt{A}$ and two sequences of random oracle queries $\mathbf{h} = \{h_0, h_1, \ldots\}$ and $\mathbf{h}' = \{h'_0, h'_1, \ldots\}$.
(2) Execute $(out, idx) \leftarrow \mathtt{A}(\rho, \mathbf{h}, inp)$.
(3) Set $j := idx$ and glue the oracle queries $\mathbf{h}$, $\mathbf{h}'$ together

$$\mathbf{h}^* = \left\{h_0, h_1, \ldots, h_{idx-1}, h'_{idx}, h'_{idx+1}, \ldots, \right\}.$$

(4) Execute $(out^*, idx^*) \leftarrow \mathtt{A}(\rho, \mathbf{h}^*, inp)$.
(5) If $idx \neq idx^*$ or $h_{idx} = h'_{idx}$, output $\perp$ and terminate. Otherwise, output (out, out', idx).

This algorithm is precisely the algorithm $\mathcal{F}^{\mathtt{A}}$ in the following lemma.

Lemma 1 (General Forking Lemma). *Let $1 \leq \eta \in poly(\lambda)$. Let $\mathtt{A}$ be any PPT algorithm which takes as input some tuple $x_{\mathtt{A}} = (x, \boldsymbol{h})$ where we let $\boldsymbol{h} = (h_0, h_1, \ldots, h_{q-1})$ be a sequence of oracle query responses (η-bit strings) and returns as output $y_{\mathtt{A}}$ either a distinguished failure symbol $\perp$ or a pair (idx, y) where $idx \in [q]^2$ and y is some output. Let $\epsilon_{\mathtt{A}}$ denote the probability that $\mathtt{A}$ does not output $\perp_{\mathtt{A}}$ (where this probability is taken over all random coins of $\mathtt{A}$, the distribution of x, and all choices $\boldsymbol{h}$). Let $\mathcal{F} = \mathcal{F}^{\mathtt{A}}$ be a forking algorithm for $\mathtt{A}$. The accepting probability of $\mathcal{F}$ satisfies $\epsilon_{\mathcal{F}} \geq \epsilon_{\mathtt{A}} \left(\frac{\epsilon_{\mathtt{A}}}{q} - \frac{1}{2^{\eta}} \right)$.*

We assume that the following game, under the random oracle model, is as hard as the usual (fixed-base) decisional Diffie-Hellman game.

Definition 2 (Random Oracle Decisional Diffie-Hellman).
We say any PPT algorithm $\mathtt{A}$ that can succeed at the following game in time at most t and with success probability at least ϵ is a (t, ϵ, q)-solver of the random oracle decisional Diffie-Hellman game.

1. *Challenger draws $\{sk_i\}_{i=0}^{q-1} \subseteq \mathbb{F}$ at random, computes $pk_i := sk_i \cdot G$, computes group elements $\mathfrak{T}_i^{(1)} := sk_i \cdot \mathcal{H}^p(pk_i)$, samples random group elements $\left\{\mathfrak{T}_i^{(0)}\right\}_{i=0}^{q-1} \subseteq \mathbb{G}$, computes $S^{(j)} := \left\{(pk_i, \mathfrak{T}_i^{(j)})\right\}_{i=0}^{q-1}$ for $j \in \{0, 1\}$, samples a random bit b, sends $S^{(b)}$ to $\mathtt{A}$, and lastly grants $\mathcal{H}^p$ access to $\mathtt{A}$.*
2. *$\mathtt{A}$ outputs a bit b', succeeding if and only if $b = b'$.*

We say random oracle Diffie-Hellman points are hard to computationally distinguish from random in $\mathbb{G}$ if any (t, ϵ, q)-solver of the random oracle decisional Diffie-Hellman game has an advantage $|\epsilon - 1/2|$ that is negligible in q, where this probability is measured over all functions $\mathcal{H}^p$.

3.2 Linkable Ring Signatures

Definition 3. *A linkable ring signature (LRS) scheme Π_{LRS} is a tuple of algorithms ($\texttt{SETUP}, \texttt{KEYGEN}, \texttt{SIGN}, \texttt{VERIFY}, \texttt{LINK}$) satisfying the following constraints.*

- *$\texttt{SETUP}$ takes as input a security parameter λ and produces as output some public setup parameters ρ.*
- *$\texttt{KEYGEN}$ takes as input (λ, ρ) and outputs a private-public keypair (sk, pk).*
- *$\texttt{SIGN}$ takes as input (λ, ρ) and a triple (m, Q, sk) and outputs a signature σ or a failure symbol $\perp$. Here m is a message, Q is an anonymity set of public keys $Q = \{pk_0, pk_1, ..., pk_{n-1}\}$, and sk is a private key.*
- *$\texttt{VERIFY}$ takes as input (λ, ρ) and a triple (m, Q, σ) and outputs a bit b. Here m is a message, Q is an anonymity set of public keys $Q = \{pk_0, pk_1, ..., pk_{n-1}\}$, and σ is a signature.*
- *$\texttt{LINK}$ takes as input (λ, ρ), a pair of triples (m, Q, σ), (m^*, Q^*, σ^*), and outputs a bit b. Here m, m^* are messages, Q, Q^* are anonymity sets of public keys, and σ, σ^* are signatures.*

We say a triple of the form (m, Q, σ) (that is to say, a triple suitable for use as input in $\texttt{VERIFY}$ and $\texttt{LINK}$) is a signature triple. We say Π_{LRS} is a d-LRS if the dimension of private and public keys is $d \geq 1$ and anonymity sets can be described as $d \times n$ matrices.

We assume ρ includes (implicitly or explicitly) descriptions of a private key space $\mathcal{SK}$, a public key space $\mathcal{PK}$, a signature space $\mathcal{SIG}$, a map $\phi : \mathcal{SK} \to \mathcal{PK}$, and a family of hash functions $\mathcal{H}$ from which we can construct $\mathcal{H}^s$ and $\mathcal{H}^p$. For example, with an elliptic curve group $\mathbb{G}$ over a field $\mathbb{F}$, for two hash functions $H_s : \{0,1\}^* \to \mathbb{F}$ and $H_p : \{0,1\}^* \to \mathbb{G}$, a Schnorr-like ring signature scheme such as [13] would pack into ρ the descriptions $\mathcal{SK} = \mathbb{F}^*$, $\mathcal{PK} = \mathbb{G}$, $\mathcal{SIG} = \mathbb{F}^n$, and a generator $G \in \mathbb{G}$ (which is sufficient to specify the map $\phi : \mathbb{F}^* \to \mathbb{G}$ defined by $x \mapsto xG$). Note each algorithm in an LRS scheme takes (λ, ρ) as input; in the sequel we exclude these from notation for LRS schemes, taking their use as implicit.

Note we do not allow a LRS scheme to be correct if the scheme allows anonymity multisets Q.

