RESEARCH BULLETIN MRL-XXXX

Ring Threshold Multisignature Schemes and Security Models

Brandon Goodell* and Sarang Noether†
Monero Research Lab
October 22, 2017

Abstract

This research bulletin extends [4] 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 [3].

1 Introduction and Background

Collaboration to construct, compute, share, and leak a secret is a central goal in the construction of cryptographic systems. Although the idea of constructing a shared secret has a very long history, modern secret sharing through threshold cryptosystems found its most successful footing in [6]. These and related ideas (as in [7]) formed the historical foundation of secure multi-party computation, threshold encryption, and threshold authentication. For example, threshold multisignatures play a critical role in multi-factor message authentication in general and off-chain transactions for cryptocurrencies in particular, e.g. the Bitcoin Lightning Network. For a more modern treatment of threshold cryptosystems, see [2].

Ring signatures are digital signatures that are verifiably signed by one of possibly several public keys (*ring members*), and can play a critical role in promoting signer during message authentication. It is therefore natural to apply the notion of threshold cryptosystems to ring signatures for implementation in cryptocurrencies, resulting in signer-ambiguous threshold multisignatures.

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 *ring threshold multisignature* (RTM) scheme allows any set of N keys to collaborate as a coalition of signers with threshold t. The coalition is assigned a shared public key $X_{\mathtt{shared}}$ such that any t coalition members may collaborate to fashion a ring signature with $X_{\mathtt{shared}}$. The ring of signatories $\mathcal{Q}$ contains the key $X_{\mathtt{shared}}$, but an adversary cannot determine which element of Q computed the signature. This may be made into a *linkable* signature scheme using key images, and this may be made into a *one-time* signature scheme by inserting a one-time key exchange step between signing and the construction of $X_{\mathtt{shared}}$. 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.

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). A 1-of-N multisignature scheme can be trivially accomplished by handing out the same private key to each coalition member.

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. These signatures can be combined into a list to obtain

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

signature sizes that vary according to the number of signers, or can be combined in more sophisticated ways that are more private (see [2]). 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. We investigate some security definitions for a one-time linkable ring threshold multisignature (OT-LRTM) scheme.

1.1 Our Contribution

We consider a formal definition of one-time linkable ring threshold multisignature (OT-LRTM) schemes. We introduce the definition of *coalition-indistinguishable keys*, *signer ambiguity*, and *existential unforgeability* in these schemes. We describe a modified implementation of t-of-N linkable ring threshold multisignature (under the restriction $N-1 \leq t \leq N$ with $2 \leq t$ and $2 \leq N$) first described by previous Monero Research Lab (MRL) contributors Shen Noether in [4] and implemented for use in Monero by contributor `Luigi`. We prove that this implementation satisfies our security definitions. Lastly, we remark on some recent advancements in thresholdizing cryptoschemes from [2] using different security definitions.

1.2 Notation and Prerequisites

We let $\mathbb{G}$ be a group with prime order $\mathtt{q}$ and we let $G \in \mathbb{G}$ denote a commonly-known point with order $\mathtt{q}$. Let H_p and H_s be secure cryptographic hash-to-point and hash-to-scalar functions. 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).

2 MLSAG and Straightforward Threshold Set-ups

We briefly describe LSAG ring signatures in the sense of [3] and their MLSAG variant as used in Monero, and then a straightforward implementation of an LRTM scheme (which is one-time if the transaction keys are one-time).

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 $\mathtt{dest}(q, Q) = (y, Y)$. With a destination user key X, the user computes a new transaction public key $Q*$, constructs an appropriate message M, computes the key image $J = qH_p(Q)$, and selects a ring of public transaction keys $\mathcal{Q} = \{Q_1, \dots Q_L\}$ 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 computes $M^* = (M, J, Q^*, \mathcal{Q})$.

The signer selects a random secret scalar u, computes an initial temporary pair of points uG and uH_k, and computes an initial commitment $c_{k+1} := H_s(M^*, uG, uH_k)$. The signer sequentially proceeds through indices $i = k+1, k+2, \dots, L, 1, 2, \dots, k-1$ by selecting a random scalar s_i, computing the next pair of points $s_iG + c_iQ_i$ and $s_iH_i + c_iJ$, and computes the next commitment $c_{i+1} := H_s(M^*, s_iG + c_iQ_i, s_iH_i + c_iJ)$. 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 $s_k := u - c_kq_k$.

