Compact Linkable Ring Signatures and Applications

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 introduce an efficient linkable ring multisignature construction, *compact linkable spontaneous anonymous group* (CLSAG) signatures. These admit d-dimensional keys with a specified linking coordinate but do not have signature sizes directly proportional to d. Compared to existing constructions used for signer-ambiguous confidential transactions without trusted setup, CLSAG signatures are smaller and more efficient in terms of both proving and verification time. CLSAG signatures also satisfy some rigorous security definitions: unforgeability depends upon the k-OMDL hardness assumption, linkability depends on unforgeability as well as the collision resistance of key aggregation, and signer-ambiguity depends on the DDH assumption. We demonstrate an application for CLSAG signatures for use in transacting multiple assets over ring signature-based transaction protocols.

1 Introduction

First introduced in [20] in the RSA setting and in [13] in the discrete logarithm setting, ring signatures permit a non-interactive signature on behalf of a set of public keys rather than a single public key. Ring signatures see myriad applications ranging from lightweight anonymous authentication as in [25] to transaction protocols like Monero in [16] and CryptoNote in [24]. 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 *signer-ambiguous* by nature because the verifier does not learn additional information from the signature about which key is the signer. We stress that methods of practical analysis such as those of [15, 19] can exploit metadata in real-life applications of signer-ambiguous protocols to reduce ambiguity.

Group signature constructions preceding [20, 13] require some degree of interactivity, a fixed set of participants, a trusted group manager (or some other trusted setup), or hardness assumptions not based on the discrete logarithm problem. Since [20], ring signatures have enjoyed many improvements, expansions, and modifications. An incomplete list of examples includes: ring signatures are constructed in the bilinear pairing setting in [26], key structures are generalized in [1], security definitions are improved in [4], signature size is improved in [7, 11], and traceability is introduced in [8].

Linkable ring signatures were first introduced in [13]; in the context of distributed ledgers like Monero, linkable ring signatures are the basis for signer-ambiguous 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 [21].

The key images 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-spend attempts is ensured using key images as described in [24]. More recent work of [16] extends the approach of [13] to enable a signer-ambiguous confidential transaction model with *ad hoc* ring member selection. In [16], 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 [16] 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, 3, 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 [5, 12] show both improvements to the trust requirement as well as improvements in efficiency.

1.1 Our contribution

We first introduce a formal definition of d-dimensional linkable ring signatures. We present an implementation of a new signature scheme, d-CLSAG signatures. These have d-dimensional keys and are *compact* linkable spontaneous anonymous group signatures in the sense that signature size scales with the *sum* of ring size and the dimension d. Equivalent MLSAG signatures of [16] produce signatures that scale with the product of ring size and d rather than the sum. Size efficiency comes from an aggregation of keys across components, similar to the approach from [14, 18], resulting in d-CLSAG signatures that are about half the size of [16], and can be generated and verified more quickly.

We present linkable ring signatures and some examples in Section 2. We present a compact d-LRS scheme we call d-CLSAG in Section 3 and measure efficiency. We present an informal description of a transaction protocol for confidentially transacting multiple asset types simultaneously, which we call a *multi-asset ring confidential transaction* or MARCT in Section 4.2. We reserve discussion of security for the appendix, where we make a new definition for unforgeability in linkable ring signatures that takes into account both insider corruption and forgeries from partially-corrupted rings. We prove that d-CLSAG signatures are unforgeable up to the hardness of the k-one-more discrete logarithm (k-OMDL) problem under this definition in the random oracle model in Appendix A. We also make some comments on signer ambiguity and linkability in Appendix B.

1.2 Notation

We denote algorithms with typefont majuscule English letters like `A`, `B`, or `O`, or typefont names like `Setup`, `KeyGen`, and so on. For any prime p, denote the field with p elements as $\mathbb{F}_p := \mathbb{Z}/p\mathbb{Z}$, and denote the non-zero elements as $\mathbb{F}_p^*$.

Group parameters are denoted as a tuple $(p, \mathbb{G}, d, G)$ where $\mathbb{G}$ is an elliptic curve group with prime order p, d is a dimension, and G is a generator of $\mathbb{G}$. We denote integers, bits, indices, and scalars in $\mathbb{F}_p$ with minuscule English letters x, y, z, b, c, i, j, k, etc. and we denote group elements with majuscule English letters, G, X, W, and so on. We use miniscule Greek letters like σ to describe signatures and majuscule calligraphic Latin letters like $\mathfrak{T}$ when describing key images.

For these group parameters, the secret key space is $\mathbb{F}_p^*$ and the public key space is $\mathbb{G}$. For any non-zero secret key $sk \in \mathbb{F}_p^*$, the corresponding public key pk is computed from the generator G through exponentiation in $\mathbb{G}$ as usual. However, we use notation for a module over the field $\mathbb{Z}_p$ to maintain consistency with, say [24] and [16].

We denote column vectors in boldface, e.g. $(x_1, \ldots, x_d)^\top = \mathbf{x}$, and matrices in underlined boldface, e.g. $((x_{1,1}, x_{1,2}, \ldots, x_{1,n}), \ldots, (x_{d,1}, \ldots, x_{d,n})) = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = \underline{\mathbf{x}}$ is a $d \times n$ matrix. We denote the Hadamard product of two vectors with $\circ$, so for any $\mathbf{x} = (x_1, x_2, \ldots, x_d) = (x_i)_{i=1}^d$, and for any $\mathbf{y} = (y_i)_{i=1}^d$, we denote the sequence $(x_i \cdot y_i)_{i=1}^d$ with $\mathbf{x} \circ \mathbf{y}$. We denote bitwise concatenation with the symbol $||$.

We distinguish oracles with calligraphic font, e.g. $\mathcal{CO}$ denotes a corruption oracle, $\mathcal{SO}$ denotes a signing oracle. If the codomain of a random oracle is the field of scalars $\mathbb{F}_p$, we denote this $\mathcal{H}^s$ (hash-to-scalar). If the codomain is $\mathbb{G}$, we denote this $\mathcal{H}^p$ (hash-to-point).

2 Linkable ring multisignatures

In this section, we recall linkable ring signature (LRS) schemes, definitions of correctness in verification and linkability, and we provide two examples.

2.1 LRS schemes

Definition 1 (LRS). *A linkable ring signature scheme is a tuple* (`Setup`, `KeyGen`, `Sign`, `Verify`, `Link`) *satisfying the following.*

- `Setup`$(1^\lambda) \to par$. `Setup` *takes as input a security parameter* 1^λ*, produces some public parameters par.*
- `KeyGen`$(1^\lambda, par) \to (sk, pk)$. `KeyGen` *takes as input a security parameter* 1^λ *and public parameters par.* `KeyGen` *produces as output a private-public keypair* (sk, pk)*.*
- `Sign`$\left(1^\lambda, par, (m, \boldsymbol{pk}, sk)\right) \to \left\{\bot_{\texttt{Sign}}, \sigma\right\}$. `Sign` *takes as input a security parameter* 1^λ*, public parameters par, an arbitrary message* $m \in \{0,1\}^*$*, an ad hoc ring of public keys* $\boldsymbol{pk} = \{pk_1, \ldots, pk_n\}$*, and a secret key sk.* `Sign`

produces as output either a distinguished failure symbol $out = \bot_{\mathtt{Sign}}$ *or a signature* $out = \sigma$*.*

- $\mathtt{Verify}\left(1^\lambda, par, (m, \boldsymbol{pk}, \sigma)\right) \to \{0,1\}$*.* $\mathtt{Verify}$ *takes as input a security parameter* 1^λ*, public parameters par, a message* m*, a ring of public keys* $\boldsymbol{pk}$*, and a signature* σ*.* $\mathtt{Verify}$ *produces as output a bit* $b \in \{0,1\}$*;* 0 *indicates the signature is not verified, and* 1 *indicates the signature is verified.*
- $\mathtt{Link}\left(1^\lambda, par, (m, \boldsymbol{pk}, \sigma), (m', \boldsymbol{pk}', \sigma')\right)$*.* $\mathtt{Link}$ *takes as input a security parameter* 1^λ*, public parameters par, and a pair of tuples* $(m, \boldsymbol{pk}, \sigma), (m', \boldsymbol{pk}', \sigma')$ *for messages* m, m'*, rings* $\boldsymbol{pk}$*,* $\boldsymbol{pk}'$*, and ring signatures* σ, σ'*.* $\mathtt{Link}$ *produces as output a bit* $b \in \{0,1\}$*;* 0 *indicates the signatures are not linked or invalid, and* 1 *indicates the signatures are valid and linked.*[3]

In the sequel, we implicitly assume all algorithms in a LRS take 1^λ as input, and all algorithms (except `Setup`) takes *par* as input. We suppress this notation in the sequel. For example, we write $par \leftarrow \mathtt{Setup}$ instead of $par \leftarrow \mathtt{Setup}(1^\lambda)$ and $\sigma \leftarrow \mathtt{Sign}(m, \mathbf{pk}, sk)$ instead of $\sigma \leftarrow \mathtt{Sign}(1^\lambda, par, m, \mathbf{pk}, sk)$.

Note the dimension of keys in Definition 1 is not specified. If the keys from `KeyGen` have dimension $d > 1$, we instead say the LRS is a d-LRS and use the vector notation introduced in Section 1.2 representing keys in boldface (*e.g.* $\mathbf{pk}$ instead of pk) and rings in underlined boldface ($\underline{\mathbf{pk}}$ instead of $\mathbf{pk}$). We always assume the first coordinate of a secret key is the *linking coordinate*, and linkability depends only on the linking coordinate of the signing key vector. For ring confidential transactions, this is important since only one such entry corresponds to an output public key for double-spend detection purposes and the rest of the keys could have been adversarially generated.

