MONERO RESEARCH LAB

RESEARCH BULLETIN MRL-0011

Compact linkable ring signatures and applications

Brandon Goodell* and Sarang Noether† and RandomRun
Monero Research Lab
June 3, 2019

Abstract

We describe an efficient linkable ring signature scheme, compact linkable spontaneous anonymous group (CLSAG) signatures, for use in confidential transactions. Compared to the existing signature scheme used in Monero, CLSAG signatures are both smaller and more efficient to generate and verify for ring sizes of interest. We generalize the construction and show how it can be used to produce signatures with coins of different type in the same transaction.

1 Introduction

Ring signatures are used in Monero to provide transaction signer ambiguity. Originally, a simple Schnorr-like linkable ring signature [4] was used that included previous one-time output public keys as possible signers. Later, the addition of Pedersen commitments to hide amounts necessitated a generalization to the signature [5] in order to prove transaction balance without revealing the signer.

The current ring signature construction used in Monero, MLSAG signatures, scale linearly with the number of ring members included in the signature. In particular, they require the inclusion of two scalars per ring member.

We note that the MLSAG signature scheme, while general enough to permit linking across multiple sets of input keys, is in practice used as a hybrid approach to a linkable ring signature. Indeed, each signature input consists of two keys: the first is a standard one-time output public key, and the second is a key computed using the associated Pedersen commitments. Linkability is only used for the first key, to ensure that double-spending does not occur.

This construction is flexible. For example, the format can be used for usual ring confidential transactions or could be modified to describe *colored ring confidential transactions* with multiple so-called coin colors, allowing users to transact with multiple assets simultaneously.

1.1 Our contribution

In this work, we introduce a d-dimensional linkable ring multisignature (d-LRMS) scheme suitable for use in a colored ring confidential transaction scheme with d colors. Our scheme is *compact* in the sense that signature size scales with the sum of ring size and the dimension d; that is, increasing d yields the same additional number of signature elements, regardless of the ring size. We prove our scheme is unforgeable up to the hardness of the k-one-more discrete logarithm problem.

We call our signature scheme d-dimensional compact linkable spontaneous anonymous group (d-CLSAG) signatures. We demonstrate how to use this scheme for ring confidential transactions in Monero by replacing multi-layered linkable spontaneous anonymous group (MLSAG) signatures with d-CLSAG signatures. The

*surae.noether@protonmail.com
†sarang.noether@protonmail.com

resulting scheme is more efficient in both signature size and verification time than current Monero transaction structures, and can be seamlessly integrated into the Monero transaction protocol.

2 Notation

In this document, we denote algorithms with typefont majuscule English letters like $\texttt{A}$, $\texttt{B}$, or $\texttt{O}$, or typefont names like $\texttt{Setup}$, $\texttt{KeyGen}$, and so on.

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 $\mathbb{Z}/p\mathbb{Z}$ 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 linkability tags.

We denote column vectors in boldface, e.g. $(x_1, \dots, x_d)^\top = \mathbf{x}$, and matrices in underlined boldface, e.g. $((x_{1,1}, x_{1,2}, \dots, x_{1,n}), \dots, (x_{d,1}, \dots, x_{d,n})) = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) = \underline{\mathbf{x}}$ is a $d \times n$ matrix. We denote the Hadamard product of two vectors with $\circ$, so for any $\mathbf{x} = (x_1, x_2, \dots, x_d) = (x_i)_{i=1}^d$, 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 $\mathbb{Z}/p\mathbb{Z}$, we denote this $\mathcal{H}^s$ (hash-to-scalar). If the codomain is $\mathbb{G}$, we denote this $\mathcal{H}^p$ (hash-to-point).

3 Linkable ring multisignatures

In this section, we recall linkable ring signature (LRS) schemes, we modify the definition to account for d-dimensional keys for linkable ring multisignature (LRMS) schemes, and we provide two examples. We emphasize that our description does not present multisignatures as computed between mutually untrusted parties. We use the term to mean a d-of-d threshold multisignature with all keys controlled by the same signer; in particular, we have none of the communication steps necessary for computing a multisignature.

3.1 LRS and LRMS schemes

Definition 3.1 (LRS)**.** A *linkable ring signature scheme* is a tuple $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}, \texttt{Link})$ satisfying the following.

- $\texttt{Setup}(1^\lambda) \to par$. $\texttt{Setup}$ takes as input a security parameter 1^λ, produces some public parameters par.
- $\texttt{KeyGen}(1^\lambda, par) \to (sk, pk)$. $\texttt{KeyGen}$ takes as input a security parameter 1^λ and public parameters par. $\texttt{KeyGen}$ produces as output a private-public keypair (sk, pk).
- $\texttt{Sign}\left(1^\lambda, par, (m, \mathbf{pk}, sk)\right) \to \left\{\perp_{\texttt{Sign}}, (\sigma, \mathfrak{T})\right\}$. $\texttt{Sign}$ 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 $\mathbf{pk} = \{pk_1, \dots, pk_n\}$, and a secret key sk. $\texttt{Sign}$ produces as output either a distinguished failure symbol $out = \perp_{\texttt{Sign}}$ or a signature-tag pair $out = (\sigma, \mathfrak{T})$.
- $\texttt{Verify}\left(1^\lambda, par, (m, \mathbf{pk}, (\sigma, \mathfrak{T}))\right) \to \{0,1\}$. $\texttt{Verify}$ takes as input a security parameter 1^λ, public parameters par, a message m, a ring of public keys $\mathbf{pk}$, and a signature-tag pair $(\sigma, \mathfrak{T})$. $\texttt{Verify}$ produces as output a bit $b \in \{0,1\}$.

- $\texttt{Link}\left(1^\lambda, par, (\sigma, \mathfrak{T}), (\sigma', \mathfrak{T}')\right)$. `Link` takes as input a security parameter 1^λ, public parameters *par*, and a pair of signature-tag pairs $(\sigma, \mathfrak{T}), (\sigma', \mathfrak{T}')$. `Link` produces as output a bit $b \in \{0, 1\}$.

We extend this to a scheme with d-dimensional keys.

Definition 3.2 (d-LRMS)**.** A *d-dimensional linkable ring multisignature scheme* is a tuple of algorithms $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}, \texttt{Link})$ satisfying the following.