Now $\sigma = (c_1, s_1, \ldots, s_L)$ is the signature on M^*. A verifier checks this M^* signed by at least one member of $\mathcal{Q}$ in the following way. Given a message M^* and signature σ, the verifier parses $M^* = (M, J, Q^*, \mathcal{Q})$ and $\sigma = (c_1, s_1, \ldots, s_L)$. For each $i = 1, 2, \ldots, L$, 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 $c_{i+1} = H_s(M^*, z_i, z_i')$. After computing $c_2, c_3, \ldots, c_L, c_1$, 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 $\underline{q} = (q_1, \ldots, q_w)$, 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

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

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 [8], 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

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 $B = bG$. The private-public transaction key pair (q, Q) takes the form $Q = (S, P)$, $q = (s, p)$ where s is a scalar, $S = sG$, $p = H_s(aS) + b$, and $P = pG$[‡].

Given a message M, the coalition of user keys C compute their shared public key as $X_{\texttt{shared}} := (\sum_i A_i, \sum_i B_i)$, which is published. The coalition members secretly compute and share $a^* = \sum_i a_i$ and share this secret view key among themselves. With a destination user key X, the coalition computes a new transaction public key Q^*. 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 $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_L\}$ 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 $J_j = b_j H_p(Q)$, 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 $J = H_s(a^* S) + \sum_j J_j$. The coalition constructs an appropriate message $M^* = (M, J, Q^*, \mathcal{Q})$.

The coalition computes the 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

[‡]In the CryptoNote whitepaper, the private key p was denoted x, the random scalar s wa denoted r, and the random point $S = sG$ was denoted $R = rG$. The point $S = sG$ (originally denoted R) in the original CryptoNote whitepaper was the *public transaction key*, the value P was the *public output key*. This allowed for many outputs (each with their own signature) to have a common transaction output key. However, in MLSAG, we only use one output per signature anyway, so the value in distinguishing between S as a transaction key and P as an output key is diminished.

random values $s_{k+1}, \ldots, s_L, s_1, \ldots, s_{k-1}$. Any member in C may compute the commitments

$$c_{k+1} := H_s(M^*, u_k G, u_k H_k) \text{ and}$$
$$c_{i+1} := H_s(M^*, s_i G + c_i Q_i, s_i H_i + c_i J) \text{ for } i = k+1, \ldots, k-1.$$

All coalition members then use c_k to compute their personal $s_{k,j} = u_j - c_k b_j$. The signers share their $s_{k,j}$ with the other signers. Any threshold member may then compute the value $s_k = (\sum_j s_{k,j}) - c_k H_s(a^* S)$ and publishes as before. 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 set-up extends naturally to an $(N-1)$-of-N set-up. As before, a set of N public keys $\{X_1, X_2, \ldots, X_N\}$ form a coalition. Each pair of users has a shared secret scalar $z_{i,j} = H_s(x_i X_j)$ with associated point $Z_{i,j} = z_{i,j} G$. 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 members in C do not directly learn the private transaction key $q = H_s(a^* S) + \sum_j b_j$ and do not directly reveal their own private *spend* keys b_j (although they collaborate to compute their shared private view key a^*. 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.

Remark 2.2.2. Note that the same basic extension works for coalitions containing coalitions also: each time a member of a coalition must make a decision in a signing protocol, the coalition may simply compute a sum. 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 2.2.3. 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. Moreover, if the computation of sums is done inappropriately, the summands may be accidentally revealed. Using hash functions and encrypt-then-authenticate communication, we may resolve these problems.

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). In the case of CryptoNote keys of the form $((a,b),(A,B))$, select constants μ_j, γ_j and set $a^* = H_s(a, \mu_j)$, $b^* = H_s(b, \gamma_j)$.

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. Also note that the above does not take into account the possibility of malicious coalition members.

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`, `Merge`, `TxnKeyGen`, `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) $\texttt{Sign}(M, X, \mathcal{Q}, k, y)$ takes as input message M, a destination user key X, ring of public transaction keys $\mathcal{Q} = \{Q_1, \ldots, Q_L\}$, secret index k, and a set of private user keys y. A transaction key Q^* addressed to X is computed $Q^* \leftarrow \texttt{TxnKeyGen}(1^\lambda, X)$, a key image J is generated, and a modified message $M^* = (M, J, Q^*, \mathcal{Q})$ is constructed. `Sign` outputs (σ, M^*).