Definition 4. *Let $b \in \{0,1\}$, $sk, sk^* \in \mathcal{SK}$, let Q, $Q^* \subset \mathcal{PK}$ be sets, let m, m^*, m' be messages, let $\sigma, \sigma^*, \sigma' \in \mathcal{SIG}$ be any purported signatures. Define the following events.*

- *$E_1(sk, pk)$ is the event that some $(sk, pk) \leftarrow \texttt{KEYGEN}$.*
- *$E_2(sk, Q)$ is the event that $\phi(sk) \in Q$.*
- *$E_3(m, Q, \sigma)$ is the event that $\texttt{VERIFY}(m, Q, \sigma) = 1$.*
- *$E_4(b, m, m^*, m', Q, Q^*, Q', \sigma, \sigma^*, \sigma')$ is the event that*

$$b = \texttt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = \texttt{LINK}((m^*, Q^*, \sigma^*), (m', Q', \sigma')).$$

- *$E_5(m, m^*, Q, Q^*, \sigma, \sigma^*, sk)$ is the event that*

$$\phi(sk) \in Q \cap Q^*,\ \sigma \leftarrow \texttt{SIGN}(m, Q, sk),\ \text{and}\ \sigma^* \leftarrow \texttt{SIGN}(m^*, Q^*, sk).$$

– $E_6(m, m^*, Q, Q^*, \sigma, \sigma^*, sk, sk^*)$ *is the event that*

$$\sigma \leftarrow \texttt{SIGN}(m, Q, sk),\ \sigma^* \leftarrow \texttt{SIGN}(m^*, Q^*, sk^*),\ \textit{and}\ sk \neq sk^*.$$

We say Π_{LRS} is correct if all of the following properties hold, where these probabilities are computed over all random coins and over all choices of hash function.

(i) $\mathbb{P}\left[\phi(sk) = pk \mid E_1(sk, pk)\right] \approx 1$
(ii) $\mathbb{P}\left[\texttt{SIGN}(m, Q, sk) = \perp \mid \overline{E_2}(sk, Q)\right] \approx 1$
(iii) $\mathbb{P}\left[\texttt{VERIFY}(m, Q, \texttt{SIGN}(m, Q, sk)) = 1 \mid E_2(sk, Q)\right] \approx 1$
(iv) $\mathbb{P}\left[\texttt{LINK}((m, Q, \sigma), (m, Q, \sigma)) = 1 \mid E_3(m, Q, \sigma)\right] \approx 1$
(v) $\mathbb{P}\left[\texttt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = \texttt{LINK}((m^*, Q^*, \sigma^*), (m, Q, \sigma))\right] \approx 1$
(vi) $\mathbb{P}\left[b = \texttt{LINK}((m, Q, \sigma), (m', Q', \sigma')) \mid E_4(b, m, m^*, m', Q, Q^*, Q', \sigma, \sigma^*, \sigma')\right] \approx 1$
(vii) $\mathbb{P}\left[\texttt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = 1 \mid E_5(m, m^*, Q, Q^*, \sigma, \sigma^*, sk)\right] \approx 1$
(viii) $\mathbb{P}\left[\texttt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = 0 \mid E_6(m, m^*, Q, Q^*, \sigma, \sigma^*, sk, sk^*)\right] \approx 1$

3.3 Linkability

We use two distinct, but related, definitions of linkability. The first is from [2]. This definition allows signing oracle queries with any anonymity sets Q consisting of at least one challenge key (which can be simulated via backpatching). This definition also allows the challenger to succeed at the linkability game using signature triples (m, Q, σ), (m^*, Q^*, σ^*) that have semi-corrupted anonymity sets Q, Q^* (just so long as the keys in $Q \cup Q^*$ have not been tampered with too severely). The second is from [3], which allows the adversary total control over key selection, declaring success when they produce more unlinked ring signatures than total ring members. Such an adversary doesn't need key generation, corruption, or signature oracle access.

Definition 5 (ACST Linkability With an Adversarial Key). *We say any PPT algorithm* `A` *that can succeed at the following game in time at most t and with success probability at least ϵ is a (t, ϵ, q)-solver of ACST linkability.*

1. *Challenger draws $\{(sk_i, pk_i)\}_{i=0}^{q-1}$ from* `KEYGEN` *and sends the challenge public keys $S = \{pk_i\}_{i=0}^{q-1}$ to* `A`.
2. `A` *is granted* `SO` *and* `CO` *access.*
   – `CO` *takes as input a public key pk. If $pk \in S$,* `CO` *outputs the corresponding private key sk. Otherwise,* `CO` *outputs a failure symbol, $\perp$. Corrupted keys are tracked in a table C.*
   – `SO` *operates as follows:*
      (i) Takes as input a message m, an anonymity set $Q = \{pk'_i\}_{i=0}^{n-1}$, and an index l.
      (ii) If $0 > l$ or $l \geq n$ or if the $pk'_l \notin S$, `SO` *outputs a failure symbol $\perp$.*
      (iii) Otherwise, there exists some i such that $pk'_l = pk_i \in S$. `SO` *retrieves the private key sk_i and outputs a valid signature triple (m, Q, σ) such that $\sigma \leftarrow \texttt{SIGN}(m, Q, sk_i)$.*

3. *$\mathtt{A}$ outputs a pair of signature triples, (m, Q, σ), (m^*, Q^*, σ^*), succeeding if and only if all of the following conditions are satisfied.*
   (i) $\mathtt{VERIFY}(m, Q, \sigma) = \mathtt{VERIFY}(m^*, Q^*, \sigma^*) = 1$
   (ii) $\mathtt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = 0$
   (iii) σ, σ^* *are not output from any* $\mathtt{SO}$ *query.*
   (iv) $\left|(Q \cup Q^*) \cap (C \cup \overline{S})\right| \leq 1$

We say that a scheme is ACST linkable if every PPT algorithm $\mathtt{A}$ that is a (t, ϵ, q)-solver ACST linkability game has a negligible acceptance probability ϵ.

Definition 6 (q-Pigeonhole Linkability). *We say any PPT algorithm $\mathtt{A}$ that can succeed at outputting q public keys $\{pk_i\}_{i=0}^{q-1}$ and $q+1$ valid, unlinked signature triples $\{(m_j, Q_j, \sigma_j)\}_{j=0}^{q}$ such that $\cup_j Q_j \subseteq \{pk_i\}_{i=0}^{q-1}$ in time at most t and probability at least ϵ is a (t, ϵ, q)-solver of pigeonhole linkability. We say that a scheme is q-pigeonhole linkable if every PPT algorithm $\mathtt{A}$ that is a (t, ϵ, q)-solver of q-pigeonhole linkability has a negligible acceptance probability ϵ.*

These definitions are distinct. Pigeonhole linkability does not necessarily imply ACST linkability. Indeed, a successful player of the game in Definition 5 may only succeed with sufficiently large rings, in which case that player may only be used to succeed at the game of Definition 6 for a sufficiently large q. Similarly, ACST linkability may not imply pigeonhole linkability. A player of the game in Definition 6 that requires knowledge of many private keys may be too corruption-hungry to succeed at Definition 5.

For a scheme that is pigeonhole linkable, Definition 6 says nothing about an adversary that can keep signing with one key multiple times to construct new unlinked signatures, just so long as they continue to use different anonymity sets.