- $\texttt{Setup}(1^\lambda) \to par$. `Setup` works as described in 3.1.
- $\texttt{KeyGen}(1^\lambda, par) \to (\mathbf{sk}, \mathbf{pk})$. `KeyGen` takes as input a security parameter 1^λ and public parameters *par*. `KeyGen` produces as output a private-public keypair $(\mathbf{sk}, \mathbf{pk})$ where each key is a d-dimensional vector. We refer to the first coordinate of each of these keys as the *linking key* and the $d-1$ remaining coordinates (if there are any) are the *auxiliary keys*.
- $\texttt{Sign}\left(1^\lambda, par, (m, \underline{\mathbf{pk}}, \mathbf{sk})\right) \to \left\{\perp_{\texttt{Sign}}, (\sigma, \mathfrak{T})\right\}$. `Sign` takes as input a security parameter 1^λ, public parameters *par*, a message m, a vector of public keys $\underline{\mathbf{pk}} = \{\mathbf{pk}_1, \ldots, \mathbf{pk}_n\}$ where each $\mathbf{pk}_i$ is a d-dimensional public key, and a d-dimensional secret key $\mathbf{sk}$. `Sign` produces as output either a distinguished failure symbol $\perp_{\texttt{Sign}}$ or a signature-tag pair $(\sigma, \mathfrak{T})$.
- $\texttt{Verify}\left(1^\lambda, par, (m, \underline{\mathbf{pk}}, (\sigma, \mathfrak{T}))\right) \to \{0, 1\}$. `Verify` takes as input a security parameter 1^λ, public parameters *par*, a message m, a matrix of public keys $\underline{\mathbf{pk}}$, and a signature-tag pair $(\sigma, \mathfrak{T})$. `Verify` produces as output a bit $b \in \{0, 1\}$.
- $\texttt{Link}\left(1^\lambda, par, (\sigma, \mathfrak{T}), (\sigma', \mathfrak{T}')\right)$. `Link` works as described in 3.1.

We refer to the linkability tag as the *key image* for consistency with terminology in Monero. We also use the following terminology if the scheme satisfies the respective properties.

Correctly verified: For any message m, d-dimensional secret key $\mathbf{sk}$ with corresponding public key $\mathbf{pk}$, and appropriately-sized matrix $\underline{\mathbf{pk}} = \{\mathbf{pk}_1, \ldots, \mathbf{pk}_n\}$, if there exists an index $1 \leq \ell \leq n$ such that $\mathbf{pk} = \mathbf{pk}_\ell$, then $\texttt{Verify}(\texttt{Sign}(m, \underline{\mathbf{pk}}, \mathbf{sk})) = 1$.

Correctly linkable: For any messages m, m', any d-dimensional secret keys $\mathbf{sk}$ and $\mathbf{sk}'$ with equal linking keys (that is, with $\mathbf{sk}_0 = \mathbf{sk}'_0$) and with corresponding public keys $\mathbf{pk}$ and $\mathbf{pk}'$ respectively, any appropriately-sized pair of matrices $\underline{\mathbf{pk}}$ and $\underline{\mathbf{pk}}'$, if $(\sigma, \mathfrak{T}) \leftarrow \texttt{Sign}(m, \underline{\mathbf{pk}}, \mathbf{sk})$ and $(\sigma', \mathfrak{T}') \leftarrow \texttt{Sign}(m', \underline{\mathbf{pk}}', \mathbf{sk}')$ and there exists a pair of indices ℓ and ℓ' such that $\underline{\mathbf{pk}}_\ell = \mathbf{pk}$ and $\underline{\mathbf{pk}}_{\ell'} = \mathbf{pk}'$, then $\texttt{Link}((\sigma, \mathfrak{T}), (\sigma', \mathfrak{T}')) = 1$.

Note that linkability is only considered with respect to a single component of a matrix entry. When applied to Monero, this is important since only one such entry corresponds to an output public key for double-spend detection purposes.

3.2 Examples

Example 3.1. The LSAG signature scheme from [4] is a 1-LRMS. In this example, `Setup` always deterministically sets $d := 1$ (so we only use the linking key and there are no auxiliary keys). `Setup` selects some $G \in \mathbb{G}$ to be a group generator for the group parameters $(p, \mathbb{G}, 1, G)$, two cryptographic hash functions $\mathcal{H}^s : \{0,1\}^* \to \mathbb{Z}/p\mathbb{Z}$ and $\mathcal{H}^p : \{0,1\}^* \to \mathbb{G}$. `Setup` outputs $par = (p, \mathbb{G}, 1, G, \mathcal{H}^s, \mathcal{H}^p)$.

`KeyGen` produces as output a private key $x \in \mathbb{Z}/p\mathbb{Z}$ and a public key $X = xG$. `Sign` takes as input a private key $y \in \mathbb{Z}/p\mathbb{Z}$, a message m, and a ring $\mathbf{X} = \{X_1, \ldots, X_n\}$, and produces as output either a distinguished failure symbol $\perp_{\texttt{Sign}}$ or a signature-tag pair $(\sigma, \mathfrak{T})$.

The signature scheme originally described in [4] signs a message m with a ring of keys $\mathbf{X} = \{X_1, \dots, X_n\}$ and a secret key x corresponding to some X_ℓ, using the linkability tag (key image) $\mathfrak{T} := x\mathcal{H}^p(\mathbf{X})$. Unfortunately, this key image is unsuitable for use in Monero, as changing ring members will change the key image. This would allow the same key to be used twice in two different ring signatures. For use in Monero, we modify this key image to be independent of the non-signing ring members, $\mathfrak{T} := x\mathcal{H}^p(X_\ell)$. This allows these key images to be used for double-spend protection.

An LSAG signature on a message m with ring $\mathbf{X}$ using these key images is computed in the following way. First, the signer samples $\alpha, s_{\ell+1}, s_{\ell+2}, \dots, s_{\ell-1} \in \mathbb{Z}/p\mathbb{Z}$ at random. Next, the signer computes basepoints $H_i = \mathcal{H}^p(X_i)$. Next, the signer computes $c_{\ell+1} = \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, \alpha G \,||\, \alpha H_\ell)$ and the signer computes each $c_{i+1} = \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, s_i G + c_i X_i \,||\, s_i H_i + c_i \mathfrak{T})$ for $i = \ell+1, \dots, \ell-1$, naturally identifying index 1 with index $n+1$. The signer finishes by computing $s_\ell = \alpha - c_\ell x_\ell$ and publishing the signature-tag pair $(\sigma, \mathfrak{T})$ where $\sigma = (c_1, s_1, \dots, s_n)$.

