RESEARCH BULLETIN MRL-0005

Ring Confidential Transactions

February 2016

Shen Noether*, Adam Mackenzie and Monero Core Team

*Correspondence:
lab@getmonero.org
shen.noether@gmx.com
Monero Research Lab

Abstract

This article introduces a method of hiding transaction amounts in the strongly decentralized anonymous cryptocurrency Monero. Similar to Bitcoin, Monero is a cryptocurrency which is distributed through a proof of work "mining" process. The original Monero protocol was based on CryptoNote, which uses ring signatures and one-time keys to hide the destination and origin of transactions. Recently the technique of using a commitment scheme to hide the amount of a transaction has been discussed and implemented by Bitcoin Core Developer Gregory Maxwell. In this article, a new type of ring signature, A Multi-layered Linkable Spontaneous Anonymous Group signature is described which allows for hidden amounts, origins and destinations of transactions with reasonable efficiency and verifiable, trustless coin generation. Some extensions of the protocol are provided, such as Aggregate Schnorr Range Proofs, and Ring Multisignature. The author would like to note that early drafts of this were publicized in the Monero Community and on the bitcoin research irc channel. Blockchain hashed drafts are available in [14] showing that this work was started in Summer 2015, and completed in early October 2015. An eprint is also available at http://eprint.iacr.org/2015/1098.

1 Introduction

1.1 Spontaneous (Ad Hoc) Ring Signatures in CryptoCurrencies

Recall that in Bitcoin each transaction is signed by the owner of the coins being sent and these signatures verify that the owner is allowed to send the coins. This is entirely analogous to the signing of a check from your bank.

CryptoNote [16] and Ring Coin [2] advanced this idea by using "ring signatures" which were originally described in [15] as a "digital signature that specifies a group of possible signers such that the verifier can't tell which member actually produced the signature." The idea therefore is to have the origin pubkey of a transaction hidden in a group of pubkeys all of which contain the same amount of coins, so that no one can tell which user actually sent the coins.

The original CryptoNote protocol as described in [16] implements a slight modification of this to prevent double spends. Namely in [16] a "traceable ring signature," which is a slight modification of those described in [6] is employed. This type of ring signature has the benefit of not allowing the owner of a coin to sign two different ring signatures with the same pubkey without being noticed on the blockchain. The obvious reason for this is to prevent "double-spending" which, in Bitcoin, refers to spending a coin twice. Ring coin [2,3] uses a more efficient linkable ring signature which is a slight modification of the Linkable Spontaneous Anonymous Group signatures described in [8].

One benefit of using the above types of ring signatures over other anonymizing techniques, such as CoinJoin [10] or using coin mixing services, is that they allow for "spontaneous" mixing. With CoinJoin or coin mixers, it is similarly possible to hide the originator of a given transaction, however these techniques in practice need some sort of centralized group manager, such as a centralized CoinJoin server, where transactions are combined by a trusted party. In the case that the trusted party is compromised, the anonymity of the transaction is also compromised.

Some coins such as Dash (originally called Darkcoin) [5], attempt to negate this by using a larger number of trusted mixers (in this case masternodes) but this number is still much smaller than the users of the coin. In contrast, with a spontaneous ring signature, transactions can be created by the owner of a given pubkey (this is the spontaneous, or "ad-hoc" property) without relying on any trusted server, and thus providing for safer anonymity.

One possible attack against the original CryptoNote or ring-coin protocol [2,16] is blockchain analysis based on the amounts sent in a given transaction. For example, if an adversary knows that .9 coins have been sent at a certain time, then they may be able to narrow down the possibilities of the sender by looking for transactions containing .9 coins. This is somewhat negated by the use of the one-time keys used in [16] since the sender can include a number of change addresses in a transaction, thus obfuscating the amount which has been sent with a type of "knapsack mixing." However this technique has the downside that it can create a large amount of "dust" transactions on the blockchain, i.e. transactions of small amounts that take up proportionately more space than their importance. Additionally, the receiver of the coins may have to "sweep" all this dust when they want to send it, possibly allowing for a smart adversary to keep track of which keys go together in some manner. Furthermore, it is easy to establish an upper and lower bound on the amounts sent.

Another downside to the original CryptoNote set-up is that it requires a given pair of (P, A) of pubkey P and amount A to be used in a ring signature with other pubkeys having the same amount. For less common amounts, this means there may be a smaller number of potential pairs (P', A') available on the blockchain with $A' = A$ to ring signature with. Thus, in the original CryptoNote protocol, the potential anonymity set is perhaps smaller than may be desired. Analysis of the above weaknesses is covered in [9].

1.2 Ring CT for Monero

An obvious way to negate the downsides of the CryptNote protocol, as described in the previous section, would be to implement hidden amounts for any transaction. In this paper, I describe a modification to the Monero protocol, a proof-of-work cryptocurrency extending the original CryptoNote protocol, which allows the amounts sent in a transaction to be hidden. This modification is based on the Confidential Transactions [11] which are used on the Elements side-chain in Bitcoin, except it allows for their use in ring signatures. Therefore, the modification is given the obvious name of Ring Confidential Transactions for Monero.

In order to preserve the property that coins cannot be double spent, a generalization of the LSAG's of [8] is described, a Multilayered Linkable Spontaneous Anonymous Group Signature (MLSAG) which allows for combining Confidential Transactions with a ring signature in such a way that using multiple inputs and outputs is possible, anonymity is preserved, and double-spending is prevented. The author notes that an essentially similar protocol was proposed by Connor Frenkenecht about a month after the second drafts of this were originally publicized.

1.3 Strongly Decentralized Anonymous Payment Schemes

The Ring CT protocol allows hidden amounts, origins, and destinations for transactions which is somewhat similar to Zerocash [4]. One possible differentiator is that the use of proof of work for coin generation is possible with Ring CT as opposed to in ZeroCash, where it seems all coins must be pregenerated by a trusted group.

Note that one of the biggest innovations in Bitcoin [13], was the decentralized distribution model allowing anyone willing to put their computing power to work to participate in the generation of the currency. Some of the benefits of this type of proof-of-work include trustless incentives for securing the network and stronger decentralization (for example, to protect against poison-pill type attacks).

One final obvious benefit of the proof-of-work coin generation is it makes Ring CT immune to a powerful actor somehow acquiring all the pieces of the master key used in coin generation. Since there is an obvious large incentive (the ability to generate free money [1]) to acquire all pieces of the trusted generation key, this is fairly important.

