RESEARCH BULLETIN MRL-0003

Monero is Not That Mysterious

25 September 2014

Shen Noether* and Sarang Noether

*Correspondence: lab@monero.cc
Monero Research Lab

1 Introduction

Recently, there have been some vague fears about the CryptoNote source code and protocol floating around the internet based on the fact that it is a more complicated protocol than, for instance, Bitcoin. The purpose of this note is to try and clear up some misconceptions, and hopefully remove some of the mystery surrounding Monero Ring Signatures. I will start by comparing the mathematics involved in CryptoNote ring signatures (as described in [CN]) to the mathematics in [FS], on which CryptoNote is based. After this, I will compare the mathematics of the ring signature to what is actually in the CryptoNote codebase.

2 CryptoNote Origins

As noted in ([CN], 4.1) by Saberhagen, group ring signatures have a history starting as early as 1991 [CH], and various ring signature schemes have been studied by a number of researchers throughout the past two decades. As claimed in ([CN] 4.1), the main ring signature used in CryptoNote is based on [FS], with some changes to accomodate blockchain technology.

2.1 Traceable Ring Signatures

In [FS], Fujisaki and Suzuki introduce a scheme for a ring signature designed to "leak secrets anonymously, without the risk of identity escrow." This lack of a need for identity escrow allows users of the ring signature to hide themselves in a group with an inherently higher level of distrust compared to schemes relying on a group manager.

In ring-signature schemes relying on a group manager, such as the original ring signatures described in [CH], a designated trusted person guards the secrets of the group participants. While anonymous, such schemes rely, of course, on the manager not being compromised. The result of having a group-manager, in terms of currencies, is essentially the same as having a trusted organization or node to mix your coins.

In contrast, the traceable ring signature scheme given in [FS] has no group manager.

According to [FS], there are four formal security requirements to their traceable ring signature scheme:

- Public Traceability - Anyone who creates two signatures for different messages with respect to the same tag can be traced. (In CryptoNote, if the user opts not to use a one-time key for each transaction, then they will be traceable, however if they desire anonymity, then they will use the one-time key. Thus as stated on page 5 of [CN], the traceability property is weakened in CryptoNote.)
- Tag-Linkability - Every two signatures generated by the same signer with respect to the same tag are linked. (This aspect in CryptoNote refers to each transaction having a key image which prevents double spending.)
- Anonymity - As long as a signer does not sign on two different messages with respect to the same tag, the identity of the signer is indistinguishable from any of the possible ring members. In addition, any two signatures generated with respect to two distinct tags are always unlinkable. (In terms of CryptoNote, if the signer attempts to use the same key-image more than once, then they can be identified out of the group. The unlinkability aspect is retained and is a key part of CryptoNote.)
- Exculpability - An honest ring member cannot be accused of signing twice with respect to the same tag. In other words, it should be infeasible to counterfeit a tag corresponding to another person's secret key. (In terms of CryptoNote, this says that key images cannot be faked.)

In addition, [FS] provide a ring signature protocol on page 10 of their paper, which is equivalent to the CryptoNote ring signature algorithm, as described on page 9-10 of [CN]. It is worthwhile to note that [FS] is a publicly peer-reviewed publication appearing in Lecture Notes in Computer Science, as opposed to typical cryptocurrency protocol descriptions, where it is unclear whether or not they have been reviewed or not.

2.2 Traceability vs CryptoNote

In the original traceable ring signature algorithm described in [FS], it is possible to use the tag corresponding to a signature multiple times. However, multiple uses of the tag allow the user to be traced; in other words, the signer's index can be determined out of the group of users signing. It is worthwhile to note that, due to the exculpability feature of the protocol ([FS] 5.6, [CN], A2), keys cannot be stolen this way, unless an attacker is able to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP) upon which a large portion of modern cryptography is based ([Si] XI.4).

The process to trace a tag used more than once is described on ([FS], page 10). In the CryptoNote protocol, however, key images (tags) used more than once are rejected by the blockchain as double-spends, and hence traceability is not an aspect of CryptoNote.

2.3 Tag-Linkability vs CryptoNote

In essence, the tag-linkability aspect of the traceable ring signature protocol is what prevents CryptoNote transactions from being double-spends. The relevant protocols are referred to as "Trace" in ([FS], 5) and "LNK" in the CryptoNote paper. Essentially all that is required is to be able to keep track of the key images which have been used before, and to verify that a key image is not used again.