Definition 2 (Correctly verified). *Let* m *be any message,* sk *be any secret key with corresponding public key* pk*, and let* $\boldsymbol{pk}$ *be a multiset of public keys* $\boldsymbol{pk} = \{pk_1, \ldots, pk_n\}$*. If there exists an index* $1 \leq \ell \leq n$ *satisfying* $pk = pk_\ell$*, then*

$$\mathtt{Verify}(m, \boldsymbol{pk}, \mathtt{Sign}(m, \boldsymbol{pk}, sk)) = 1.$$

Soundness in verification is the property of unforgeability. See Appendix A.

Definition 3 (Correctly linkable). *Let* m, m' *be any messages,* sk, sk' *be any secret keys with corresponding public keys* pk*,* pk'*. Let* $\boldsymbol{pk}$*,* $\boldsymbol{pk}'$ *be multisets of public keys with indices* ℓ, ℓ'*, respectively, such that* $\boldsymbol{pk}_\ell = pk$ *and* $\boldsymbol{pk}'_{\ell'} = pk'$*. Let* sk, sk' *be the respective private keys for* pk, pk' *with respective linking coordinates* sk_0 *and* sk'_0*. Correct linkability means being both positively and negatively linkable, in the following sense.*

- ***Positively linkable:*** *If the linking coordinates satisfy* $sk_0 = sk'_0$*, then*

  $$Link((m, \boldsymbol{pk}, \mathtt{Sign}(m, \boldsymbol{pk}, sk)), (m', \boldsymbol{pk}', \mathtt{Sign}(m', \boldsymbol{pk}', sk'))) = 1$$

  except with negligible probability.

[3] This is the opposite of the conventions in, say, [23], which outputs 0 to indicate two signatures are linked (*i.e.* rejected) and outputs 1 to indicate two signatures are not linked (*i.e.* accepted).

- ***Negatively linkable:*** *If the linking coordinates satisfy* $sk_0 \neq sk'_0$*, then*

$$\mathit{Link}((m, \boldsymbol{pk}, \boldsymbol{\mathtt{Sign}}(m, \boldsymbol{pk}, sk)), (m', \boldsymbol{pk}', \boldsymbol{\mathtt{Sign}}(m', \boldsymbol{pk}', sk'))) = 0$$

  except with negligible probability.

Soundness in linkability means informally that an adversarial algorithm can only produce tuples $y = (m, \mathbf{pk}, \sigma)$ and $y' = (m', \mathbf{pk}', \sigma')$ such that $\mathtt{Link}(y, y') = 1$ by computing σ, σ' from $\mathtt{Sign}$ using secret keys sk, sk' with matching linking coordinates. See Appendix B.

2.2 Examples

In the following examples, all verifiers must list the keys in **pk** in an agreed-upon order for the above verification to work; either they should agree upon lexicographic or some other ordering. Linking occurs merely by comparing key images: two valid signatures with the same key image were signed with the same secret key (and, in transaction applications, would signal an attempt to double-spend funds).

Example 1. The signature scheme of [13] is an LRS; we present the key image variant from [24] with a key image appropriate for linking signatures key-by-key. The signature scheme originally described in [13] signs a message m with a ring of keys $\mathbf{pk} = \{pk_1, \ldots, pk_n\}$ and a secret index-key pair (ℓ, sk) corresponding to some pk_ℓ, using the key image $\mathfrak{I} := sk \cdot \mathcal{H}^p(\mathbf{pk})$.

Unfortunately, this key image is unsuitable for transaction applications, as changing ring members will change the key image, allowing the same key to sign twice. For use in a transaction protocol and following [24], we modify this key image from that of [13] to be $\mathfrak{I} := sk \cdot \mathcal{H}^p(pk_\ell)$, which is independent of the non-signing ring members. This allows these key images to be used for double-spend protection, as discussed previously.

$\mathtt{Setup}$ always deterministically sets $d := 1$ so we only use the linking key and there are no auxiliary keys. $\mathtt{Setup}$ selects a generator $G \in \mathbb{G}$ to be a group generator for the group parameters $(p, \mathbb{G}, d, G)$, two cryptographic hash functions $\mathcal{H}^s : \{0,1\}^* \to \mathbb{F}_p$ and $\mathcal{H}^p : \{0,1\}^* \to \mathbb{G}$. $\mathtt{Setup}$ outputs $par = (p, \mathbb{G}, d, G, \mathcal{H}^s, \mathcal{H}^p)$.

$\mathtt{KeyGen}$ produces as output a secret key $sk \in \mathbb{F}_p^*$ and the corresponding public key $\mathtt{pk} = sk \cdot G \in \mathbb{G}$.

$\mathtt{Sign}$ takes as input a (non-zero) private key $sk \in \mathbb{F}_p^*$, a message m, a ring $\mathbf{pk} = \{pk_1, \ldots, pk_n\}$, and produces as output either a distinguished failure symbol $\bot_{\mathtt{Sign}}$ or a signature σ, computed as follows. First, the signer samples $\alpha, s_{\ell+1}, s_{\ell+2}, \ldots, s_{\ell-1} \in \mathbb{F}_p$ at random. Next, the signer computes basepoints $H_i = \mathcal{H}^p(pk_i)$ and the key image $\mathfrak{I} = sk \cdot \mathcal{H}^p(pk_\ell)$, the first challenge

$$c_{\ell+1} = \mathcal{H}^s(\mathbf{pk} \,||\, m \,||\, \alpha G \,||\, \alpha H_\ell)$$

and each subsequent challenge

$$c_{i+1} = \mathcal{H}^s(\mathbf{pk} \,||\, m \,||\, s_i G + c_i \cdot pk_i \,||\, s_i H_i + c_i \mathfrak{I})$$

for $i = \ell+1, \ldots, \ell-1$, naturally identifying index 1 with index $n+1$. The signer finishes by computing $s_\ell = \alpha - c_\ell \cdot sk$ and publishing the signature σ where $\sigma = (c_1, s_1, \ldots, s_n, \mathfrak{T})$.

`Verify` takes as input a message m, a ring $\mathbf{pk}$, and a purported signature σ'. `Verify` parses $(c_1, s_1, \ldots, s_n, \mathfrak{T}) \leftarrow \sigma$. If $c_1 \notin \mathbb{F}_p$ or any $s_i \notin \mathbb{F}_p$ or $\mathfrak{T} \notin \mathbb{G}$, then `Verify` outputs 0. Otherwise, the verifier computes $H_i := \mathcal{H}^p(X_i)$ for each ring member, sets $c_1' := c_1$, and computes the challenges

$$c_{i+1}' = \mathcal{H}^s(\mathbf{pk} \mid\mid m \mid\mid s_i G + c_i' \cdot pk_i \mid\mid s_i H_i + c_i' \mathfrak{T})$$

for $i = 1, 2, \ldots, n$. The verifier outputs 1 when $c_{n+1}' = c_1$ and 0 otherwise.

`Link` checks the validity of both signatures. If both are valid, `Link` parses the key images $\mathfrak{T}$ and $\mathfrak{T}'$. If either are not in $\mathbb{G}$, the linker outputs 0 and terminates. Otherwise, the linker outputs 1 when $\mathfrak{T} = \mathfrak{T}'$ and 0 otherwise.

Remark 1. The key image modification in Example 1 is due to the basepoint of the key image $\mathfrak{T}$. As noted in [13], variations on key image formats may be desirable. How or whether the security properties of LSAG signatures are retained in practical use given more flexible key image formats, while interesting, is beyond the scope of this work.

Example 2. This example extends the LRS of the previous example to a so-called MLSAG scheme [16], which is a 2-LRS for use in signer-ambiguous confidential transactions. `Setup` always deterministically sets $d := 2$, so we use one linking key and one auxiliary key, but otherwise works as before.

`KeyGen` produces as output a secret key $\mathbf{sk} = (x, z) \in \mathbb{F}_p^* \times \mathbb{F}_p^*$ and the corresponding public key $\mathbf{pk} := \mathbf{sk} \circ \mathbf{G} = (xG, zG) \in \mathbb{G}^2$ where $\mathbf{G} = (G, G) \in \mathbb{G}^2$. The key x is the linking key. The auxiliary key z is a blinder that opens a Pedersen commitment to zero demonstrating transaction balance in ring confidential transactions in the style of [16].