1.4 Acknowledgements

I would like to thank Monero team for lots of help and discussion in the creation of this paper and the Monero and Bitcoin Community for support and discussion. With respect to disclosure, the author received several donations totalling between 2 and 3 bitcoins from the Monero community in gratitude for his work on this research.

2 Multilayered Linkable Spontaneous Anonymous Group Signatures

In this section, I define the Multilayered Linkable Spontaneous Anonymous Group signatures (MLSAG) used by the the Ring CT protocol. Note that I define these as a general signature, and not necessarily in their use case for Ring Confidential Transactions. An MLSAG is essentially similar to the LSAG's described in [8], but rather than having a ring signature on a set of n keys, instead, an MLSAG is a ring signature on a set of n key-vectors.

Definition 2.1 A **key-vector** is just a collection $\overline{y} = (y_1, ..., y_r)$ of public keys with corresponding private keys $\overline{x} = (x_1, ..., x_r)$.

[1]The author previously had assumed this would allow the unmasking of transactions as well, but the newer ZeroCash paper claims this is not possible.

2.1 LWW signatures vs FS signatures

The ring signatures used in Monero and the original CryptoNote protocol are derived from the traceable ring signatures of [6]. The CryptoNote [16] ring signatures come with a "key-image" which means that a signer can only sign one ring on the block-chain with a given public and private key pair or else their transaction will be marked as invalid. Because of this, one-time keys are used in CryptoNote, which further helps anonymity.

In [3], Adam Back noticed that the Linkable Spontaneous Anonymous Group (LSAG) signatures of [8] can be modified to give a more efficient linkable ring signature producing the same effect as the [6] ring signatures. This modification reduces the storage cost on the blockchain essentially in half.

First I recall almost verbatim the modification given in [3]:

Keygen: Find a number of public keys $P_i, i = 0, 1, ..., n$ and a secret index j such that $xG = P_j$ where G is the ed25519 base-point and x is the signers spend key. Let $I = xH_p(P_j)$ where H_p is a hash function returning a point [2] Let $\mathfrak{m}$ be a given message.

SIGN: Let $\alpha, s_i,\ i \neq j,\ i \in \{1, ..., n\}$ be random values in $\mathbb{Z}_q$ (the ed25519 base field).

Compute

$$L_j = \alpha G$$

$$R_j = \alpha H_p(P_j)$$

$$c_{j+1} = h(\mathfrak{m}, L_j, R_j)$$

where h is a hash function returning a value in $\mathbb{Z}_q$. Now, working successively in j modulo n, define

$$L_{j+1} = s_{j+1}G + c_{j+1}P_{j+1}$$

$$R_{j+1} = s_{j+1}H_p(P_{j+1}) + c_{j+1} \cdot I$$

$$c_{j+2} = h(\mathfrak{m},\ L_{j+1},\ R_{j+1})$$

$$\ldots$$

$$L_{j-1} = s_{j-1}G + c_{j-1}P_{j-1}$$

$$R_{j-1} = s_{j-1}H_p(P_{j-1}) + c_{j-1} \cdot I$$

$$c_j = h(\mathfrak{m}, L_{j-1},\ R_{j-1})$$

[2]In practice $H_p(P) = Keccak(P) \cdot G$ where G is the ed25519 basepoint, although note that for the commitment scheme I will use $toPoint(Keccak(P))$, hashing successively until $Keccak(P)$ returns a multiple of the basepoint.

so that $c_1, ..., c_n$ are defined.

Let $s_j = \alpha - c_j \cdot x_j \ mod \ l$, ($l$ being the ed25519 curve order) hence $\alpha = s_j + c_j x_j \ mod \ l$ so that

$$L_j = \alpha G = s_j G + c_j x_j G = s_j G + c_j P_j$$

$$R_j = \alpha H_p(P_j) = s_j H_p(P_j) + c_j I$$

and

$$c_{j+1} = h(\mathfrak{m},\ L_j,\ R_j)$$

and thus, given a single c_i value, the P_j values, the key image I, and all the s_j values, all the other $c_k,\ k \neq i$ can be recovered by an observer. The signature therefore becomes:

$$\sigma = (I, c_1, s_1, ..., s_n)$$

which represents a space savings over [16, 4.4] where the ring signature would instead look like:

$$\sigma = (I, c_1, ..., c_n, s_1, ..., s_n)$$

Verification proceeds as follows. An observer computes L_i, R_i, and c_i for all i and checks that $c_{n+1} = c_1$. Then the verifier checks that

$$c_{i+1} = h(\mathfrak{m}, L_i, R_i)$$

for all $i \mod n$

LINK: Signatures with duplicate key images I are rejected.

Note that proofs of unforgeability, anonymity, and linkability hold for the above protocol which are only insignificant modifications to the proofs given in [8]. I will give a more generalized version of these proofs for the MLSAG's.

2.2 MLSAG Description

For the Ring CT protocol, which will be described in section 4, I require a generalization of the Back LSAG signatures described in the previous section which allows for key-vectors (Definition 2.1) rather than just keys.

Suppose that each signer of a (generalized) ring containing n members has exactly m keys $\left\{P_i^j\right\}_{j=1,...,m}^{i=1,...,n}$. The intent of the MLSAG ring signature is the following:

- To prove that one of the n signers knows the secret keys to their entire key vector.
- To enforce that if the signer uses any one of their m signing keys in another MLSAG signature, then the two rings are linked, and the second such MLSAG signature (ordered by the Monero block chain) is discarded.

The algorithm proceeds as follows: Let $\mathfrak{m}$ be a given message. Let π be a secret index corresponding to the signer of the generalized ring. For $j = 1, ..., m$, let $I_j = x_j H\left(P_\pi^j\right)$, and for $j = 1, ..., m$, $i = 1, ..., \hat{\pi}, ...n$ (where $\hat{\pi}$ means omit the index π) let s_i^j be some random scalars (elements of $\mathbb{Z}_q$). Now, in a manner analogous to subsection 2.1, define

$$L_\pi^j = \alpha_j G$$

$$R_\pi^j = \alpha_j H\left(P_\pi^j\right)$$

for random scalars α_j and $j = 1, ..., m$. Now, again analogously to section 2.1, set:

$$c_{\pi+1} = H\left(\mathfrak{m}, L_\pi^1, R_\pi^1, ..., L_\pi^m, R_\pi^m\right).$$

$$L_{\pi+1}^j = s_{\pi+1}^j G + c_{\pi+1} P_{\pi+1}^j$$

$$R_{\pi+1}^j = s_{\pi+1}^j H\left(P_{\pi+1}^j\right) + c_{\pi+1} I_j$$

and repeat this, incrementing i modulo n until we arrive at

$$L_{\pi-1}^j = s_{i-1}^j G + c_{i-1} P_{i-1}^j$$

$$R_{\pi-1}^j = s_{i-1}^j H\left(P_{i-1}^j\right) + c_{i-1} \cdot I_j$$

$$c_\pi = H\left(\mathfrak{m}, L_{\pi-1}^1, R_{\pi-1}^1, ..., L_{\pi-1}^m, R_{\pi-1}^m\right).$$