For a scheme that is ACST linkable, a user providing honest signatures to an adversary acts like a signature oracle, so the adversary could attempt to construct a bad/cloned signature that links to some honest one (possibly with adversarially selected ring members), only ever outputting a single signature. Definition 5 says nothing about such an attacker. This attacker could stand between Alice and Bob, provide cloned signatures to the Bob ostensibly signed by Alice, fooling him. When Alice eventually contacts Bob without the adversary in the middle, Alice would be unable to assert culpability over the cloned signatures. Bob concludes that σ^* and σ were signed by the same private key, despite that the adversary does not know the private key that generated σ.

However, Definition 6 implies Definition 5 under certain circumstances. For one (of many) examples, if there exists a $(t, \epsilon, 1)$-solver of the game in Definition 5 that produces two signature triples (m, Q, σ), (m', Q', σ') such that $Q = Q'$ and $|Q| = 1$, then $\mathtt{A}$ succeeds at Definition 6.

A more general look at the relationships between linkability definitions is beyond the scope of this work. Since neither definition is sufficient for our purposes alone and these definitions are distinct in general, we use both.

3.4 Unforgeability and Non-Slanderability

The idea of existential unforgeability in usual digital signatures is to task an algorithm to produce any valid, non-oracle signature on any message such that the signature appears to be signed by an uncorrupted challenge key. The clear analogue for LRS schemes is to task an algorithm to produce any valid, non-oracle signature on any message such that the signature links to a challenge signature computed from an uncorrupted challenge key. In fact, the implementation in Section 4 compares linking tags as if they were verification keys; in this sense, a forger is tasked to construct a valid signature with some verification keys to which they do not know the corresponding private key. We consider this to be a textbook forgery, and so we take these occurrences into account in our definition.

Definition 7 (Existential Unforgeability of Linkable Ring Signatures Against Adversarially-Selected Keys and Insider Corruption). *We say any PPT algorithm* `A` *that can succeed at the following game in time at most* t *and with probability at least* ϵ *is a* (t, ϵ, q)*-forger.*

1. *Challenger draws* $\{(sk_i, pk_i)\}_{i=0}^{q-1}$ *from* `KEYGEN` *and sets* $S := \{pk_i\}_{i=0}^{q-1}$*. For each* $0 \leq i < q$*, the challenger uniformly samples secret* $\emptyset \neq \widetilde{Q}_i \in \mathcal{P}(S)$*, defines* $Q_i := \widetilde{Q}_i \cup \{pk_i\}$*, retrieves the index* l_i *such that* $pk_i \in Q_i \cap S$ *has index* l_i *in* Q_i*, samples secret random messages* $\{m_i\}_{i=0}^{q-1}$*, and computes secret challenge signatures* $\sigma_i \leftarrow$ `SIGN`(m_i, Q_i, sk_i)*. Challenger sends* S *to* `A`*.*
2. *Challenger grants* `SO` *and* `CO` *access to* `A`*:*
   - `CO` *takes as input a challenge public key* $pk_i \in S$ *and produces as output the corresponding private key* sk_i*, keeping a list of all corrupted public keys in an internal table* C*.*
   - `SO` *operates as follows:*
     (i) Takes as input a message m*, an anonymity set* Q*, and an index* l*.*
     (ii) If the key at index l *in* Q *is not a challenge public key from* S*,* `SO` *outputs a failure symbol* $\perp$*.*
     (iii) Otherwise, `SO` *retrieves the index* i *such that* $pk_i \in Q \cap S$ *has index* l *in* Q*, retrieves the corresponding private key* sk_i*, and outputs a signature* $\sigma \leftarrow$ `SIGN`(m, Q, sk_i)*.*
3. `A` *outputs a message* m*, a ring* Q*, a signature* σ*, succeeding if and only if:*
   (i) `VERIFY`$(m, Q, \sigma) = 1$*,*
   (ii) there does not exist a query made to `SO` *whose output is* σ*,*
   (iii) `LINK`$((m, Q, \sigma), (m_i, Q_i, \sigma_i)) = 1$ *for some* $pk_i \in S \cap Q \setminus C$*.*

Moreover, we say that a linkable ring signature scheme is unforgeable if every (t, ϵ, q)*-solver of this game has a negligible acceptance probability* ϵ*.*

Remark 1. Access to a corruption oracle in this unforgeability definition is one reason security reduces (weakly) to the κ-one-more discrete logarithm problem, not the usual discrete logarithm problem: an algorithm executing a forgery algorithm in a black box cannot simulate the corruption oracle for the forgery without, itself, gaining corruption oracle access or without resorting to the generic group model.

Remark 2. If $\mathtt{A}$ is an algorithm that produces signatures from an LRS satisfying Definition 7, then (except with negligible probability), the signatures aren't forgeries, so some property 3(i)-3(iii) must be violated. If the signature is valid and not from an oracle query, then 3(iii) fails. In particular, for an LRS scheme satisfying Definition 7, if an algorithm produces a valid non-oracle triple (m, Q, σ) that links to a challenge key in Q, then that challenge key has been corrupted.

Non-slanderability is introduced in [22]. This definition is refined twice in [2], allowing $\mathtt{A}$ to succeed whenever it publishes any signature that links with any signature from a query $\mathtt{A}$ made to $\mathtt{SO}$, excepting certain conditions. We modify the definition from [2] to allow signing oracle queries with anonymity sets containing adversarially selected members.

Definition 8 (Chosen-Target Non-Slanderability Against Adversarial Keys and Insider Corruption). *We say any PPT algorithm $\mathtt{A}$ that can succeed at the following game in time at most t and with probability at least ϵ is a (t, ϵ, q)-solver of non-slanderability against adversarially chosen keys.*

1. *Challenger draws $\{(sk_i, pk_i)\}_{i=0}^{q-1}$ from KeyGen and sends the challenge public keys $S = \{pk_i\}_{i=0}^{q-1}$ to $\mathtt{A}$.*
2. *A is granted $\mathtt{SO}$ and $\mathtt{CO}$ access just as in Definition 7.*
3. *A outputs a triple (m, Q, σ), succeeding if and only if:*
   - *(i) $\mathtt{VERIFY}(m, Q, \sigma) = 1$, and*
   - *(ii) there does not exist a query made to $\mathtt{SO}$ whose output is σ, and*
   - *(iii) there exists a query made to $\mathtt{SO}$, say $\sigma^* \leftarrow \mathtt{SO}(m^*, Q^*, l^*)$, such that*
     - *(a) $\mathtt{LINK}((m, Q, \sigma), (m^*, Q^*, \sigma^*)) = 1$, and*
     - *(b) $pk^*_{l^*} \in S \cap Q^* \cap Q \setminus C$.*

Moreover, we say that a linkable ring signature scheme is non-slanderable if every (t, ϵ, q)-solver of this game has a negligible acceptance probability ϵ.