`Sign` takes as input a (non-zero) private key $\mathbf{sk} \in \mathbb{F}_p^* \times \mathbb{F}_p^*$, a message m, a ring $\underline{\mathbf{pk}} \in \mathbb{G}^{2\times n}$, and produces as output either a distinguished failure symbol $\bot_{\texttt{Sign}}$ or a signature σ, computed as follows. First, the signer samples rows of signature data $\alpha, \alpha', s_{\ell+1}, s_{\ell+1}', s_{\ell+2}, s_{\ell+2}', \ldots, s_{\ell-1}, s_{\ell-1}' \in \mathbb{F}_p$ at random. Next, the signer computes the basepoints $H_i = \mathcal{H}^p(X_i)$ from the linking keys X_i of each ring member $\mathbf{pk}_i = (X_i, Z_i)$. The linking key image $\mathfrak{T} = xH_\ell$ is computed from the linking key. An auxiliary key image with the same basepoint but discrete logarithm z_ℓ is computed, $\mathfrak{D} = z_\ell H_\ell$. The signer computes the challenges

$$c_{\ell+1} = \mathcal{H}^s(\mathbf{pk} \mid\mid m \mid\mid \alpha G \mid\mid \alpha H_\ell \mid\mid \alpha' G \mid\mid \alpha' H_\ell)$$

and

$$c_{i+1} = \mathcal{H}^s(\mathbf{pk} \mid\mid m \mid\mid s_i G + c_i X_i \mid\mid s_i H_i + c_i \mathfrak{T} \mid\mid s_i' G + c_i Z_i \mid\mid s_i' H_i + c_i \mathfrak{D}).$$

The values $s_\ell = \alpha - c_\ell x_\ell$ and $s_\ell' = \alpha' - c_\ell z_\ell$ are computed. The signature is set $\sigma := (c_1, s_1, s_1', \ldots, s_n, s_n', \mathfrak{T}, \mathfrak{D})$ and is output.

`Verify` takes as input a message m, a ring $\underline{\mathbf{pk}}$, and a signature σ. The verifier parses $(c_1, s_1, s'_1, \dots, s_n, s'_n, \mathfrak{T}, \mathfrak{D}) \leftarrow \sigma$. If this is not possible, or $c_1 \notin \mathbb{F}_p$, or any s_i or $s'_i \notin \mathbb{F}_p$, or if $\mathfrak{T} \notin \mathbb{G}$, then the verifier outputs 0. Otherwise, the verifier parses $(\mathbf{pk}_1, \dots, \mathbf{pk}_{n'}) \leftarrow \underline{\mathbf{pk}}$. If any $\mathbf{pk}_i \notin \mathbb{G}^2$, or if $n \neq n'$, the verifier outputs 0. Otherwise, the verifier parses each $\mathbf{pk}_i$ as (X_i, Z_i), computes each $H_i = \mathcal{H}^p(X_i)$, sets $c'_1 := c_1$, and computes the challenges

$$c'_{i+1} = \mathcal{H}^s(\mathbf{pk} \mid\mid m \mid\mid s_iG + c'_iX_i \mid\mid s_iH_i + c'_i\mathfrak{T} \mid\mid s'_iG + c'_iZ_i \mid\mid s'_iH_i + c'_i\mathfrak{D})$$

for $i = 1, 2, \dots, n$. The verifier outputs 1 when $c'_{n+1} = c_1$ and 0 otherwise.

Lastly, `Link` works as before, by checking key images.

3 A compact d-LRS scheme

We informally say that a d-LRS scheme is *compact* or *concise* if signature sizes are not directly proportional to d. We present a multisignature variant of LSAG signatures that is asymptotically more compact than the previous examples. We call this scheme d-CLSAG. We take a look at efficiency for usage in ring confidential transactions by comparing 2-CLSAG signatures against equivalent MLSAG signatures.

3.1 Implementation

Definition 4 (d-CLSAG). *The tuple (`Setup`, `KeyGen`, `Sign`, `Verify`, `Link`) satisfying the following is a d-CLSAG signature scheme.*

- *`Setup` → 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 $\mathcal{H}^s_0, \dots, \mathcal{H}^s_{d-1}$ 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* $:= \left(p, \mathbb{G}, d, G, \left\{\mathcal{H}^s_j\right\}_{j=0}^{d-1}, \mathcal{H}^p\right)$.[4]
- *`KeyGen` → $(\boldsymbol{sk}, \boldsymbol{pk})$. When queried for a new key, `KeyGen` samples a fresh secret key and computes the associated public key:*

$$\begin{aligned} \boldsymbol{sk} =& (x, z_1, \dots, z_{d-1}) \leftarrow (\mathbb{F}_p^*)^d \\ \boldsymbol{pk} :=& \boldsymbol{sk} \circ \boldsymbol{G} = (X, Z_1, \dots, Z_{d-1}) \in \mathbb{G}^d \end{aligned}$$

  where $\boldsymbol{G} = (G, \dots, G) \in \mathbb{G}^d$. `KeyGen` outputs $(\boldsymbol{sk}, \boldsymbol{pk})$. We say x is the linking key and the remaining keys $\{z_j\}$ are the auxiliary keys.
- *`Sign` $\left(m, \underline{\boldsymbol{pk}}, \boldsymbol{sk}\right) \to \left\{\bot_{Sign}, \sigma\right\}$. `Sign` takes as input a message $m \in \{0,1\}^*$, a ring $\underline{\boldsymbol{pk}} = (\boldsymbol{pk}_1, \dots, \boldsymbol{pk}_n)$ for ring members $\boldsymbol{pk}_i = (X_i, Z_{i,1}, \dots, Z_{i,d-1}) \in \mathbb{G}^d$, and a secret key $\boldsymbol{sk} = (x, z_1, \dots, z_{d-1}) \in (\mathbb{F}_p^*)^d$. `Sign` does the following.*

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

1. *If* $\underline{\boldsymbol{pk}} \notin \mathbb{G}^{d\times n}$ *for some* n, ***Sign*** *outputs* $\perp_{\texttt{Sign}}$ *and terminates.*
2. *Otherwise,* ***Sign*** *parses*[5] $\underline{\boldsymbol{pk}}$ *to obtain each* $\boldsymbol{pk}_i$. *If the public key associated with the input* $\boldsymbol{sk}$ *is not a ring member in* $\underline{\boldsymbol{pk}}$, *then* ***Sign*** *outputs* $\perp_{\texttt{Sign}}$ *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 key image* $\mathfrak{T}$, *the auxiliary key images* $\{\mathfrak{D}_j\}_{j=1}^{d-1}$, *the aggregated public keys and their key images:*

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

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* $n+1$ *with index* 1*),* ***Sign*** *computes*

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

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

4. ***Sign*** *sets the signature* $\sigma = (c_1, s_1, \ldots, s_n, \mathfrak{T}, \{\mathfrak{D}_j\}_{j=1}^{d-1})$ *and publishes the signature* σ.

- ***Verify*** $(m, \underline{\boldsymbol{pk}}, \sigma) \to \{0,1\}$. ***Verify*** *takes as input a message* m, *a matrix* $\underline{\boldsymbol{pk}} = (\boldsymbol{pk}_1, \ldots, \boldsymbol{pk}_n)$, *and a signature* σ.
  1. *If* $\underline{\boldsymbol{pk}} \notin \mathbb{G}^{d\times n}$ *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] $(\boldsymbol{pk}_1, \ldots, \boldsymbol{pk}_n) \leftarrow \underline{\boldsymbol{pk}}$ *for keys* $\boldsymbol{pk}_i \in \mathbb{G}^d$ *for* $i = 1, \ldots, n$, *and parses each public key* $(X_i, Z_{i,1}, \ldots, Z_{i,d-1}) \leftarrow \boldsymbol{pk}_i$. ***Verify*** *also parses* $(c_1, s_1, \ldots, s_n, \mathfrak{T}, \mathfrak{D}_1, \ldots, \mathfrak{D}_{d-1}) \leftarrow \sigma$. ***Verify*** *computes each* $H_i = \mathcal{H}^p(X_i)$, *computes the aggregation coefficients, and computes aggregated public keys and their images:*

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

  3. ***Verify*** *sets* $c_1' := c_1$ *and, for* $i = 1, 2, \ldots, n-1$, *computes the following.*

$$L_i := s_i G + c_i' W_i, \quad R_i := s_i H_i + c_i' \mathfrak{W}, \quad c_{i+1}' := \mathcal{H}_0^s\left(\underline{\boldsymbol{pk}} \mid\mid m \mid\mid L_i \mid\mid R_i\right)$$

[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`.

4. *If $c'_{n+1} = c_1$, **Verify** outputs 1, and otherwise outputs 0.*

- ***Link*** $\big((m, \underline{\mathbf{pk}}, \sigma), (m', \underline{\mathbf{pk}}', \sigma')\big) \to \{0,1\}$. *Link takes as input two message-ring-signature triples.*
  1. *If **Verify**$(m, \underline{\mathbf{pk}}, \sigma) = 0$ or **Verify**$(m', \underline{\mathbf{pk}}', \sigma') = 0$, **Link** outputs 0 and terminates.*
  2. *Otherwise, **Link** parses[7] the signatures to obtain*

$$(c_1, s_1, \ldots, s_n, \mathfrak{T}, \mathfrak{D}_1, \ldots, \mathfrak{D}_{d-1}) \leftarrow \sigma \text{ and}$$
$$(c'_1, s'_1, \ldots, s'_n, \mathfrak{T}', \mathfrak{D}'_1, \ldots, \mathfrak{D}'_{d-1}) \leftarrow \sigma'.$$

__Link__ outputs 1 if $\mathfrak{T} = \mathfrak{T}'$ and 0 otherwise.

Later, we discuss the security of this implementation.

3.2 Efficiency

Consider the space and time efficiency of Definition 4. We disregard additional information broadcast alongside the signature, such as descriptions of the ring members.

A d-CLSAG signature with a ring size of n contains $n+1$ scalars and d group elements, so this scheme is compact; signature size is $k_s(n+1) + k_p d$ where k_s describes the size of scalar field elements and k_p describes the size of points on the curve.

To examine the verification time complexity, instead let k_s and k_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 $k^{(i)}$ be the time complexity to evaluate a scalar-point linear combination of i terms; using specialized algorithms like Straus [22] or Pippenger [17] multiexponentiation (or others, based on i), 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 points 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 $(n+d)k_s + nk_p + 2nk^{(d+1)}$.