Finally, solve for each s_π^j using $\alpha_j = s_\pi^j + c_\pi x_j \mod \ell$. The signature is then given as $\left(I_1, ..., I_m, c_1, s_1^1, ..., s_1^m, s_2^1, ..., s_2^m, ..., s_n^1, ..., s_n^m\right)$, so the complexity is $O\left(m\left(n+1\right)\right)$. Verification proceeds by regenerating all the L_i^j, R_i^j starting from $i = 1$ as in section 2.1 (which is the special case that $m = 1$) and verifying the hash $c_{n+1} = c_1$. If these are being used in a blockchain setting such as Monero, signatures with key images I_j which have already appeared are then rejected. One can easily show, in a manner similar to [8]:

- The probability of a signer generating a valid signature without knowing all "m" private keys belonging to their key vector for index π is negligible.
- The probability of a signer not signing for any key of index π is negligible. (In other words, the key images in the signature necessarily all come from index π.)
- If a signer signs two rings using at least one of the same public keys, then the two rings are linked.

I expand on these points below with security proofs.

2.3 MLSAG Security Model

An MLSAG will satisfy the following three properties of Unforgeability, Linkability, and Signer Ambiguity which are very similar to the definitions given in [8].

Definition 2.2 (Unforgeability) An MLSAG signature scheme is unforgeable if for any probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ with signing oracle $\mathcal{SO}$ producing valid signatures, given a list of n public key vectors chosen by $\mathcal{A}$, then $\mathcal{A}$ can only with negligible probability produce a valid signature when $\mathcal{A}$ does not know one of the corresponding private key vectors.

Remark 2.3 In the following definition, note that I include rejecting duplicate key images as part of the verification criteria for the MLSAG, which gives a slightly different Linkability definition than the one in [8].

Definition 2.4 (Linkability) Let L be the set of all public keys in a given setting (e.g. in a given blockchain). An MLSAG signature scheme on L is key-image linked if the probability of a PPT adversary $\mathcal{A}$ creating two signatures σ, σ' signed with respect to key-vectors $\overline{y}$ and $\overline{y}'$ each containing the same public key $y_i = y_i'$ in L and each verifying without being marked duplicate, is negligible.

Definition 2.5 (Signer Ambiguity) An MLSAG signature scheme is said to be signer ambiguous if given any verifying signature σ on key-vectors $(\overline{y}_1, ..., \overline{y}_n)$ and any set of t private keys, none of the same index, nor of the secret index, then the probability of guessing the secret key is less than $\frac{1}{n-t} + \frac{1}{Q(k)}$

Proofs of the above definitions for the MLSAG signatures are given in the appendix.

3 Background on Confidential Transactions

3.1 Confidential Transactions in Bitcoin

In [11], Greg Maxwell describes Confidential Transactions which are a way to send Bitcoin transactions with the amounts hidden. The basic idea is to use a Pedersen Commitment and the method is well described in the cited source. In this paper I make a slight modification the the Confidential Transactions machinery in that rather than taking the commitments to sum to zero, I instead sign for the commitment, to prove I know a private key. This is described in more detail in the next section.

3.2 Modification for Ring Signatures

Let G be the ed25519 basepoint. Let[3]

$$H = toPoint\left(cn\_fast\_hash\left(G\right)\right)$$

Note that not every hash gives a point in the group of the basepoint (i.e. $H = \psi G$ for some unknown ψ) (which is contrary to what happens in secp256k1, the curve used by Bitcoin). However, it seems that choosing the basepoint itself works (I previously used H(123456G) which seemed more secure to me, but the basepoint is certainly a more natural choice). Choosing $H = \gamma G$ for some unknown γ is necessary so that all the usual elliptic curve math holds.

[3] $H = MiniNero.getHForCT()$ in terms of the code at [14]

Under the discrete logarithm assumption on ed25519, the probability of an adversary discovering γ is negligible. Define $C(a,x) = xG + aH$, the commitment to the value a with mask x. Note that as long as $log_G H$ is unknown, and if $a \neq 0$, then $log_G C(a,x)$ is unknown. On the other hand, if $a = 0$, then $log_G C(a,x) = x$, so it is possible to sign with sk-pk keypair $(x, C(0,x))$.

In [11], there are input commitments, output commitments, and the network checks that

$$\sum Inputs = \sum Outputs.$$

However, this does not suffice in Monero: Since a given transaction contains multiple possible inputs $P_i, i = 1, ..., n$, only one of which belong to the sender, (see [16, 4.4]), then if we are able to check the above equality, it must be possible for the network to see which P_i belongs to the sender of the transaction. This is undesirable, since it removes the anonymity provided by the ring signatures. Thus instead, commitments for the inputs and outputs are created as follows (suppose first that there is only one input)

$$C_{in} = x_c G + aH$$

$$C_{out-1} = y_1 G + b_1 H$$

$$C_{out-2} = y_2 G + b_2 H$$

such that $x_c = y_1 + y_2 + z$, $x_c - y_1 - y_2 = z$, y_i are mask values, $z > 0$ and $a = b_1 + b_2$. Here x_c is a special private key the "amount key" known only to the sender, and to the person who sent them their coins, and must be different than their usual private key. In this case,

$$C_{in} - \sum_{i=1}^{2} C_{out-i}$$

$$= x_c G + aH - y_1 G - b_1 H - y_2 G - b_2 H$$

$$= zG.$$

Thus, the above summation becomes a commitment to 0, with $sk = z$, and $pk = zG$, rather than an actual equation summing to zero. Note that z is not computable to the originator of x_c's coins, unless they know both of the y_1, y_2, but even this can be simply mitigated by including an additional change address (the usual case is that the second commitment, with y_2 as mask, is sent to yourself as change).