Theorem 1 (Non-Slanderability is Unforgeability). *A correct LRS is unforgeable under Definition 7 if and only if it is non-slanderable under Definition 8.*

Proof. If $\mathtt{A}$ is a (t, ϵ, q)-solver of the game of Definition 7, then $\mathtt{A}$ can always be wrapped in an algorithm that executes $\mathtt{A}$ in a black box and regurgitates the results, except querying the signing oracle with the linking challenge key before outputting the results of $\mathtt{A}$. Such an algorithm succeeds at the game of Definition 8 with exactly equal success probability and asymptotically equal timing as $\mathtt{A}$. For the converse, let $\mathtt{A}$ be a (t, ϵ, q)-solver of the game of Definition 8. With probability at least ϵ, $\mathtt{A}$ outputs a signature triple (m, Q, σ) such that the triple is valid and σ is not output from any $\mathtt{SO}$ query and there exists some query, say $\sigma^* \leftarrow \mathtt{SO}(m^*, Q^*, pk^*_{l^*})$ in the transcript between $\mathtt{A}$ and $\mathtt{SO}$. The challenge key $pk^*_{l^*} = pk_i \in S$ for some i and the signature σ^* links with σ. By the transitivity of $\mathtt{LINK}$, we obtain that σ links to σ_i.

3.5 Linkable Anonymity

We use a modification of the definition of linkable anonymity from [3].

Definition 9 (Chosen-Target Linkable Anonymity With Adversarial Keys). *We say any PPT algorithm* `A` *that can succeed at the following game in time at most* t *and with probability at least* ϵ *is a* (t, ϵ, q)*-solver of the linkable anonymity game.*

1. *Challenger samples a secret random bit* $b \in \{0, 1\}$*, draws* $\{(sk_i, pk_i)\}_{i=0}^{q-1}$ *from* `KEYGEN`*, and sends the challenge public keys* $S := \{pk_i\}_{i=0}^{q-1}$ *to* `A`*.*
2. `A` *outputs a pair of indices* $0 \leq i_0, i_1 < q$ *such that* $pk_{i_0}, pk_{i_1} \in S$*, indicating the target keys.*
3. `A` *is granted access to a signing oracle* `SO`*:*
   (i) *Takes as input a message* m*, an anonymity set* Q*, and a public key* $pk \in Q$*.*
   (ii) *If* $\{pk_{i_0}, pk_{i_1}\} \not\subseteq Q$ *or* $pk \notin \{pk_{i_0}, pk_{i_1}\}$*, then* `SO` *outputs a valid signature* σ *linked to* pk*.*
   (iii) *Otherwise,* $pk = pk_{i_c}$ *for a bit* c*. The bit* $c' = (1 - c)b + c(1 - b)$ *is computed and the oracle outputs* $\sigma \leftarrow \mathtt{Sign}(m, Q, sk_{i_{c'}})$*.*
4. `A` *outputs a bit* b'*, succeeding if* $b' = b$*.*

Moreover, we say that a scheme is linkably anonymous if every (t, ϵ, q)*-solver of the linkable anonymity game has a negligible acceptance advantage* ϵ *over* $1/2$*.*

4 Construction

In this section, we describe d-CLSAG, our implementation of a concise d-LRS construction.

Definition 10 (d-CLSAG). *The tuple* $(\mathtt{Setup}, \mathtt{KeyGen}, \mathtt{Sign}, \mathtt{Verify}, \mathtt{Link})$ *as follows is a* d*-LRS signature scheme.*

- $\mathtt{Setup} \rightarrow par$*.* `Setup` *selects a prime* p*, a group* $\mathbb{G}$ *with prime order* p*, selects a group generator* $G \in \mathbb{G}$ *uniformly at random, selects* d *cryptographic hash functions* $\mathcal{H}_0^s, \ldots, \mathcal{H}_{d-1}^s$ *with codomain* $\mathbb{F}_p$*, selects a cryptographic hash function* $\mathcal{H}^p$ *with codomain* $\mathbb{G}$*.* `Setup` *outputs the group parameter tuple and the hash functions,* $par := \left(p, \mathbb{G}, d, G, \left\{\mathcal{H}_j^s\right\}_{j=0}^{d-1}, \mathcal{H}^p\right)$*.*[4]
- $\mathtt{KeyGen} \rightarrow (\boldsymbol{sk}, \boldsymbol{pk})$*. When queried for a new key,* `KeyGen` *samples a fresh secret key and computes the associated public key:*

$$\boldsymbol{sk} = (z_0, z_1, \ldots, z_{d-1}) \leftarrow (\mathbb{F}_p^*)^d$$
$$\boldsymbol{pk} := \boldsymbol{sk} \circ \boldsymbol{G} = (Z_0, Z_1, \ldots, Z_{d-1}) \in \mathbb{G}^d$$

  where $\boldsymbol{G} = (G, \ldots, G) \in \mathbb{G}^d$*.* `KeyGen` *outputs* $(\boldsymbol{sk}, \boldsymbol{pk})$*. We say* z_0 *is the linking key, the remaining keys* $\{z_j\}_{j=1}^{d-1}$ *are the auxiliary keys, and we denote the linking key with* x*.*

[4] Note that domain separation can be used here to take one $\mathcal{H}^s$ and construct each $\mathcal{H}_j^s$ by defining $\mathcal{H}_j^s(x) := \mathcal{H}^s(j \mid\mid x)$.

– $\mathtt{Sign}(m, Q, \boldsymbol{sk}) \to \{\perp_{\mathtt{Sign}}, \sigma\}$. *`Sign` takes as input a message* $m \in \{0,1\}^*$*, a ring* $Q = (\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1})$ *for ring members* $\boldsymbol{pk}_i = (X_i, Z_{i,1}, \ldots, Z_{i,d-1}) \in \mathbb{G}^d$*, and a secret key* $\boldsymbol{sk} = (x, z_1, \ldots, z_{d-1}) \in (\mathbb{F}_p^*)^d$*.* `Sign` *does the following.*

1. *If* $Q \not\subseteq \mathbb{G}^{d\times n}$ *for some* n*,* `Sign` *outputs* $\perp_{\mathtt{Sign}}$ *and terminates.*
2. *Otherwise,* `Sign` *parses*[5] Q *to obtain each* $\boldsymbol{pk}_i$*. If the public key associated with the input* $\boldsymbol{sk}$ *is not a ring member in* Q*, then* `Sign` *outputs* $\perp_{\mathtt{Sign}}$ *and terminate.*
3. *Otherwise,* `Sign` *finds the signing index* ℓ *such that* $\boldsymbol{pk}_\ell = \boldsymbol{sk} \circ (G, \ldots, G)$*.* `Sign` *samples* $\alpha \in \mathbb{F}_p$*, samples* $\{s_i\}_{i\neq\ell} \in (\mathbb{F}_p)^{n-1}$*, and computes the points* $H_i = \mathcal{H}^p(X_i)$ *for each* i*.* `Sign` *computes the aggregation coefficients* μ_X *and* $\{\mu_j\}_{j=1}^{d-1}$*, the linking tag* $\mathfrak{T}$*, the auxiliary group elements* $\{\mathfrak{D}_j\}_{j=1}^{d-1}$*, and the aggregated public keys:*