A purported LSAG signature-tag pair on a message m with ring $\mathbf{X}$ is verified in the following way. The verifier sets $c'_1 = c_1$ and computes $c'_{i+1} = \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, s_i G + c'_i X_i \,||\, s_i H_i + c'_i \mathfrak{T})$ for $i = 1, 2, \dots, n$. The verifier outputs 1 when $c'_{n+1} = c_1$ and 0 otherwise. Note that all verifiers must list the keys in $\mathbf{X}$ in an agreed-upon order for the above verification to work; either they should agree upon lexicographic or some other ordering. LSAG signature-tag pairs are linked merely by comparing key images: two valid signatures with the same key image were signed with the same secret key (and, in the case of Monero, would signal an attempt to double-spend funds).

Remark 3.1. The key image modification in Example 3.1 is due to the basepoint of the key image $\mathfrak{T}$. As noted in [4], easy variations on key image formats are available. 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 3.2. The MLSAG signature scheme from [5] is a 2-LRMS. In this example, `Setup` always deterministically sets $d := 2$. Public keys are taken to be pairs of group elements $(X, Z) \in \mathbb{G}^2$. `Sign` takes as input a private key vector (x, z), a message m, and a matrix of public keys $\underline{\mathbf{pk}} = \{\mathbf{pk}_1, \dots, \mathbf{pk}_n\}$ where each $\mathbf{pk}_i = (X_i, Z_i)$, each X_i is the public key of a one-time transaction output, and each Z_i is computed using combinations of Pedersen commitments that depend on the particular transaction type in use.

A signature-tag pair $(\sigma, \mathfrak{T})$ is computed in the following way. Two rows of random signature data are sampled, say $\alpha, \alpha', s_{\ell+1}, s'_{\ell+1}, s_{\ell+2}, s'_{\ell+2}, \dots, s_{\ell-1}, s'_{\ell-1}$ and the basepoints $H_i = \mathcal{H}^p(X_i)$ are computed from the signing keys X_i of each ring member (X_i, Z_i). The key image $\mathfrak{T} = x_\ell H_\ell$ is computed. The following challenges are computed

$$\begin{aligned} c_{\ell+1} =& \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, \alpha G \,||\, \alpha H_\ell \,||\, \alpha' G') \\ c_{i+1} =& \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, s_i G + c_i X_i \,||\, s_i H_i + c_i \mathfrak{T} \,||\, s'_i G' + c_i Z_i). \end{aligned}$$

The values $s_\ell = \alpha - c_\ell x_\ell$ and $s'_\ell = \alpha' - c_\ell z_\ell$ are computed. The signature is set $\sigma := (c_1, s_1, s'_1, \dots, s_n, s'_n)$ and the signature-tag pair is published $(\sigma, \mathfrak{T})$. Verification proceeds as one would expect, succeeding if $c'_{n+1} = c_1$ after the verifier computes each H_i and each challenge

$$\begin{aligned} c'_2 =& \mathcal{H}^s(\mathbf{X} \,||\, m \,||\, s_1 G + c_1 X_1 \,||\, s_1 H_1 + c_1 \mathfrak{T} \,||\, s'_1 G' + c_1 Z_1) \\ c'_{i+1} =& \mathcal{H}^s\left(\mathbf{X} \,||\, m \,||\, s_i G + c'_i X_i \,||\, s_i H_i + c'_i \mathfrak{T} \,||\, s'_i G' + c'_i Z_i\right). \end{aligned}$$

Note this scheme is not compact in the sense that doubling key dimension going from $d = 2$ in Example 3.1 to $d = 2$ results in an asymptotic doubling of signature size.

4 A compact d-LRMS scheme

We present a multisignature variant of LSAG signatures that is more compact than the previous examples. We loosely say that a d-LRMS scheme is compact if its signature sizes are not proportional to d. We call this scheme d-CLSAG to be concise. We show how to apply a 2-CLSAG for use in ring confidential transactions, and how to apply d-CLSAG for colored ring confidential transactions. We make remarks on efficiency for both applications. In particular, we show that our 2-CLSAG construction is more efficient in both space and verification time than the equivalent MLSAG construction.

4.1 Implementation

Definition 4.1 (d-CLSAG)**.** The following tuple $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Sign}, \texttt{Verify}, \texttt{Link})$ is a compact d-LRMS.

- $\texttt{Setup}(1^\lambda, d) \to \mathit{par}$. `Setup` takes as input a security parameter 1^λ and a public dimension d. `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 $\mathbb{Z}/p\mathbb{Z}$, selects a cryptographic hash function $\mathcal{H}^p$ with codomain $\mathbb{G}$. `Setup` outputs $\mathit{par} = \left(p, \mathbb{G}, d, G, \left\{\mathcal{H}_j^s\right\}_{j=0}^{d-1}, \mathcal{H}^p\right)$.[‡]

- $\texttt{KeyGen}(1^\lambda, \mathit{par}) \to (\mathbf{sk}, \mathbf{pk})$. `KeyGen` takes as input the security parameter 1^λ, public parameters par. `KeyGen` samples $\mathbf{sk} \leftarrow (\mathbb{Z}/p\mathbb{Z})^d$, which we denote $\mathbf{sk} = (x, z_1, \ldots, z_{d-1})$, and computes $\mathbf{pk} := \mathbf{sk} \circ \mathbf{G}$ for $\mathbf{G} = (G, G, \ldots, G) \in \mathbb{G}^d$. `KeyGen` outputs $(\mathbf{sk}, \mathbf{pk})$. We say x is the *linking key* and the remaining keys $\{z_j\}$ are the *auxiliary keys*.

- $\texttt{Sign}\left(1^\lambda, \mathit{par}, (m, \underline{\mathbf{pk}}, \mathbf{sk})\right) \to \left\{\perp_{\texttt{Sign}}, (\sigma, \mathfrak{T})\right\}$. `Sign` takes as input the security parameter 1^λ, public parameters par, a message $m \in \{0,1\}^*$, a matrix of public keys $\underline{\mathbf{pk}} = (\mathbf{pk}_1, \ldots, \mathbf{pk}_n)$ where each $\mathbf{pk}_i = (X_i, Z_{i,1}, \ldots, Z_{i,d-1}) \in \mathbb{G}^d$, and a secret key vector $\mathbf{sk} = (x, z_1, \ldots, z_{d-1}) \in (\mathbb{Z}/p\mathbb{Z})^d$. `Sign` does the following.