Since it is undesirable to show which input belongs to the sender, a ring signature consisting of all the input commitments $C_i, i = 1, ..., s, ..., n$ (where s is the secret index of the commitment of the sender), adding the corresponding pubkey (so commitments and pubkeys are paired (C_i, P_i) only being allowed to be spent together) and subtracting $\sum C_{out}$ is created:

$$\left\{P_1 + C_{1,in} - \sum_j C_{j,out}, ..., P_s + C_{s,in} - \sum_j C_{j,out}, ..., P_n + C_{n,in} - \sum_j C_{j,out}\right\}.$$

This is a ring signature which can be signed since we know one of the private keys (namely $z + x'$ with z as above and $x'G = P_s$). In fact, since we know, for each i, both the private key for P_i and the private key for $P_i + C_{i,in} - \sum_j C_{j,out}$, we can perform a signature as in section 2.2. This precise details are described in Definition 4.1.

As noted in [11], it is important to prove that the output amounts[4] $b_1, ...b_n$ all lie in a range of positive values, e.g. $(0, 2^{16})$. This can be accomplished essentially the same way as in [11] and is described in more detail in section 5.

Finally, note that in the above, I have not made any mention of the tag-linkability property which is used in Monero and Cryptonote to prevent double-spends. The tag-linkability property here will result from combining the above discussion with the MLSAG signatures as described in Definition 4.1.

4 Ring CT For Monero Protocol

4.1 Protocol Description

Definition 4.1 (Tag-Linkable Ring-CT with Multiple Inputs and One-time Keys)

- Let $\left\{\left(P_\pi^1, C_\pi^1\right), ..., \left(P_\pi^m, C_\pi^m\right)\right\}$ be a collection of addresses / commitments with corresponding secret keys x_j, $j = 1, ..., m$.
- Find $q+1$ collections $\left\{\left(P_i^1, C_i^1\right), ..., \left(P_i^m, C_i^m\right)\right\}, i = 1, ..., q+1$ which are not already tag linked in the sense of [6, page 6].
- Decide on a set of output addresses $(Q_i, C_{i,out})$ such that $\sum_{j=1}^m C_\pi^j - \sum_i C_{i,out}$ is a commitment to zero.
- Let

$$\mathfrak{R} := \left\{\left\{\left(P_1^1, C_1^1\right), ..., \left(P_1^m, C_1^m\right), \left(\sum_j P_1^j + \sum_{j=1}^m C_1^j - \sum_i C_{i,out}\right)\right\},\right.$$

$$...,$$

$$\left.\left\{\left(P_{q+1}^1, C_{q+1}^1\right), ..., \left(P_{q+1}^m, C_{q+1}^m\right), \left(\sum_j P_{q+1}^j + \sum_{j=1}^m C_{q+1}^j - \sum_i C_{i,out}\right)\right\}\right\}.$$

  be the generalized ring which we wish to sign. Note that the last column is a Ring-CT ring in the sense of section 4.
- Compute the MLSAG signature Σ on $\mathfrak{R}$.

[4]Since input commitments could potentially be just inherited from the previous transaction, it suffices to consider the output amounts.

In this case, by Theorem A.2, $P_\pi^j, j = 1, ..., m$ cannot be the signer of any additional non-linked Ring Signatures in the given superset $\mathcal{P}$ of all such pairs $\mathcal{P} = \{(P, C)\}$ after signing Σ.

Remark 4.2 Space complexity of the above protocol. Note that the size of the signature Σ on $\mathfrak{R}$ according to definition 4.1 is actually smaller, for $m > 1$, than a current CryptoNote [16] ring signature based transaction which includes multiple inputs. This is because of the size improvements, given by [8], to each column. Note also, it is probably not necessary to include the key-image of the commitment entry of the above signature. Further size optimizations are likely possible.

4.2 Conversion from Visible Denominations to Commitments

As Monero currently uses Blockchain visible scalars to represent amounts, it is important that there is a way to convert from visible amounts to commitments while preserving anonymity. In fact, this is not difficult. Given a pair (P, a) where P is a public key and a represents an amount, this may be used as the input to a transaction as (P, aH), and it must be checked by the verifier that the input amount a multiplied by the masking point H, indeed gives aH. Thus at the first step, the input amounts will not be hidden, but the outputs of this transaction can be hidden, and all the necessary relations outlined in section 4 hold. Note that a range proof is not necessary for such an input.

Remark 4.3 The obvious benefit of this method of converting from visible amounts to commitments is that the amount of coins generated by the mining process is trustlessly verifiable. This is an advantage of the Ring CT protocol over payment schemes such as [4] which rely on a trusted setup phase.

4.3 Transaction Fees

As Monero is strongly decentralized (i.e. proof of work) it is necessary to pay miners a transaction fee for each transaction. This helps with the network security to prevent blockchain bloat. These fees must be paid "unmasked" i.e. just as bH, rather than $xG + bH$, and for some standardized amount b so that the miner can verify that $b \cdot H = bH$ and thus there is enough money for the transaction fee while still having the equations in terms of H so the necessary relations of section 4 hold.

4.4 Ring Multisignature

Note that a simple version of a t of m of n ring- multisignature can be done with the MLSAG signatures. This allows a group of m participants to create a multi-signature such that t out of m must sign for the signature to be accepted as valid, and in addition, it is signer ambiguous which m keys are the participants out of the n keys.

- The n participants in the multisignature create a shared secret x_e and public key P_e and share multisig key-images

  $$I_j = x_e H(P_e|P_j)$$

  with P_j the public key of the participant.

- Any participant of the multisignature selects $n - m$ additional public keys on the block-chain and creates an MLSAG signature with the first row the total n keys, and the second row having each entry the shared key P_e.
- Each signer transmitting part of the multisignature provides the initial $I_j,\ j = 1, ..., m$ so that the signature is accepted after t verifying signatures have been transmitted on the blockchain, each corresponding to one of the key images I_j.

An expanded writeup of the ring-multisignature is in the works.

