Expensive Coffee (Ristretto)

(Preventing) exploitation of subgroup cofactors for fun and profit

Brandon Goodell
Contributor at Monero Research Lab
One of several workgroups at The Monero Project

April 18, 2019

My name is Brandon Goodell.

My name is Brandon Goodell. I work pseudonymously as *Surae Noether* for *Monero Research Lab*, a cryptocurrency research group focused on *Monero*.

My name is Brandon Goodell. I work pseudonymously as *Surae Noether* for *Monero Research Lab*, a cryptocurrency research group focused on *Monero*.

Monero is different from Bitcoin. Different protocol (Cryptonote, not Bitcoin), not a fork...digital currency, an emphasis on user privacy.

My name is Brandon Goodell. I work pseudonymously as *Surae Noether* for *Monero Research Lab*, a cryptocurrency research group focused on *Monero*.

Monero is different from Bitcoin. Different protocol (Cryptonote, not Bitcoin), not a fork... digital currency, an emphasis on user privacy.

Privacy is important: Microsoft does not want to reveal information about their spending habits to IBM, and citizens of Tyrrania (North Korea? Venezuela?) want to purchase banned books/evade gov't capital controls without punishment.

I have a personal Clemson tradition.

I have a personal Clemson tradition.

Album recommendation: *Polygondwanaland* by *King Gizzard and the Lizard Wizard* (or maybe *Nonagon Infinity*, which plays forever on a loop, both great albums).

In January of 2017, an easily-overlooked bug in Monero was submitted to our team that would allow a malicious party to "mint" infinite Monero coins.

Okay, now that's out of the way...

In January of 2017, an easily-overlooked bug in Monero was submitted to our team that would allow a malicious party to "mint" infinite Monero coins.

We fixed the bug, we contacted other CryptoNote developers about the bug, and after a period of time, we made a public disclosure of the bug.

In January of 2017, an easily-overlooked bug in Monero was submitted to our team that would allow a malicious party to "mint" infinite Monero coins.

We fixed the bug, we contacted other CryptoNote developers about the bug, and after a period of time, we made a public disclosure of the bug.

The bug was due to a cofactor problem.

In January of 2017, an easily-overlooked bug in Monero was submitted to our team that would allow a malicious party to "mint" infinite Monero coins.

We fixed the bug, we contacted other CryptoNote developers about the bug, and after a period of time, we made a public disclosure of the bug.

The bug was due to a cofactor problem.

What are the consequences? Bytecoin (also Cryptonote), made no fixes, saw 100 BILLION tokens minted (about 1M USD) the day of the Monero's public disclosure ...

Okay, now that's out of the way...

In January of 2017, an easily-overlooked bug in Monero was submitted to our team that would allow a malicious party to "mint" infinite Monero coins.

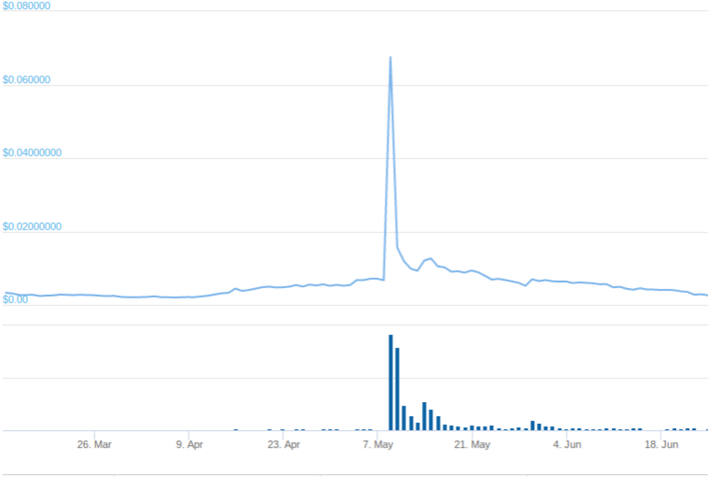
We fixed the bug, we contacted other CryptoNote developers about the bug, and after a period of time, we made a public disclosure of the bug.

The bug was due to a cofactor problem.

What are the consequences? Bytecoin (also Cryptonote), made no fixes, saw 100 BILLION tokens minted (about 1M USD) the day of the Monero's public disclosure ...

What should happen to the exchange rate if the supply skyrockets?