  1. `Sign` looks for the signing index ℓ such that $(x, z_1, \ldots, z_{d-1}) \circ (G, G, \ldots, G) = \mathbf{pk}_\ell$. If no such index exists, `Sign` outputs $\perp_{\texttt{Sign}}$ and terminates.
  2. Otherwise, `Sign` samples $\alpha \in \mathbb{Z}/p\mathbb{Z}$ and $\{s_i\}_{i \neq \ell} \in (\mathbb{Z}/p\mathbb{Z})^{n-1}$.
  3. `Sign` computes the *aggregation coefficients* μ_X and $\{\mu_j\}_{j=1}^{d-1}$, the *linkability tag* or *key image* $\mathfrak{T}$, and auxiliary key images $\{\mathfrak{D}_j\}_{j=1}^{d-1}$:

$$\begin{aligned} \mathfrak{T} &\leftarrow x\mathcal{H}^p(X_\ell) & \mu_X &\leftarrow \mathcal{H}_0^s(\underline{\mathbf{pk}} \,||\, \mathfrak{T} \,||\, \{\mathfrak{D}_j\}_{j=1}^{d-1}) \\ \{\mathfrak{D}_j\} &\leftarrow \{z_j \mathcal{H}^p(X_\ell)\} & \mu_j &\leftarrow \mathcal{H}_1^s(\underline{\mathbf{pk}} \,||\, \mathfrak{T} \,||\, \{\mathfrak{D}_j\}_{j=1}^{d-1}). \end{aligned}$$

  4. For $i = \ell, \ell+1, \ldots, \ell-1$, identifying index n with index 1 as usual, `Sign` computes

$$\begin{aligned} L_\ell &= \alpha G & L_i &= s_i G + c_i \left(\mu_X X_i + \sum_{j=1}^{d-1} \mu_j Z_{i,j}\right) \\ R_\ell &= \alpha \mathcal{H}^p(X_\ell) & R_i &= s_i \mathcal{H}^p(X_i) + c_i \left(\mu_X \mathfrak{T} + \sum_{j=1}^{d-1} \mu_j \mathfrak{D}_j\right) \\ c_{\ell+1} &= \mathcal{H}_0^s(\underline{\mathbf{pk}} \,||\, m \,||\, L_\ell \,||\, R_\ell) & c_{i+1} &= \mathcal{H}_0^s(\underline{\mathbf{pk}} \,||\, m \,||\, L_i \,||\, R_i) \end{aligned}$$

     and lastly computes $s_\ell = \alpha - c_\ell(\mu_X x_\ell + \sum_{j=1}^{d-1} \mu_j z_{\ell,j})$.

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

5. `Sign` sets the signature $\sigma = (c_1, s_1, \dots, s_n, \{\mathfrak{D}_j\}_{j=1}^{d-1})$ and publishes the signature-tag pair $(\sigma, \mathfrak{T})$.

- `Verify` $\left(1^\lambda, par, m, \underline{\mathbf{pk}}, (\sigma, \mathfrak{T})\right) \rightarrow \{0, 1\}$. `Verify` takes as input the security parameter 1^λ, public parameters *par*, a message m, a matrix $\underline{\mathbf{pk}} = ((X_i, Z_{i,1}, \dots, Z_{i,d-1}))_{i=1}^n$, and a signature-tag pair $(\sigma, \mathfrak{T})$. `Verify` does the follwoing.

  1. If $n > N$, or any coordinate of any ring member is not in $\mathbb{G}$, or if σ cannot be parsed as $(c_1, s_1, \dots, s_n, \mathfrak{D}_1, \dots, \mathfrak{D}_{d-1})$ for some $c_1 \in \mathbb{Z}/p\mathbb{Z}$, some $s_i \in \mathbb{Z}/p\mathbb{Z}$, and some $\mathfrak{D}_j \in \mathbb{G}$, or if $\mathfrak{T} \notin \mathbb{G}$, `Verify` produces 0 as output and terminates.
  2. Otherwise, `Verify` parses $(c_1, s_1, \dots, s_n, \{\mathfrak{D}_j\}_{j=1}^{d-1}) \leftarrow \sigma$, computes each $\mathcal{H}^p(X_i)$, and compute the aggregation coefficients as above.
  3. `Verify` sets $c'_1 := c_1$ and, for $i = 1, 2, \dots, n-1$, computes the following.

$$\begin{aligned} L_i :=& s_i G + c'_i \left(\mu_X X_i + \sum_{j=1}^{d-1} \mu_j Z_{i,j} \right) \\ R_i :=& s_i \mathcal{H}^p(X_i) + c'_i \left(\mu_X \mathfrak{T} + \sum_{j=1}^{d-1} \mu_j \mathfrak{D}_j \right) \\ c'_{i+1} :=& \mathcal{H}_0^s \left(\underline{\mathbf{pk}} \,||\, m \,||\, L_i \,||\, R_i \right) \end{aligned}$$

  4. If $c'_{n+1} = c_1$, `Verify` produces 1 as output. Otherwise, `Verify` produces 0 as output.

- `Link` $\left(1^\lambda, par, (\sigma_1, \mathfrak{T}_1), (\sigma_2, \mathfrak{T}_2)\right) \rightarrow \{0, 1\}$. `Link` takes as input the security parameter 1^λ, public parameters *par*, and two signature-tag pairs $((\sigma_1, \mathfrak{T}_1), (\sigma_2, \mathfrak{T}_2))$. If $\mathfrak{T}_1 \in \mathbb{G}$ and $\mathfrak{T}_2 \in \mathbb{G}$ and $\mathfrak{T}_1 = \mathfrak{T}_2$, `Link` produces 1 as output. Otherwise, `Link` produces 0 as output.

We use the usual definition of correct verification and correct linkability, both of which are straightforward to verify directly from the implementation. Later, we discuss the security of this implementation.

Remark 4.1. It may be possible to compress signatures further by computing the coefficients μ_j without the inputs $\mathfrak{D}_j$ and publishing $\sigma = (c_1, s_1, \dots, s_n, \sum_j \mu_j \mathfrak{D}_j)$ only, reducing signature sizes for $d > 1$ further.

The security of this variant is not proven here and is a good avenue for future work. The variant is not relevant for Monero, since $d = 1$ for single-asset ring confidential transactions.

Remark 4.2. Note that the signature is computed with the aggregated secret $w = \mu_X x_\ell + \sum_j \mu_j z_{\ell,j}$. This method of aggregating keys has two properties of importance.