To compare to the efficiency of an MLSAG implementation, note that 2-CLSAG has equivalent functionality to an MLSAG signature (which is a 2-LRS). An MLSAG signature used in this way requires $2n+1$ scalars 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, 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.

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

	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

4 Applications

4.1 Single-asset ring confidential transactions

As mentioned above, it is possible to use 2-CLSAG as a replacement for MLSAG signatures in ledger applications (like Monero) for equivalent functionality. For example, Monero currently uses MLSAG signatures for two different transaction types: full and simple.

Full transactions are only used when spending a single input. They leverage the fact that in a balanced transaction, the difference between input and output commitments is a commitment to zero; the signer can therefore use such differences as the second component of key vectors in the signature and sign using the known secret key at the signing index.

Simple transactions are used when spending multiple inputs. Each spent input requires a separate signature, as a naive extension of full transactions presents an index linking issue. The signer first generates auxiliary commitments for each spent input using the same value but a different blinder. This means it is possible to use the difference between input and auxiliary commitments as a commitment to zero for the purpose of signing. By choosing all blinders at random except one, the signer can construct the auxiliary commitments such that the difference between auxiliary and output commitments is zero, proving balance.

Both transaction types can be constructed with 2-CLSAG signatures since linkability is not considered for the second key component used in the transaction protocols.

4.2 Multi-asset ring confidential transactions (MARCTs)

It is possible to use a straightforward d-CLSAG construction to accommodate transactions spending $d-1$ *types* or *colors* of assets separately within the same transaction and signature. To do so, transaction outputs are extended to have a separate commitment to each asset type value. When spending an output, either a full or simple transaction (discussed above) is used; we simply copy the method used to compute commitment public keys in the signature to additional dimensions of the d-CLSAG signature, using only the commitments for a particular asset type in each. This separation ensures that the transaction balances in each asset type separately, while taking advantage of the scaling benefits of d-CLSAG compared to the equivalent MLSAG signature construction.

4.3 Informal description of MARCTs with unmixable colors

We informally describe an example of MARCTs with two unmixable colors using 3-CLSAG. Let $(\texttt{Prove}, \texttt{Ver})$ be a zero-knowledge sound range proving scheme, such as that described in [6], and let $(\texttt{Com}, \texttt{Open})$ be a Pedersen commitment scheme such that $\texttt{Com}(r, v) = rG + vG'$.

For the sake of this example, we define a public trading key to be a tuple (X, C, D, P) where $X, C, D \in \mathbb{G}$, C and D are amount commitments and P is a batched range proof from $\texttt{Prove}$ covering the values of both C and D. Here, C and D play the role of the Z_j points, and P is additional data required for the transaction protocol.

We define a transaction key to be a tuple $(m, \mathbf{Q}, \mathbf{O}, (f_C, f_D), \sigma, aux)$ where $\mathbf{Q}$ is a ring of n public trading keys $\mathbf{Q} = \{(X_i, C_i, D_i, P_i)\}_{i=1}^{n}$, $\mathbf{O}$ is a set of n' output public trading keys $\mathbf{O} = \{(X'_i, C'_i, D'_i, P'_i)\}_{i=1}^{n'}$, f_C is a plaintext list of fees to be paid from C, f_D is a plaintext list of fees to be paid from D, and σ is a 3-CLSAG signature. We say a transaction key is valid if the following are satisfied:

- every input ring member $(X_i, C_i, D_i, P_i) \in \mathbf{Q}$ has a valid range proof P_i so $\texttt{Ver}(P_i) = 1$; and
- every output range proof P'_k is valid so $\texttt{Ver}(P'_k) = 1$; and
- for the ring
$$\mathbf{pk} = \begin{pmatrix} X_1 & X_2 & \cdots & X_n \\ Z_1 & Z_2 & \cdots & Z_n \\ Z'_1 & Z'_2 & \cdots & Z'_n \end{pmatrix}$$
where each $Z_i = C_i - f_C G' - \sum_k C'_k$ and each $Z'_i = D_i - f_D G' - \sum_k D'_k$, $\texttt{Verify}(m, \underline{\mathbf{pk}}, \sigma) = 1$.

This 3-CLSAG signature demonstrates knowledge of the discrete logarithm of some x_ℓ, knowledge of the opening information for the input and output commitments, and that the transaction amounts balance with the fees f_C and f_D. After all, when the amounts in C_ℓ and D_ℓ balance with the fees f_C and f_D together with the sum of the amounts in each C'_k and D'_k, and when the signer knows all the openers for all these commitments, Z_ℓ and Z'_ℓ can be regarded as usual public keys with basepoint G whose secret key is known by the signer.

Unfortunately, this model does not allow for exchanging amounts of the first color with amounts of the second color. In the next section, we present another toy model that allows for transferring between colors at a fixed/pegged exchange rate.

Informal description of MARCTs with a fixed exchange rate We modify the previous example to use a 2-CLSAG. Consider the canonical example of colored currency with a fixed peg between two colors: dollars and pennies with a 100 : 1 exchange rate between them. Define an exchange rate by determining a constant ξ and some constants γ_C, γ_D on $\left\{1, 2, \ldots, 2^{\xi-1}\right\}$, (in this example, $\gamma_C = 1$ and $\gamma_D = 100$).

As before, we define a public trading key to be a tuple (X, C, D, P) and a transaction key to be a tuple $(m, \mathbf{Q}, \mathbf{O}, (f_C, f_D), \sigma, aux)$. We interpret these identically as in the previous step, except using 2-CLSAG signatures instead of 3-CLSAG signatures, and of course we compute them differently. We say a simple transaction key is valid if the following are satisfied:

- every input ring member $(X_i, C_i, D_i, P_i) \in \mathbf{Q}$ has a valid range proof P_i so $\texttt{Ver}(P_i) = 1$; and
- every output range proof P'_k is valid so $\texttt{Ver}(P'_k) = 1$; and
- for the modified ring
$$\mathbf{pk} = \begin{pmatrix} X_1 & X_2 & \cdots & X_n \\ Z_1 & Z_2 & \cdots & Z_n \end{pmatrix}$$
where each $Z_i = \gamma_C(C_i - f_C G' - \sum_k C'_k) + \gamma_D(D_i - f_D G' - \sum_k D'_k)$, the signature σ passes the 2-CLSAG verification, $\texttt{Verify}(m, \underline{\mathbf{pk}}, \sigma) = 1$.

Unlike the previous example, this example allows for the fixed exchange rate between colors determined by γ_C and γ_D.

A Security: Unforgeability

We prove the unforgeability of our implementation in Definition 4. A good forgery game should grant the adversary the power to persuade otherwise honest users to hand over their keys (modeled by a corruption oracle) or sign adversarially selected messages with adversarially selected rings (modeled by a signing oracle), and our algorithm is based on the random oracle model. Of course, any oracle queries made by a forgery algorithm $\texttt{A}$ being run in a black box must be handled by whatever algorithm is executing $\texttt{A}$, so we describe how these are simulated in Section A.2.

A.1 Hardness Assumption

Unforgeability comes from the k-OMDL hardness assumption.

Definition 5 (k-OMDL problem). *Let $k \in \mathbb{N}$. We say a PPT algorithm $\texttt{A}$ is a (t, ϵ)-solver of the k-OMDL problem if, within time at most t and with probability at least ϵ, $\texttt{A}$ can succeed at the following.*

1. *The challenger uses group parameters $(p, \mathbb{G}, G)$ and picks a collection of group elements $G_1, G_2, \ldots, G_k, G_{k+1} \in \mathbb{G}$ (the targets) uniformly at random from $\mathbb{G}$. The challenger sends the group parameters and $\{G_i\}$ to $\texttt{A}$.*

2. *$\mathtt{A}$ is granted access to a corruption oracle $\mathcal{CO}$ that takes as input some G_i sent to $\mathtt{A}$ and produces as output the discrete logarithm of G_i with respect to G, i.e. some $x_i \in \mathbb{F}_p^*$ such that $G_i = x_i G$.*
3. *$\mathtt{A}$ produces as output a sequence of $k+1$ scalars $x_1, \ldots, x_{k+1} \in \mathbb{F}_p^*$, counting as a success if:*
   - *(i) for each x_i, there exists some index $1 \leq j(i) \leq k+1$ such that $G_{j(i)} = x_i G$ and*
   - *(ii) $\mathtt{A}$ made no more than k queries to $\mathcal{CO}$.*

A.2 The Oracles

Random oracle queries made by $\mathtt{A}$ are handled by looking to a random tape $\mathbf{h}$ available to the simulator to generate hashes for new queries made to the oracle $\mathcal{H}^s$. These are stored in a hash table for consistency in later queries. We assume whatever algorithm is executing $\mathtt{A}$ in a black box has pre-generated a set of private-public key pairs for use in simulating a corruption oracle $\mathcal{CO}$ for $\mathtt{A}$. This leaves only the signing oracle, which is simulated through back-patching in the following way.

The simulator reserves the next random oracle query on the random tape $\mathbf{h}$ to select $c_{\ell+1}$ for back-patching. The random signature data α and $\{s_i\}_{i \neq \ell}$ are sampled as usual, and each challenge c_{i+1} for $i \neq \ell$ is simulated from $\mathbf{h}$ as described before and stored in a hash table. After all challenges are computed, the simulator compute the group points L_ℓ, R_ℓ, and back-patches their hash table to force $\mathcal{H}^s(\underline{\mathbf{pk}} \mid\mid m \mid\mid L_\ell \mid\mid R_\ell) \leftarrow c_{\ell+1}$.