Wrong. A: Pump-and-dump schemes with ostensibly no long-term impact.



To understand what happened.

1. Review of how Cryptonote accomplishes double-spend protection.

To understand what happened.

1. Review of how Cryptonote accomplishes double-spend protection.
2. Description of how the exploit of early 2017 worked.

To understand what happened.

1. Review of how Cryptonote accomplishes double-spend protection.
2. Description of how the exploit of early 2017 worked.
3. How Monero contributors intend to solve such problems at the ground level with Ristretto and Decaf (undergoing present development).

Monero is focused on privacy... and understanding cryptocurrency often boils down to thinking analogously about checks/cheques.

Monero is focused on privacy... and understanding cryptocurrency often boils down to thinking analogously about checks/cheques.

To spend a check, we have to sign the check, and the signature... well... gives you away.

Double-spend protection and signer ambiguity

Monero is focused on privacy... and understanding cryptocurrency often boils down to thinking analogously about checks/cheques.

To spend a check, we have to sign the check, and the signature... well... gives you away.

Q: How to authenticate a sig without revealing the signer?

Double-spend protection and signer ambiguity

Monero is focused on privacy... and understanding cryptocurrency often boils down to thinking analogously about checks/cheques.

To spend a check, we have to sign the check, and the signature... well... gives you away.

Q: How to authenticate a sig without revealing the signer? A: Monero uses ring signatures (trustless setup), Zcash uses SNARKs (trusted setup).

From now on: let G be an elliptic curve group with prime order q and some generator $g \in G$ chosen uniformly at random, let $H_s : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$ be a hash function, and denote concatenation with $||$.

From now on: let G be an elliptic curve group with prime order q and some generator $g \in G$ chosen uniformly at random, let $H_s : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$ be a hash function, and denote concatenation with $||$.

Instead of verifying σ with pubkey $X = g^x$ using privkey x as usual, verify σ with multiset $L = \{X_1, \ldots, X_n\}$. The signature shows the signer knows at least one x_ℓ wihtout revealing which.

From now on: let G be an elliptic curve group with prime order q and some generator $g \in G$ chosen uniformly at random, let $H_s : \{0,1\}^* \to \mathbb{Z}/q\mathbb{Z}$ be a hash function, and denote concatenation with $||$.

Instead of verifying σ with pubkey $X = g^x$ using privkey x as usual, verify σ with multiset $L = \{X_1, \ldots, X_n\}$. The signature shows the signer knows at least one x_ℓ wihtout revealing which.

This is what it means to "spend" x_ℓ.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $c_{\ell+1} = H(L \,||\, m \,||\, g^r)$.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $c_{\ell+1} = H(L \mid\mid m \mid\mid g^r)$.
- For $i = \ell+1, \ell+2, \ldots, \ell-1$ (identifying index n with index 1), compute $c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X_i^{c_i})$.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $c_{\ell+1} = H(L \mid\mid m \mid\mid g^r)$.
- For $i = \ell+1, \ell+2, \ldots, \ell-1$ (identifying index n with index 1), compute $c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X_i^{c_i})$.
- Solve $s_\ell = r - c_\ell x_\ell$.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $c_{\ell+1} = H(L \mid\mid m \mid\mid g^r)$.
- For $i = \ell+1, \ell+2, \ldots, \ell-1$ (identifying index n with index 1), compute $c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X_i^{c_i})$.
- Solve $s_\ell = r - c_\ell x_\ell$.
- Publish $(c_1, s_1, \ldots, s_n)$ along with m and L.

Sign a message m this way:

- Sample $r \in \mathbb{Z}/q\mathbb{Z}$ and, for $1 \leq i \leq n, i \neq \ell$, $s_i \in \mathbb{Z}/q\mathbb{Z}$.
- Compute $c_{\ell+1} = H(L \mid\mid m \mid\mid g^r)$.
- For $i = \ell+1, \ell+2, \ldots, \ell-1$ (identifying index n with index 1), compute $c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X_i^{c_i})$.
- Solve $s_\ell = r - c_\ell x_\ell$.
- Publish $(c_1, s_1, \ldots, s_n)$ along with m and L.

Anyone can sequentially compute $c_2' = H(L \mid\mid m \mid\mid g^{s_1} X_1^{c_1})$, c_3', and so on; valid if $c_{n+1}' = c_1$.

