MRL: Notes

July 6, 2015

Abstract

In this note I examine possibilities for simple improvements to the Monero protocol. This is a preliminary draft, so there may be errors. I have also included an experimental extension of gmaxwell's CT protocol to CryptoNote. Please contact me with comments and suggestions at Shen.Noether@gmx.com

Contents

1 Size Improvements in Ring Signatures

In this section, I give a brief explanation of a ring signature scheme which is similar to the one used in Monero as it exists circa June 2015, but offers a small size improvements. This scheme is copied (the original post is poorly typeset, so I have reproduced it below) from an explanation given by Adam Back in a bitcointalk.org, and appears to have been originally designed by Liu, Wei, and Wong in [LWW].

In the current Monero ring signature, which is the same as it's parent protocol (CryptoNote), ring signatures follow [CN, 4.4]. The proposed algorithm is the four following steps:

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 basepoint and x is the signers spend key. Let $I = xH(P_j)$ where H is a hash function returning a point (in practice $toPoint(Keccak(P_k))$).

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

Compute

$$L_j = \alpha G$$

$$R_j = \alpha H\left(P_j\right)$$

$$c_{j+1} = h\left(P_1, ..., P_n, L_j, R_j\right)$$

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

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

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

$$c_{j+2} = h\left(P_1, ..., P_n,\ L_{j+1},\ R_{j+1}\right)$$

$$\ldots$$

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

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

$$c_j = h\left(P_1, ..., P_n, L_{j-1},\ R_{j-1}\right)$$

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

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

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

$$R_j = \alpha H\left(P_j\right) = s_j H\left(P_j\right) + c_j I$$

and

$$c_{j+1} = h\left(P_1, ..., P_n,\ L_j,\ R_j\right)$$

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 [CN, 4.4].

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

$$c_{i+1} = h\left(P_1, ..., P_n, L_i, R_i\right)$$

for all i.

LINK: Signatures with duplicate key images I are rejected.

1.1 Compatibility

The above scheme uses no additional elliptic curve functions than already exist in Monero, and thus should be relatively easy to implement (perhaps a prototype in MiniNero is called for). A hard-fork could be accomplished by (perhaps temporarilty) including both the current and proposed schemes, and then deciding which to verify based on the size of the signature. Note that the key image is the same in each scheme, so it is impossible to double spend utilizing both schemes in combination.

1.2 Threshold Version

It is simple to get a t of n threshold version of the above signature by simply concatenating t such signatures. Using a t-of n ring signature (sometimes denoted (t, n)) will allow you to take t inputs, so you can use this in place of a "sweeping" transaction.

2 Knapsack Scheme for Anonymity

Recall the one time public keys implemented in the CryptoNote protocol [CN, 4.3]: if Alice wants to send a payment to Bob, she gets Bob's public

key (A, B), generates a random $r \in [1, l-1]$ (where l is the ed25519 curve order) and computes a one-time public key $P = h(rA)G + B$ and sends a message to P containing $R = rG$ as additional information.

Bob uses his private key (a, b) to then compute $P' = h(aR)G + B = h(rA)G + B = P$ and when he finds a match he computes the private key $x = h(aR) + b$, so that $P = xG$.

The above is standard CryptoNote protocol. Now let's suppose that Alice wants to hide the amount D which she sends to Bob. In this case, Alice could partition the amount D into $d_1 + d_2 + ... + d_n = D$ and send the amounts $d_1, ..., d_n$ to one-time public keys $P_i = h(r_i A)G + B,\ i \in \{1, ..., n\}$. To save space, the P_i could be created by choosing a single r_1, and then (for example) deterministically letting $P_2 = h(2r_1 A)G + B$, $P_3 = h(3r_1 A)G + B = h(3aR_1)G + B$. This deterministic manner allows that only one $R = r_1 G$ needs to be included in a given transaction.