A.3 Defining forgeries

We use a modified version of the definition of existential forgery with insider corruption for a ring signature by Bender, Katz, and Morselli [4]. In contrast with the definition of unforgeability with respect to insider corruption in [4], our modification allows for a forger to succeed at the forgery game with partially corrupted rings.

Let $n(-)$ be a positive polynomial. Let $\mathcal{H}^s : \{0,1\}^* \to \mathbb{F}_p$ be modeled as a random oracle. Let $\mathcal{CO}$ be a corruption oracle that takes as input a public key $\mathbf{pk}$ from the list of challenge keys and produces as output the corresponding secret key $\mathbf{sk}$ and the key image $\mathfrak{I}$. Let $\mathcal{SO}$ be a signing oracle that takes as input some $(m, \underline{\mathbf{pk}}', \ell)$ such that $\underline{\mathbf{pk}'}$ is a matrix of challenge key vectors (*i.e.* each column is in $\mathbf{pk}$) and produces as output a signature σ such that $\mathtt{Verify}(m, \underline{\mathbf{pk}}', \sigma) = 1$ and such that the key image $\mathfrak{I} \in \sigma$ is the key image for the ℓ^{th} key in $\underline{\mathbf{pk}}'$.

Definition 6 (Existential unforgeability of linkable ring signatures with respect to insider corruption).

We say a PPT algorithm $\mathtt{A}$ is a $(t, \epsilon, q_h, q_c, q_s, n(-))$-forger of a linkable ring signature scheme if, within time at most t and with at most q_h oracle queries to $\mathcal{H}^s$, at most q_c oracle queries to $\mathcal{CO}$, and at most q_s queries to $\mathcal{SO}$, $\mathtt{A}$ can succeed at the following game with probability at least ϵ.

1. *Challenge keys* $\{(\boldsymbol{sk}_i, \boldsymbol{pk}_i)\}_{i=1}^{n(\lambda)} \leftarrow \texttt{KeyGen}(1^{\lambda})$ *are selected and the public keys* $\underline{\boldsymbol{pk}} = \{\boldsymbol{pk}_i\}_{i=1}^{n(\lambda)}$ *are sent to* `A`*.*
2. `A` *is granted access to a corruption oracle* $\mathcal{CO}$*, random oracle* $\mathcal{H}^s$*, and the signing oracle* $\mathcal{SO}$*.*
3. `A` *outputs a message* m*, a ring of at most* n *public keys* $\underline{\boldsymbol{pk}}'$*, and a signature* σ*. This output is a success if*
   (a) σ *is not output from any query made to* $\mathcal{SO}$*; and*
   (b) *the key image* $\mathfrak{I}$ *does not correspond to a corrupted ring member; and*
   (c) $\texttt{Verify}(m, \underline{\boldsymbol{pk}}', \sigma) = 1$*.*

A forgery challenger can play the forgery game of Definition 6 with `A` in a black box, can simulate $\mathcal{SO}$ and $\mathcal{CO}$, and can check whether the purported forgeries by `A` are successful; the challenger simply keeps a history of all oracle queries, and computes key images of corrupted keys to check.

This definition allows the attacker to attempt a successful forgery by re-using messages, rings, or indices from previous $\mathcal{SO}$ queries (but not by reusing queries *per totum*). It also allows key images that are unrelated to the ring members and it allows key images found in previous $\mathcal{SO}$ queries. Hence, proving the implementation of Definition 4 unforgeable under this definition implies these attempts will fail. A forger gains no advantage by re-using messages (or rings or indices) that have already been used in $\mathcal{SO}$. The forger gains no advantage by including key images that are unrelated to the ring members, or by using the key images from $\mathcal{SO}$ queries.

Moreover, consider some $(m, \underline{\mathbf{pk}}', \sigma)$ output by an alleged forger `A`. If the scheme is unforgeable, one of the above conditions must fail. If the signature passes verification, one of the first two conditions must fail, so any valid signature must violate one of these first two conditions. If the first condition is violated, the forger is merely attempting to pass off an oracle signature or some previously computed signature as their own. If the second condition is violated, then $\mathfrak{I}$ corresponds to a corrupted ring member (in which case `A` knows the key) or corresponds to none of the ring members.

That is to say: if `A` produces a valid signature, then either `A` knows the key or the key image does not correspond to a ring member.

Moreover, presume that `A` produces a message m, a ring $\underline{\mathbf{pk}}$, and a signature σ that passes verification, and yet such that the key image $\mathfrak{I} \in \sigma$ does not correspond to any ring member. Under the random oracle model, `A` cannot simultaneously satisfy the verification equations

$$\begin{aligned}
c_2 =& \mathcal{H}_0^s(\underline{\mathbf{pk}} \mid\mid m \mid\mid s_1 G + c_1 W_1 \mid\mid s_1 H_1 + c_1 \mathfrak{W}) \\
&\vdots \\
c_n =& \mathcal{H}_0^s(\underline{\mathbf{pk}} \mid\mid m \mid\mid s_{n-1} G + c_{n-1} W_{n-1} \mid\mid s_{n-1} H_{n-1} + c_{n-1} \mathfrak{W}) \\
c_1 =& \mathcal{H}_0^s(\underline{\mathbf{pk}} \mid\mid m \mid\mid s_n G + c_n W_n \mid\mid s_n H_n + c_n \mathfrak{W})
\end{aligned}$$

except with negligible probability, because the discrete logarithm of $\mathfrak{W}$ cannot be written as $x_i \mathcal{H}^p(X_i)$ for any i. We conclude that `A` must have known the discrete logarithm of the signing key to produce this signature.

Remark 2. Note that if the corruption oracle merely acted by computing arbitrary discrete logarithms, then an adversary could do the following: take some target **pk** from the challenge set, apply a permutation to the coordinates of **pk**, pass the permuted key through $\mathcal{CO}$, obtain the discrete logarithm of the first (signing) key of **pk**, compute the key image for this signing key, and lastly produce a signature using `Sign`. For example, to find the discrete logarithm of the linking key in $\mathbf{pk} = (A, B)$, the adversary may query $\mathcal{CO}$ with (B, A), bypassing our definition.

Such a signature would pass validation and not be described as a forgery according to our definition. Our definition avoids this problem by requiring the corruption oracle only be queried with challenge keys. This has the added benefit that it is possible to simulate the corruption oracle for a the black box execution of `A`.

A.4 The Forking Lemma

To prove that the existence of a forger implies that of a k-OMDL solver, we use the forking lemma. In the following, we presume the bit length η is used to describe group elements in $\mathbb{G}$ and scalars in $\mathbb{F}_p$, *i.e.* $\eta = O(|p|)$.

Lemma 1 (General Forking Lemma). *Let $q, \eta \geq 1$. Let $\mathtt{A}$ be any PPT algorithm which takes as input some $x_{\mathtt{A}} = (x, \boldsymbol{h})$ where $\boldsymbol{h} = (h_1, \ldots, h_q)$ is 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, all choices $\boldsymbol{h}$). Let $\mathcal{F} = \mathcal{F}^{\mathtt{A}}$ be the forking algorithm for $\mathtt{A}$ described below. 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 describe the general forking algorithm below, and refer the reader to [2] for a proof of this lemma, which demonstrates that if executing some `A` has non-negligible acceptance probability, then forking `A` does as well. Since all queries before the $(j^*)^{th}$ query are identical in both transcripts, the input of the $(j^*)^{th}$ query is also identical. Since oracle queries $h'_{j^*}, h'_{j^*+1}, \ldots$ are newly sampled upon receiving the first output from `A`, the queries $h_{j^*} \neq h'_{j^*}$ except with negligible probability. All subsequent computations in the signature that are common in both transcripts will have the same results only with negligible probability.

Using a forger in the Forking Lemma Note that a forger according to Definition 6 is not directly compatible with the forking lemma; the output is some $y = (m, \mathbf{pk}, \sigma)$ and no *idx* is included. However, without loss of generality, we can execute `A` in a black box that extracts from the transcript of `A` some $idx = (i^*, j^*)$ for $j^* = j(i^*)$ in the following way.

For each query for any c_{i+1} that appears in the successful forgery, there exists a corresponding index $j(i)$ that satisfies $c_{i+1} = h_{j(i)}$. The black box executing $\mathtt{A}$ looks at the transcript and extracts the index pair $\mathit{idx} = (i^*, j^*)$ that indicates where in the random oracle transcript we can find the very first oracle query made by $\mathtt{A}$ to $\mathcal{H}^s$ for some challenge used in signature verification

$$c_{i^*+1} = \mathcal{H}_0^s\left(\underline{\mathbf{pk}} \mid\mid m \mid\mid L_{i^*} \mid\mid R_{i^*}\right)$$

used in the successful forgery. Such a pair (i^*, j^*) can be found by merely inspecting the transcript, so the algorithm wrapping $\mathtt{A}$ can output (idx, y) without harming its advantage.

Without loss of generality, we can assume that $\mathtt{A}$ has been appropriately wrapped so is compatible with the forking lemma without impacting its advantage.