$$\begin{aligned} \mathfrak{T} &:= xH_\ell & \{\mathfrak{D}_j\} &:= \{z_j H_\ell\} \\ \mu_X &:= \mathcal{H}_0^s(Q \mid\mid \mathfrak{T} \mid\mid \{\mathfrak{D}_j\}_{j=1}^{d-1}) & \mu_j &:= \mathcal{H}_j^s(Q \mid\mid \mathfrak{T} \mid\mid \{\mathfrak{D}_j\}_{j=1}^{d-1}) \\ W_i &:= \mu_X X_i + \sum_{j=1}^{d-1} \mu_j Z_{i,j} & \mathfrak{W} &:= \mu_X \mathfrak{T} + \sum_{j=1}^{d-1} \mu_j \mathfrak{D}_j \end{aligned}$$

and the aggregated secret key $w_\ell := \mu_X x + \sum_{j=1}^{d-1} \mu_j z_j$*. For* $i = \ell, \ell+1, \ldots, \ell-1$*, (and by identifying index* n *with index* 0*),* `Sign` *computes*

$$\begin{aligned} L_\ell &= \alpha G & R_\ell &= \alpha H_\ell & c_{\ell+1} &= \mathcal{H}_0^s(Q \mid\mid m \mid\mid L_\ell \mid\mid R_\ell) \\ L_i &= s_i G + c_i W_i & R_i &= s_i H_i + c_i \mathfrak{W} & c_{i+1} &= \mathcal{H}_0^s(Q \mid\mid m \mid\mid L_i \mid\mid R_i) \end{aligned}$$

and lastly computes $s_\ell = \alpha - c_\ell w_\ell$*.*

4. `Sign` *sets the signature* $\sigma = (c_0, s_0, s_1, \ldots, s_{n-1}, \mathfrak{T}, \{\mathfrak{D}_j\}_{j=1}^{d-1})$ *and publishes the signature* σ*.*

– $\mathtt{Verify}(m, Q, \sigma) \to \{0,1\}$. *`Verify` takes as input a message* m*, a matrix* $Q = (\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1})$*, and a signature* σ*.*

1. *If* $Q \not\subseteq \mathbb{G}^{d\times n}$ *for some* n*, or if* $\sigma \notin \mathbb{F}_p^{n'+1} \times \mathbb{G}^d$ *for some* n'*,* `Verify` *outputs* 0 *and terminates. Otherwise, if* $n' \neq n$*,* `Verify` *outputs* 0 *and terminates.*
2. `Verify` *parses*[6] $(\boldsymbol{pk}_0, \ldots, \boldsymbol{pk}_{n-1}) \leftarrow Q$ *for keys* $\boldsymbol{pk}_i \in \mathbb{G}^d$ *for* $i \in [0, n-1]$*, and parses each public key* $(X_i, Z_{i,1}, \ldots, Z_{i,d-1}) \leftarrow \boldsymbol{pk}_i$*.* `Verify` *also parses* $(c_0, s_0, \ldots, s_{n-1}, \mathfrak{T}, \mathfrak{D}_1, \ldots, \mathfrak{D}_{d-1}) \leftarrow \sigma$*.* `Verify` *computes each* $H_i = \mathcal{H}^p(X_i)$*, computes the aggregation coefficients, and computes aggregated public keys:*

$$\begin{aligned} \mu_X &:= \mathcal{H}_0^s(Q \mid\mid \mathfrak{T} \mid\mid \{\mathfrak{D}_j\}_{j=1}^{d-1}) & \mu_j &:= \mathcal{H}_j^s(Q \mid\mid \mathfrak{T} \mid\mid \{\mathfrak{D}_j\}_{j=1}^{d-1}) \\ W_i &:= \mu_X X_i + \sum_{j=1}^{d-1} \mu_j Z_{i,j} & \mathfrak{W} &:= \mu_X \mathfrak{T} + \sum_{j=1}^{d-1} \mu_j \mathfrak{D}_j \end{aligned}$$

[5] Note that this parsing always succeeds if `Sign` does not fail in the previous step.
[6] This parsing is always successful if the previous step does not terminate `Verify`.

3. *`Verify` sets* $c'_0 := c_0$ *and, for* $i = 1, 2, \ldots, n-1$*, computes the following.*

$$L_i := s_i G + c'_i W_i, \quad R_i := s_i H_i + c'_i \mathfrak{W}, \quad c'_{i+1} := \mathcal{H}^s_0 (Q \,||\, m \,||\, L_i \,||\, R_i)$$

4. *If* $c'_n = c_0$*,* `Verify` *outputs* 1*, and otherwise outputs* 0*.*

– `Link`$((m, Q, \sigma), (m', Q', \sigma')) \to \{0, 1\}$. `Link` *takes as input two message-ring-signature triples.*

1. *If* `Verify`$(m, Q, \sigma) = 0$ *or* `Verify`$(m', Q', \sigma') = 0$*,* `Link` *outputs* 0 *and terminates.*
2. *Otherwise,* `Link` *parses*[7] *the signatures to obtain the individual linking tags* $(\mathfrak{T}, \{\mathfrak{D}_j\}_j), (\mathfrak{T}', \{\mathfrak{D}'_j\}_j) \leftarrow \sigma, \sigma'$*.* `Link` *outputs* 1 *if* $\mathfrak{W} = \mathfrak{W}'$ *and* 0 *otherwise.*

This implementation has *full-key-oriented* linkability with linkability tags $\mathfrak{W}$: two signatures will link if they not only are signed using the same linking and auxiliary keys, but also the same ring. We can replace the `Link` algorithm with *single-key-oriented* linkability:

– `Link`$((m, Q, \sigma), (m', Q', \sigma')) \to \{0, 1\}$. `Link` takes as input two message-ring-signature triples.

1. If `Verify`$(m, Q, \sigma) = 0$ or `Verify`$(m', Q', \sigma') = 0$, `Link` outputs 0 and terminates.
2. Otherwise, `Link` parses[8] the signatures to obtain the individual linking tags $(\mathfrak{T}, \{\mathfrak{D}_j\}_j), (\mathfrak{T}', \{\mathfrak{D}'_j\}_j) \leftarrow \sigma, \sigma'$. `Link` outputs 1 if $\mathfrak{T} = \mathfrak{T}'$ and 0 otherwise.

5 Proofs of Security

The following lemma follows immediately from the random oracle model we use for $\mathcal{H}^s$.

Lemma 2. *For any* $Q \subseteq \mathcal{PK}$*, for any private key* $sk = (x, \{z_j\}_j) \in Q$*, the map* $sk \mapsto \mu_X x + \sum_j \mu_j z_j$ *where* μ_X*,* μ_j *are computed as in Definition 10 is a collision-resistant function.*

We prove d-CLSAG is unforgeable in Theorem 2 by showing that if some PPT algorithm produces some valid, non-oracle triple (m, Q, σ) that links to an anonymity set member in Q (malicious or otherwise), then that algorithm can be rewound to compute the discrete logarithm of that anonymity set member. This theorem is standard for Schnorr-like signatures in the programmable random oracle model.

[7] As before with `Verify`, this parsing is always successful if the previous step does not terminate `Link`.

[8] As before with `Verify`, this parsing is always successful if the previous step does not terminate `Link`.