First, under the random oracle model, the map defined by mapping $(x, z_1, \dots, z_{d-1}) \mapsto w$ is collision resistant and anyone with knowledge of $(x_\ell, z_{\ell,1}, \dots, z_{\ell,d-1})$ can compute the aggregated secret key w. More than one choice of $(x_\ell, z_{\ell,1}, \dots, z_{\ell,d-1})$ could map to the same aggregated key, but collision resistance implies second pre-image resistance. Hence, given some $W = wG$ for an unknown aggregated secret key w aggregated from some secret key $(x_\ell, z_{\ell,1}, \dots, z_{\ell,d-1})$, an attacker has at most a negligible chance of finding a second secret key $(x'_\ell, z'_{\ell,1}, \dots, z'_{\ell,d-1})$ that aggregates to the same secret w. This prevents the attacker from using an adversarially generated secret key that coincidentally aggregates to the same w as some challenge key.

Second, under the random oracle model, the map is resistant to key cancellation. That is to say, an attacker has at most a negligible chance of selecting a secret key $(x_\ell, z_{\ell,1}, \dots, z_{\ell,d-1})$ with coordinates that cancel in the sum w; except with negligible probability, w is dependent upon all bits of all coordinates of a secret key $(x, z_1, \dots, z_{d-1})$.

4.2 Efficiency

Consider the space and time efficiency of Definition 4.1. A d-CLSAG signature with a ring size of n contains $n + 1$ scalars and d group elements, so this scheme is compact. In practice, signatures are broadcast with additional information such as references to the ring members. However, this is outside the scope of our definitions.

To examine the verification time complexity, 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 or Pippenger 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 current MLSAG scheme in Monero, note that 2-CLSAG has equivalent functionality. The current Monero implementation requires $2n + 1$ scalars and 1 group element.

To determine the feasibility of implementation in Monero, we produced a test implementation using the Monero codebase 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 ring sizes of interest to Monero (the current network-enforced size is 11), 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.

	Verify		Sign	
Ring size	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

5 Applications

5.1 Single-asset ring confidential transactions

As mentioned above, it is possible to use 2-CLSAG as a replacement for MLSAG signatures in Monero for equivalent functionality. Currently, Monero 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 used with 2-CLSAG, since linkability is not considered for the second key component used in the transaction protocols.

5.2 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, Monero 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.

5.2.1 Informal description of MARCT implementation

In this section, we leverage $\Pi_C^{(d)}$ for use in ring confidential transactions on $d-1$ different assets (or colors) with a pegged exchange rate between colors. We analyze the cost (in terms of weight and sync time) to the blockchain of our proposal compared to a similar proposal using d-MLSAG signatures. We present only the case $d = 3$ in this section to represent two colors, leaving extensions for future work.

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 some constants γ_C, γ_D on $\left\{1, 2, \ldots, 2^{\xi-1}\right\}$, (in this example, $\gamma_C = 1$ and $\gamma_D = 100$). Let G, G' be public group elements with unknown discrete logarithms with respect to each other where G is the output from `Setup` for a d-CLSAG. Let $(\texttt{Prove}, \texttt{Ver})$ be a zero-knowledge sound range proving scheme, such as that described in [3], 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 commitments and P is a batched range proof from `Prove` covering the values of both C and D. We define a transaction key to be a tuple $(m, \mathbf{Q}, \mathbf{O}, (f_C, f_D), (\sigma, \mathfrak{T}), aux)$ 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, $S = (\sigma, \mathfrak{T})$ is a CLSAG signature-tag pair.

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

- the signature-tag pair $(\sigma, \mathfrak{T})$ passes d-CLSAG verification $\texttt{Verify}(m, \underline{\mathbf{pk}}, (\sigma, \mathfrak{T})) = 1$ 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 $Z_i = \gamma_C(C_i - f_C G' - \sum_k C'_k) + \gamma_D(D_i - f_D G' - \sum_k D'_k)$.

As before, this 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 opening information for the commitments 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, the point $\gamma_C(C_\ell - f_C G' - \sum_k C'_k) + \gamma_D(D_\ell - f_D G' - \sum_k D'_k)$ will be a usual public key with basepoint G whose secret key is known by the signer.

5.2.2 Flexible pegs, coin spectra, privacy, and further extensions

The validity of a MARCT without any exchange at all between assets is also possible, allowing for assets to be segregated.

On the other hand, different exchange coefficients γ_C and γ_D are certainly possible. Just so long as there is a way for validators to come to an agreement upon exchange rate coefficients, validators can also convince themselves that transactions accordingly balance. Extending the model of simple ring confidential transactions with two colors to more than two colors is a straightforward exercise.

Further extensions with more immediate value may be possible. One open question is how to modify the above scheme to mask exchange rates. Another open question involves formalizing flexible-peg models of full colored ring confidential transactions between many colors (a spectrum?). These extensions represent fundamental building blocks for a system of smart transactions that respect user privacy.

A Security: Unforgeability

A.1 Hardness

Unforgeability comes from the k-OMDL hardness assumption.

Definition A.1 (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 $G_1, G_2, \ldots, G_k, G_{k+1} \in \mathbb{G}$ (the targets) uniformly at random from $\mathbb{G}$. The challenger sends the group parameters and $\{G_i\}$ to $\texttt{A}$.

2. $\texttt{A}$ is granted access to a corruption oracle $\mathcal{CO}$ that takes as input some G_i sent to $\texttt{A}$ and produces as output the discrete logarithm of G_i with respect to G, i.e. some $x_i \in \mathbb{Z}/p\mathbb{Z}$ such that $G_i = x_i G$.

3. $\texttt{A}$ produces as output a sequence of $k+1$ scalars $x_1, \ldots, x_{k+1} \in \mathbb{Z}/p\mathbb{Z}$, counting as a success if:

   (i) for each x_i, there exists some index $1 \leq j(i) \leq k+1$ such that $G_{j(i)} = xG$ and

   (ii) $\texttt{A}$ made no more than k queries to $\mathcal{CO}$.

A.2 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 [2]. By using linkability tags, our definition is slightly stronger; we allow forgeries to count as successful either with partially-corrupted rings or with messages that the signing oracle has signed before, just so long as neither the signature nor the linkability tag from the forgery appear as output from any oracle query in any transcript. This prevents a malicious party with the ability to persuade users to sign maliciously selected messages with maliciously selected ring members from constructing ostensibly valid, previously-unseen signature-tag pairs.

