RESEARCH BULLETIN MRL-XXXX

Ring Threshold Multisignature Schemes and Security Models

Brandon Goodell* and Sarang Noether†
Monero Research Lab
October 7, 2017

Abstract

This research bulletin extends [3] by constructing a t-of-n threshold multi-layered linkable spontaneous anonymous group signature scheme (t-of-n MLSAG) in the same style as the LSAG schemes put forth by [2].

1 Introduction and Background

Ring signatures can play a critical role in promoting user anonymity (or at least user ambiguity) during message authentication. A one-time signature scheme inserts a barrier between user key pairs and one-time signature key pairs. Due to this utility, one-time ring signature schemes enjoy application in many cryptocurrency protocols. Multisignatures play a critical role in off-chain transactions for cryptocurrencies (e.g. the Bitcoin Lightning Network) and for message authentication in general (e.g. multi-factor authentication). A t-of-N threshold multisignature scheme specifies sets containing N public keys and thresholds t such that any subset with at least t elements may collaborate to fashion a signature. A usual digital signature scheme is a 1-of-1 multisignature scheme, so we can regard all keys as shared public keys (just perhaps with a coalition of only one member). It is therefore natural to extend the notion of ring signatures to ring threshold multisignatures for implementation in cryptocurrencies to enjoy signer-ambiguous multisignatures.

A multisignature scheme is a t-of-N *ring threshold multisignature* (RTM) scheme if any set of N keys may be specified as a coalition of signers and assigned a shared public key $X_{\texttt{shared}}$ such that any t coalition members may collaborate to fashion a ring signature. The ring of signatories R contains the key $X_{\texttt{shared}}$, but an adversary cannot determine which element of R computed the signature.

If the number of users cooperating in the construction of a signature is not secret, naive multisignature schemes can be constructed from any signature scheme (ring signature or otherwise) by simply requiring each participating user to present a separate signature. More sophisticated implementations combine these together using boolean AND circuits as in the Borromean ring set-up described in [] for efficiency reasons.

If a user does not desire to reveal to an adversary how many devices were used for some multi-factor authentication, it should be difficult for an adversary to determine the size of a coalition behind some shared public key. This property should be satisfied even if the adversary can persuade the party (or parties) controlling the shared public key to sign arbitrary messages chosen by the adversary. We introduce the security definition of *coalition-indistinguishable* multisignature schemes against adaptive chosen message attacks: given a shared t-of-N public key $X_{\texttt{shared}}$, an adversary should be unable to guess any information about t or N.

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

1.1 Our Contribution

We consider a formal definition of one-time linkable ring threshold multisignature (OT-LRTM) schemes. We investigate modifications to security definitions that take threshold behaviors into account and a new security model. We describe a modified implementation of t-of-N linkable ring threshold multisignature (under the restriction $N-1 \leq t \leq N$) first described by previous MRL contributors Shen Noether in [3] and implemented for use in Monero by contributor `Luigi`, and we prove that this implementation satisfies our security definitions.

1.2 Notation and Prerequisites

We let $\mathbb{G}$ be a group with prime order $\mathtt{q}$ and we let G denote a commonly known point with order $\mathtt{q}$. Denote the user and transaction key spaces, respectively, as $\mathcal{K}_{\mathtt{user}}, \mathcal{K}_{\mathtt{txn}}$. In implementations involving cryptocurrencies, there exists a function $\mathtt{dest} : \mathcal{K}_{\mathtt{txn}} \to \mathcal{K}_{\mathtt{user}}$ describing the user key pair to whom a certain transaction key pair is addressed. For any transaction key pair $(q, Q) \in \mathcal{K}_{\mathtt{txn}}$, we say $\mathtt{dest}(q, Q)$ is the *destination* user key pair for (q, Q). We sometimes write Q as Q_X to emphasize the destination user key X associated with the transaction key Q.

2 MLSAG and Straightforward Threshold Set-ups

We briefly describe LSAG ring signatures in the sense of [2] and their MLSAG variant used in Monero, and then a straightforward implementation of an LRTM scheme.

2.1 MLSAGs

A user with user key pair (y, Y) wishes to spend an old transaction output with private-public transaction key pair $(q, Q) \in \mathcal{K}_{\mathtt{txn}}$ such that $\mathtt{dest}(q, Q) = (y, Y)$. The user constructs an appropriate message M, a destination user key X, computes the key image $J = qH_p(Q)$, and selects a ring of public transaction keys $R = \{Q_1, \dots Q_L\}$ such that, for a secret distinguished index k, $Q_k = Q$. For each $i = 1, \dots, L$, the signer computes an elliptic curve point from the i^{th} ring member public key as $H_i := H_p(Q_i)$. The signer selects a random secret scalar u, computes an initial temporary pair of points uG and uH_k, and computes an initial commitment $c_{k+1} := H_p(M, uG, uH_k)$.

The signer sequentially proceeds through indices $i = k+1, k+2, \dots, n, 1, 2, \dots, k-1$ in the following way. The signer selects a random scalar s_i, computes the next temporary pair of points $s_iG + c_iQ_i$ and $s_iH_i + c_iJ$, and computes the next commitment $c_{i+1} := H_p(m, s_iG + c_iQ_i, s_iH_i + c_iJ)$. The signer continues proceeding through the set of keys until commitments c_i have been computed for each $i = 1, \dots, L$. The signer then computes $s_k := u - c_kq_k$ and publishes the signature $\sigma = (c_1, s_1, \dots, s_n)$ and the key image J in a signature-tag pair (σ, J).