(v) $\texttt{Verify}(M^*, \sigma)$ takes as input a message $M^* = (M, J, Q^*, \mathcal{Q})$ and a signature σ, and outputs a bit $b \in \{0, 1\}$.

(vi) $\texttt{Link}((M_0^*, \sigma_0), (M_1^*, \sigma_1))$ takes as input two signatures on two messages and 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 $2 \leq t$ and $2 \leq N$ and $N - 1 \leq t \leq N$ do not hold then `Merge` outputs $\perp$ instead of a key.

Definition 3.1.2. Assume for each $i \in \{0, 1\}$ that M_i is an arbitrary message, X_i is an arbitrary t_i-of-N_i shared public user key with coalition C_i, $\mathcal{Q}_i = \{Q_{i,j}\}_{j=1}^{|\mathcal{Q}_i|}$ is an arbitrary ring of public transaction keys with associated secret index 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 is generated fairly from $\sigma_i \leftarrow \texttt{Sign}(M_i, X_i, \mathcal{Q}_i, k_i, y_i)$. Further say $M_i^* = (M_i, J_i, Q_i^*, \mathcal{Q}_i)$. We say an OT-LRTM scheme is *complete* if

(a) $\texttt{VER}(M_i^*, \sigma_i) = 1$ and

(b) if $Q_{i,k_i} = Q_{j,k_j}$ then $\texttt{LNK}((M_i^*, \sigma_i), (M_j^*, \sigma_j)) = 1$.

Recall Remark 2.2.3 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 values $s_{k,j} = u_j - c_k b_j$, and then finally $s_k = (\sum_j s_{k,j}) - c_k H_s(a^* S)$. Denote the coalition of user key pairs for $X_{\texttt{shared}} = (A^*, B^*)$ as $\{((a_j, b_j), (A_j, B_j))\}$. The coalition for $X_{\texttt{shared}}$ may use Π to share their $(H_s(a_j S) + b_j) \cdot H_p(Q)$ and compute the key image $J = (H_s(a^* S) + \sum_j b_j) H_p(Q)$. 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, $J_{j,i} = b_{j,i} H_p(Q)$, this sub-coalition uses Π to compute $J_j = (H_s(a_j S) H_p(Q) + \sum_i J_{j,i}$, and the sub-coalition outputs J_j.

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 them to the rest of the coalition with Π. 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. In Definition 3.2.1, we formalize the notion that keys cannot be feasibly determined by a PPT algorithm to be threshold keys or 1-of-1 keys. If an OT-LTRM scheme is secure under that definition, 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 above to prevent recursive formation of coalition keys is not feasible.

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 t_i, N_i such that $2 \leq t_i \leq N_i \leq L(\lambda)$. 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 $C_1 \subseteq S$, $\tau_1 \in \mathbb{N}$, $\tau_1 \leq |C_1|$, $\tau_0 \neq \tau_1$, 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. 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 and has the ability to compute discrete logs, then $\mathcal{A}$ cannot successfully run `Merge` for each threshold 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 corruption and discrete log oracle access 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 $(M, X, \mathcal{Q}, k)$ (a message, a destination public user key, a ring of public transaction keys, and an index k) and outputs a valid signature-tag pair $(\sigma, J) \leftarrow \texttt{Sign}(M, X, \mathcal{Q}, k, y)$ for some set y of private user keys.

Definition 3.3.1. [Linkable Signer Ambiguity v. Adversarially Generated Keys] Let $N(-)$ 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^* addressed to X_i is generated. The set $\mathcal{Q}^* := \{Q_i^*\} \subseteq \mathcal{K}_{\texttt{txn}}$ is constructed, randomly permuted, and then sent to $\mathcal{A}$.

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

(v) A random bit b is chosen. The message $M^* = (M, J, Q^*, \mathcal{Q})$ and signature $\sigma \leftarrow \mathcal{SO}(M^*, X, \mathcal{Q}, i_b)$ are sent to $\mathcal{A}$.