We use the following in the next definition. Let $n(-)$ be a positive polynomial. Let $\mathcal{H}^s : \{0,1\}^* \to \mathbb{Z}/p\mathbb{Z}$ be modeled as a random oracle. Let $\mathcal{CO}$ be a corruption oracle that takes as input a public key $\mathbf{pk}$ and produces as output the corresponding secret key $\mathbf{sk}$ and the linkability tag $\mathfrak{T}$. Let C be the set of all keys in the transcript of queries made by $\mathtt{A}$ to $\mathcal{CO}$. Let T_C be the set of all linkability tags that appear as output from such a query. Let $\mathcal{SO}$ be a signing oracle that takes as input some $(m, \underline{\mathbf{pk}}', \ell)$ such that $pk_\ell \in \underline{\mathbf{pk}} \cap \underline{\mathbf{pk}}'$ and produces as output a signature-tag pair $(\sigma, \mathfrak{T})$ such that $\mathtt{Verify}(m, \underline{\mathbf{pk}}', \sigma, \mathfrak{T}) = 1$.

Definition A.2 (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. The challenger selects $\{(\mathbf{sk}_i, \mathbf{pk}_i)\}_{i=1}^{n(\lambda)} \leftarrow \mathtt{KeyGen}(1^\lambda)$ and sends $\underline{\mathbf{pk}} = \{\mathbf{pk}_i\}_{i=1}^{n(\lambda)}$ to $\mathtt{A}$.
2. $\mathtt{A}$ is granted access to a corruption oracle $\mathcal{CO}$, random oracle $\mathcal{H}^s$, and the signing oracle $\mathcal{SO}$.
3. $\mathtt{A}$ outputs a message m, a ring of at most n public keys $\underline{\mathbf{pk}}'$, and a signature-tag pair $(\sigma, \mathfrak{T})$. This output is a success if $\underline{\mathbf{pk}}' \subseteq \underline{\mathbf{pk}}$ and $\mathfrak{T} \notin T_C$ and σ has not been output by $\mathcal{SO}$ and $\mathtt{Verify}(m, \underline{\mathbf{pk}}', \sigma, \mathfrak{T}) = 1$.

These success requirements ensure that (i) any ring member whose linkability tag is $\mathfrak{T}$ has not yet had its corresponding private key corrupted by $\mathcal{CO}$; and (ii) the signature itself has not been produced by the signing oracle. This allows the attacker to perhaps re-use a message m and linkability tag $\mathfrak{T}$ from a previous oracle query. If the attacker can produce a new signature σ' on a message that has been signed before by the oracle then the new signature should still count as a forgery.

A.3 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 $\mathbb{Z}/p\mathbb{Z}$, i.e. $\eta = O(|p|)$.

Lemma A.1 (General Forking Lemma)**.** *Let $q, \eta \geq 1$. Let $\mathtt{A}$ be any PPT algorithm which takes as input some $x_{\mathtt{A}} = (x, \underline{h})$ where $\underline{h} = (h_1, \ldots, h_q)$ 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 $\underline{h}$). 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 refer the reader to [1] for a proof of this lemma, which demonstrates that if executing some $\mathtt{A}$ has non-negligible acceptance probability, then forking $\mathtt{A}$ 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 $\mathtt{A}$, 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.

A.3.1 Using a forger in the Forking Lemma

Note that a forger according to Definition A.2 is not directly compatible with the forking lemma; the output is some $y = (m, \mathbf{pk}, \sigma, \mathfrak{T})$ and no idx is included. However, without loss of generality, we can execute $\mathtt{A}$ in a black box that extracts from the transcript of $\mathtt{A}$ some $idx = (j^*, i^*)$ 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 $idx = (i^*, j^*)$ 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 any challenge c_{i^*+1} used in the successful forgery. If such a pair (i^*, j^*) can be found, the algorithm wrapping $\mathtt{A}$ can then output (idx, y) with only a negligible difference in advantage.

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. In particular, the challenge could not have been guessed without making the query (except with negligible success). Indeed, a probabilistic algorithm could flip coins to guess the hash output, but this is successful with negligible probability. Of course, although the index i^* may not have been decided by $\mathtt{A}$ when the query was made, but by the time the forgery is complete, the index i^* has been assigned.

Hence, 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. One algorithm $\mathcal{F}^{\mathtt{A}}$ that works in Lemma A.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 $\underline{h} = (h_1, \ldots, h_q)$ at random by flipping coins, and $\mathcal{F}$ executes $y_{\mathtt{A}} \leftarrow \mathtt{A}(x, \underline{h})$.
3. If $y_{\mathtt{A}} = \perp_{\mathtt{A}}$, then $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $y_{\mathtt{A}} = (idx, y)$ for some $idx = (i^*, j^*)$ 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 $\underline{h}' = (h_1, \ldots, h_{j^*-1}, h'_{j^*}, h'_{j^*+1}, \ldots, h'_q)$.
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 $y'_{\mathtt{A}} \leftarrow \mathtt{A}(x, \underline{h}')$.
5. If $y'_{\mathtt{A}} = \perp_{\mathtt{A}}$, then $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $y'_{\mathtt{A}} = (idx', y')$. If $idx \neq idx'$, $\mathcal{F}$ outputs $\perp_{\mathcal{F}}$ and terminates. Otherwise, $\mathcal{F}$ outputs $(idx, y, \underline{h}, y', \underline{h}')$.

We note that $\mathcal{F}^{\mathtt{A}}$ executed in a black box can be fed the oracle queries $\underline{h}$ and $\underline{h}'$ 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 (negligible) time it takes $\mathcal{F}^{\mathtt{A}}$ to select the random tape for $\mathtt{A}$, select the oracle query sequences $\underline{h}$ and $\underline{h}'$, and output the results.

A.4 The Oracles

Of course, any oracle queries made by $\mathtt{A}$ must be handled by $\mathcal{F}^{\mathtt{A}}$ somehow, and we have already described how $\mathcal{F}^{\mathtt{A}}$ handles random oracle queries made by $\mathtt{A}$: by flipping coins at random and keeping a hash table of the results to maintain consistency with future queries. The forger also has corruption oracle and signing oracle access. We assume that $\mathcal{F}^{\mathtt{A}}$ is granted access to a corruption oracle (say, through the k-OMDL game challenger), so we only need to describe the signing oracle.