Theorem 2 (Hardness of Discrete Logarithms of Linear Combinations Implies Unforgeability). *If a (t, ϵ, q)-solver of the unforgeability exists for the scheme of Definition 10 that makes κ' corruption oracle queries, then there exists a $(2(t+t_0)+t_1, \epsilon\left(\frac{\epsilon}{q}-\frac{1}{2^n}\right)-\mu, \lfloor\frac{q}{d}\rfloor)$-solver of the $2d\kappa'$-one-more discrete logarithm of linear combinations problem in $\mathbb{G}$ for some negligible μ and some constants t_0, t_1.*

Proof. Assume $\mathtt{A}$ is a (t, ϵ, q)-solver of the non-slanderability game of Definition 8. We wrap $\mathtt{A}$ in an algorithm $\mathtt{B}$. The algorithm $\mathtt{B}$ executes $\mathtt{A}$ in a black box, handling oracle queries for $\mathtt{A}$. Then, $\mathtt{B}$ regurgitates the output of $\mathtt{A}$ together with an index idx. This way, $\mathtt{B}$ is suitable for use in the forking lemma. We wrap $\mathcal{F}^{\mathtt{B}}$ in a master algorithm $\mathtt{M}$ that is a $(2(t+t_0)+t_1, \epsilon\left(\frac{\epsilon}{q}-\frac{1}{2^n}\right)-\mu, \lfloor\frac{q}{d}\rfloor)$-solver of the κ-one-more discrete logarithm of linear combinations problem in $\mathbb{G}$.

If $\mathtt{A}$ produces a successful forgery, each verification query of the form $c_{\ell+1} = \mathcal{H}^s(m \mid\mid Q \mid\mid R_\ell \mid\mid L_\ell)$ occurs in the transcript between $\mathtt{A}$ and the random oracle $\mathcal{H}^s$. Indeed, the signature triple produced by $\mathtt{A}$ passes verification, so each challenge $c_{\ell+1}$, whether made with oracle queries in the transcript or not, must be matched by random oracle queries made by the verifier. The prover cannot guess the output of such a query before making it except with negligible probability. Hence, if $\mathtt{A}$ outputs a valid signature, all verification challenges are computed by an actual oracle query. See [13] for a formal proof of this fact. Since all verification challenges are found through genuine oracle queries, which are well-ordered, there exists a first $\mathcal{H}^s$ query made by $\mathtt{A}$ for computing verification challenges, say $c = \mathcal{H}^s(m \mid\mid Q \mid\mid R^* \mid\mid L^*)$. This was not necessarily the first query made to $\mathcal{H}^s$ overall, though; say it was the k^{th} query. Although the ring index for this query may not have been decided when this query was first issued by $\mathtt{A}$, by the end of the transcript the ring index has been decided.

We construct $\mathtt{B}$ in the following way. We grant $\mathtt{B}$ access to the same oracles as $\mathtt{A}$. Any oracle queries made by $\mathtt{A}$ are passed along by $\mathtt{B}$ to the oracles. The responses are recorded and then passed back to $\mathtt{A}$. The algorithm $\mathtt{B}$ works by finding two indices to augment the output of $\mathtt{A}$. First, $\mathtt{B}$ finds the $\mathcal{H}^s$ query index k corresponding to the first verification challenge computed by $\mathtt{A}$ used in verifying the purported foergery. Second, $\mathtt{B}$ inspects the transcript of $\mathtt{A}$ to find the anonymity set index ℓ in the transcript such that $c = c_{\ell+1}$ and $R^* = R_\ell$ and $L^* = L_\ell$. Now $\mathtt{B}$ outputs $idx = (k, \ell)$ along with whatever $\mathtt{A}$ outputs. Clearly, $\mathtt{B}$ makes the same number of corruption oracle queries as $\mathtt{A}$.

Note $\mathtt{B}$ succeeds whenever $\mathtt{A}$ does and runs in time at most t just like $\mathtt{A}$, except for some additional time t_0 to search the transcript for idx. Since $\mathtt{B}$ is suitable for use in the forking lemma, we can use $\mathcal{F}^{\mathtt{B}}$ to construct $\mathtt{M}$.

The algorithm $\mathcal{F}^{\mathtt{B}}$ is granted oracle access to the same oracles as $\mathtt{B}$ except $\mathcal{H}^s$ and $\mathtt{SO}$. The algorithm $\mathcal{F}^{\mathtt{B}}$ simulates $\mathtt{SO}$ queries made by $\mathtt{B}$ by simple back-patching of $\mathcal{H}^s$ and simulates the other queries made to $\mathcal{H}^s$ queries made by $\mathtt{B}$ using the random tapes $\mathbf{h}, \mathbf{h}^*$ as described in Section 3.1. All other oracle queries made by $\mathtt{B}$ are passed along by $\mathcal{F}^{\mathtt{B}}$ to the actual oracles and handed back to $\mathtt{B}$.

Note that $\mathcal{F}^{\mathtt{B}}$ runs in time $2(t+t_0)$ and (with probability at least $\epsilon\left(\frac{\epsilon}{q}-\frac{1}{2^{\eta}}\right)$) outputs a pair of valid signature triples (m, Q, σ), (m', Q', σ'). The messages and anonymity sets are selected before the fork point in the transcripts, so $m = m'$ and $Q = Q'$. Moreover, $\mathcal{F}^{\mathtt{B}}$ makes at most $2\kappa'$ corruption queries. The challenges for the two transcripts are distinct since the forking algorithm outputs the failure symbol $\perp$ and terminates if the challenges for $c_{\ell+1}$ are the same in each transcript.

$$c_{\ell+1} \leftarrow \mathcal{H}_0^s(m \mid\mid Q \mid\mid R_\ell \mid\mid L_\ell) \rightarrow c'_{\ell+1}.$$

We wrap $\mathcal{F}^{\mathtt{B}}$ in an algorithm $\mathtt{M}$ that plays the κ-one-more discrete logarithm game of Definition 1 for $\kappa = 2 \cdot d \cdot \kappa'$. The algorithm $\mathtt{M}$ has corruption oracle access and runs $\mathcal{F}^{\mathtt{B}}$ in a black box, passing corruption oracle queries made by $\mathcal{F}^{\mathtt{B}}$ along. The algorithm $\mathtt{M}$ finds the following system of equations in the transcripts by inspecting the verification challenge queries

$$R_\ell = s_\ell G + c_\ell X_\ell = s'_\ell G + c'_\ell X_\ell,$$

$$L_\ell = s_\ell H_\ell + c_\ell \mathfrak{W} = s'_\ell H_\ell + c'_\ell \mathfrak{W}.$$

This $\mathtt{M}$ has enough information to compute

$$W_\ell = \frac{s_\ell - s'_\ell}{c'_\ell - c_\ell} G, \mathfrak{W} = \frac{s_\ell - s'_\ell}{c'_\ell - c_\ell} H_\ell.$$

and therefore the private key $w = \frac{s_\ell - s'_\ell}{c'_\ell - c_\ell}$. Formally, $\mathtt{M}$ operates as follows.