Problem: Alice can generate σ with $L = \{X, Y\}$ and σ' for $L' = \{X, Z\}$.

Problem: Alice can generate σ with $L = \{X, Y\}$ and σ' for $L' = \{X, Z\}$. Bob assumes inter-key collaboration doesn't happen and concludes one of the following mutually exclusive possibilities must have occurred:

- X signed σ and Z signed σ' xor
- Y signed σ and X signed σ' xor
- Y signed σ and Z signed σ'.

Problem: Alice can generate σ with $L = \{X, Y\}$ and σ' for $L' = \{X, Z\}$. Bob assumes inter-key collaboration doesn't happen and concludes one of the following mutually exclusive possibilities must have occurred:

- X signed σ and Z signed σ' xor
- Y signed σ and X signed σ' xor
- Y signed σ and Z signed σ'.

So this is a bad way to spend X: she can double-spend and Bob can't be sure she did so.

Let $H_p : \{0,1\}^n \to G$ be a hash function with independent output compared to H_s. For a private key x and public key $X = g^x$, define a *key image* $\mathfrak{I} = h^x$ where $h = H_p(g^x) \ldots$

Let $H_p : \{0,1\}^n \to G$ be a hash function with independent output compared to H_s. For a private key x and public key $X = g^x$, define a *key image* $\mathfrak{I} = h^x$ where $h = H_p(g^x) \ldots$ (denote $H_p(X_i)$ with h_i for the i^{th} ring member).

Let $H_p : \{0,1\}^n \to G$ be a hash function with independent output compared to H_s. For a private key x and public key $X = g^x$, define a *key image* $\mathfrak{I} = h^x$ where $h = H_p(g^x)\ldots$ (denote $H_p(X_i)$ with h_i for the i^{th} ring member).

Both X and $\mathfrak{I}$ have the same discrete logarithms, although not wrt the same bases, $\mathfrak{I}$ looks random (DDH-ish)...

Let $H_p : \{0,1\}^n \to G$ be a hash function with independent output compared to H_s. For a private key x and public key $X = g^x$, define a *key image* $\mathfrak{I} = h^x$ where $h = H_p(g^x) \ldots$ (denote $H_p(X_i)$ with h_i for the i^{th} ring member).

Both X and $\mathfrak{I}$ have the same discrete logarithms, although not wrt the same bases, $\mathfrak{I}$ looks random (DDH-ish)...

Now the signer computes challenges with

$$c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X_i^{c_i} \mid\mid h_i^{s_i} \mathfrak{I}^{c_i})$$

and publishes $\mathfrak{I}$ along with the signature σ, message m, and ring L in a signature-tag pair $(\sigma, \mathfrak{I})$; link sigs with matching key images.

Problem: Monero uses ring signatures with the Twisted Edwards curve Ed25519, which is not a prime order group!

Let G be a group with order q...

Problem: Monero uses ring signatures with the Twisted Edwards curve Ed25519, which is not a prime order group!

The Ed25519 we use has an order divisible by 8.

Problem: Monero uses ring signatures with the Twisted Edwards curve Ed25519, which is not a prime order group!

The Ed25519 we use has an order divisible by 8.

Factor the group order $\rightarrow$ the large prime-order subgroup of Ed25519 has a cofactor 8. There are cofactor 4 cases that behave very similarly but require less work to fix (see `https://ristretto.group`).

Problem: Monero uses ring signatures with the Twisted Edwards curve Ed25519, which is not a prime order group!

The Ed25519 we use has an order divisible by 8.

Factor the group order $\rightarrow$ the large prime-order subgroup of Ed25519 has a cofactor 8. There are cofactor 4 cases that behave very similarly but require less work to fix (see `https://ristretto.group`).

This means that not all group elements are public keys with corresponding private keys in the large prime subgroup!

Easy problem to fix

Computing key image $\mathfrak{I} = H_p(g^x)^x = h^x$ causes trouble when h is not in the prime-order subgroup.

Easy problem to fix

Computing key image $\mathfrak{I} = H_p(g^x)^x = h^x$ causes trouble when h is not in the prime-order subgroup.

Monero's present solution is a naive fix: have honest parties replace H_p with

$$\widehat{H}_p(X) := H_p(X)^{8I(2|\texttt{ord}(H_p(X)))}$$

where I is an indicator function.