5 Aggregate Schnorr Range Proofs

In [11], the confidential transactions without ring signatures uses a type of ring signature based on [1] called a Borromean ring signature, which helps to prove a committed value lies within a certain range. In this article, I will outline an alternative method, inspired by [7], which has the same space savings, but perhaps simpler security proofs. The motivation for this is as follows: Suppose that a given transaction has input commitments

$$C_{in} = a_{in}G + 10H$$

and output commitments

$$C_{out,1} = a_{out,1}G + 5H,\ C_{out,2} = a_{out,2}G + 5H$$

this scenario is valid as it is possible to sign for

$$C_{in} - C_{out,1} - C_{out,2} = (a_{in} - a_{out,1} - a_{out,2})\, G$$

However, note that (without range proofs) it would be possible to alternatively set output commitments

$$C_{out,1} = a_{out,1}G - H,\ C_{out,2} = a_{out,2}G + 11H$$

as -1 is a very large number modulo the curve group order, free money has been created. It is therefore necessary to prove that the $C_{out,i}$ are commitments to values which are positive and lie in a restricted range $[0, 2^n]$ for some n. To do this, one decomposes each output value into binary:

$$b = b_0 2^0 + b_1 2^1 + b_2 2^2 + \cdots + b_n 2^n$$

and computes commitments $C^j_{out,i}$ to $b_j \cdot 2^j$ and such that

$$C^1_{out,i} + C^2_{out,i} + \cdots + C^n_{out,i} = C_{out,i}$$

Finally, using secret key b_j, one computes a ring signature on

$$(C^j_{out,i}, C^j_{out,i} - 2^j H)$$

for all j and provides the $C^j_{out,i}$ to the verifying parties (in this case, the miners).

For space savings, one can either use a Borromean ring signature (as in [11]) to combine all of these simple ring signatures, or a type of aggregate ring signature defined as follows:

5.0.1 Aggregate Schnorr Non-linkable Ring Signature (ASNL) Generation

Let (x^j_i, P^j_1, P^j_2) be a set of keys, $j = 1, ..., n$ with x^j_i the secret key of P^j_i

- For each j, let $i' := i + 1 \ mod \ 2$, set α_j a random scalar, and compute $L^j_i = \alpha_j G$.
- Set $c^j_{i'} = H_s(L^j_i)$, where H_s is a cryptographic hash function returning a scalar, and after choosing $s^j_{i'}$ random, compute

$$L^j_{i'} = s_{i'} G + c_{i'} H.$$

- Set $c_i = H_s(L^j_{i'})$ and compute

$$s^j_i = \alpha - c^j_i x_i \ mod \ \ell$$

- Return (L^j_1, s^j_2) for all j and $s = \sum_j s^j_1$.

5.0.2 Aggregate Schnorr Non-linkable Ring Signature (ASNL) Verification

Start with $(P^j_1, P^j_2, L^j_1, s^j_2)$ for $j = 1, ..., n$ and s.

- For all j, compute $c^j_2 = H_s(L^j_1)$, $L^j_2 = s^j_2 G + c^j_2 H$, and $c^j_1 = H_s(L^j_2)$.
- If $\sum_{j=1}^n L^j_1 = sG + (c^j_1 + \cdots + c^j_n)H$ then return 0 for a valid signature. Otherwise return -1.

Theorem 5.1 The Aggregate Schnorr Non-linkable ring signature is unforgeable under the discrete logarithm assumption.

Proof I sketch a proof in the case $n = 2$. The general case is similar. Suppose that an adversary $\mathcal{A}$ is able to forge an ASNL signature on

$$\{(x^1_i, P^1_1, P^1_2), (x^2_i, P^2_1, P^2_2)\}$$

with non-negligible probability while knowing at most one of the x^j_i (suppose without loss of generality that $\mathcal{A}$ knows x^1_i). For any given such forgery:

$$\{s, (P^j_1, P^j_2, L^j_1, s^j_2)\}, \ j = 1, 2,$$

I solve the discrete logarithm of P^2_1 with non-negligible probability. Following the verification algorithm, let $c^j_1 = H_s\left(s^j_2 G + H_s(L^j_1)H\right)$. It must then be true that

$$L^1_1 + L^2_1 = sG + \left(c^1_1 P^1_1 + c^2_1 P^2_1\right).$$

Supposing that $L^1_1 = aG$ and $L^2_1 = bG$ with a, b known to $\mathcal{A}$, then

$$aG + bG - sG - c^1_1 P^1_1 = c^2_1 P^2_1$$

so, as c_1^2 is determined by the verification protocol, it must be the case that $\mathcal{A}$ knows the private key of P_1^2,

$$x_1^2 := \frac{a + b - s - c_1^1 x_1^1}{c_1^2} \mod \ell$$

This contradicts the discrete logarithm assumption for the given group. □

5.1 Representing Amounts

Amounts in the Ring CT protocol be represented in a similar manner as in [11], however modified slightly to fit Monero. CryptoNote coins organize denominations of coins into a databases of indices, so that they may be referenced in order of their appearance as inputs for ring signatures. As all values in CryptoNote coins are represented as 64-bit unsigned integers, we can use a range proof over the entire register to encode the output amount to the recipient. Beneficially, this also allows all outputs to be mixed with each other, so only a single index of outputs in order of appearance is required. A range proof of this is approximately 5 kilobytes. Ring signatures for inputs then become much more resilient to analysis, as the anonymity set is widely expanded.

While the range proofs are large, using a transaction hashing scheme that signs the transaction "prefix" (both input indices and outputs) and stores the range proof, along with ring signatures, separately enables large amounts of space saving when syncing to a checkpoint or operating as an SPV node. Similar to Sidechains Elements [12], we would store $H(H(prefix)||H(signatures))$ in the merkle tree. Then, nodes syncing to and trusting a checkpoint need only download the prefix and a hash of the signatures, effectively reducing the amount of bandwidth require to sync dramatically. Because the transaction prefix uses a varint for the outputs it references in its ring signatures and outputs funds to single public keys, they are miniscule in comparison to the ring signature data1in the inputs and range proofs. Later, the ring signatures may even be pruned from the database by some nodes to further save space, storing instead only the comparatively small UTXO set.

5.1.1 *Passing Amounts to Receiver*

Now, given any output amount, $b = b_0 2^0 + b_1 2^1 + \cdots b_n 2^n$, a sender computes a new private /public key pair and corresponding shared ECDH secret ss and makes the following information available in their transaction:

- $C_j = a_j G + (b_j 2^j) H$ where a_i are some random numbers for $j = 0, ..., n$.
- The data $\left\{ (L_1^i, s_2^j), s \right\}$.
- ECDH public key and $a + ss \ mod \ \ell$ where $a = a_0 + \cdots + a_n$.

The receiver then:

- Computes their shared secret ss and computes a from $a + ss \ mod \ \ell$.
- Computes $C = \sum C_i$, computes $C - aG = bH$, and finds b by comparing to all bH in the given range $[0, 2^n]$. (In practice this will be a quick 500 kilobyte lookup with $n = 14$ as in the previous section. If on the other hand 2^{32} were to be chosen as the upper limit value, as in [11], the search would become computationally intensive).