A signature-tag pair (σ^*, J^*) on m can be verified to have been generated by at least one ring member in the following way: for each $i = 1, 2, \dots, L$, the verifier computes $z_i = s_i^*G + c_i^*Q_i$ and $z_i' = s_i^*H_i + c_i^*J^*$ and uses these to compute the $(i+1)^{th}$ commitment $c_{i+1} = H_p(m, z_i, z_i')$. After computing $c_2, c_3, \dots, c_L, c_1$, the verifier approves of the signature if and only if $c_1 = c_1^*$. A verifier can check against double spends by comparing the key images of two signature-tag pairs.

Remark 2.1.1. The MLSAG generalization, where transaction keys are represented by vectors, is straightforward. With transaction keys $\underline{q} = (q_1, \dots, q_w)$, each component of $\underline{q}$ is used to generate a temporary pair of

points starting with $u_j G$, $u_j H_k$ and then $s_{j,i}G + c_i Q_{j,i}$, $s_{j,i}H_{j,i} + c_i J$, providing the associated commitments

$$c_{i+1} := H_p\left(m, \{(s_{j,i}G + c_i Q_{j,i}, s_{j,i}H_{j,i} + c_i J)\}_{j=1}^{w}\right).$$

Remark 2.1.2. Note that user keys are not used above except as a destination for the transaction key. Anyone with a destination public user key in mind and knowledge of a transaction private key may fashion a signature like the one above. In the reference CryptoNote protocol, the private transaction key q associated to some public transaction key Q is only feasibly computable by a user who knows the private destination user key in $\texttt{dest}(q, Q)$ or by an adversary who can solve the discrete logarithm problem.

2.2 Extending to Threshold Signatures

Note that a 1-of-N threshold signature scheme may be accomplished by simply handing out an identical set of keys to N individuals: whoever decides to use them first will be able to fashion a signature. We consider this a degenerate case. On the other hand, a 1-of-1 signature scheme is a usual signature scheme, as we mentioned.

Generally, we wish to allow a coalition of (distinct) public user keys $C = \{X_1, X_2, \ldots, X_N\} \subseteq \mathcal{K}_{\texttt{user}}$ (where each $X_i = x_i G$) to collaboratively fashion a shared public user key $X_{\texttt{shared}}$ such that, for any transaction key pair $(q, Q) \in \mathcal{K}_{\texttt{txn}}$ such that $\texttt{dest}(q, Q) = X_{\texttt{shared}}$, any subset of at least t of the users in C can collaborate to fashion a signature on a message M corresponding to public transaction key Q. Certainly we wish that no member of C reveal their own private user key x_i, but moreover we wish that coalition members cannot feasibly learn the private transaction key q.

In the N-of-N case, we may use CryptoNote-styled user and transaction keys to see an example implementation. For this example, we assume user private-public key pairs (x, X) take the form $x = (a, b)$ for some scalars a, b and $X = (A, B)$ where $A = aG$ and $B = bG$ for some common point G. The private-public transaction key pair (q, Q) takes the form $Q = (S, P)$, $q = (s, p)$ where s is a scalar, $S = sG$, $p = H_s(aR) + b$, and $P = pG$. Given message M, a coalition of user keys C, the coalition members compute their shared public key as $X_{\texttt{shared}} := \sum_{j=1}^{N} X_j$, which is published. Assume $(q, Q) = ((s, p), (S, P))$ is a transaction key pair such that $\texttt{dest}(q, Q) = X_{\texttt{shared}}$.

The coalition C selects a ring of public transaction keys $\mathcal{R} = \{Q_1, Q_2, \ldots, Q_L\}$ such that $Q = Q_k$ for some secret index k. For each $j = 1, \ldots, N$, the j^{th} coalition member in C computes a partial key image $J_j = (H_s(a_j S) + b_j) \cdot H_p(Q)$, picks a random secret scalar u_j, and computes $H_i = H_p(Q_i)$ for each $Q_i \in R$. The coalition C computes the key image $J = \sum_j J_j$, the random points $u_k G = \sum_j u_j G$, and $u_k H_k = \sum_j u_j H_k$. The coalition C decides upon random values $s_{k+1}, \ldots, s_L, s_1, \ldots, s_{k-1}$. Using these, any member in C may compute the commitments

$$\begin{aligned} c_{k+1} :=& H_p(m, u_k G, u_k H_k) \text{ and} \\ c_{i+1} :=& H_p(m, s_i G + c_i P_i, s_i H_i + c_i J) \text{ for } i = k+1, \ldots, k-1. \end{aligned}$$

All coalition members then use c_k to compute their personal $s_{k,j} = u_j - c_k(H_s(a_j S) + b_j)$. The signers share their $s_{k,j}$ with the other signers. Any threshold member may then compute the value $s_k = \sum_j s_{k,j}$ and publish the signature-tag pair (σ, J) where $\sigma = (c_1, s_1, \ldots, s_L)$ as usual. Any user may verify this signature corresponds to the N-of-N shared public key $X_{\texttt{shared}}$ using the same method as above.

This implementation satisfies our immediate two properties: members in C do not learn the private transaction key $q = \sum_j H_s(a_j S) + b_j$ and do not reveal their own private keys. Assuming at least one contributing user key in C was honestly generated from a uniform random scalar, an adversary who has

learned a public threshold address cannot determine the number of summands contributing to it, let alone determine the summands.

This set-up extends naturally to an $(N-1)$-of-N set-up. As before, a set of N public keys $\{X_1, X_2, \dots, X_N\}$ form a coalition. Each pair of users has a shared secret scalar $z_{i,j} = H_s(x_i X_j)$ with associated point $Z_{ij} = z_{i,j}G$. There are $\frac{N(N-1)}{2}$ such pairs; if any $N-1$ members get together, all of the associated shared secrets are known. Hence, we may simply instantiate the $(N-1)$-of-N threshold as an N^*-of-N^* set-up with $N^* = \frac{N(N-1)}{2}$. All values $Z_{i,j}$ are necessary to compute the shared public user key, $Z_{\texttt{shared}}$ and all values of $z_{i,j}$ are necessary to fashion a signature with a public transaction key Q with $\texttt{dest}(Q) = Z_{\texttt{shared}}$.