Forking the forger at this query preserves the input to the query $(\underline{\mathbf{pk}}, m, L_{i^*}, R_{i^*})$ but does not preserve the challenge c_{i^*+1}. Moreover, each c_{i+1} used in the signature verification is computed by $\mathtt{A}$ by querying $\mathcal{H}^s$ in the transcript of $\mathtt{A}$ leading to a successful forgery; the outputs of these queries cannot be guessed except with negligible probability, and so the oracle must have actually been queried (see, for example, [13]). Of course, although the index i^* may not have been decided by $\mathtt{A}$ when the query was made, but the index i^* is assigned before the end of the transcript.

That is to say, a forked forger presents two forgeries with the same ring, message, and idx except with negligible probability, with the pair of points L_{i^*}, R_{i^*} common in both transcripts.

A forking algorithm $\mathcal{F}^{\mathtt{A}}$ satisfying Lemma 1 works in the following way.

1. $\mathcal{F}$ takes as input some x and $\mathcal{F}$ selects the random tape for $\mathtt{A}$.
2. $\mathcal{F}$ selects some $\mathbf{h} = (h_1, \ldots, h_q)$ at random by flipping coins, and $\mathcal{F}$ executes $y_{\mathtt{A}} \leftarrow \mathtt{A}(x, \mathbf{h})$.
3. If $y_{\mathtt{A}} = \perp_{\mathtt{A}}$, then $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $y_{\mathtt{A}} = (\mathit{idx}, y)$ for some $\mathit{idx} = (i^*, j^*)$ and some output y and $\mathcal{F}$ selects new oracle queries $h'_{j^*}, h'_{j^*+1}, \ldots, h'_q$, and glues the hash challenges together:
$$\mathbf{h}' = (h_1, \ldots, h_{j^*-1}, h'_{j^*}, h'_{j^*+1}, \ldots, h'_q)$$
4. If $h_{j^*} = h'_{j^*}$, then $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $h_{j^*} \neq h'_{j^*}$ and $\mathcal{F}$ executes $y'_{\mathtt{A}} \leftarrow \mathtt{A}(x, \mathbf{h}')$.
5. If $y'_{\mathtt{A}} = \perp_{\mathtt{A}}$, then $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $y'_{\mathtt{A}} = (\mathit{idx}', y')$. If $\mathit{idx} \neq \mathit{idx}'$, $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $\mathcal{F}$ outputs the tuple $(\mathit{idx}, y, \mathbf{h}, y', \mathbf{h}')$.

We note that $\mathcal{F}^{\mathtt{A}}$ executed in a black box can be fed the oracle queries $\mathbf{h}$ and $\mathbf{h}'$ and so these can be assumed to be output as well without loss of generality or impacting acceptance probability. Of course, if $\mathtt{A}$ runs in time at most t, $\mathcal{F}^{\mathtt{A}}$ runs in time at most $2t + s$ where s denotes the time it takes $\mathcal{F}^{\mathtt{A}}$ to select the random tape for $\mathtt{A}$, select the oracle query sequences $\mathbf{h}$ and $\mathbf{h}'$, perform lookups in hash tables to maintain oracle query consistency, and outputting the results. These times are all negligible, so $\mathcal{F}^{\mathtt{A}}$ runs in $O(2t)$ time.

A.5 Playing k-OMDL

We now construct a master algorithm $\mathtt{M}$ that plays the k-OMDL game for $k = 2d \cdot q_c + d - 1$ that operates in the following way. Recall that $\mathtt{M}$ is granted access to up to k queries at a discrete logarithm oracle.

1. $\mathtt{M}$ receives group parameters $(p, \mathbb{G}, G)$ and target group elements $G_1, \ldots, G_{k+1}$ from the k-OMDL challenger.
2. $\mathtt{M}$ blocks $(G_1, \ldots, G_{k+1})$ into d-length blocks and reserve them for public keys using the following equations.

$$\begin{aligned} \mathbf{pk}_1 =& (G_1, \ldots, G_d) \\ \mathbf{pk}_2 =& (G_{d+1}, \ldots, G_{2d}) \\ &\vdots \\ \mathbf{pk}_{2q_c+1} =& (G_{2q_c d+1}, \ldots, G_{k+1}) \end{aligned}$$

   $\mathtt{M}$ uses $\{\mathbf{pk}_i\}_{i=1}^{2q_c+1}$ as input for $\mathcal{F}^{\mathtt{A}}$, responding to corruption oracle queries made by $\mathcal{F}^{\mathtt{A}}$ for a key $\mathbf{pk}_i$ by querying $\mathcal{CO}$ directly with each coordinate and responding with the result. Denote $X_i := G_{(i-1)d+1}$ and $Z_{i,j} := G_{(i-1)d+1+j}$ for consistency with our earlier notation.
3. If $\mathcal{F}^{\mathtt{A}}$ outputs $\perp$, so does $\mathtt{M}$ and $\mathtt{M}$ terminates. Otherwise, $\mathcal{F}^{\mathtt{A}}$ succeeds executing $\mathtt{A}$ twice, each time taking no more than q_c queries to corrupt d-dimensional keys, resulting in no more than $2 \cdot d \cdot q_c$ queries to the discrete logarithm oracle $\mathcal{CO}$. $\mathcal{F}^{\mathtt{A}}$ produces $(idx, y, \mathbf{h}, y', \mathbf{h}')$ where $y = (m, \underline{\mathbf{pk}}, \sigma)$ and $y' = (m, \underline{\mathbf{pk}}, \sigma')$ are forgeries using oracle queries $\mathbf{h}$ and $\mathbf{h}'$, respectively, and $idx = (i^*, j^*)$ as described in Section A.4.
   The messages and rings are identical in these forgeries because they must have been selected before the first challenge query, except with negligible probability. So $\mathtt{M}$ can parse

$$\begin{aligned} y =& (m, \underline{\mathbf{pk}}, \sigma) & \sigma =& (c_1, s_1, \ldots, s_n, \mathfrak{T}, \{\mathfrak{D}_j\}_j) \\ y' =& (m, \underline{\mathbf{pk}}, \sigma') & \sigma' =& (c_1', s_1', \ldots, s_n', \mathfrak{T}', \{\mathfrak{D}_j'\}_j) \end{aligned}$$

   except with negligible probability (in which case $\mathtt{M}$ outputs $\perp_{\mathtt{M}}$ and terminates).
4. In the transcript of $\mathcal{F}^{\mathtt{A}}$, $\mathtt{M}$ can find $c_{i^*+1} = \mathbf{h}_{j^*}$ and in the second transcript $c_{i^*+1} = \mathbf{h}_{j^*}'$ for some $\mathbf{h}_{j^*} \neq \mathbf{h}_{j^*}'$, except with negligible probability (in which case $\mathtt{M}$ outputs $\perp_{\mathtt{M}}$ and terminates).
5. $\mathtt{M}$ parses the random oracle transcript to find the query yielding the signature challenge $c_{i^*+1} \leftarrow \mathcal{H}^s(\underline{\mathbf{pk}} \mid\mid m \mid\mid L_{i^*} \mid\mid R_{i^*})$. Both transcripts match until this line, and the oracle responses stored in c_{i^*+1} in these transcripts don't match (i.e. $\mathbf{h}_{j^*} \neq \mathbf{h}_{j^*}'$ except with negligible probability). The algorithm $\mathtt{M}$ parses the first transcript to look for constants s_{i^*}, c_{i^*} and the group point $W = \mu_X X_{i^*} + \sum_j \mu_j Z_{i^*,j}$ such that $L_{i^*} = s_{i^*} G + c_{i^*} W$. From the second transcript, and with the same L_{i^*} and W, $\mathtt{M}$ parses to find constants s_{i^*}', c_{i^*}' such that $L_{i^*} = s_{i^*}' G + c_{i^*}' W$.

6. If $\mu_X = 0$ then `M` outputs $\perp$ and terminates. Otherwise, `M` computes the discrete logarithm
$$w = \frac{s'_{i^*} - s_{i^*}}{c_{i^*} - c'_{i^*}}$$
without querying $\mathcal{CO}$.
7. `M` makes up to $d-1$ queries to $\mathcal{CO}$ to find the discrete logarithms of the elements of any $(d-1)$-subset of $\{X_{i^*}, Z_{i^*,1}, \ldots, Z_{i^*,d-1}\}$.
8. `M` uses w to solve for the final discrete logarithm.
9. `M` outputs the $2{\cdot}d{\cdot}q_c$ corruptions queries and the d-vector $(x_{i^*}, z_{i^*,1}, \ldots, z_{i^*,d-1})$, totaling $k+1$ discrete logarithms.

Note that if `M` does not terminate and output $\perp$, then `M` makes up to $2 \cdot d \cdot q_c$ queries to $\mathcal{CO}$ for $\mathcal{F}^{\mathtt{A}}$ and makes an an additional $d-1$ queries to $\mathcal{CO}$, and yet produces as output $d \cdot (q_c + 1) > d \cdot q_c + d - 1$ discrete logarithms, *i.e.* `M` successfully plays the k-OMDL game for $k = 2 \cdot d \cdot q_c + d - 1$. Furthermore, if `M` already corrupted these discrete logarithms, even fewer queries could be made, tightening k and making `M` a more powerful solver.

Also note that, as previously mentioned, since the map $(x, z_1, \ldots, z_{d-1}) \mapsto w$ is collision resistant, `M` can skip steps and guess w in step 7 only with negligible success.

Recall that $\mathcal{F}^{\mathtt{A}}$ takes time $O(2t)$ to execute. `M` executes $\mathtt{F}^{\mathtt{A}}$ and then performs parsing of transcripts and some additional computations, so `M` takes time $O(2t + s')$ for some s' due to parsing and processing transcript data.