6 Conclusion

The Ring Confidential Transactions protocol provides a strongly decentralized cryptocurrency (i.e. there is no privileged party) which has provable security estimates regarding the hiding of amounts, origins and destinations. In addition, coin generation in the Ring Confidential Transactions protocol is trustless and verifiably secure. These five factors are a necessity of a cash-like crypto-currency such as Monero.

References

1. Masayuki Abe, Miyako Ohkubo, and Koutarou Suzuki. 1-out-of-n signatures from a variety of keys. *Advances in Cryptology?Asiacrypt 2002*, pages 415–432, 2002.
2. Adam Back. Bitcoins with homomorphic value (validatable but encrypted). https://bitcointalk.org/index.php?topic=305791.0, 2013. [Online; accessed 1-May-2015].
3. Adam Back. Ring signature efficiency. https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684, 2015. [Online; accessed 1-May-2015].
4. Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.
5. Evan Duffield and Kyle Hagan. Darkcoin: Peertopeer cryptocurrency with anonymous blockchain transactions and an improved proofofwork system. 2014.
6. Eiichiro Fujisaki and Koutarou Suzuki. Traceable ring signature. In *Public Key Cryptography–PKC 2007*, pages 181–200. Springer, 2007.
7. Javier Herranz. Aggregate signatures. http://www.iiia.csic.es/~jherranz/papers/Nijmegen_seminar_aggregate.pdf, oct 2005.
8. Joseph K Liu, Victor K Wei, and Duncan S Wong. Linkable spontaneous anonymous group signature for ad hoc groups. In *Information Security and Privacy*, pages 325–335. Springer, 2004.
9. Adam Mackenzie, Surae Noether, and Monero Core Team. Improving obfuscation in the cryptonote protocol. "https://lab.getmonero.org/pubs/MRL-0004.pdf", January 2015.
10. Greg Maxwell. Coinjoin: Bitcoin privacy for the real world, august 2013. Bitcoin Forum. https://bitcointalk.org/index.php?topic=279249.0, 2013. [Online; accessed 1-July-2015].
11. Greg Maxwell. Confidential Transactions. https://people.xiph.org/~greg/confidential_values.txt, 2015. [Online; accessed 1-June-2015].
12. Greg Maxwell. Elements project. https://github.com/ShenNoether/MiniNero, 2015.
13. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Consulted*, 1(2012):28, 2008.
14. Shen Noether. Mininero. https://github.com/ShenNoether/MiniNero, 2015.
15. Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *Advances in Cryptology???ASIACRYPT 2001*, pages 552–565. Springer, 2001.
16. Nicolas van Saberhagen. Cryptonote v 2. 0. *HYPERLINK https://cryptonote.org/whitepaper.pdf*, 2013.

Appendix A: Appendix: Security Proofs

A.1 MLSAG Unforgeability

This follows similarly to [8, Theorem 1]. Let H_1 and H_2 random oracles, and $\mathcal{SO}$ be a signing oracle which returns valid MLSAG signatures. Assume there is a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ with the ability to forge an MLSAG from a list of key vectors L with non-negligible probability

$$Pr\left(\mathcal{A}\left(\mathcal{L}\right) \to (m, \sigma) : Ver\left(L, m, \sigma\right) = True\right) > \frac{1}{Q_1\left(k\right)}$$

where Q_1 is a polynomial inputting a security parameter k and where (m, σ) is not one of the signatures returned by $\mathcal{SO}$. Assume that $\mathcal{A}$ makes no more than $q_H + nq_S$ (with n the number of keys in $\mathcal{L}$) queries to the signing oracles H_1, H_2 and $\mathcal{SO}$ respectively. The oracles H_1 and H_2 are assumed independent and random and are consistent given duplicate queries. The signing oracle $\mathcal{SO}$ is also allowed to query H_1 and H_2. Given $\mathcal{A}$, I will show it is possible to create a PPT adversary $\mathcal{M}$ which uses $\mathcal{A}$ to find the discrete logarithm of one of the keys in $\mathcal{L}$.

If L is a set of key vectors $\{\overline{y_1}, ..., \overline{y_n}\}$ each of size r, (i.e. $\overline{y_i} = \left(y_1^i, ..., y_r^i\right)$ with $y_1, ..., y_r$ public keys) then a forged signature

$$\sigma = (c_1, s_1, ..., s_n, y_0)$$

produced by $\mathcal{A}$ must satisfy

$$c_{i+1} = H\left(\mathfrak{m}, L_i^1, R_i^1, ..., L_i^m, R_i^m\right)$$

where the i are taken mod n, and the L_i^j, R_i^j are defined as in section 2.2. The new adversary $\mathcal{M}$ may call $\mathcal{A}$ to forge signatures a polynomial number of times and will record each Turing script $\mathcal{T}$ whether or not the forgery is successful.

Lemma A.1 [8, Lemma 1] Let $\mathcal{M}$ invoke $\mathcal{A}$ to obtain a transcript $\mathcal{T}$. If $\mathcal{T}$ is successful, then $\mathcal{M}$ rewinds $\mathcal{T}$ to a header H and re-simulates $\mathcal{A}$ to obtain transcript $\mathcal{T}'$. If $Pr\left(\mathcal{T} \text{ succeeds}\right) = \epsilon$, then $Pr\left(\mathcal{T}' \text{ succeeds}\right) = \epsilon$.

Proof Follows easily from the cited Theorem. □

Theorem A.2 The probability that an adversary $\mathcal{A}$ forges a verifying MLSAG signature is negligible under the discrete logarithm assumption.

Proof I follow the notation introduced above. Similarly to [8, Theorem 1], since the probability of guessing the output of a random oracle is negligible, therefore, for each successful forgery $\mathcal{A}$ completes with transcript $\mathcal{T}$, there are $m_{\mathcal{T}}$ queries to H_1 matching the n queries used to verify the signature. Thus let $X_{i_1}, ..., X_{i_m}$ denote these queries used in verification for the i^{th} such forgery and let π be the index corresponding to the last such verification query for a given forgery

$$X_{i_m} = H_1\left(m, L_{\pi-1}^1, R_{\pi-1}^1, ..., L_{\pi-1}^{m_{\mathcal{T}}}, R_{\pi-1}^{m_{\mathcal{T}}}\right).$$

(Intuitively, π corresponds to what would be the secret index of the forged signature, since it corresponds to the last call to the random oracle for the given signature).