Finally, Alice includes randomly generated values $d'_1, d'_2, ..., d'_m$ chosen so that

$$\sum_{i=1}^{n} d_i + \sum_{j=1}^{m} d'_j = D + D' = \mathbb{D}$$

and d'_j represents a (positive) amount sent to Alice's deterministically generated change address and $\mathbb{D} = D + D'$ represents the total input amount of the transaction.

2.1 Comparison to Other Techniques

Recall the set of all subsets of $\{d_1, ..., d_n, d'_1, ..., d'_m\}$ has size $2^{n+m} = 2^N$ Thus the total number of possible amounts sent to Bob by Alice is 2^N. The additional size added to the transaction is $N \cdot 32$ *bytes*. Thus for example, if it is desired to limit the additional size of an output to 2560 *kb* (for example, the amount hiding given in gmaxwell's Confidential Transactions for Bitcoin [CT] has an additional 2564 bytes per output) and include a change address, then $N = 2560 \cdot 2/32 = 160$, which implies that the total number of possible amounts sent to Bob by Alice is 2^{160}. In comparison, [CT] offers masking of a 32 bit value with a 2564 proof. This seems to offer about a $2^{160}/2^{32} = 2^{128}$ times improvement over CT [1] for no change to the Monero code[2], **however** this is not quite optimal as coins will eventually be split into smaller and smaller chunks. In addition, the total amount of inputs and outputs are necessarily known using this technique so not all information is hidden.

For example, here are some random amounts generated (each sent to one time keys) which might be used to send 250 out of 500 coins to someone else, and the remaining 250 coins to your one-time change addresses:

```
('all amounts:', [0.01, 0.03, 0.03, 0.04, 0.1,
0.11, 0.49, 0.69, 2.05, 2.84, 3.63, 5.65,
11.28, 14.23, 19.61, 32.05, 34.1, 36.86,
```

[1] (just in regards to hiding transaction amounts with no mention of the additional privacy offered by ring signatures, one-time keys, etc)

[2] this is possible simply with some smarter wallet software see Knapsack.py in my personal MiniNero repository for example

38.91, 43.93, 44.34, 48.47, 60.18, 100.37])

In contrast, here are some random amounts generated to send 2.5 out of 500 coins:

('all amounts:', [0.02, 0.05, 0.07, 0.59,
0.97, 1.03, 1.46, 2.62, 2.77, 6.59, 10.54,
11.11, 17.3, 37.05, 59.01, 61.43, 72.09, 215.3])

Each of the above lists has a powerset with cardinality around 50,000 after duplicate amounts are removed. In addition, a small optional fuzz fee[3] between 0 and t xmr, for some optional t is applied to the sent amount for further obfuscation.

The simple technique in the previous paragraph is possible, however in practice it is undesirable to have coins spread among many addresses (eventually, for convenience, you will have to sweep the coins to your main address). Thus consider the following modification. Assume for simplicity, that we are comparing again to a single output and single change address so $N = 2560 \cdot 2/32 = 160$ is the number to beat (note that the above scheme has even more significant improvements over [CT] for each additional output). In practice, masking a 16 bit value may be sufficient. ($2^{16} = 65536$ gives you 65536 possible amounts which should be enough for plausible deniability, in particular it should be sufficient to deter any sort of blockchain-wide analysis

[3]Note that in practice, this optional fuzz fee may not actually be any more expensive than using the CT technique of the next section, since currently Monero implements a .1 xmr fee per kilobyte, and the range proofs are about 2.5 kilobytes per output.

of the amounts spent). Thus, if the number of change addresses Alice chooses is chosen randomly in $\{1, 16\}$, then Bob will receive on average 8 inputs. In order to keep the average amount of inputs equal to the average amount of outputs, after subtracting $160 - 16$, we have 144, 32 byte chunks we can still fit in our transaction. Thus if Bob uses 8 inputs into the next transaction, then Bob can afford an $(8, 20)$ ring signature. Although in practice the number will not be 8, on average the values should work out.