(vi) $\mathcal{A}$ outputs a bit b'. The game counts as a success if (a) $b = b'$ and (b) if $(M', X', \mathcal{Q}', i)$ is any query from $\mathcal{A}$ to $\mathcal{SO}$, then the i^{th} element of $\mathcal{Q}'$ 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) that the transcript for $\mathcal{A}$ reveals the key images for $\{Q_{i_0}, Q_{i_1}\}$ for the first time in step (v).

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.

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 $Y \in \texttt{Merge}^{\leftarrow}(S)$, if Y is a t_Y-of-N_Y shared public user key, then at most $t_Y - 1$ coalition members from $\texttt{Merge}^{-1}(Y)$ appear in the transcript between $\mathcal{A}$ and $\mathcal{O}(-)$.

Also for notational convenience, we call $\texttt{TxnKeyGen}^{-1}(-)$ an oracle that inverts `TxnKeyGen` by taking as input a public transaction key Q_X and produces as output the user key X. Let $\mathcal{CO}(-)$ be a corruption oracle that takes as input a public user key and outputs the associated private user key.

Definition 3.4.3. [Existential Unforgeability v. Adaptive Chosen Message and Subthreshold Insider Corruption] Let $\mathcal{A}$ be a PPT adversary and $N(-)$ be a 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 $\mathcal{Q}^* := \{Q_i^*\}$ 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 $\mathcal{Q} \subseteq \mathcal{K}_{\texttt{txn}}$ of public transaction keys, and a signature σ. The game counts as a success if

  (a) $\mathcal{Q} \subseteq \mathcal{Q}^*$,

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

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

  (d) for each $Q_k \in \mathcal{Q}$, $\mathcal{CO}_{\texttt{user}}$ is not queried with the public user key $\texttt{TxnKeyGen}^{-1}(Q_k)$, and

  (e) $\mathcal{A}$ has had subthreshold $\mathcal{CO}_{\texttt{user}}$ access to the set $\left\{\texttt{TxnKeyGen}^{-1}(Q_k) \mid Q_k \in R\right\}$.

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. We assume a coordinating member of the coalition is constructing the signature and has persuaded threshold members to participate. 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 [8], 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).

(I) `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).

(II) `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 $2 \leq t$ and $2 \leq N$ and $N-1 \leq t \leq N$ where each $Z_i = (A_i, B_i)$, and outputs a shared public user key $Z_{\texttt{shared}}$.

(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 $A_i^* = a_i^* G$ and $B_i^* = b_i^* G$.

(3) If $t = N$, then each coalition member uses Π to send a_i^* to the coordinating user, who computes the shared secret view key $a^* = \sum_i a_i^*$[§] The coordinating user computes the shared public spend key $B^* = \sum_{i=1}^{N} B_i^*$. The coordinating user outputs (A^*, B^*), which may be made public.

(4) If $t = N - 1$, then

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

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

(III) `TxnKeyGen` takes as input a set of destination user public keys $\{(A_i, B_i)\}_i$. For each destination, the coordinating user selects a random scalar s_i, computes $S_i = s_i G$ and $P_i = H_s(s_i A_i)G + B_i$, outputs the (public) pairs $\{(S_i, P_i)\}_i$.

(IV) `Sign` takes as input a message M, a destination public user key $(A_{\texttt{dest}}, B_{\texttt{dest}})$, a set of input public transaction keys $\{(S_i, P_i)\}_{i=1}^{I}$, a set of output public transaction keys $\left\{(\widetilde{S}_d, \widetilde{P}_d)\right\}_{d=1}^{D}$, a secret index $1 \leq i^* \leq I$, and a set of t user private keys $y = \{(a_n^*, b_n^*)\}_{n=1}^{t}$ such that $a_n^* = a_m^*$ for each n, m.

(1) The coalition computes $(S^*, P^*) \leftarrow \texttt{TxnKeyGen}(1^\lambda, (A_{\texttt{dest}}, B_{\texttt{dest}}))$.

(2) For each $(a_i, b_i) \in y$, the points $b_i H_p(S_k, P_k)$ are computed and communicated with Π to the coalition. The sum $\sum_i b_i H_p(S_k, P_k)$ is computed together with the key image $J = (H_s(a_1^* S_k) + \sum_i b_i) H_p(S_k, P_k)$ by any member of the coalition and shared with the coalition using Π.