An attempted forgery σ produced by $\mathcal{A}$ is an (ℓ, π)-forgery if $i_1 = \ell$ and π is as above (so this forgery corresponds to queries ℓ through $\ell+\pi$). By assumption, there exists a pair (ℓ, π) such that the probability that the corresponding transcript $\mathcal{T}$ gives a successful forgery, $\epsilon_{\ell,\pi}\left(\mathcal{T}\right)$, satisfies

$$\epsilon_{\ell,\pi} \geq \frac{1}{m_{\mathcal{T}}\left(q_H + m_{\mathcal{T}} q_S\right)} \cdot \frac{1}{Q_1\left(k\right)} \geq \frac{1}{n\left(q_H + n q_S\right)} \cdot \frac{1}{Q_1\left(k\right)}.$$

Now, rewinding $\mathcal{T}$ to just before the ℓ^{th} query, and again attempting a forgery on the same set of keys, (and letting H_1 compute new coin flips for all of it's succeeding queries) then by Lemma A.1, it follows that the probability that $\mathcal{T}'$ is also a successful forgery satisfies

$$\epsilon_{\ell,\pi}\left(\mathcal{T}'\right) \geq \frac{1}{n\left(q_H + n q_S\right)} \cdot \frac{1}{Q_1\left(k\right)}.$$

Therefore, the probability that both $\mathcal{T}$ and $\mathcal{T}'$ correspond to verifying forgeries σ and σ' is non-negligible:

$$\epsilon_{l,\pi}\left(\mathcal{T} \text{ } and \text{ } \mathcal{T}'\right) \geq \left(\epsilon_{l,\pi}\left(\mathcal{T}\right)\right)^2 .$$

As new coin-flips have been computed for the random oracle outputs of H_1, it follows that with overwhelming probability there is j such that $s_\pi^j \neq s_\pi'^j$ and $c_\pi \neq c_{\pi+1}$. Thus we can solve for the private key of index π:

$$x_\pi^j = \frac{s_\pi'^j - s_\pi^j}{c_\pi - c_\pi'} \text{ } mod \text{ } q$$

which contradicts the discrete logarithm assumption. □

A.2 MLSAG Linkability

Theorem A.3 (Key-Image Linkability) The probability that a PPT adversary $\mathcal{A}$ can create two verifying (and unlinked in the given setting) signatures σ, σ' signed with respect to key vectors $\overline{y}$ and $\overline{y}'$respectively such that there exists a public key y in both $\overline{y}$ and $\overline{y}'$ is negligible.

Proof Suppose to the contrary that $\mathcal{A}$ has created two verifying signatures σ and σ' both signed with respect to key vectors $\overline{y}$ and $\overline{y}'$ respectively such that there exists a public key y in both $\overline{y}$ and $\overline{y}'$. Let y appear as element j of $\overline{y}$, and as element j' element of $\overline{y}'$. By Theorem A.2, it holds with overwhelming probability that there exists indices π and π' for the public keys in σ and σ' respectively such that

$$L_\pi^j = s_\pi^j G + c_\pi y_\pi^j$$

$$R_\pi^j = s_\pi^j H\left(y_\pi^j\right) + c_\pi I_j$$

and

$$L_{\pi'}^{j'} = s_{\pi'}^{j'} G + c_{\pi'} y_{\pi'}^{j'}$$

$$R_{\pi'}^{j'} = s_{\pi'}^{j'} H\left(y_{\pi'}^{j'}\right) + c_{\pi'} I_{j'}$$

with

$$log_G L_\pi^j = log_{H\left(y_\pi^j\right)} R_\pi^j$$

and

$$log_G L_{\pi'}^{j'} = log_{H\left(y_{\pi'}^{j'}\right)} R_{\pi'}^{j'}$$

Letting x denote the private key of y, $y = xG$, then after solving the above for I_j and $I_{j'}$ it follows that $I_j = xH\left(y_\pi^j\right) = xH\left(y\right)$ and similarly $I_{j'} = xH\left(y\right)$. Thus the two signatures include $I_j = I_{j'}$, and therefore, since duplicate key images are rejected, one of them must not verify. □

A.3 MLSAG Anonymity

To prove the anonymity of the above protocol in the random oracle model, let H_1, H_2 be random oracles modeling discrete hash functions. Let $\mathcal{A}$ be an adversary against anonymity. I construct an adversary $\mathcal{M}$ against the Decisional Diffie Helman (DDH) assumption as follows. The DDH asumption says that given a tuple $(G, aG, bG, \gamma G)$, the probability of determining whether $\gamma G = abG$ is negligible.

Theorem A.4 Ring CT protocol is signer-ambiguous under the Decisional Diffie-Helman assumption.

Proof (Similar proof to [8, Theorem 2]) Assume that the Decisional Diffie-Helman problem is hard in the cyclic group generated by G and suppose there exists a PPT adversary $\mathcal{A}$ against signer ambiguity. Thus given a list L of n public key-vectors of length m, a set of t private keys $\mathcal{D}_t = \{x_1, ..., x_t\}$, a valid signature σ on L signed by a user with respect to a key-vector $\overline{y}$ such that the corresponding private key-vector $\overline{x} = (x_1^\pi, ..., x_m^\pi)$ satisfies $x_j^\pi \notin \mathcal{D}_t$, then $\mathcal{A}$ can decide π with probability

$$Pr\left(\mathcal{A} \to \pi\right) > \frac{1}{n-t} + \frac{1}{Q\left(k\right)}$$

for some polynomial $Q\left(k\right)$. I construct a PPT adversary $\mathcal{M}$ which takes as inputs a tuple (G, aG, bG, c_iG) where $i \in \{0, 1\}$ is randomly chosen (and not a priori known to $\mathcal{M}$), $c_1 = ab$, and c_0 is a random scalar, and outputs i with probability

$$Pr\left(\mathcal{M}\left(G, aG, bG, c_iG\right) \to i\right) \geq \frac{1}{2} + \frac{1}{Q_2\left(k\right)}$$

for some polynomial $Q_2\left(k\right)$.