The additional time s' is due to:

- Lookup time in a hash table for each $\mathcal{CO}$ query, each random oracle query, and each extraction of a value from the random tapes $\mathbf{h}$, $\mathbf{h}'$ by $\mathcal{F}^{\mathtt{A}}$ throughout the transcript.
- Parsing and constructing keys in step 2.
- Parsing purported forgeries in step 3.
- Parsing transcripts and computing multi-exponentiations to verify equations in step 5.
- Computing the discrete logarithm w using field arithmetic in Step 6.
- Computing the discrete logarithm of the challenge key using field arithmetic in step 8.
- Outputting the results.

All of these are negligible, so `M` also takes time $O(2t)$.

A.6 Proof

All that remains to prove the unforgeability of the d-CLSAG scheme from Section 3.1 is to show that `M` as described has a non-negligible acceptance probability.

Theorem 1. *Let $d, q_h, q_c, q_s \in \mathbb{N}$ and let $(p, \mathbb{G}, G)$ be some group parameters. If a $(t, \epsilon, q_h, q_c, q_s, n(-))$-forger of the d-CLSAG implementation in Section 3.1 exists then a $(2t, \epsilon')$-solver of the k-OMDL problem in $\mathbb{G}$ exists for $k = d{\cdot}q_c{+}d{-}1$ where $\epsilon' \geq \epsilon\left(\frac{\epsilon}{q_c} - \frac{1}{2^\eta}\right) - p^{-1}$.*

Proof. Let where d, q_h, q_c, q_s satisfy the hypotheses and let $\mathtt{A}$ be a $(t, \epsilon, q_h, q_c, q_s, n)$-forger of the d-CLSAG scheme of Section 3.1, let $\mathcal{F}^{\mathtt{A}}$ be the forking algorithm for $\mathtt{A}$, and let $\mathtt{M}$ be the master algorithm previously described. $\mathtt{M}$ terminates and outputs $\perp$ in steps 3 and 6 only; otherwise, $\mathtt{M}$ succeeds at the k-OMDL game. Hence, if E_3 is the event that $\mathtt{M}$ outputs $\perp$ in step 3 and E_6 is the event that $\mathtt{M}$ outputs $\perp$ in step 6, then E_3, E_6 are disjoint and the acceptance probability for $\mathtt{M}$ is $1 - \mathbb{P}(E_3 \cup E_6) = 1 - \mathbb{P}(E_3) - \mathbb{P}(E_6)$. The probability that $\mathtt{M}$ outputs $\perp$ in step 6 is the probability that the hashed coefficient $\mu_X = 0$, which occurs with probability p^{-1}. $\mathtt{M}$ outputs $\perp$ in step 3 when $\mathcal{F}^{\mathtt{A}}$ produces $\perp$, but the forking lemma gives us that the acceptance probability of $\mathcal{F}^{\mathtt{A}}$ is bounded from below by $\epsilon\left(\frac{\epsilon}{q_c} - \frac{1}{2^\eta}\right)$. Hence, $\mathtt{M}$ succeeds with probability at least $\left(\epsilon\left(\frac{\epsilon}{q_c} - \frac{1}{2^\eta}\right) - p^{-1}\right)$.

Note that, as a corollary, we can conclude that if a signature passes verification, then the key image corresponds to one of the ring members except with negligible probability. Indeed, these signatures soundly prove knowledge of the discrete logarithm of the signing key as well as equality of the discrete logarithm with the signing key and the key image, so constructing a signature that convinces the verifier of one but not the other succeeds with at most negligible probability.

B Security: Definitions other than unforgeability

B.1 Linkability

Correctness in linkability for the implementation of Definition 4 is easily verified. We consider soundness in linkability. Soundness relies upon the collision resistance of the map from a secret key to the key image $\mathfrak{I}$ and unforgeability (which, in turn, relies on the k-OMDL hardness assumption).

Soundly linkable: We say Definition 4 is *soundly linkable* or *non-slanderable* if it is infeasible for an algorithm to produce messages m, m', signatures σ, σ', rings $\underline{\mathbf{pk}}$, $\underline{\mathbf{pk}}'$, and indices ℓ, ℓ' such that the key image $\mathfrak{I} \in \sigma$ corresponds to the ℓ^{th} member of $\underline{\mathbf{pk}}$, the key image $\mathfrak{I}' \in \sigma'$ corresponds to the $(\ell')^{th}$ member of $\underline{\mathbf{pk}}'$, the linking coordinate of $\mathbf{pk}_\ell$ does not equal the linking coordinate of $\mathbf{pk}'_{\ell'}$, and $\mathtt{Link}((m, \underline{\mathbf{pk}}, \sigma), (m', \underline{\mathbf{pk}}', \sigma')) = 1$.

Assume an algorithm attempts to slander and generates a pair of tuples $(m, \underline{\mathbf{pk}}, \sigma)$ and $(m', \underline{\mathbf{pk}}', \sigma')$ such that $\mathtt{Link}((m, \underline{\mathbf{pk}}, \sigma), (m', \underline{\mathbf{pk}}', \sigma')) = 1$. Since $\mathtt{Link}$ outputs 1, both signatures pass verification and the key image $\mathfrak{I}$ is the same in both signatures, $\mathfrak{I} = \mathfrak{I}'$. Since both signatures pass verification and Definition 4 is unforgeable, the slanderer has, except with negligible probability, computed the signatures $\mathtt{Sign}$, *i.e.* $\sigma \leftarrow \mathtt{Sign}(m, \underline{\mathbf{pk}}, \mathbf{sk})$ and $\sigma' \leftarrow \mathtt{Sign}(m', \underline{\mathbf{pk}}', \mathbf{sk}')$.

Since the map from $\mathbf{sk}$ to the key image $\mathfrak{I}$ is collision-resistant, this implies that the linking coordinates are equal, $\mathbf{sk}_0 = \mathbf{sk}'_0$, except with negligible probability, so no one is slandered.

B.2 Signer ambiguity

Hardness We show our scheme is computationally signer-ambiguous if the following DDH game is hard in $\mathbb{G}$.

Definition 7 (Decisional Diffie-Hellman). *Let* $\mathtt{A}$ *be any PPT algorithm,* $(p, \mathbb{G}, G)$ *and let* $n \in \mathbb{N}$.

1. *The challenger selects* $(r_{i,1}, r_{i,2}, r_{i,3}) \in (\mathbb{F}_p)^3$ *uniformly and independently for* $i = 1, \ldots, n$. *The challenger computes the public keys* $R_{i,1} = r_{i,1}G$, $R_{i,2} = r_{i,2}G$, $R_{i,3}^{(0)} = r_{i,1}r_{i,2}G$, $R_{i,3}^{(1)} = r_{i,3}G$.
2. *The challenger selects a bit* b *independently and uniformly from* $\{0, 1\}$ *and sends* $\left\{(R_{i,1}, R_{i,2}, R_{i,3}^{(b)})\right\}_{i=1}^{n}$ *to* $\mathtt{A}$.
3. $\mathtt{A}$ *outputs a bit* b', *succeeding if* $b = b'$.

Note any algorithm can flip a coin and guess correctly half the time. We say the advantage of $\mathtt{A}$ *is the difference between the probability of success for* $\mathtt{A}$ *and* $1/2$. *If* $\mathtt{A}$ *can solve this with an advantage at least* ϵ *in time at most* t, *we say* $\mathtt{A}$ *is a* (t, ϵ)*-solver of the DDH problem in* $\mathbb{G}$.

We note that due to the random self-reducibility of the DDH game, in the sense that solving one instance of the problem has complexity no worse than solving a sequence of random instances of the problem, the classic DDH game is no harder than Definition 7.

Definition 8 (Signer Ambiguity). *We say* $\mathtt{A}$ *is a* (t, ϵ, n_1, n_2)*-solver of the signer ambiguity game if it can succeed with non-negligible advantage at the following game.*

1. *The challenger selects* n_1 *secret keys* $\{\boldsymbol{sk}_i\} \subseteq (\mathbb{F}_p^*)^d$, *computes the corresponding public keys* $\boldsymbol{pk}_i = \boldsymbol{sk}_i \circ \boldsymbol{G}$, *and sends* $\{\boldsymbol{pk}_i\}$ *to* $\mathtt{A}$.
2. $\mathtt{A}$ *outputs an arbitrary message* m *and a ring of* n_2 *distinct members* $\underline{\boldsymbol{pk}}' \subseteq \{\boldsymbol{pk}_i\}$.
3. *The challenger selects a ring index* $1 \leq \ell \leq n_2$ *uniformly at random, retrieves the private key* $\boldsymbol{sk}$, *and sends a valid signature* $\sigma \leftarrow \mathtt{Sign}(m, \underline{\boldsymbol{pk}}', \boldsymbol{sk})$ *to* $\mathtt{A}$.
4. $\mathtt{A}$ *outputs an index* ℓ', *succeeding if* $\ell = \ell'$.

Note that a simulator in place of $\mathtt{A}$ *without any input can guess any index from* $\{1, \ldots, n_2\}$ *with coin flips, succeeding with probability at least* $1/n_2$. *We define the advantage of* $\mathtt{A}$ *as the difference in acceptance probability and* $1/n_2$.

Note that if the secret index ℓ is leaked in the signer ambiguity game, this is equivalent to leaking information about the bit b used in the DDH game. Also note that the game could be generalized to allow $\mathtt{A}$ repeated and adaptive access to a signing oracle, just so long as so-called *ring intersection* attacks are taken into account when defining the advantage of $\mathtt{A}$. However, such a generalization is equivalent to ours.