(1) $\mathtt{M}$ inputs the set of discrete logarithm challenge public keys $S = \left\{\widetilde{pk}_i\right\}_{i=0}^{q-1}$.
(2) $\mathtt{M}$ partitions the challenge keys into lists of d keys

$$\begin{aligned} pk_0 =& (X_0, Z_{0,1}, \ldots, Z_{0,d-1}) := (\widetilde{pk}_0, \ldots, \widetilde{pk}_{d-1}) \\ pk_1 =& (X_1, Z_{1,1}, \ldots, Z_{1,d-1}) := (\widetilde{pk}_d, \ldots, \widetilde{pk}_{2d-1}) \\ & \vdots \end{aligned}$$

obtaining $S := \{pk_i\}_{i=0}^{\lfloor \frac{q}{d} \rfloor}$.
(3) $\mathtt{M}$ picks two random tapes $\mathbf{h}, \mathbf{h}'$ to simulate oracle query responses for $\mathcal{F}^{\mathtt{B}}$.
(4) $\mathtt{M}$ executes $\mathcal{F}^{\mathtt{B}}$ in a black box, using S as input. Upon receiving a corruption query from $\mathcal{F}^{\mathtt{B}}$ on some pk_i, $\mathtt{M}$ makes a $\mathtt{CO}$ query on X_i and each $Z_{j,i}$, passing sk_i back to $\mathcal{F}^{\mathtt{B}}$. Each corruption query made by $\mathcal{F}^{\mathtt{B}}$ consists of d corruption queries made to $\mathtt{CO}$ by $\mathtt{M}$.
(5) If $\mathcal{F}^{\mathtt{B}}$ fails, or if $\mathcal{F}^{\mathtt{B}}$ succeeds with all zero coefficients μ_X and μ_j, then $\mathtt{M}$ samples a random $w \in \mathbb{F}$, samples a random subset of challenge keys $\left\{pk_j^*\right\}_j \subseteq S$, samples random coefficients $\{h_j\}_j$, outputs $w, \left\{pk_j^*\right\}_j, \{h_j\}_j$, and terminates.

(6) Otherwise, $\mathtt{M}$ obtains two signature triples with the same message and ring, $(m, Q, \sigma), (m, Q, \sigma')$ at at least one non-zero aggregation coefficient. $\mathtt{M}$ computes the challenge discrete logarithm $w = (c'_\ell - c_\ell)^{-1}(s_\ell - s'_\ell)$. $\mathtt{M}$ outputs $w, \left\{\mu_X, \{\mu_j\}_j\right\}$, and $\left\{X_\ell, \{Z_{\ell,j}\}_j\right\}$.

Denote with t_1 the time it takes for $\mathcal{M}$ to inspect the transcript, perform field operations, and process corruption queries for $\mathcal{F}^{\mathtt{B}}$. Then the algorithm $\mathtt{M}$ runs in time at most $2(t + t_0) + t_1$.

To complete the proof, consider the overall success probability and timing of $\mathtt{M}$. Since $\mathtt{A}$ is a (t, ϵ, q)-solver of the unforgeability game and these are successful signatures, there must be at least one query made to $\mathtt{SO}$ corresponding to an uncorrupted challenge key linking to these signatures. In particular, $w \cdot G = W_\ell = \mu_X X_\ell + \sum_j \mu_j Z_{\ell,j}$ for some $(X_\ell, \{Z_{\ell,j}\}_j) \in Q$. The algorithm $\mathtt{M}$ succeeds at the discrete logarithms of linear combinations game whenever $\mathcal{F}^{\mathtt{B}}$ succeeds at forking $\mathtt{B}$ and at least one coefficient μ_X and μ_j is non-zero; we denote the probability of obtaining any zero coefficients as μ. We note that μ is negligible under the random oracle model. Thus, $\mathtt{M}$ runs in time at most $2(t + t_0) + t_1$, has success probability exceeding $\epsilon\left(\frac{\epsilon}{q} - \frac{1}{2^\eta}\right) - \mu$.

The proof of Theorem 2 demonstrates that the validity of a triple implies that the aggregated private key w is the discrete logarithm of the aggregated linking tag $\mathfrak{W}$ with respect to H_ℓ and is also the discrete logarithm of the aggregated key W_ℓ with respect to G. In this way, the linking tag of a valid signature must be the linking tag corresponding to at least one ring member, except possibly with negligible probability.

Theorem 3 (No Alien Linking Tags). *If there exists a PPT algorithm $\mathtt{A}$ that produces a valid signature triple (m, Q, σ) with the scheme in Definition 10, then there exists a ring member in Q whose aggregated key W_ℓ has the same discrete logarithm w with respect to G as $\mathfrak{W}$ has with respect to H_ℓ, and this w is known to $\mathtt{A}$ (except possibly with negligible probability).*

Theorem 4. *The scheme in Definition 10 is linkable under Definition 5 and Definition 6.*

Proof. We show that valid, non-oracle signature triples from the scheme in Definition 10 satisfying the corrupted key conditions in the game of Definition 5 always link. Hence, any algorithm fails at that game except with negligible probability.

Assume that $\mathtt{A}$, while playing the game of ACST linkability from Definition 5, produces a pair of valid, non-oracle signature triples (m, Q, σ), (m^*, Q^*, σ^*) such that at most one key in $Q \cup Q^*$ is corrupted or outside of S. This algorithm can be forked and rewound as above to compute the aggregated private key used in computing each signature, say w, w^*. At most one key in $Q \cup Q^*$ is corrupted or outside of S. Since $\mathtt{A}$ has knowledge of w, then w is corrupted or outside of S, and likewise w^* is corrupted or outside of S. Since at most one key in $Q \cup Q^*$ can be corrupted or outside of S, we conclude $w = w^*$.

Since key aggregation is collision resistant and wG is the aggregated public key for some public key $(X_\ell, \{Z_{\ell,j}\}_j) \in Q \cap Q^*$, w must be aggregated from a private key $(x_\ell, \{z_{\ell,j}\}_j)$ via the collision-resistant aggregation function. In both the case of single-key-oriented linkability and full-key-oriented linkability, the linkability tags are therefore exactly equal. Hence, with probability 1, the pair of triples (m, Q, σ), (m^*, Q^*, σ^*) are linked, and `A` fails at ACST linkability except with negligible probability.

Similarly, an algorithm that outputs $q+1$ unlinked signatures can be rewound to compute $2(q+1)$ signatures from which $q+1$ aggregated keys can be computed. Moreover, if these signatures are unlinked, then the $q+1$ aggregated keys are distinct, violating q-pigeonhole linkability.

Theorem 5. *The scheme in Definition 10 is linkably anonymous.*

Proof. We assume `A` is a (t, ϵ, q)-solver of linkable anonymity that has access to the random oracles $\mathcal{H}^p$, $\mathcal{H}^s$ and a signing oracle `SO`. We construct an algorithm `B` has $\mathcal{H}^p$ oracle access, executes `A` as a black box, simulates $\mathcal{H}^s$ queries made by `A` with coin flips, simulates `SO` queries made by `A` via usual back-patching, and plays the random oracle decisional Diffie-Hellman game of Definition 2. The algorithm `B` works formally as follows.