Easy problem to fix

Computing key image $\mathfrak{I} = H_p(g^x)^x = h^x$ causes trouble when h is not in the prime-order subgroup.

Monero's present solution is a naive fix: have honest parties replace H_p with

$$\widehat{H}_p(X) := H_p(X)^{8I(2|\mathtt{ord}(H_p(X)))}$$

where I is an indicator function. We do this when dealing with other adversarially-selected group elements, too.

Easy problem to fix

Computing key image $\mathfrak{I} = H_p(g^x)^x = h^x$ causes trouble when h is not in the prime-order subgroup.

Monero's present solution is a naive fix: have honest parties replace H_p with

$$\widehat{H}_p(X) := H_p(X)^{8I(2|\texttt{ord}(H_p(X)))}$$

where I is an indicator function. We do this when dealing with other adversarially-selected group elements, too.

The exponent "clears" the torsion part of the group element. More generally, we can exponentiate any adversarially-generated group element that isn't on the large prime subgroup.

Easy problem to fix

Computing key image $\mathfrak{I} = H_p(g^x)^x = h^x$ causes trouble when h is not in the prime-order subgroup.

Monero's present solution is a naive fix: have honest parties replace H_p with

$$\widehat{H}_p(X) := H_p(X)^{8I(2|\mathtt{ord}(H_p(X)))}$$

where I is an indicator function. We do this when dealing with other adversarially-selected group elements, too.

The exponent "clears" the torsion part of the group element. More generally, we can exponentiate any adversarially-generated group element that isn't on the large prime subgroup.

But... front-end fixes for a back-end problems play badly with security proofs.

So what about non-honest parties?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8.

So what about non-honest parties?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8. This can be used by some malicious Matthew Macauley to avoid linking, since $\mathfrak{I} \neq \mathfrak{I}_{bad}$ but the exponentiated key images match.

So what about non-honest parties?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8. This can be used by some malicious Matthew Macauley to avoid linking, since $\mathfrak{I} \neq \mathfrak{I}_{bad}$ but the exponentiated key images match.

How to find $\mathfrak{I}_{bad}$?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8. This can be used by some malicious Matthew Macauley to avoid linking, since $\mathfrak{I} \neq \mathfrak{I}_{bad}$ but the exponentiated key images match.

How to find $\mathfrak{I}_{bad}$? Assume c is a hash digest under the random oracle model. and h_{tor} is any non-id element from the 8-torsion subgroup of Ed25519.

So what about non-honest parties?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8. This can be used by some malicious Matthew Macauley to avoid linking, since $\mathfrak{I} \neq \mathfrak{I}_{bad}$ but the exponentiated key images match.

How to find $\mathfrak{I}_{bad}$? Assume c is a hash digest under the random oracle model. and h_{tor} is any non-id element from the 8-torsion subgroup of Ed25519. If $8 \mid c$, then h_{tor}^c is the group identity and $\mathfrak{I}^c = (h_{tor}\mathfrak{I})^c = h_{tor}^c \mathfrak{I}^c$. Moreover, $8 \mid c$ with probability $1/8$.

So what about non-honest parties?

For any pubkey g^x with key image $\mathfrak{I}$ in the prime-order subgroup of Ed25519, $\exists \mathfrak{I}_{bad} \in G \setminus \{\mathfrak{I}\}$ such that $\mathfrak{I}^c = \mathfrak{I}_{bad}^c$ whenever c is divisible by 8. This can be used by some malicious Matthew Macauley to avoid linking, since $\mathfrak{I} \neq \mathfrak{I}_{bad}$ but the exponentiated key images match.

How to find $\mathfrak{I}_{bad}$? Assume c is a hash digest under the random oracle model. and h_{tor} is any non-id element from the 8-torsion subgroup of Ed25519. If $8 \mid c$, then h_{tor}^c is the group identity and $\mathfrak{I}^c = (h_{tor}\mathfrak{I})^c = h_{tor}^c \mathfrak{I}^c$. Moreover, $8 \mid c$ with probability $1/8$.

Okay, how to exploit?

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.

The Key Image Exploit I

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.

The Key Image Exploit I

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.
3. Matthew computes a usual ring signature-tag pair $(\sigma, \mathfrak{I})$.