Proof of Signer Ambiguity If $\mathbb{G}$ satisfies the DDH hardness assumption, then the distribution of the triple of group elements (r_1G, r_2G, r_3G) is computationally indistinguishable from the triple (r_1G, r_2G, r_1r_2G), where the r_i are independently uniform on $\mathbb{F}_p$. If $\mathcal{H}^p : \{0,1\}^* \to \mathbb{G}$ is modeled as a random oracle with output that is independent of its input, the distribution of a tuple (r_1G, r_2G, r_3G) is identical to the distribution of $(r_1G, \mathcal{H}^p(r_1G), r_3G)$ where r_1, r_3 are independently uniform on $\mathbb{F}_p$. Hence, under the random oracle model and assuming $\mathbb{G}$ is DDH-hard, the distribution of triples $(r_1G, \mathcal{H}^p(r_1G), r_1\mathcal{H}^p(r_1G))$ where r_1 is uniformly random from $\mathbb{F}_p$ is computationally indistinguishable from the distribution of triples $(r_1G, \mathcal{H}^p(r_1G), r_3G)$ where r_1, r_3 are uniformly random from $\mathbb{F}_p$.

Now note that a solver of the signer ambiguity game is given X_i and $\mathcal{H}^p(X_i)$ for each ring member and the key image $\mathfrak{T} = x_\ell \mathcal{H}^p(X_\ell)$. The solver with a non-negligible advantage at guessing ℓ has a non-negligible advantage in distinguishing whether a given triple $(X_i, \mathcal{H}^p(X_i), \mathfrak{T})$ satisfies $\mathfrak{T} = x_i\mathcal{H}^p(X_i)$ or not.

Theorem 2. *If a (t, ϵ, n_1, n_2)-solver of the signer-ambiguity game exists, there exists a $(t, \frac{\epsilon}{2})$-solver of the DDH game.*

Proof. We assume `A` is an algorithm that can succeed at the game in Definition 8 with non-negligible advantage. We construct a master algorithm `M` that plays the game in Definition 7 by executing `A` in a black box such that `M` plays the role of the challenger in Definition 8.

`M` receives a set of DDH challenge tuples $\left\{(R_{i,1}, R_{i,2}, R_{i,3}^{(b)})\right\}_{i=1}^{n}$. `M` keeps two internal hash tables to maintain consistency between oracle queries made to $\mathcal{H}^p$ and $\mathcal{H}^s$, and flips coins to determine hash outcomes except as specified below. `M` sets $X_i := R_{i,1}$, backpatches the key image basepoints $\mathcal{H}^p(X_i) := R_{i,2}$, and sets the purported key images $\mathfrak{T}_i := R_{i,3}^{(b)}$. The algorithm selects $Z_{i,j}$ at random and sets $\mathbf{pk}_i := (X_i, Z_{i,1}, \ldots, Z_{i,d-1})$. The algorithm `M` then operates in the following way:

1. `M` sends the public keys $\underline{\mathbf{pk}} = \{\mathbf{pk}_i\}_{i=1}^{n}$ to `A`.
2. `A` outputs a message m and a ring $\underline{\mathbf{pk}}'$.
3. If $\underline{\mathbf{pk}}' \not\subseteq \underline{\mathbf{pk}}$, `M` outputs $\perp$ and terminates. Otherwise, the algorithm `M` can find a one-to-one correspondence between ring indices in $\underline{\mathbf{pk}}'$ and key indices in $\underline{\mathbf{pk}}$, so that for each ring index $1 \leq \ell \leq n_2$ in $\underline{\mathbf{pk}}'$, there exists some key index $1 \leq i(\ell) \leq n_1$ in $\underline{\mathbf{pk}}$ such that the ring member is $X_{i(\ell)} = R_{i(\ell),1}$, has key image basepoint $\mathcal{H}^p(X_{i(\ell)}) = R_{i(\ell),2}$, and has key image $R_{i(\ell),3}^{b}$.
4. `M` simulates a signature in the following way.
   (a) `M` selects a random index $1 \leq \ell \leq n_2$, selects a random scalar $c_{\ell+1} \in \mathbb{F}_p$, and selects random scalars $s_1, s_2, \ldots, s_n \in \mathbb{F}_p$.

(b) For $i = \ell+1, \ell+2, \dots, n-1, n, 1, 2, \dots, \ell-1$, `M` computes

$$\begin{aligned} L_i :=& s_i G + c_i \left(\mu_X X_i + \sum_j \mu_j Z_{i,j} \right) \\ R_i :=& s_i \mathcal{H}^p(X_i) + c_i \left(\mu_X \mathfrak{T}_{i(\ell)} + \sum_j \mu_j \mathfrak{D}_j \right) \\ c_{i+1} :=& \mathcal{H}^s \left(\underline{\mathbf{pk}'} \mid\mid m \mid\mid L_i \mid\mid R_i \right) \end{aligned}$$

(c) `M` computes c_ℓ, L_ℓ, and R_ℓ as above. If $\mathcal{H}^s$ has been queried before with $\left(\underline{\mathbf{pk}'} \mid\mid m \mid\mid L_\ell \mid\mid R_\ell\right)$, `M` outputs $\perp$ and terminates. Otherwise, `M` backpatches $\mathcal{H}^s\left(\underline{\mathbf{pk}'} \mid\mid m \mid\mid L_\ell \mid\mid R_\ell\right) \leftarrow c_{\ell+1}$.

(d) `M` sends to `A` the signature $\left(\sigma, \mathfrak{T}_{i(\ell)}\right)$ where the signature $\sigma = (c_1, s_1, \dots, s_n, \{\mathfrak{D}_j\}_j)$.

5. `A` outputs a signing index ℓ'. If $\ell = \ell'$, `M` outputs $b' = 0$. Otherwise, `M` flips a coin and outputs a bit b' selected uniformly at random.

Note that `M` only terminates and outputs $\perp$ if `A` asks for a signature with a ring containing a key that is not a DDH challenge key or if $\mathcal{H}^s$ has been queried with $\left(\underline{\mathbf{pk}'} \mid\mid m \mid\mid L_\ell \mid\mid R_\ell\right)$ before step 4*c*. We can assume `A` never asks for a signature with a bad ring like this. Moreover, the points L_ℓ and R_ℓ are uniformly distributed, so the probability that any algorithm can guess the input for backpatching is negligible. Hence, `M` carries out the game in Definition 7 except with negligible probability.

The law of total probability gives us that $\mathbb{P}\left[\texttt{M} \text{ wins}\right] = \frac{1}{2}\mathbb{P}\left[1 \leftarrow \texttt{M} \mid b = 1\right] + \frac{1}{2}\mathbb{P}\left[0 \leftarrow \texttt{M} \mid b = 0\right]$. Moreover, the event that $1 \leftarrow \texttt{M}$ is exactly the event that $\ell' \leftarrow \texttt{A}$ and $\ell' \neq \ell$, and the event that $0 \leftarrow \texttt{M}$ is exactly the event that $\ell' \leftarrow \texttt{A}$ and $\ell' = \ell$. If $b = 1$, then `M` received random points, not the DDH exchange key, so the signature sent to `A` consists of uniformly random points and scalars. `A` can do no better than to guess the index ℓ' uniformly at random. So $\mathbb{P}\left[1 \leftarrow \texttt{M} \mid b = 1\right] = \mathbb{P}\left[\ell' \leftarrow \texttt{A}, \ell' \neq \ell \mid b = 1\right] = \frac{n-1}{n}$. On the other hand, if $b = 0$, then `M` received the DDH exchange key. In this case, `A` has an advantage ϵ at guessing the successful index, so $\mathbb{P}\left[\ell' \leftarrow \texttt{A}, \ell = \ell' \mid b = 0\right] = \frac{1}{n} + \epsilon$. Hence, `M` succeeds at the DDH game with probability $\frac{1}{2}\left(1 - \frac{1}{n}\right) + \frac{1}{2}\left(\frac{1}{n} + \epsilon\right) = \frac{1}{2} + \frac{\epsilon}{2}$.

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. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Proceedings of the 13th ACM Conference on Computer and Communications Security. pp. 390–399. CCS '06, ACM, New York, NY, USA (2006). https://doi.org/10.1145/1180405.1180453, http://doi.acm.org/10.1145/1180405.1180453

3. 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)
4. 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)
5. Bowe, S., Grigg, J., Hopwood, D.: Halo: Recursive proof composition without a trusted setup
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 (full version) (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. Maxwell, G., Poelstra, A., Seurin, Y., Wuille, P.: Simple Schnorr multi-signatures with applications to Bitcoin. Designs, Codes and Cryptography pp. 1–26 (2018)
15. 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)
16. Noether, S., Mackenzie, A., et al.: Ring confidential transactions. Ledger **1**, 1–18 (2016)
17. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
18. Qian, H., Xu, S.: Non-interactive multisignatures in the plain public-key model with efficient verification. Information Processing Letters **111**(2), 82–89 (2010)
19. Quesnelle, J.: On the linkability of Zcash transactions. arXiv preprint arXiv:1712.01210 (2017)
20. 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)
21. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of cryptology **4**(3), 161–174 (1991)
22. Straus, E.G.: Addition chains of vectors (problem 5125). American Mathematical Monthly **70**(806-808), 16 (1964)

23. 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)
24. Van Saberhagen, N.: CryptoNote v 2.0 (2013)
25. 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)
26. 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)