3 Confidential Transactions for CryptoNote

In this section, I discuss how to make a slight extension to gmaxwells Confidential Transactions protocol, described in[CT], so that it works in Monero. This section is fairly experimental at this point, and will need to be checked carefully before I am confident in it's security. The benefit of the [CT] techniques are that they accomplish more mathematically and precisely the previous section accomplishes. However some downsides include: it might be the (possibly much) larger size of the ring signatures which are used, and the fact that it would require a fairly complex hard fork to the current code to use these, for potentially not that much benefit to the techniques of the previous section (Since senders and receivers are, in any case, hidden in Monero, there is a question of how precisely do the amounts being sent actually need to be hidden).

Let G be the ed25519 basepoint. Let[4]

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

It is difficult to find an x such that $xG = H$.

Define $C\left(a, x\right) = xG + aH$, 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 $log_G C\left(a, x\right)$ is unknown. On the other hand, if $a = 0$, then $log_G C\left(a, x\right) = x$, so it is possible to sign with sk-pk keypair $\left(x, C\left(0, x\right)\right)$.

In [CT], 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 [CN, 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)

$$C_{in} = xG + aH$$

[4] $H = MiniNero.getHForCT()$

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

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

such that $x + z = y_1 + y_2$, $z > 0$ and $a = b_1 + b_2$. In this case,

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

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

$$= zG.$$

Thus, the above summation becomes a commitment to 0, with $sk = z$, and $pk = zG$, rather than an actual equation summing to zero.

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:

$$\left\{ P_1 + C_1 - \sum C_{out}, ..., P_s + C_s - \sum C_{out}, ..., P_n + C_n - \sum C_{out} \right\}.$$

This is a ring signature which can be signed since we know one of the private keys (namely $z + x_s$ where z as above and $x_s G = P_s$). The technique of section 1 may be used.

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

- Prove first $C^{(j)}_{out-i} \in \{0, 2^j\}$ for all $j \in \{0, 1, ..., 16\}$.
- By carefully choosing the blinding values for each j, ensure that

$$\sum_{j=1}^{16} C^{(j)}_{out-i} = C_{out-i}.$$

- By homomorphicity of the Commitments, $b_i = \sum_j \delta_{ji} 2^j$, where δ_{ji} is the j^{th} digit in the binary expansion of b_i.

Thus in total, by the above, the sum of inputs into a transaction equals the outputs, yet the specific input (and it's index!) is hidden. In addition, the outputs are positive values.

Some prototype ring signature and commitment code will be included in future versions of https://github.com/ShenNoether/MiniNero

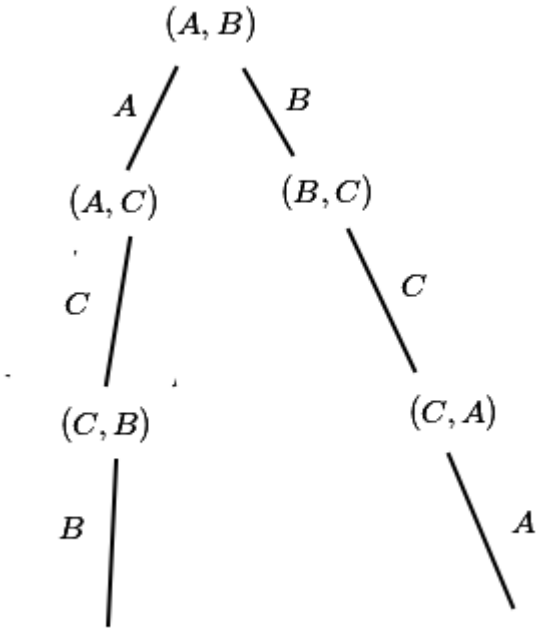
4 PigeonHole Problem

This section gives a solution to an attack on CryptoNote where using a tree analysis, some outputs may be proved to be spent.