(1) `B` takes as input $S^{(b)} = \left\{(\widetilde{pk}_i, \widetilde{\mathfrak{T}}_i^{(b)})\right\}_{i=0}^{q-1}$.
(2) `B` partitions the challenge keys into lists of d keys

$$\begin{aligned} pk_0 =& (X_0, Z_{0,1}, \ldots, Z_{0,d-1}) := (\widetilde{pk}_0, \ldots, \widetilde{pk}_{d-1}) \\ pk_1 =& (X_1, Z_{1,1}, \ldots, Z_{1,d-1}) := (\widetilde{pk}_d, \ldots, \widetilde{pk}_{2d-1}) \\ & \vdots \end{aligned}$$

obtaining $S := \{pk_i\}_{i=0}^{\lfloor \frac{q}{d} \rfloor}$.
(3) `B` sends S to `A`, who outputs a piar of indices i_0, i_1.
(4) `B` grants `SO` access to `A`, simulating oracle queries for non-challenge keys via back-patching and responding with honestly generated signatures for challenge keys.
(5) `A` outputs a bit b'; `B` outputs the same b'.

Note that if `A` succeeds with a non-negligible advantage $\left|\epsilon - \frac{1}{2}\right|$, then `B` succeeds with the same advantage. Moreover, `B` takes only some additional time to process keys and simulate `SO` queries, both of which we assume are dominated by the time it takes `A` to succeed.

6 Efficiency

Consider the space and time efficiency of Definition 10. We disregard any additional information typically broadcast alongside the signature, such as representations of the ring members.

A d-CLSAG signature with a ring size of n contains $n+1$ field elements and d group elements; signature size is $k_s(n+1)+k_p d$ where k_s describes the size of field elements and k_p describes the size of group elements.

To examine the verification time complexity, let t_s and t_p be the time complexity of evaluating the hash-to-scalar functions $\mathcal{H}^s$ and of evaluating the hash-to-point function $\mathcal{H}^p$, respectively. Let $t^{(i)}$ be the time complexity to evaluate a linear combination of i terms; using specialized algorithms for multiscalar multiplication [21, 17], such a linear combination can be evaluated much more quickly than a simple term-by-term computation. We note that it is also possible to cache multiples of group elements that are reused within verification for faster linear combination evaluation, but we do not differentiate this here. Using these, the time complexity of d-CLSAG verification is $(n+d)t_s + nt_p + 2nt^{(d+1)}$.

To compare to the efficiency of an MLSAG implementation from [15], observe that 2-CLSAG has equivalent functionality to an MLSAG signature (which is a 2-LRS). An MLSAG signature used in this way produces $2n+1$ field elements and 1 group element.

We produced a test implementation in C++ and tested signing and verification for MLSAG and 2-CLSAG on a 2.1 GHz Opteron processor. Table 1 shows the results for different ring sizes. In particular, we note that for smaller anonymity sizes, 2-CLSAG is uniformly faster than MLSAG. However, at very large ring sizes, MLSAG is faster due to additional computations involved in computing aggregation coefficients and key prefixing. Despite this eventual inefficiency, we note that the linear space requirements generally preclude the use of very large ring sizes in practice, making 2-CLSAG an efficient improvement over MLSAG in both space and time.

	Verify		Sign	
Anonymity set	MLSAG	CLSAG	MLSAG	CLSAG
2	2.4	2.0	2.3	2.7
4	4.7	4.0	4.6	4.6
8	9.5	7.8	9.4	8.5
16	18.9	15.9	18.9	16.5
32	37.8	32.3	37.8	33.0
64	75.4	67.5	75.9	68.3
128	150	147	151	148
256	301	344	303	346

Table 1. Signing and verification times (ms) for MLSAG and 2-CLSAG

References

1. Abe, M., Ohkubo, M., Suzuki, K.: 1-out-of-n signatures from a variety of keys. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 415–432. Springer (2002)

2. Au, M.H., Chow, S.S., Susilo, W., Tsang, P.P.: Short linkable ring signatures revisited. In: European Public Key Infrastructure Workshop. pp. 101–115. Springer (2006)
3. Backes, M., Döttling, N., Hanzlik, L., Kluczniak, K., Schneider, J.: Ring signatures: Logarithmic-size, no setup—from standard assumptions. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–311. Springer (2019)
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: 23rd {USENIX} Security Symposium ({USENIX} Security 14). pp. 781–796 (2014)
5. Bender, A., Katz, J., Morselli, R.: Ring signatures: Stronger definitions, and constructions without random oracles. In: Theory of Cryptography Conference. pp. 60–79. Springer (2006)
6. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 315–334. IEEE (2018)
7. Fujisaki, E.: Sub-linear size traceable ring signatures without random oracles. In: Cryptographers' Track at the RSA Conference. pp. 393–415. Springer (2011)
8. Fujisaki, E., Suzuki, K.: Traceable ring signature. In: International Workshop on Public Key Cryptography. pp. 181–200. Springer (2007)
9. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 305–326. Springer (2016)
10. Groth, J., Kohlweiss, M., Maller, M., Meiklejohn, S., Miers, I.: Updatable and universal common reference strings with applications to zk-SNARKs. In: Annual International Cryptology Conference. pp. 698–728. Springer (2018)
11. Gu, K., Wu, N.: Constant size traceable ring signature scheme without random oracles. IACR Cryptology ePrint Archive **2018**, 288 (2018)
12. Hoffmann, M., Klooß, M., Rupp, A.: Efficient zero-knowledge arguments in the discrete log setting, revisited. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2093–2110 (2019)
13. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups. In: Australasian Conference on Information Security and Privacy. pp. 325–335. Springer (2004)
14. Möser, M., Soska, K., Heilman, E., Lee, K., Heffan, H., Srivastava, S., Hogan, K., Hennessey, J., Miller, A., Narayanan, A., et al.: An empirical analysis of traceability in the Monero blockchain. Proceedings on Privacy Enhancing Technologies **2018**(3), 143–163 (2018)
15. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
16. Paillier, P., Vergnaud, D.: Discrete-log-based signatures may not be equivalent to discrete log. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 1–20. Springer (2005)
17. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
18. Quesnelle, J.: On the linkability of Zcash transactions. arXiv preprint arXiv:1712.01210 (2017)
19. Rivest, R.L., Shamir, A., Tauman, Y.: How to leak a secret. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 552–565. Springer (2001)

20. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
21. Straus, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**(806-808), 16 (1964)
22. Tsang, P.P., Wei, V.K., Chan, T.K., Au, M.H., Liu, J.K., Wong, D.S.: Separable linkable threshold ring signatures. In: International Conference on Cryptology in India. pp. 384–398. Springer (2004)
23. Van Saberhagen, N.: CryptoNote v 2.0 (2013)
24. Yang, X., Wu, W., Liu, J.K., Chen, X.: Lightweight anonymous authentication for ad hoc group: A ring signature approach. In: International Conference on Provable Security. pp. 215–226. Springer (2015)
25. Zhang, F., Kim, K.: ID-based blind signature and ring signature from pairings. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 533–547. Springer (2002)