(3) The coalition computes $M^* = (M, J, S^*, P^*, \mathcal{Q})$.

(4) 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 Π.

[§]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^*.

(5) For each j, the j^{th} signatory selects a random scalar $u_{k,j}$, computes $H_i := H_p(B_i)$ for each index $1 \leq i \leq L$, and computes the points $u_{k,j}G$ and $u_{k,j}H_k$. The coalition uses Π to collaboratively compute $u_kG := \sum_j u_{k,j}G$ and $u_kH_k := \sum_j u_{k,j}H_k$.

(6) Some threshold member computes

$$\begin{aligned} c_{k+1} =& H_p(M^*, u_kG, u_kH_k) \text{ and} \\ c_{i+1} =& H_p(M^*, s_iG + c_iP_i, s_iH_i + c_iJ) \text{ for } i = k+1, k+2, \ldots, k-1. \end{aligned}$$

(7) The threshold member from the previous step uses Π to send c_k to all other signers with authentication. These signers may check that their received c_k matches their expected computations.

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

(9) Any signatory may now publish the signature $\sigma = (c_1, s_1, \ldots, s_N)$, and any verifier may use the usual MLSAG verification to check that some member of the ring $\{(S_i, P_i)\}$ fashioned the signature.

Remark 4.0.1. The resulting signature takes the same form as LSAG signatures as in [3]. 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`.

Remark 4.0.4. When a transaction is relayed on the Monero network, the transaction key pair (S, P) is scanned by any of the coalition members. Any of these members can use their shared secret view key a^* to check whether $B^* = P - H_s(a^*S)G$. If so, they can execute `Sign` to sign a message with the private key p without revealing it.

5 Security

Recall the critical fact proven in [5] 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. If H_s is a secure hash function under the random oracle model, 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 the keys defined in `Merge` are $A^* = \sum_i H_s(a_i, \mu_i)G$ 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 [5], 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. If the Decisional Diffie Hellman hardness assumption holds in $\mathbb{G}$, then the OT-LRTM implementation from Section 4 is LSA-AGK.

Proof. Assume $\mathcal{A}$ has a non-negligible advantage in the game of Definition 3.3.1 played with the above implementation, namely let us say there exists some polynomial $f_1(\lambda)$ such that

$$\mathbb{P}\left[\mathcal{A} \text{ succeeds in Definition 3.3.1}\right] > \frac{1}{2} + \frac{1}{f_1(\lambda)}$$

We construct an algorithm $\mathcal{M}$ that gains a non-negligible advantage in solving the DDH game by running $\mathcal{A}$ as a subroutine, leading us to a contradiction. A base point G is chosen and given to $\mathcal{M}$. Random scalars r, s, and t are selected, a random bits $c \in \{0, 1\}$ are chosen. The points $Q_0 = tG$, $Q_1 = rsG$, $Q_2 = rG$, $Q_3 = sG$ are computed. The points (Q_c, Q_2, Q_3) are given to $\mathcal{M}$, who outputs a bit c'. The game counts as a success if $c = c'$.

The algorithm $\mathcal{M}$ receives (Q_1^*, Q_2^*, Q_3^*) and executes part (i) from Definition 3.3.1 by calling $\texttt{UserKeyGen}(1^\lambda)$, and calls $\mathcal{A}$ to execute (ii) as usual. In (iii), $\mathcal{M}$ constructs $\mathcal{Q}^*$ to include Q_i^* for $i = 1, 2, 3$ in the following way. First, $\mathcal{M}$ instructs $\mathcal{A}$ to select three user public keys $X_1, X_2, X_3 \in S$, picks a random permutation $\tau \in S_3$, and assigns $\texttt{dest}(Q_i^*) := X_{\tau(i)}$ for each $i = 1, 2, 3$. Next, $\mathcal{M}$ calls `TxnKeyGen` using the other members of S as usual to build $\mathcal{Q}^*$. Denote the index for Q_i^* in $\mathcal{Q}^*$ as j_i for $i = 1, 2, 3$. Now $\mathcal{M}$ instructs $\mathcal{A}$ to select $i_0 = j_1$ and $i_1 \in \{j_2, j_3\}$. Steps (v) and (vi) execute as usual. If $\mathcal{A}$ succeeds, then $\mathcal{M}$ outputs $c' = 1$, otherwise selects a random bit as an output. We have the following from the law of total probability.