Remark 2.2.1. Note that the above extension works for coalitions containing t-of-N keys also. For example, when the j^{th} coalition member computes the partial key image J_j, if this coalition member is some sub-coalition, then each member of the sub-coalition can compute their own partial key image $J_{j,k}$ and the sub-coalition can report the sum $J_j = \sum_k J_{j,k}$. In this manner, signatures involving nested coalitions may be executed recursively. We elaborate on this in Section 3.1.

Remark 2.2.2. Note that an adversary with knowledge of some set of public keys can compute the sums of all subsets to brute-force test whether a certain user key X is a threshold key. Using hash functions and encrypt-then-authenticate communication, we may resolve the brute force problem.

Consider modifying the N-of-N implementation by having coalition members select secret scalars μ_j associated with the threshold t and coalition C, e.g.:

$$\mu_j = H_s(\text{“multisig constant for escrow at local coffee shop”}, \texttt{secret salt})$$

Now merely require that participating members not use their private user keys in the construction of their shared user key pair, but instead the i^{th} coalition member selects a constant μ_i associated with their coalition and instead uses $x_i^* = H(x_i, \mu_i)$ as their private key (or in the $(N-1)$-of-N case, computing $z_{i,j}^* = H_s(x_i^* X_j^*)$ instead of $z_{i,j} = H_s(x_i X_j)$ and communicating the points $Z_{i,j}^*$ to the coalition).

With this modification, an adversary cannot use strictly public information to determine if a certain key is a threshold key or not. The possibility remains that the adversary may overhear the associated public points X_i^* (or $Z_{i,j}^*$) being communicated within the coalition, allowing the adversary to fall back on the brute force approach again. Hence, the points X_i^* (or $Z_{i,j}^*$) should be communicated with a secure encrypt-then-authenticate scheme. Note that either step alone (hashing, then encrypt-then-authenticating) is insufficient to prevent the adversary from using brute force.

3 Security Models

3.1 One-Time Linkable Ring Threshold Multisignatures

We begin by defining a one-time linkable ring threshold multisignature (OT-LRTM) scheme. A central idea to our security models is that a coalition of user keys may be merged into a new user key, which may then be again merged with other user keys.

Definition 3.1.1. [One-Time Linkable Ring Threshold Multisignature Scheme] A one-time linkable ring threshold multisignature scheme is a set of PPT algorithms, (`UserKeyGen`, `TxnKeyGen`, `Merge`,`Sign`,`Verify`, `Link`) that, respectively, generates usual private-public keypairs for users, generates public transaction keys, merges user keys into new shared user keys, fashions signatures on messages given a ring of public transaction keys, verifies signatures, and links signatures. Formally:

(i) $\texttt{UserKeyGen}(1^\lambda)$ outputs a random user key pair (x, X) called a 1-*of*-1 *user key pair* where x is a private user key with associated public user key X.

(ii) $\texttt{Merge}(t, C)$ takes as input a positive integer (threshold) t and coalition of private-public user keypairs $C = \{(x_1, X_1), (x_2, X_2), \ldots, (x_n, X_N)\}$ and outputs a public user key $X_{t,C}$ called a *t-of-N user key pair* or a *shared user key pair*.

(iii) $\texttt{TxnKeyGen}(1^\lambda, X)$ takes as input a public user key X called the *destination key*. A one-time random private-public transaction key pair (q_X, Q_X) is generated. `TxnKeyGen` outputs Q_X.

(iv) $\texttt{ImageGen}(1^\lambda, Q, y)$ takes as input a public transaction key Q with a set of private user keys $y = \{x_1, \ldots\}$ and outputs a point J_Q called the *key image* for the private transaction key q and outputs J_Q.

(v) $\texttt{Sign}(M, X, R, k, y)$ takes as input message M, a destination user key X, ring of public transaction keys $R = \{Q_1, \ldots, Q_L\}$, secret index k, and a set of private user keys y. `Sign` obtains $Q \leftarrow \texttt{TxnKeyGen}(1^\lambda, X)$, $J \leftarrow \texttt{ImageGen}(1^\lambda, Q, y)$. `Sign` generates a signature σ and publishes the signature-image pair (σ, J).

(vi) $\texttt{Verify}(M, R, \sigma)$ takes as input a message M, a ring of public transaction keys R, and a signature σ, and outputs a bit $b \in \{0, 1\}$.

(vii) $\texttt{Link}((M_0, R_0, (\sigma_0, J_0)), (M_1, R_1, (\sigma_1, J_1)))$ takes as input two (possibly distinct) messages, two (possibly distinct) rings of transaction public keys, and two signature-image pairs. `Link` outputs a bit $b \in \{0, 1\}$.

Note that a 1-of-1 user key may be regarded as a "usual" user key in a one-time linkable ring signature scheme. In this way, we may regard all user keys as t-of-N shared user keys by simply regarding 1-of-1 keys as having a coalition of a single member. We consider only *restricted* OT-LRTM schemes where `Merge` is modified such that (i) if $t = 1$ and $|C| = 1$, then `Merge` returns the public user key in C, (ii) if the inequalities $2 \leq N - 1 \leq t \leq N$ do not hold then `Merge` outputs $\perp$ instead of a key.