The Key Image Exploit I

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.
3. Matthew computes a usual ring signature-tag pair $(\sigma, \mathfrak{I})$.
4. Matthew selects a torsion element from G, say h'.

The Key Image Exploit I

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.
3. Matthew computes a usual ring signature-tag pair $(\sigma, \mathfrak{I})$.
4. Matthew selects a torsion element from G, say h'.
5. Matthew computes a bad key image $\mathfrak{I}_{\mathtt{bad}} = h'\mathfrak{I}$.

The Key Image Exploit I

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.
3. Matthew computes a usual ring signature-tag pair $(\sigma, \mathfrak{I})$.
4. Matthew selects a torsion element from G, say h'.
5. Matthew computes a bad key image $\mathfrak{I}_{\texttt{bad}} = h'\mathfrak{I}$.
6. Matthew uses $c_{i+1} = H(L \,||\, m \,||\, g^{s_i} X^{c_i} \,||\, h^{s_i} \mathfrak{I}_{bad}^{c_i})$ to look for a bad ring signature-tag pair $(\sigma_{bad}, \mathfrak{I}_{bad})$.

1. Matthew wants to double-sign with his key x (public key $g^x = X$) using ring L by signing a message m.
2. Matthew constructs a key image basepoint $h = H_p(g^x)$ and the key image $\mathfrak{I} = h^x$.
3. Matthew computes a usual ring signature-tag pair $(\sigma, \mathfrak{I})$.
4. Matthew selects a torsion element from G, say h'.
5. Matthew computes a bad key image $\mathfrak{I}_{\texttt{bad}} = h'\mathfrak{I}$.
6. Matthew uses $c_{i+1} = H(L \mid\mid m \mid\mid g^{s_i} X^{c_i} \mid\mid h^{s_i} \mathfrak{I}_{bad}^{c_i})$ to look for a bad ring signature-tag pair $(\sigma_{bad}, \mathfrak{I}_{bad})$.
7. Matthew publishes $(\sigma, \mathfrak{I})$ and $(\sigma_{bad}, \mathfrak{I}_{bad})$ or tries again with a new message or ring.

The Key Image Exploit II

Detecting this? Similar to the basepoint trick from earlier.

Detecting this? Similar to the basepoint trick from earlier. Good key images are in the large prime subgroup and all other group elements that are "bad key images" are 8-torsion. So check if $\mathfrak{I}^q = 1_G$ (expensive check!) and reject if so.

Detecting this? Similar to the basepoint trick from earlier. Good key images are in the large prime subgroup and all other group elements that are "bad key images" are 8-torsion. So check if $\mathfrak{I}^q = 1_G$ (expensive check!) and reject if so.

This fix, even without Ristretto, is so easy, it makes one wonder why the Bytecoin team didn't implement this when we disclosed the bug to them.

Summing up the problem

The problem began with the cofactor of the large prime subgroup and ended with counterfeited/double-spent currency. Neat.

The problem began with the cofactor of the large prime subgroup and ended with counterfeited/double-spent currency. Neat.

This may be a good point for me to re-iterate that math mistakes are multimillion dollar mistakes leading to space probes to slamming into planets, degradation of global currencies, etc. This job ain't good for stress-related heart palpitations, if you get my meaning.

Decaf was first described by Mike Hamburg at CRYPTO 2015. This presentation is based on documents on https://ristretto.group by Henry de Valence, Isis Lovecruft, and Tony Arcieri (each notably are big critics of Monero).

Ristretto? Decaf?

Decaf was first described by Mike Hamburg at CRYPTO 2015. This presentation is based on documents on https://ristretto.group by Henry de Valence, Isis Lovecruft, and Tony Arcieri (each notably are big critics of Monero).

- Ristretto uses a new type for group elements.

Decaf was first described by Mike Hamburg at CRYPTO 2015. This presentation is based on documents on https://ristretto.group by Henry de Valence, Isis Lovecruft, and Tony Arcieri (each notably are big critics of Monero).

- Ristretto uses a new type for group elements.
- Ristretto uses an equivalence relation to define equality of these types.

Ristretto? Decaf?

Decaf was first described by Mike Hamburg at CRYPTO 2015. This presentation is based on documents on https://ristretto.group by Henry de Valence, Isis Lovecruft, and Tony Arcieri (each notably are big critics of Monero).