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

4.1 Example of the attack

Suppose that (a, b), (b, c), (c, a) are rings. By drawing out the possible choices of which could be spent as in the following picture:



we see that all of a, b, and c have been spent. Thus if (a, d) is another ring, then d must be the spender as a has already been spent.

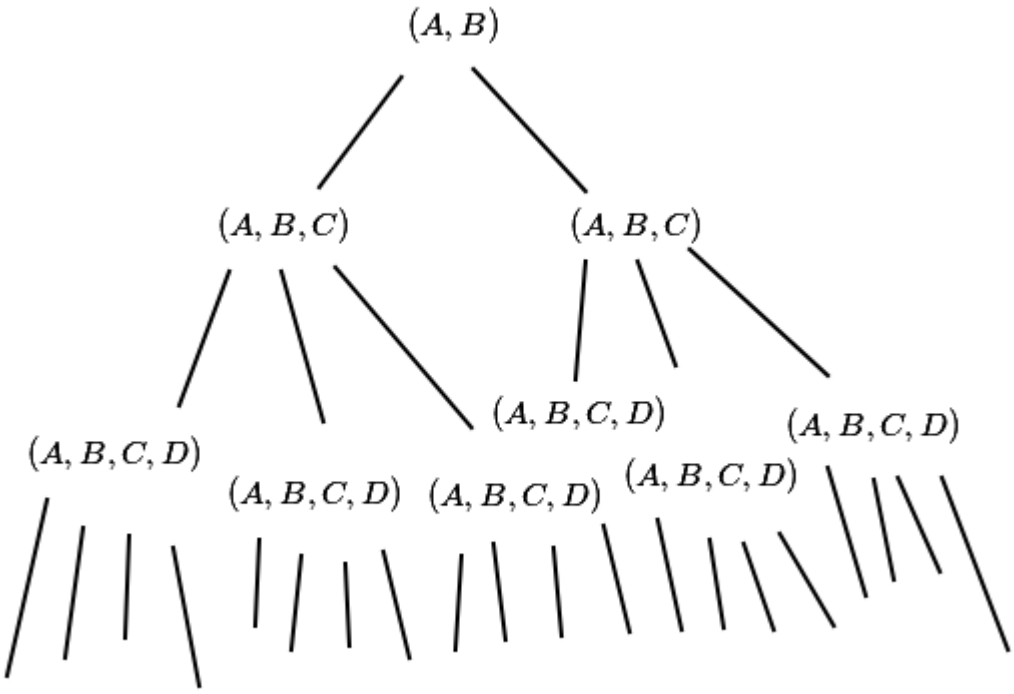
4.2 Preventing this attack

This attack is made possible through something called the Pigeonhole principle, which states that, if there are n items put into m containers, and $n > m$, then at least one container must contain more than one item.

In terms of ring signatures, the "containers" are pubkeys and the "items" are rings. So if there are n rings with m pubkeys are $n > m$ then there must be one pubkey in more than one ring (a double spend). However double spends are impossible by the key image, but what can happen is if $n = m$, then you can prove that all the pubkeys have been spent.

Thus to prevent the attack, keep track of the number of rings each pubkey has been in, and then enforce that for each subsequent ring it is a part of must have at least one greater member than the number of rings it has been in. This solves the problem since at each step of the tree there are more pubkeys than rings (see the image on the next page).

Note that if you have a large and unique amount in an input, then there should be a value **MAX_MIX** above which you are not allowed to mix, but instead the input should first be split into slightly smaller and divisible amounts (there is no million dollar bill) to emulate cash, which can then be mixed.



References

[AB] https://bitcointalk.org/index.php?topic=972541.msg10619684#msg10619684

[CN] van Saberhagen, Nicolas. "Cryptonote v 2. 0." HYPERLINK "https://cryptonote. org/whitepaper. pdf" https://cryptonote. org/whitepaper. pdf (2013).

[CT] https://people.xiph.org/~greg/confidential_values.txt

[LWW] https://eprint.iacr.org/2004/027.pdf