Definition 3.1.2. Assume for each $i = 0, 1$, M_i is an arbitrary message, X_i is an arbitrary t_i-of-N_i shared public user key with coalition C_i, $R_i = \{Q_{i,j}\}_{j=1}^{|R_i|}$ is an arbitrary ring of public transaction keys with associated secret indices k_i such that $\texttt{dest}(Q_{i,k_i}) = X_i$, each y_i is a set of private user keys such that $y_i \subseteq C_i$ and $t_i \leq |y_i|$, and each signature-tag pair (σ_i, J_i) is honestly generated as $(\sigma_i, J_i) \leftarrow \texttt{Sign}(M_i, X_i, R_i, k_i, y_i)$. We say an OT-LRTM scheme is *complete* if

(a) $\texttt{VER}(M_i, R_i, (\sigma_i, J_i)) = 1$ and

(b) if $Q_{i,k_i} = Q_{j,k_j}$ then $\texttt{LNK}((\sigma_i, J_i), (\sigma_j, J_j)) = 1$.

Recall Remark 2.2.2 and consider the hash-then-encrypt-then-authenticate approach to computing shared public keys. We let $\Pi = (\texttt{UserKeyGen}^*, \texttt{Enc}^*, \texttt{Auth}^*, \texttt{Ver}^*, \texttt{Dec}^*)$ be a secure encrypt-then-authenticate scheme (where $\Pi_{\texttt{enc}} = (\texttt{Gen}^*, \texttt{Enc}^*, \texttt{Dec}^*)$ is a secure encryption sub-scheme and $\Pi_{\texttt{auth}} = (\texttt{Gen}^*, \texttt{Auth}^*, \texttt{Ver}^*)$ is a secure message authentication sub-scheme). Augmenting the implementation of Section 2.2 with Π allows the coalition for $X_{\texttt{shared}}$ to compute the appropriate values to participate in the signing of a message in a recursive fashion. To see how, note that when the implementation of Section 2.2 is carried out, this t-of-N shared public user key $X_{\texttt{shared}}$ must first compute the partial key image, next select a random secret scalar u_j, then compute the commitments c_k, and finally must compute the value $s_{k,j} = u_j - c_k(H_s(a_j R) + b_j)$.

Denote the coalition of user key pairs for $X_{\texttt{shared}}$ as $\{((a_j, b_j), (A_j, B_j))\}$. The coalition for $X_{\texttt{shared}}$ may use $\Pi_{\texttt{auth}}$ to share their $(H_s(a_j S) + b_j) \cdot H_p(Q)$ and compute the key image $J = (\sum_j H_s(a_j)S + b_j)H_p(Q)$. If some

index, say j, corresponds to a user key pair $((a_j, b_j), (A_j, B_j))$ that is a t_j-of-N_j public key, then the secrets a_j and b_j are not known by the coalition and so the j^{th} share of the key image, J_j, must be collaboratively computed by at least t_j coalition members for the shared key $((a_j, b_j), (A_j, B_j))$. Denote the coalition for this key as $\{((a_{j,i}, b_{j,i}), (A_{j,i}, B_{j,i}))\}_{i=1}^{N_j}$. Each member computes their share, $J_{j,i} = (H_s(a_{j,i}S) + b_{j,i})H_p(Q)$, this sub-coalition uses $\Pi_{\texttt{auth}}$ to compute $J_j = \sum_i J_{j,i}$, and the sub-coalition reports J_j when prompted.

Similarly the random scalar u_j is computed as a sum $u_j = \sum_i u_{j,i}$ using Π. Now any of these coalition members may compute the commitments c_k and disseminate this to the rest of the coalition with $\Pi_{\texttt{auth}}$. At that point each member of the coalition may compute their individual $s_{k,j,i} = u_{j,i} - c_k x_{j,i}$ and the coalition may use $\Pi_{\texttt{auth}}$ to compute $s_{k,j} = \sum_i s_{k,j,i}$. In this way, sub-coalitions are simply handled recursively.

Remark 3.1.3. If an OT-LTRM scheme is secure under the CIK model from Definition 3.2.1 in Section 3.2, then it is not feasible for any PPT algorithm to check whether the keys used as input for `Merge` are 1-of-1, so modifying the straightforward implementation by banning composite coalition keys is not feasible. Recursion seems to be a natural design choice.

3.2 Coalition Indistinguishable Keys

Definition 3.2.1 formalizes the idea that an adversary should not be able to determine information about the input of `Merge` based on its output except with negligible probability.

Definition 3.2.1 (Coalition Indistinguishable Keys)**.** Let $\mathcal{A}$ be a PPT adversary. Let $N(-)$, $L(-)$ be polynomials.

(i) A set of user key pairs $S^* \subseteq \mathcal{K}_{\texttt{user}}$ with $|S| = N(\lambda)$ is generated where the i^{th} key pair is t_i-of-N_i public user key for some $2 \leq t_i \leq N_i \leq L(\lambda)$. The set of public keys $S = \{X_i \mid \exists (x_i, X_i) \in S^*\}$ is sent to $\mathcal{A}$.

(ii) $\mathcal{A}$ outputs (τ_0, C_0) where $C_0 \subseteq S$, $\tau_0 \in \mathbb{N}$, and $\tau_0 \leq |C_0|$.

(iii) A random pair (τ_1, C_1) is selected where $C_1 \subseteq S$, $\tau_1 \in \mathbb{N}$, $\tau_0 \neq \tau_1$, and $C_1 \neq C_0$. A random bit b is selected. The key $X_{\tau_b, C_b} \leftarrow \texttt{Merge}(\tau_b, C_b)$ is sent to $\mathcal{A}$.