If one key-image is detected on the blockchain before another key-image, then the second key image is detected as a double-spend transaction. As key-images cannot be forged, being exculpable, the double-spender must in fact be the same person, and not another person trying to steal a wallet.

3 One-Time Ring Signatures (mathematics)

The security of the ring signature scheme as described in ([FS] 10, [CN] 10) and implemented in the CryptoNote source relies on the known security properties of Curve25519. Note that this is the same curve used in OpenSSH 6.5, Tor, Apple iOS, and many other[1] security systems.

3.1 Twisted Edwards Curves

The basic security in the CryptoNote Ring Signature algorithm is guaranteed by the ECDLP ([Si], XI.4) on the Twisted Edwards curve ed25519. The security properties of curve ed25519 are described in [Bern], by noted cryptographer Daniel Bernstein, and in ([BCPM]) by a team from Microsoft Research. Bernstein notes about ed25519 the "every known attack is more expensive than performing a brute-force search on a typical 128-bit secret-key cipher."

The curve ed25519 is a singular curve of genus 1 with a group law, and described by $-x^2 + y^2 = 1 + \left(\frac{-121665}{121666}\right) x^2 y^2$. This curve is considered over the finite field $\mathbb{F}_q$, $q = 2^{255} - 19$. For those readers unfamiliar with algebraic geometry, an algebraic curve is considered as a one dimensional sort of space, consisting of all points (x, y) satisfying the above equation. All points are also considered modulo q. By virtue of its genus, ed25519 has a "group structure" which, for the purpose of this discussion, means if $P = (x_1, y_1)$ is a point on the curve, and $Q = (x_2, y_2)$ is another point on the curve, then these points can be added (or subtracted) and the sum (or difference), $P + Q$ (or $P - Q$) will also be on the curve. The addition is **not** the naive adding of $x_1 + x_2$ and $y_1 + y_2$, but instead points are added using the rule

$$P + Q = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 + x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right)$$

where $d = \left(\frac{-121665}{121666}\right)$ ([BBJLP] 6, [BCPM]). The mathematics of curves of genus one are explained in great detail in [Si] for the interested reader.

Based on the above, we can compute $P + P$ for any such point. In order to shorten notation, we rely on our algebraic intuition and denote $2P = P + P$. If $n \in \mathbb{Z}$, then nP denotes the repeated sum

$$\underbrace{P + P + \cdots + P}_{n \text{ times}}$$

using the above nonlinear addition law. As an example of how this differs from ordinary addition, consider the following system of equations:

$$\begin{aligned} aP + bQ &= X \\ aP' + bQ' &= Y \end{aligned}$$

[1] http://ianix.com/pub/curve25519-deployment.html

where a, b, c, d are integers and P, Q, X are points. If this were a standard system of linear equations then one could use linear algebraic techniques to easily solve for a and b, assuming that P, Q, X, Y, P', and Q' are known. However, even if a, b are very small the above system is extremely difficult to solve using the ed25519 addition law. For example, if $a = 1$ and $b = 1$, we have

$$\begin{aligned}\left(\frac{x_P y_Q + y_P x_Q}{1 + d x_P x_Q y_P y_Q}, \frac{y_P y_Q + x_P x_Q}{1 - d x_P x_Q y_P y_Q}\right) &= (x_X, y_X) \\ \left(\frac{x_{P'} y_{Q'} + y_{P'} x_{Q'}}{1 + d x_{P'} x_{Q'} y_{P'} y_{Q'}}, \frac{y_{P'} y_{Q'} + x_{P'} x_{Q'}}{1 - d x_{P'} x_{Q'} y_{P'} y_{Q'}}\right) &= (x_Y, y_Y)\end{aligned}$$

So in reality, this is a system of 4 nonlinear equations. To convince yourself that it is in fact difficult to figure out a and b, try writing the above systems assuming $a = 2$, $b = 1$. It should become clear that the problem is extremely difficult when a, b are chosen to be very large. As of yet, there are no known methods available to efficiently solve this system for large values of a and b.

Consider the following problem. Suppose your friend has a random integer q, and computes qP using the above form of addition. Your friend then tells you the x and y coordinates $qP = (x, y)$, but not what q itself is. Without asking, how do you find out what q is? A naive approach might be to start with P and keep adding $P + P + P$... until you reach qP (which you will know because you will end up at (x, y)). But if q is very large then this naive approach might take billions of years using modern supercomputers. Based on what mathematicians currently know about the problem and the number of possible q, none of the currently known attacking techniques can, as a general rule, do better in any practical sense than brute force.