The signing oracle access granted to $\mathtt{A}$ is simulated by $\mathcal{F}^{\mathtt{A}}$ through backpatching in the following way. $\mathcal{F}^{\mathtt{A}}$ selects $c_{\ell+1}$ at random without knowing the corresponding query to $\mathcal{H}^s$. $\mathcal{F}^{\mathtt{A}}$ uses the query results $\underline{h}$ to get each c_{i+1} and stores the results in a hash table for consistency in later queries. $\mathcal{F}^{\mathtt{A}}$ finally computes c_ℓ, $\mathcal{F}^{\mathtt{A}}$ and can compute the group points L_ℓ, R_ℓ such that $c_{\ell+1} = \mathcal{H}^s(\underline{\mathbf{pk}} \,||\, m \,||\, L_\ell \,||\, R_\ell)$. $\mathcal{F}^{\mathtt{A}}$ back-patches the hash table.

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.

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. $\mathtt{M}$ sets $\mathbf{pk}_i := (G_{d(i-1)+1}, G_{d(i-1)+2}, \ldots, G_{di})$ for $i = 1, \ldots, \frac{k+1}{d}$ and uses $\{\mathbf{pk}_i\}_{i=1}^{\frac{k+1}{d}}$ as input for $\mathcal{F}^{\mathtt{A}}$, responding to 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.
3. If $\mathcal{F}^{\mathtt{A}}$ outputs $\perp$, so does $\mathtt{M}$ and $\mathtt{M}$ terminates.
4. 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 $(idx, y_1, \underline{h}, y_2, \underline{h}')$ where $y_1 = (m_1, \mathbf{pk}'_1, \sigma_1, \mathfrak{T}_1)$ and $y_2 = (m_2, \mathbf{pk}'_2, \sigma_2, \mathfrak{T}_2)$ are forgeries using oracle queries $\underline{h}$ and $\underline{h}'$, respectively, and $idx = (i^*, j^*)$ as described previously. 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

$$\begin{aligned} y_1 &= (m, \underline{\mathbf{pk}'}, \sigma_1, \mathfrak{T}_1) & \sigma_1 &= (c_1, s_1, \ldots, s_n, \{\mathfrak{D}_j\}_j) \\ y_2 &= (m, \underline{\mathbf{pk}'}, \sigma_2, \mathfrak{T}_2) & \sigma_2 &= (c'_1, s'_1, \ldots, s'_n, \{\mathfrak{D}'_j\}_j) \end{aligned}$$

5. In the transcript of $\mathcal{F}^{\mathtt{A}}$, $\mathtt{M}$ finds $c_{i^*+1} = h_{j^*}$ and in the second transcript $c_{i^*+1} = h'_{j^*}$ for some $h_{j^*} \neq h'_{j^*}$. In both transcripts, c_{i^*+1} is the response to the query $\mathcal{H}^s(\mathbf{X} \,||\, m \,||\, L \,||\, R)$ for the same group elements L, R. Moreover, since this is the query for c_{i^*+1} in both transcripts, the $\mathtt{M}$ finds that $L = s_{i^*}G + c_{i^*}W$ where $W = \mu_X X_{i^*} + \sum_j \mu_j Z_{i^*,j}$ in the first transcript and that $L = s'_{i^*}G + c'_{i^*}W$ in the second transcript, where L and W are common to both transcripts.
6. If $\mu_X = 0$ then $\mathtt{M}$ outputs $\perp$ and terminates.
7. Otherwise, $\mathtt{M}$ computes the discrete logarithm $w = \frac{s'_{i^*} - s_{i^*}}{(c_{i^*} - c'_{i^*})}$ without querying $\mathcal{CO}$.
8. $\mathtt{M}$ makes up to $d-1$ queries to $\mathcal{CO}$ to find the discrete logarithms of the elements of any $(d-1)$-subset of $\{X_{i^*}, Z_{i^*,1}, \ldots, Z_{i^*,d-1}\}$.

9. M uses w to solve for the final discrete logarithm.

10. M outputs the $2 \cdot d \cdot q_c$ queries made by $\mathcal{F}^{\mathtt{A}}$ and outputs the vector $(x_{i^*}, z_{i^*,1}, \ldots, z_{i^*,d-1})$.

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.

We previously noted that if A runs in time $O(t)$, $\mathcal{F}^{\mathtt{A}}$ runs in time $O(2t)$. M takes an additional time s due to: simulating oracle queries with coin flips and hash table modifications throughout the execution, constructing keys in step 2, making termination checks in steps 3, 6, parsing the transcripts in step 4, retrieving L and W from the transcript in step 5, computing the discrete logarithm w in step 7, computing the d-linear system of equations in step 9.

A.6 Security proof

All that remains to prove the unforgeability of the d-CLSAG scheme from Section 4.1 is to show that M as described has a non-negligible acceptance probability.

Theorem A.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 4.1 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 A be a $(t, \epsilon, q_h, q_c, q_s, n)$-forger of the d-CLSAG scheme of Section 4.1, let $\mathcal{F}^{\mathtt{A}}$ be the forking algorithm for A, and let M be the master algorithm previously described. M terminates and outputs $\perp$ in steps 3 and 6 only; otherwise, M succeeds at the k-OMDL game. Hence, if E_3 is the event that M outputs $\perp$ in step 3 and E_6 is the event that M outptus $\perp$ in step 6, then E_3, E_6 are disjoint and the acceptance probability for M is $1 - \mathbb{P}(E_3 \cup E_6) = 1 - \mathbb{P}(E_3) - \mathbb{P}(E_6)$. The probability that M outputs $\perp$ in step 6 is the probability that the hashed coefficient $\mu_X = 0$, which occurs with probability p^{-1}. 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, M succeeds with probability at least $\left(\epsilon \left(\frac{\epsilon}{q_c} - \frac{1}{2^\eta} \right) - p^{-1} \right)$. □

B Security: Signer ambiguity

B.1 Hardness

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

Definition B.1 (Decisional Diffie-Hellman)**.** Let A be any computationally unbounded PPT algorithm, $(p, \mathbb{G}, G)$ and let $n \in \mathbb{N}$.

1. The challenger selects $(r_{i,1}, r_{i,2}, r_{i,3}) \, (\mathbb{Z}/p\mathbb{Z})^3$ 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, the classic DDH game is no harder than Definition B.1

B.2 Defining signer ambiguity

Definition B.2. 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 $\{\mathbf{sk}_i\} \subseteq (\mathbb{Z}/p\mathbb{Z})^d$, computes the corresponding public keys $\mathbf{pk}_i = \mathbf{sk}_i \circ \mathbf{G}$, and sends $\{\mathbf{pk}_i\}$ to $\mathtt{A}$.