(iv) $\mathcal{A}$ outputs a bit b'. This counts as a success if $b = b'$.

We say an OT-LRTM scheme has Coalition Indistinguishable Keys (CIK) if the adversary can succeed with probability only negligibly more than $1/2$.

Remark 3.2.2. Even taking the above modification into account, each $H_s(X_j, \mu_j)G$ must be communicated to the coalition. An adversary who can learn these points may simply check whether a given public key X is the sum of some observed values of $H_s(X_j, \mu_j)G$, determining non-trivial information about coalition size. Hence, these points should be communicated with Π if an OT-LRTM scheme is to satisfy Definition 3.2.1.

Remark 3.2.3. We may be tempted to strengthen Definition 3.2.1 to take into account corruption oracle access on the part of the adversary. Unfortunately this leads to certain problems with the security definition. However, by using the hash-then-encrypt-then-authenticate method of computing shared public keys, without knowledge of the values of μ_j, even if the adversary corrupts all the public keys in S, then $\mathcal{A}$ cannot successfully run `Merge` for each value $1 \leq t \leq |S|$ to check the results by hand in comparison against the key X_{t_b, C_b}. Thus, if the participating coalition members keep each μ_j and $\mu_j G$ secret, then even a very powerful adversary with oracle access for computing discrete logs will still be unable to discern whether some user key is a coalition key or not.

3.3 Signer Ambiguity

In addition to coalition indistinguishability, we desire the ring signature property of *signer ambiguity*. Variations of security models appear in [1].

Double spend protection in Monero relies on a one-time linkable ring signature scheme that is not signer ambiguous with respect to adversarially generated keys according to the definition presented in [1]. Indeed, in Monero, `Link` simply checks if two key images J_i are identical. In this way, the signer ambiguity game falls apart: $\mathcal{A}$ can obtain signature-tag pair (σ_0, J_0) on M_0 using ring R_0 with $Q_{i_0} \in R_0$ and a pair (σ_1, J_1) on M_1 using ring R_1 with $Q_{i_1} \in R_1$. Then, upon receipt of (σ, J) in step (v), $\mathcal{A}$ can check if $J = J_0$ or $J = J_1$. The definition may be modified, however, to take key images into account.

Let $\mathcal{SO}(-,-,-,-)$ be a signing oracle that takes as input (M, X, R, k) (a message, a destination public user key, a ring of public transaction keys, and an index k) and outputs a valid signature-tag pair $(\sigma, J) \leftarrow \texttt{Sign}(M, X, R, k, y)$ for some set y of private user keys.

Definition 3.3.1. [Linkable Signer Ambiguity v. Adversarially Generated Keys] Let $N(-), L(-)$ be a positive polynomial. Let $\mathcal{A}$ be a PPT adversary. Let $\mathcal{A}$ have access to $\mathcal{SO}$. Consider the following game:

(i) A set of user key pairs $S^* \subseteq \mathcal{K}_{\texttt{user}}$ is selected with $|S^*| = N(\lambda)$. The public keys in S^* are sent to $\mathcal{A}$.

(ii) $\mathcal{A}$ outputs a set of user key pairs $S \subseteq S^*$.

(iii) For each public user key $X_i \in S$, a public transaction key $Q_i^* \leftarrow \texttt{TxnKeyGen}(1^\lambda, X_i)$ is generated and the set $R^* := \{Q_i^*\}$ is generated, randomly permuted, and then sent to $\mathcal{A}$.

(iv) $\mathcal{A}$ selects a message M, a destination public user key $X \in \mathcal{K}_{\texttt{user}}$, a ring of transaction public keys $R = \{Q_1, \ldots, Q_L\} \subseteq \mathcal{K}_{\texttt{txn}}$, and two indices $i_0 \neq i_1$ such that $\{Q_{i_0}, Q_{i_1}\} \subseteq R^* \cap R$.

(v) A random bit b is chosen. The signature-tag pair $(\sigma, J) \leftarrow \mathcal{SO}(M, X, R, i_b)$ is sent to $\mathcal{A}$.

(vi) $\mathcal{A}$ outputs a bit b'. The game counts as a success if (a) $b = b'$ and (b) if (M', X', R', i) is a query from $\mathcal{A}$ to $\mathcal{SO}$, then the i^{th} element of R' is not Q_{i_0} or Q_{i_1}.

We say the scheme is *linkably signer ambiguous against adversarially generated keys* (LSA-AGK) if the probability that $\mathcal{A}$ succeeds is negligibly close to $1/2$ (with respect to λ).

Definition 3.3.1 essentially modifies the signer ambiguity game in [1] by adding requirements in step (vi) requiring that $\mathcal{A}$ see in step (v) either the key image for Q_{i_0} or the key image for Q_{i_1} for the first time.

3.4 Unforgeability

Unforgeability of any threshold signature scheme must take into account subthreshold corruption oracle access. Multisignatures must not be forgeable by a subthreshold collection of malicious coalition members, otherwise they have no utility as signatures, of course. A naive definition may be something like this:

Definition 3.4.1. [Prototype: Subthreshold Oracle Access] Given a $S = \{X_1, \ldots, X_N\} \subseteq \mathcal{K}_{\texttt{user}}$ where each $X_i \in S$ is a t_i-of-N_i public user key, we say that any PPT adversary $\mathcal{A}$ with access to an oracle $\mathcal{O}(-)$ has had *subthreshold oracle access* to S if, for any $X_i \in S$, at most $t_i - 1$ coalition members for X_i appear in the transcript between $\mathcal{A}$ and $\mathcal{O}(-)$.