In CryptoNote, your secret key is essentially just a very, very large number x (for other considerations, see section 4.3.3, we choose x to be a multiple of 8). There is a special point G on the curve ed25519 called "the base point" of the curve which is used as the starting point to get xG. Your public key is just xG, and you are protected by the above problem from someone using known information to determine the private key.

3.2 Relation to Diffie Helman

Included in a ring signature are the following equations involving your secret key x:

$$\begin{aligned}P &= xG \\ I &= x\mathcal{H}_p(P) \\ r_s &= q_s - c_s x.\end{aligned}$$

Here s is a number giving the index in the group signature to your public key, and $\mathcal{H}_p(P)$ is a hash function which deterministically takes the point P to another point $P' = x'G$, where x' is another very large uniformly chosen number. The value q_s is chosen uniformly at random, and c_s is computed using another equation involving random values. The particular hash function used in CryptoNote is Keccak1600,

used in other applications such as SHA-3; it is currently considered to be secure ([FIPS]). The CryptoNote use of a single hash function is consistent with the standard procedure of consolidating distinct random oracles (in proofs of security in [FS], for example) into a single strong hash function.

The above equations can be written as follows:

$$\begin{aligned} P &= xG \\ P' &= xx'G' \\ r_s &= q_s - c_s x \end{aligned}$$

Solving the top two equations is equivalent to the ECDH (as outlined in a previous note ([SN])) and is the same practical difficulty as the ECDLP. Although the equations appear linear, they are in fact highly non-linear, as they use the addition described in 3.1 and above. The third equation (with unknowns q_s and x), has the difficulty of finding a random number (either q_s or x) in $\mathbb{F}_q$, a very large finite field; this is not feasible. Note that as the third equation has two unknowns, combining it with the previous two equations does not help; an attacker needs to determine at least one of the random numbers q_s or x.

3.3 Time Cost to Guess q or x

Since q and x are assumed to be random very large numbers in $\mathbb{F}_q$, with $q = 2^{255} - 19$ (generated as 32-byte integers), this is equivalent to a 128-bit security level ([BCPM]), which is known to take billions of years to compute with current supercomputers.

3.4 Review of Proofs in Appendix

In the CryptoNote appendix, there are four proofs of the four basic properties required for security of the one-time ring-signature scheme:

- Linkability (protection against double-spending)
- Exculpability (protection of your secret key)
- Unforgeability (protection against forged ring signatures)
- Anonymity (ability to hide a transaction within other transactions)

These theorems are essentially identical to those in [FS] and show that the ring signature protocol satisfies the above traits. The first theorem shows that only the secret keys corresponding to the public keys included in a group can produce a signature for that group. This relies on the ECDLP for the solution of two simultaneous (non-linear) elliptic curve equations, which, as explained in 3.2, is practically unsolvable. The second theorem uses the same reasoning, but shows that in order to create a fake signature that passes verification, one would need to be able to solve the ECDLP. The third and fourth theorems are taken directly from [FS].

4 One-Time Ring Signatures (Application)

To understand how CryptoNote is implementing the One-Time Ring signatures, I built a model in Python of Crypto-ops.cpp and Crypto.cpp from the Monero source code using naive Twisted Edwards Curve operations (taken from code by

Bernstein), rather than the apparently reasonably optimized operations existing in the CryptoNote code. Functions are explained in the code comments below. Using the model will produce a working ring signature that differs slightly from the Monero ring signatures only because of hashing and packing differences between the used libraries. The full code is hosted at the following address:

https://github.com/monero-project/mininero

Note that most of the important helper functions in crypto-ops.cpp in the CryptoNote source are pulled from the reference implementation of Curve25519. This reference implentation was coded by Matthew Dempsky (Mochi Media, now Google)[2].

In addition, after comparing the python code to the paper, and in turn comparing the python code to the actual Monero source, it is fairly easy to see that functions like `generate_ring_sig` are all doing what they are supposed to based on the protocol described in the whitepaper. For example, here is the ring signature generation algorithm used in the CryptoNote source:

Algorithm 1 Ring Signatures

$i \leftarrow 0$
while $i < numkeys$ **do**
  if $i = s$ **then**
    $k \leftarrow random\ \mathbb{F}_q\ element$
    $L_i \leftarrow k \cdot G$
    $R_i \leftarrow k \cdot \mathcal{H}_p(P_i)$
  else
    $k1 \leftarrow random\ \mathbb{F}_q\ element$
    $k2 \leftarrow random\ \mathbb{F}_q\ element$
    $L_i \leftarrow k_1 P_i + k_2 G$
    $R_i \leftarrow k_1 I + k_2 \mathcal{H}_p(P_i)$
    $c_i \leftarrow k_1$
    $r_i \leftarrow k_2$
  end if
  $i \leftarrow i + 1$
end while
$h \leftarrow \mathcal{H}_s(prefix + L'_i s + R'_i s))$
$c_s \leftarrow h - \sum_{i \neq s} c_i$
$r_s \leftarrow k - x c_s$
return $(I, \{c_i\}, \{r_i\})$

Comparing this with [CN] shows that it agrees with the whitepaper. Similarly, here is the algorithm used in the CryptoNote source to verify ring signatures:

Algorithm 2 VER

$i = 0$
while $i < numkeys$ **do**
  $L'_i \leftarrow c_i P_i + r_i G$
  $R'_i \leftarrow r_i \mathcal{H}_p(P_i) + c_i I$
  $i \leftarrow i + 1$
end while
$h \leftarrow \mathcal{H}_s(prefix + L'_i s + R'_i s))$
$h \leftarrow h - \sum_{i \neq s} c_i$
return $(h == 0 (mod\ q)) == 0$

4.1 Important Crypto-ops Functions

Descriptions of important functions in Crypto-ops.cpp. Even more references and information is given in the comments in the MiniNero.py code linked above.

[2] http://nacl.cr.yp.to/

4.1.1 ge_frombytes_vartime

Takes as input some data and converts to a point on ed25519. For a reference of the equation used, $\beta = uv^3 \left(uv^7\right)^{(q-5)/8}$, see ([BBJLP], section 5).

4.1.2 ge_fromfe_frombytesvartime

Similar to the above, but compressed in another form.

4.1.3 ge_double_scalarmult_base_vartime

Takes as inputs two integers a and b and a point A on ed25519 and returns the point $aA + bG$, where G is the ed25519 base point. Used for the ring signatures when computing, for example, L_i with $i \neq s$ as in ([CN], 4.4)

4.1.4 ge_double_scalarmult_vartime

Takes as inputs two integers a and b and two points A and B on ed25519 and outputs $aA + bB$. Used, for example, when computing the R_i in the ring signatures with $i \neq s$ ([CN], 4.4)

4.1.5 ge_scalarmult

Given a point A on ed25519 and an integer a, this computes the point aA. Used for example when computing L_i and R_i when $i = s$.

4.1.6 ge_scalarmult_base

Takes as input an integer a and computes aG, where G is the ed25519 base point.

4.1.7 ge_p1p1_to_p2

There are different representations of curve points for ed25519, this converts between them. See MiniNero for more reference.

4.1.8 ge_p2_dbl

This takes a point in the "p2" representation and doubles it.

4.1.9 ge_p3_to_p2

Takes a point in the "p3" representation on ed25519 and turns it into a point in the "p2" representation.

4.1.10 ge_mul8

This takes a point A on ed25519 and returns $8A$.

4.1.11 sc_reduce

Takes a 64-byte integer and outputs the lowest 32 bytes modulo the prime q. This is not a CryptoNote-specific function, but comes from the standard ed25519 library.

4.1.12 sc_reduce32

Takes a 32-byte integer and outputs the integer modulo q. Same code as above, except skipping the 64→32 byte step.

4.1.13 sc_mulsub

Takes three integers a, b, c in $\mathbb{F}_q$ and returns $c - ab$ modulo q.

4.2 Important Hashing Functions

4.2.1 cn_fast_hash

Takes data and returns the Keccak1600 hash of the data.

4.3 Crypto.cpp Functions

4.3.1 random_scalar

Generates a 64-byte integer and then reduces it to a 32 byte integer modulo q for 128-bit security as described in section 3.3.

4.3.2 hash_to_scalar

Inputs data (for example, a point P on ed25519) and outputs $\mathcal{H}_s(P)$, which is the Keccak1600 hash of the data. The function then converts the hashed data to a 32-byte integer modulo q.

4.3.3 generate_keys

Returns a secret key and public key pair, using `random_scalar` (as described above) to get the secret key. Note that, as described in [Bern], the key set for ed25519 actually is only multiples of 8 in $\mathbb{F}_q$, and hence `ge_scalarmult_base` includes a `ge_mul8` to ensure the secret key is in the key set. This prevents transaction malleability attacks as described in ([Bern], *c.f.* section on "small subgroup attacks"). This is part of the **GEN** algorithm as described in ([CN], 4.4).