Consider an algorithm SIMNIZKP (similar to the one defined in [6]) which takes as input scalars a, c , a private key vector $\overline{x}$, a set of public key-vectors $\overline{y}_i, i = 1, ..., m$, an index π, and a message $\mathfrak{m}$ and acts on these as follows:

1. Generate random scalars $s_1, ..., s_m$ and, a random scalar $c_\pi \leftarrow H$.
2. For j indexing $\overline{x}$, set

$$L_\pi^1 = aG$$

$$R_\pi^1 = cG$$

and for all other j

$$L_\pi^j = s_\pi^j G + c_\pi y_\pi^j$$

$$R_\pi^j = s_\pi^j H\left(y_\pi^j\right) + c_\pi x^j H\left(y_\pi^j\right)$$

3. Compute a random output from the random oracle

$$c_{\pi+1} \leftarrow H\left(\mathfrak{m}, L_\pi^1, R_\pi^1, ..., L_\pi^m, R_\pi^m\right).$$

4. For each i, working mod m, compute

$$L_i^j = s_i^j G + c_i y_\pi^j$$

$$R_i^j = s_i^j H\left(y_i^j\right) + c_i x^j H\left(y_i^j\right)$$

$$c_{i+1} \leftarrow H\left(\mathfrak{m}, L_i^1, R_i^1, ..., L_i^m + R_i^m\right).$$

and note that at the last step when $i = \pi - 1$, then c_{i+1} is already determined, to maintain consistency with the random oracle output.

Note that regardless of whether $\overline{x}$ is the actual private key corresponding to $\overline{y}$, due to the fact that consistency is maintained by the random oracles in subsequent calls, the above signature verifies. If $\overline{x}$ is actually the private key-vector of $\overline{y}$, then there is no difference between SIMNIZKP and an actual signature.

Finally, given a tuple (G, aG, bG, c_iG) where a, b are randomly selected scalars, with $c_1 = ab$, c_0 a random element, $i \in \{0, 1\}$, $\mathcal{M}$ takes the following steps to solve the Decisional Diffie Helman Problem with non-negligible probability. $\mathcal{M}$ grabs a random $\gamma \leftarrow H$ from the random oracle and takes a private / public key-vector pair $(\overline{x}, \overline{y})$, and then computes s such that $a = s + \gamma x$. Now $\mathcal{M}$ performs SIMNIZKP with arbitrarily selected key-vectors $\{\overline{y_i}\}_{i=1,...,n}$ such that $\overline{y} = \overline{y}_\pi$, $a \to a$, $c_i \to c$ some message $\mathfrak{m}$, and $\overline{x} \to \overline{x}$.

If it is the case that $i = 1$, then $c = ab$, then

$$log_G aG = log_{bG} cG = a$$

and due to the fact that $\mathcal{A}$ is assumed to be able to find π with non-negligible probability, then there is a non-negligible probability over $\frac{1}{2}$ that $\mathcal{A}$ returns 1 (upon which $\mathcal{M}$ returns 1). If $i = 0$, then $\mathcal{A}$ returns 1 only with probability $\frac{1}{2}$, and so for some non-negligible probability over $\frac{1}{2}$, $\mathcal{M}$ returns the same value as $\mathcal{A}$, and thus solves the Decisional Diffie-Helman problem for randomly chosen scalars with non-negligible probability over $\frac{1}{2}$, which is a contradiction. □

Appendix B: Ring CT Demo Code

In the repository at [14] I have created a simple demonstration of the Ring Confidential Transactions protocol utilizing the MLSAG signatures of section 2 and the ASNL signatures of section 5:

```
  H_ct = RingCT.getHForCT()
print("H", H_ct)
sr, Pr = PaperWallet.skpkGen() #receivers private/ public
se, pe, ss = ecdh.ecdhgen(Pr) #compute shared secret ss
digits = 14 #in practice it will be 14
print("inputs")
Cia, L1a, s2a, sa, ska = RingCT.genRangeProof(10000, digits)
print("outputs")
Cib, L1b, s2b, sb, skb = RingCT.genRangeProof(7000, digits)
Cic, L1c, s2c, sc, skc = RingCT.genRangeProof(3000, digits)
```

```
print("verifying range proofs of outputs")
RingCT.verRangeProof(Cib, L1b, s2b, sb)
RingCT.verRangeProof(Cic, L1c, s2c, sc)
x, P1 = PaperWallet.skpkGen()
P2 = PaperWallet.pkGen()
C2 = PaperWallet.pkGen()
#some random commitment grabbed from the blockchain
ind = 0
Ca = RingCT.sumCi(Cia)
Cb = RingCT.sumCi(Cib)
Cc = RingCT.sumCi(Cic)
sk = [x, MiniNero.sc_sub_keys(ska, MiniNero.sc_add_keys(skb, skc))]
pk = [[P1, P2], [MiniNero.subKeys(Ca, MiniNero.addKeys(Cb, Cc)), \
MiniNero.subKeys(C2, MiniNero.addKeys(Cb, Cc)) ] ]
II, cc, ssVal = MLSAG.MLSAG_Sign(pk, sk, ind)
print("Sig verified?", MLSAG.MLSAG_Ver(pk, II, cc, ssVal) )
print("Finding received amount corresponding to Cib")
RingCT.ComputeReceivedAmount(pe, sr, MiniNero.addScalars(ss, skb), Cib)
print("Finding received amount corresponding to Cic")
RingCT.ComputeReceivedAmount(pe, sr, MiniNero.addScalars(ss, skc), Cic)
```

Here is an example transaction with input $10,000$ and outputs $3,000$ and $7,000$.

```
 ('H', '61fe7f0f5a607a33427d01dd1fded5ffa03fae2e9df9ebccf2e0a2f5bd77a204')
inputs
('b, b in binary', 10000, [0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1])
Generating Aggregate Schnorr Non-linkable Ring Signature
outputs
('b, b in binary', 7000, [0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0])
Generating Aggregate Schnorr Non-linkable Ring Signature
('b, b in binary', 3000, [0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0])
Generating Aggregate Schnorr Non-linkable Ring Signature
verifying range proofs of outputs
Verifying Aggregate Schnorr Non-linkable Ring Signature
Verified
Verifying Aggregate Schnorr Non-linkable Ring Signature
Verified
('Generating MLSAG sig of dimensions ', 2, 'x ', 2)
('verifying MLSAG sig of dimensions ', 2, 'x ', 2)
('c',
['80a3cfd06dd2862307cd75c2a1566f20cd743dbb0b9feb22d79dcbecb9023f42',
'a9b7342ba7bf2f102505ca19dab734fde638916c0a29f5b30e49833ab51393ea',
'80a3cfd06dd2862307cd75c2a1566f20cd743dbb0b9feb22d79dcbecb9023f42'])
('sig verifies?', True)
('Sig verified?', True)
Finding received amount corresponding to Cib
('received ', 7000,
'a488ec68732fb551841c2c6dcc7ffac895d98ec7e9378275ed20ea12805fc18e')
Finding received amount corresponding to Cic
('received ',3000,
'1b46626858e130a0f3884c74c9fdeabc4d812c519103ea16a35a3f82a3d0ed6d')
```