However, since `Merge` allows inputs of arbitrary (possibly threshold) user keys, this definition is insufficient. For notational convenience, we call $\mathcal{M}(-)$ an oracle that inverts `Merge` by taking as input a public

t-of-N key X and producing as output (t, C), the threshold t and coalition C such that $X = \texttt{Merge}(t, C)$. For any subset $S \subseteq \mathcal{K}_{\texttt{user}}$, define $\mathcal{M}(S) = \cup_{Y \in S} \mathcal{M}(Y)$. This provides the iterative definition $\mathcal{M}^{i+1}(S) = \cup_{Y \in S} \mathcal{M}^i(Y)$. Define $\mathcal{M}^{\leftarrow}(Y) := \cup_i \mathcal{M}^i(Y)$.

Definition 3.4.2. [Subthreshold Oracle Access] Let S be a set of public user keys $S = \{X_1, \ldots, X_N\}$ where each X_i is a t_i-of-N_i public user key. We say that any PPT adversary $\mathcal{A}$ with access to an oracle $\mathcal{O}(-)$ has had *subthreshold oracle access* to S if, for any public user key $Y \in \mathcal{M}^{\leftarrow}(S)$, if Y is a t_Y-of-N_Y shared public user key, then at most $t_Y - 1$ coalition members from $\mathcal{M}(Y)$ appear in the transcript between $\mathcal{A}$ and $\mathcal{O}(-)$.

Also for notational convenience, we call $\mathcal{T}(-)$ an oracle that inverts `TxnKeyGen` by taking as input a public transaction key Q_X and produces as output the user key X.

Definition 3.4.3. [Existential Unforgeability v. Adaptive Chosen Message and Subthreshold Insider Corruption] Let $\mathcal{A}$ be a PPT adversary and $L(-)$ be polynomials. $\mathcal{A}$ is given access to a signing oracle $\mathcal{SO}$, a corruption oracle $\mathcal{CO}_{\texttt{user}}$. Consider the following game:

(i) A set of user key pairs $S^* \subseteq \mathcal{K}_{\texttt{user}}$ is selected with $|S^*| = N(\lambda)$. The public keys in S^* are sent to $\mathcal{A}$.

(ii) $\mathcal{A}$ outputs a set of user key pairs $S \subseteq S^*$.

(iii) For each public user key $X_i \in S$, a public transaction key $Q_i^* \leftarrow \texttt{TxnKeyGen}(1^\lambda, X_i)$ is generated and the set $R^* := \{Q_i^*\}$ is constructed, randomly permuted, and then sent to $\mathcal{A}$.

(iv) $\mathcal{A}$ outputs a message M, a destination public user key $X \in \mathcal{K}_{\texttt{user}}$, ring $R \subseteq \mathcal{K}_{\texttt{txn}}$ of public transaction keys, and a signature σ. The game counts as a success if

  (a) $R \subseteq R^*$,

  (b) $\texttt{VER}(M, R, \sigma) = 1$,

  (c) for each index k in R, (M, X, R, k) does not appear in the queries between $\mathcal{A}$ and $\mathcal{SO}$

  (d) for each $Q_k \in R$, $\mathcal{CO}_{\texttt{user}}$ is not queried with the public user key $\mathcal{T}(Q_k)$, and

  (e) $\mathcal{A}$ has had subthreshold $\mathcal{CO}_{\texttt{user}}$ access to the set $\{\mathcal{T}(Q_k) \mid Q_k \in R\}$.

A scheme in which an adversary is only negligibly likely to succeed is said to be *existentially unforgeable with respect to adaptive chosen message attacks and subthreshold insider corruption* (or *st-EUF* for subthreshold existentially unforgeable).

4 Proposed Implementation

We provide an implementation of a restricted OT-LRTM scheme allowing only for $N - 1 \leq t \leq N$ in the spirit of the original CryptoNote methodology. User secret keys and public keys are both ordered pairs of keys, i.e. private key (a, b) and public key (A, B). Following terminology from [5], we refer to (a, A) as the *view keypair* and (b, B) and the *spend keypair*. We let $\Pi = (\texttt{Gen}^*, \texttt{Enc}^*, \texttt{Auth}^*, \texttt{Ver}^*, \texttt{Dec}^*)$ be a secure encrypt-then-authenticate scheme (where $\Pi_{\texttt{enc}} = (\texttt{Gen}^*, \texttt{Enc}^*, \texttt{Dec}^*)$ is a secure encryption sub-scheme and $\Pi_{\texttt{auth}} = (\texttt{Gen}^*, \texttt{Auth}^*, \texttt{Ver}^*)$ is a secure message authentication sub-scheme).

`UserKeyGen` generates the secret key $z = (a, b)$ by selecting a, b from an i.i.d. uniform distribution on $\mathbb{Z}_q$, and computing $Z = (A, B)$ with $A := aG$ and $B := bG$. `UserKeyGen` then outputs (z, Z).

`Merge` takes as input a threshold t and a set of key pairs $C = \{(z_1, Z_1), \ldots, (z_n, Z_N)\}$ such that $2 \leq N - 1 \leq t \leq N$ where each $Z_i = (A_i, B_i)$. If $N = 1$, `Merge` outputs Z_1. Otherwise:

(1) Each member of the coalition selects constants μ_i, γ_i for the multisig address.

(2) Each member derives a partial secret keypair (a_i^*, b_i^*) where $a_i^* = H_s(a_i, \mu_i)$ and $b_i^* = H_s(b_i, \gamma_i)$ and computes their associated public points $A_i^* = a_i^* G$, $B_i^* = b_i^* G$.