4.3.4 check_key

Inputs a public key and outputs if the point is on the curve.

4.3.5 secret_key_to_public_key

Inputs a secret key, checks it for some uniformity conditions, and outputs the corresponding public key, which is essentially just 8 times the base point times the point.

4.3.6 hash_to_ec

Inputs a key, hashes it, and then does the equivalent in bitwise operations of multiplying the resulting integer by the base point and then by 8.

4.3.7 generate_key_image

Takes as input a secret key x and public key P, and returns $I = x\mathcal{H}_p(P)$, the key image. This is part of the **GEN** algorithm as described in ([CN], 4.4).

4.3.8 generate_ring_signature

Computes a ring signature, performing **SIG** as in ([CN], 4.4) given a key image I, a list of n public keys P_i, and a secret index. Essentially there is a loop on i, and if the secret-index is reached, then an if-statement controls the special computation of L_i, R_i when i is equal to the secret index. The values c_i and r_i for the signature are computed throughout the loop and returned along with the image to create the total signature $(I, c_1, ..., c_n, r_1, ..., r_n)$.

4.3.9 check_ring_signature

Runs the **VER** algorithm in ([CN], 4.4). The verifier uses a given ring signature to compute $L'_i = r_i G_i$, $R'_i = r_i \mathcal{H}_p(P_i) + c_i I$, and finally to check if $\sum_{i=0}^{n} c_i = \mathcal{H}_s(m, L'_0, ..., L'_n, R'_0, ..., R'_n) \ mod\ l$.

4.3.10 generate_key_derivation

Takes a secret key b, and a public key P, and outputs $8 \cdot bP$. (The 8 being for the purpose of the secret key set, as described in 4.3.3). This is used in derive_public_key as part of creating one-time addresses.

4.3.11 derivation_to_scalar

Performs $\mathcal{H}_s(-)$ as part of generating keys in ([CN], 4.3, 4.4). It hashes an output index together with the point.

4.3.12 derive_public_key

Takes a derivation rA (computed via `generate_key_derivation`), a point B, and an output index, computes a scalar via `derivation_to_scalar`, and then computes $\mathcal{H}_s(rA) + B$.

4.3.13 generate_signature

This takes a prefix, a public key, and a secret key, and generates a standard (not ring) transaction signature (similar to a Bitcoin transaction signature).

4.3.14 check_signature

This checks if a standard (not ring) signature is a valid signature.

5 Conclusion

Despite the ring signature functions in the original CryptoNote source being poorly commented, the code can be traced back to established and used sources, and is relatively straightfoward. The Python implementation provided with this review gives further indication of the code's correctness. Furthermore, the elliptic curve mathematics underlying the ring signature scheme has been extremely well-studied; the concept of ring signatures is not novel, even if their application to cryptocurrencies is.

References

BBJLP. Bernstein, Daniel J., et al. "Twisted edwards curves." Progress in Cryptology–AFRICACRYPT 2008. Springer Berlin Heidelberg, 2008. 389-405.

BCPM. Bos, Joppe W., et al. "Selecting Elliptic Curves for Cryptography: An Efficiency and Security Analysis." IACR Cryptology ePrint Archive 2014 (2014): 130.

Bern. Bernstein, Daniel J. "Curve25519: new Diffie-Hellman speed records." Public Key Cryptography-PKC 2006. Springer Berlin Heidelberg, 2006. 207-228.

CH. Chaum, David, and Eugene Van Heyst. "Group signatures." Advances in Cryptology—EUROCRYPT'91. Springer Berlin Heidelberg, 1991.

CN. van Saberhagen, Nicolas. "CryptoNote v 2.0." (2013).

FIPS. SHA, NIST DRAFT. "standard: Permutation-based hash and extendable-output functions." DRAFT FIPS 202 (2014).

Fu. Fujisaki, Eiichiro. "Sub-linear size traceable ring signatures without random oracles." IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences 95.1 (2012): 151-166.

FS. Fujisaki, Eiichiro, and Koutarou Suzuki. "Traceable ring signature." Public Key Cryptography–PKC 2007. Springer Berlin Heidelberg, 2007. 181-200.

IAN. IANIX http://ianix.com/pub/curve25519-deployment.html
Si. Silverman, Joseph H. The arithmetic of elliptic curves. Vol. 106. Dordrecht: Springer, 2009.
SN. http://lab.monero.cc/pubs/multiple_equations_attack.pdf