$$\mathbb{P}\left[\mathcal{M} \text{ succeeds at DDH}\right] = \mathbb{P}\left[\mathcal{M}(Q_1^*, Q_2^*, Q_3^*) = c \mid c = 0\right]\mathbb{P}\left[c = 0\right] + \mathbb{P}\left[\mathcal{M}(Q_1^*, Q_2^*, Q_3^*) = c \mid c = 1\right]\mathbb{P}\left[c = 1\right]$$

Certainly, $\mathbb{P}\left[c = 0\right] = \mathbb{P}\left[c = 1\right] = \frac{1}{2}$. If $c = 0$, then $\mathcal{A}$ did not succeed so $\mathcal{M}$ can only guess. In this case, $\mathbb{P}\left[\mathcal{M}(Q_1^*, Q_2^*, Q_3^*) = c \mid c = 0\right] = \frac{1}{2}$. We have

$$\mathbb{P}\left[\mathcal{M} \text{ succeeds at DDH}\right] = \frac{1}{4} + \frac{1}{2}\mathbb{P}\left[\mathcal{M}(Q_1^*, Q_2^*, Q_3^*) = c \mid c = 1\right]$$

Moreover, $\mathcal{M}$ outputs 1 if and only if $\mathcal{A}$ has succeeded:

$$\begin{aligned}\mathbb{P}\left[\mathcal{M} \text{ succeeds at DDH}\right] =& \frac{1}{4} + \frac{1}{2}\mathbb{P}\left[\mathcal{A} \text{ succeeds in Definition 3.3.1}\right] \\ >& \frac{1}{4} + \frac{1}{2}\left(\frac{1}{2} + \frac{1}{f(\lambda)}\right) \\ >& \frac{1}{2} + \frac{1}{2f(\lambda)}\end{aligned}$$

□

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 [3].

□

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

In [2], a very general fully homomorphic thresholdizing set-up is described, leading to fully homomorphic threshold signatures and encryption. The threshold signature scheme defined in [2] generates a unique verification key (akin to our transaction keys) for each coalition and access structure, so the scheme in [2] vacuously satisfies CIK. On the other hand, in our proposed implementation, keys are being combined into new keys, so it is critical to check that combined keys do not reveal anything about the participating members.

Coalition indistinguishability on keys is related to two distinct security notions defined in [2], where a highly general fully homomorphic thresholdizing scheme using only one round of communication is presented.

Threshold signatures described in [2] are required to be *anonymous* and *succinct*. Anonymity in [2] is defined such that an adversary should be unable to discern which subset of the coalition produced the signature, and succinctness is defined such that signature size is dependent only on the security parameter.

Our proposed implementation is certainly *succinct* because it produces a signature in the format of a usual MLSAG signature, regardless of the details of the coalition. Our scheme is anonymous because all signatures are, without loss of generality, N-of-N, so each signature is fashioned with a full subset.

A natural question is to ask whether the definition of CIK can be expanded to coalition indistinguishability on keys and signatures (say CIKS) by demanding that signatures also provide no information about the signing subset or signing coalition. With an LRTM, an expansion of Definition 3.2.1 to take into account an adaptive chosen message attack is nontrivial but straightforward. A CIKS LRTM scheme is necessarily succinct and anonymous according to the definition in [2]. Moreover, a succinct and anonymous threshold signature scheme from [2] that generates verification keys for each coalition randomly is also CIKS. In this sense, CIKS is equivalent to succinct anonymity.

Additional helpful definitions appear in [2]. For example, a threshold cryptoscheme may be regarded as *private* if no subthreshold collection of key shares can reveal any information about the shared secrets, and *robust* if maliciously generated evaluation key shares can always be detected. Since our implementation never reveals the shared secret key even with a threshold number of coalition members participating, our implementation is private. However, in the presence of maliciously generated choices in the signature process, we have no guarantee of robustness: although a malicious subthreshold party will only enjoy a negligible advantage in constructing a forgery, their attempt will merely appear to be an invalid signature.

Special Thanks: We would like to issue a special thanks to the members of the Monero community who used the `GetMonero.org` Community Forum Funding System to support the Monero Research Lab. Readers may also 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] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption.

[3] 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.

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

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

[6] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[7] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981.

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