(3) If $t = N$, then the coalition uses Π to collaboratively compute the shared secret view key $a^* = \sum_i a_i^{*\ddagger}$, uses $\Pi_{\texttt{auth}}$ to collaboratively compute the shared public spend key $B^* = \sum_{i=1}^N B_i^*$. If $t = N - 1$, then

   (a) For each i, j, a partial shared secret view key $\alpha_{i,j} := H_s(a_i^* A_i^*)$ and a partial shared secret spend key $\beta_{i,j} := H_s(b_i^* B_i^*)$ is computed by either participant i or j.

   (b) Set $N^* := \frac{N(N+1)}{2}$, $S^* := \{((\alpha_{i,j}, \beta_{i,j}), (\alpha_{i,j} G, \beta_{i,j} G))\}_{1 \le i < j \le N}$, and run $\texttt{Merge}(N^*, S^*)$.

(4) Every coalition member now knows the shared secret view key a^* and the shared public spend key B^*.

`TxnKeyGen` takes as input a destination user public key (A, B), selects a random scalar r, computes $R = rG$ and $P = H_s(rA)G + B$, and outputs (R, P).

`ImageGen` takes as input a set of private user keys $y = \{(a_1, b_1), \ldots, (a_N, b_N)\}$ and a public transaction key (R, P). For each $i = 1, \ldots, N$, the i^{th} member of y computes partial key image $J_i = (H_s(a_i R) + b_i) H_p(R, P)$. The participating members use $\Pi_{\texttt{auth}}$ to compute $J = \sum_i J_i$.

`Sign` takes as input a message M, a destination public user key $(A_{\texttt{dest}}, B_{\texttt{dest}})$, a set of public transaction keys $\{(R_1, P_1), \ldots, (R_L, P_L)\}$, a secret index $1 \le k \le L$, and a set of t private keys $y = \{(a_i^*, b_i^*)\}_{i=1}^t$.

(1) The points $(R^*, P^*) \leftarrow \texttt{TxnKeyGen}(1^\lambda, (A_{\texttt{dest}}, B_{\texttt{dest}}))$ are computed.

(2) The owners of $y = \{(a_i^*, b_i^*)\}$ (the *signatories*) run $J \leftarrow \texttt{ImageGen}(1^\lambda, (R_k, P_k), y)$

(3) A set $\{s_{k+1}, s_{k+2}, \ldots, s_{k-1}\}$ of i.i.d. observations of uniform random variables are generated by the coalition and shared among the coalition using $\Pi_{\texttt{auth}}$.[§]

(4) For each j, the j^{th} signatory selects a random scalar u_j, computes $H_i := H_p(B_i)$ for each index $1 \le i \le L$, and computes the points $u_j G$ and $u_j H_k$. The coalition uses $\Pi_{\texttt{auth}}$ to collaboratively compute $u_k G := \sum_j u_j G$ and $u_k H_k := \sum_j u_j H_k$.

(5) Some threshold member computes

$$\begin{aligned} c_{k+1} &= H_p(m, u_k G, u_k H_k) \text{ and} \\ c_{i+1} &= H_p(m, s_i G + c_i B_i, s_i H_i + c_i J) \text{ for } i = k+1, k+2, \ldots, k-1. \end{aligned}$$

(6) The threshold member from the previous step uses $\Pi_{\texttt{auth}}$ to send c_k to all other signers with authentication. These signers may check that their received c_k matches their expected computations.

(7) If $t = N$, each signatory computes their personal $s_{k,j} := u_j - c_k b_j^*$. If $t = N - 1$, each signatory computes $s_{k,j} = u_j - c_k \sum_{i=1}^L z_{i,j}$. The coalition uses $\Pi_{\texttt{auth}}$ to collaboratively compute $s_k = \sum_j s_{k,j}$ and construct the signature $\sigma = (c_1, s_1, s_2, \ldots, s_L)$.

(8) Any signatory may now publish the signature-tag pair (σ, J) where $\sigma = (c_1, s_1, \ldots, s_N)$ together with the public transaction key (R^*, P^*).

[‡]Note that although secret information is about a_i not being directly shared with the coalition, the result of the computation is, in fact, a secret key, a^*.

[§]We recommend that a coordinating user randomly selects these using a cryptographic random number generator; only the user coordinating the signature needs these values.

Remark 4.0.1. The resulting signature takes the same form as LSAG signatures as in [2]. Modifying the above to appropriately to take into account key vectors provides the generalization to MLSAG signatures. Thus the verification algorithm for these signatures is identical to the verification algorithm for usual MLSAG signatures and we omit its description. Similarly, `Link` merely outputs a bit signifying whether two key images are identical, so we don't describe it further either.

Remark 4.0.2. Each u_j is kept secret from the other users and is generated randomly when the signature process begins. Certainly if u_j is revealed to another signatory, since the values of s_j and c_i are communicated in with authentication but not encryption, revealing the value $u_j - c_{i'}x_j$ can lead an observer to deduce x_j. Encryption does not solve the problem if threshold members are untrustworthy.

Similarly, if some value of u_j is re-used twice with the same private key, an observer can deduce the private key. Indeed, assuming we are using a hash function resistant to second pre-image attacks, the commitments from two signature processes $c_{i'}, c^*_{i'}$ are unequal except with negligible probability even if the other threshold members are colluding. Hence since $s_{i',j} = u_j - c_{i'}x_j$ and $s^*_{i',j} = u_j - c^*_{i'}x_j$, an observer may solve for the private key x_j. Don't re-use values of u_j, keep them secret, generate them randomly.

Remark 4.0.3. Note that users in $(N-1)$-of-N processes are prompted to select constants μ, γ multiple times for multiple sets of keys. If our hash function $H_s(-)$ is suitably secure, the lazy user can re-use the same constants μ and γ without concern; nevertheless, it is recommended that users do not re-use constants in `Merge`.