2. $\mathtt{A}$ outputs an arbitrary message m and a ring of n_2 distinct members $\underline{\mathbf{pk}'} \subseteq \{\mathbf{pk}_i\}$.

3. The challenger selects a ring index $1 \leq \ell \leq n_2$ uniformly at random, retrieves the private key $\mathbf{sk}$, and sends a valid signature-tag pair $(\sigma, \mathfrak{T}) \leftarrow \mathtt{Sign}(m, \underline{\mathbf{pk}'}, \mathbf{sk})$ 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.

B.3 Security proof

If $\mathbb{G}$ satisfies the DDH hardness assumption, then the distribution of the triple (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 $\mathbb{Z}/p\mathbb{Z}$. 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 $\mathbb{Z}/p\mathbb{Z}$. 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 $\mathbb{Z}/p\mathbb{Z}$ 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 $\mathbb{Z}/p\mathbb{Z}$.

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 linkability tag $\mathfrak{T} = x_\ell \mathcal{H}^p(X_\ell)$. 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 B.1. *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 $\mathtt{A}$ is an algorithm that can succeed at the game in Definition B.2 with non-negligible advantage. We construct a master algorithm $\mathtt{M}$ that plays the game in Definition B.1 by executing $\mathtt{A}$ in a black box such that $\mathtt{M}$ plays the role of the challenger in Definition B.2.

$\mathtt{M}$ receives a set of DDH challenge tuples $\left\{(R_{i,1}, R_{i,2}, R_{i,3}^{(b)})\right\}_{i=1}^{n}$. $\mathtt{M}$ 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. $\mathtt{M}$ 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 $\mathbf{pk}_i := (X_i, Z_{i,1}, \dots, Z_{i,d-1})$. The algorithm $\mathtt{M}$ then operates in the following way:

1. $\mathtt{M}$ sends the public keys $\underline{\mathbf{pk}} = \{\mathbf{pk}_i\}_{i=1}^{n}$ to $\mathtt{A}$.
2. $\mathtt{A}$ outputs a message m and a ring $\underline{\mathbf{pk}}'$.
3. If $\underline{\mathbf{pk}}' \not\subseteq \underline{\mathbf{pk}}$, $\mathtt{M}$ outputs $\perp$ and terminates. Otherwise, the algorithm $\mathtt{M}$ 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 $1 \le \ell \le n_2$ in $\underline{\mathbf{pk}}'$, there exists some key index $1 \le i(\ell) \le n_1$ 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. $\mathtt{M}$ simulates a signature in the following way.
   (a) $\mathtt{M}$ selects a random index $1 \le \ell \le n_2$, selects a random scalar $c_{\ell+1} \in \mathbb{Z}/p\mathbb{Z}$, and selects random scalars $s_1, s_2, \dots, s_n \in \mathbb{Z}/p\mathbb{Z}$.
   (b) For $i = \ell+1, \ell+2, \dots, n-1, n, 1, 2, \dots, \ell-1$, $\mathtt{M}$ computes

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

   (c) $\mathtt{M}$ computes c_ℓ, L_ℓ, and R_ℓ as above. If $\mathcal{H}^s$ has been queried before with $\left(\underline{\mathbf{pk}}' \,||\, m \,||\, L_\ell \,||\, R_\ell\right)$, $\mathtt{M}$ outputs $\perp$ and terminates. Otherwise, $\mathtt{M}$ backpatches $\mathcal{H}^s \left(\underline{\mathbf{pk}}' \,||\, m \,||\, L_\ell \,||\, R_\ell\right) \leftarrow c_{\ell+1}$.
   (d) $\mathtt{M}$ sends to $\mathtt{A}$ the signature-tag pair $\left(\sigma, \mathfrak{T}_{i(\ell)}\right)$ where $\sigma = (c_1, s_1, \dots, s_n, \{\mathfrak{D}_j\}_j)$.
5. $\mathtt{A}$ outputs a signing index ℓ'. If $\ell = \ell'$, $\mathtt{M}$ outputs $b' = 0$. Otherwise, $\mathtt{M}$ flips a coin and outputs a bit b' selected uniformly at random.

Note that $\mathtt{M}$ only terminates and outputs $\perp$ if $\mathtt{A}$ 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 $\left(\underline{\mathbf{pk}}' \,||\, m \,||\, L_\ell \,||\, R_\ell\right)$ before step 4*c*. We can assume $\mathtt{A}$ 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, $\mathtt{M}$ carries out the game in Definition B.1 except with negligible probability.

The law of total probability gives us that $\mathbb{P}\left[\mathtt{M} \text{ wins}\right] = \frac{1}{2}\mathbb{P}\left[1 \leftarrow \mathtt{M} \mid b = 1\right] + \frac{1}{2}\mathbb{P}\left[0 \leftarrow \mathtt{M} \mid b = 0\right]$. Moreover, the event that $1 \leftarrow \mathtt{M}$ is exactly the event that $\ell' \leftarrow \mathtt{A}$ and $\ell' \neq \ell$, and the event that $0 \leftarrow \mathtt{M}$ is exactly

the event that $\ell' \leftarrow \mathtt{A}$ and $\ell' = \ell$. If $b = 1$, then $\mathtt{M}$ received random points, not the DDH exchange key, so the signature sent to $\mathtt{A}$ consists of uniformly random points and scalars. $\mathtt{A}$ can do no better than to guess the index ℓ' uniformly at random. So $\mathbb{P}\left[1 \leftarrow \mathtt{M} \mid b = 1\right] = \mathbb{P}\left[\ell' \leftarrow \mathtt{A}, \ell' \neq \ell \mid b = 1\right] = \frac{n-1}{n}$. On the other hand, if $b = 0$, then $\mathtt{M}$ received the DDH exchange key. In this case, $\mathtt{A}$ has an advantage ϵ at guessing the successful index, so $\mathbb{P}\left[\ell' \leftarrow \mathtt{A}, \ell = \ell' \mid b = 0\right] = \frac{1}{n} + \epsilon$. Hence, $\mathtt{M}$ 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] Mihir Bellare and Gregory Neven. 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*, CCS '06, pages 390–399, New York, NY, USA, 2006. ACM.

[2] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *Theory of Cryptography Conference*, pages 60–79. Springer, 2006.

[3] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[4] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Australasian Conference on Information Security and Privacy*, pages 325–335. Springer, 2004.

[5] Shen Noether, Adam Mackenzie, et al. Ring confidential transactions. *Ledger*, 1:1–18, 2016.