5 Security

Recall the critical fact proven in [4] that the sum of a uniform random variable with any indepenent random variable in $\mathbb{Z}/m\mathbb{Z}$ results in a uniform random variable (and conversely when m is prime). Hence, no PPT algorithm will be able to distinguish between a uniform random variable U and a sum of uniform random variables, $\sum_i U_i$.

Assume H_s, H_p in the OT-LRTM implementation from Section 4 are cryptographic hash functions under the random oracle model whose outputs are statistically indistinguishable from a uniform distribution except with non-negligible probability, and whose outputs are independent of one another. Assume `UserKeyGen` produces keys from a distribution that is statistically indistinguishable from a uniform distribution.

Theorem 5.0.1. The restricted OT-LRTM implementation from Section 4 is CIK.

Proof. Either the key pair $X_b = (A, B)$ received by $\mathcal{A}$ in step (iii) of Definition 3.2.1 is N-of-N for some $N \geq 2$, $(N-1)$-of-N for some $N \geq 3$, or a mere 1-of-1 user key. Of course, $(N-1)$-of-N key pairs are N^*-of-N^* key pairs. Thus, we really only need to deal with two cases: an N-of-N key pair with $N > 1$ or a 1-of-1 key pair.

If (A, B) is an N-of-N key pair, then $A^* = \sum_i H_s(a_i, \mu_i)G$ and $B^* = \sum_i H_s(b_i, \gamma_i)G$. Since H_s is a random oracle, any one of its outputs is uniformly random, and so any sum of its outputs is uniformly random [4], so no PPT adversary may determine the number of signatories. On the other hand, if (A, B) is a 1-of-1 key pair, then A and B are each independent uniform random variables from `UserKeyGen`, so no PPT algorithm can determine whether A or B is a sum or not.

□

Theorem 5.0.2. The OT-LRTM implementation from Section 4 is LSA-AGK.

Proof. $\mathcal{A}$ selects transaction public keys as ring members $(R_0, P_0) = (r_0 G, H_s(r_0 A_0)G + B_0)$ and $(R_1, P_1) = (r_1 G, H_s(r_1 A_1)G + B_1)$, a message M, a destination key pair $(A_{\texttt{dest}}, B_{\texttt{dest}})$ and receives a signature-tag pair (σ, J_b). $\mathcal{A}$ can compute $H_p(R_b, P_b)$ for each $b \in \{0, 1\}$, but without knowing the secrets a_b, b_b, computing $J_b = (H_s(a_b R_b) + b_b)H_p(R_b, P_b)$ for either b is infeasible for PPT $\mathcal{A}$. Moreover, σ has the same security properties as in [2].¶ □

Theorem 5.0.3. The OT-LRTM implementation from Section 4 is st-EUF.

Proof. Assume $\mathcal{A}$ is a PPT adversary that can succeed at the game in Definition **??** with non-negligible advantage. The adversary has sub-threshold access to the ring, $\mathcal{A}$ cannot execute `Sign` fairly and must attempt a forgery by generating additional random numbers. In this case, the adversary is merely attempting to forge a usual LSAG signature, and the security proof reduces to the one presented by [2].

□

6 Further Analysis

6.1 Efficiency and comparisons

The signatures resulting from the OT-LRTM scheme are indistinguishable from MLSAG signatures, so space complexity and verification time complexity of the OT-LRTM scheme is identical to that of MLSAGs. Due to the use of Π and several (possibly recursive) rounds of communication between coalition participants, efficiency of signatures is greatly reduced.

For a coalition consisting of 1-of-1 keys, `Merge` takes one round of communication inside the coalition with Π, `ImageGen` takes one round of communication inside the coalition with $\Pi_{\texttt{auth}}$, `Sign` takes three distinct rounds of communication with $\Pi_{\texttt{auth}}$ and calls `TxnKeyGen` and `ImageGen` each once. In total, this amounts to five rounds of communication inside the coalition per signature. For a coalition containing shared user keys for sub-coalitions, the rounds of communication in `Sign` must also take place inside each sub-coalition (and sub-sub-coalition, and so on).

In total, if we define the *depth* of 1-of-1 public user key as depth 0, and the *depth* of any t-of-N public user key X with coalition C as $\max\{\texttt{depth}(X) \mid X \in C\} + 1$. Then for a signature with a coalition of depth D where each coalition has at most N members, we require at most $2 + 3 \cdot D^N$ rounds of communication to fashion a signature.

6.2 Elaborations

We speculate that modifications to the CryptoNote-styled constructions of key images may allow for stronger notions of signer ambiguity in the future without sacrificing robustness against double-spend attacks. For example, by taking key images as homomorphic commitments, two commitments may be linked if their difference is a commitment to zero without revealing their masks. This is beyond the scope of this document.

Can the notion of CIK be expanded to CIKS so as to include signatures and chosen messages? With an LRTM, this expansion is nontrivial but straightforward. With an OT-LRTM, this becomes a delicate generalization due to the one-time transaction keys.

Special Thanks: We would like to issue a special thanks to the members of the Monero community who used the GetMonero.org Forum Funding System to support the Monero Research Lab. Readers may also

¶WORK ON THIS

regard this as a statement of conflict of interest, since our funding is denominated in Monero and provided directly by members of the Monero community by the Forum Funding System.

References

[1] Adam Bender, Jonathan Katz, and Ruggero Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In *TCC*, volume 6, pages 60–79. Springer, 2006.

[2] Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *ACISP*, volume 4, pages 325–335. Springer, 2004.

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

[4] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables. *Statistics & probability letters*, 18(4):313–314, 1993.

[5] Nicolas van Saberhagen. Cryptonote v 2. 0, 2013.