- Ristretto uses a new type for group elements.
- Ristretto uses an equivalence relation to define equality of these types.
- Ristretto encodes group elements so that equivalent representatives are encoded identically into bits.

Ristretto? Decaf?

Decaf was first described by Mike Hamburg at CRYPTO 2015. This presentation is based on documents on https://ristretto.group by Henry de Valence, Isis Lovecruft, and Tony Arcieri (each notably are big critics of Monero).

- Ristretto uses a new type for group elements.
- Ristretto uses an equivalence relation to define equality of these types.
- Ristretto encodes group elements so that equivalent representatives are encoded identically into bits.
- Ristretto decodes group elements with automatic validation.
- Ristretto defines a map from bitstrings to group elements for use, e.g. in a hash function with the codomain equal to the Ristretto group.

The ground level approach of Ristretto is what Monero is moving to over the next few years.

The ground level approach of Ristretto is what Monero is moving to over the next few years.

Decaf and Ristretto solve problems like this by constructing a prime order group and defining a new type of point to represent an isomorphism class of an element in a diagram of isomorphisms.

Decaf/Ristretto uses three elliptic curves:

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a, d)$, Twisted Edwards for parameters a, d: points (x, y) such that $ax^2 + y^2 = 1 + dx^2y^2$.

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a, d)$, Twisted Edwards for parameters a, d: points (x, y) such that $ax^2 + y^2 = 1 + dx^2y^2$.
- $\mathcal{M}(A, B)$, Montgomery Curve for parameters A, B: points (u, v) such that $Av^2 = u(u^2 + Bu + 1)$.

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a, d)$, Twisted Edwards for parameters a, d: points (x, y) such that $ax^2 + y^2 = 1 + dx^2y^2$.
- $\mathcal{M}(A, B)$, Montgomery Curve for parameters A, B: points (u, v) such that $Av^2 = u(u^2 + Bu + 1)$.

Useful lemmata: (i) $\exists$ isogenies (and duals) $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{E}(a, d)$ and $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{M}(A, B)$;

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a, d)$, Twisted Edwards for parameters a, d: points (x, y) such that $ax^2 + y^2 = 1 + dx^2y^2$.
- $\mathcal{M}(A, B)$, Montgomery Curve for parameters A, B: points (u, v) such that $Av^2 = u(u^2 + Bu + 1)$.

Useful lemmata: (i) $\exists$ isogenies (and duals) $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{E}(a, d)$ and $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{M}(A, B)$; (ii) if $\alpha = \beta^2$, then $\mathcal{J}(\alpha, \beta)$ has 2-torsion;

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha,\beta)$, Jacobi Quartic for parameters α,β: points (s,t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a,d)$, Twisted Edwards for parameters a,d: points (x,y) such that $ax^2 + y^2 = 1 + dx^2y^2$.
- $\mathcal{M}(A,B)$, Montgomery Curve for parameters A,B: points (u,v) such that $Av^2 = u(u^2 + Bu + 1)$.

Useful lemmata: (i) $\exists$ isogenies (and duals) $\mathcal{J}(\alpha,\beta) \leftrightarrow \mathcal{E}(a,d)$ and $\mathcal{J}(\alpha,\beta) \leftrightarrow \mathcal{M}(A,B)$; (ii) if $\alpha = \beta^2$, then $\mathcal{J}(\alpha,\beta)$ has 2-torsion; (iii) if both a and ad are non-square, then $\mathcal{E}(a,d)$ is complete;

Decaf/Ristretto uses three elliptic curves:

- $\mathcal{J}(\alpha, \beta)$, Jacobi Quartic for parameters α, β: points (s, t) such that $t^2 = \alpha s^4 + 2\beta s^2 + 1$.
- $\mathcal{E}(a, d)$, Twisted Edwards for parameters a, d: points (x, y) such that $ax^2 + y^2 = 1 + dx^2y^2$.
- $\mathcal{M}(A, B)$, Montgomery Curve for parameters A, B: points (u, v) such that $Av^2 = u(u^2 + Bu + 1)$.

Useful lemmata: (i) $\exists$ isogenies (and duals) $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{E}(a, d)$ and $\mathcal{J}(\alpha, \beta) \leftrightarrow \mathcal{M}(A, B)$; (ii) if $\alpha = \beta^2$, then $\mathcal{J}(\alpha, \beta)$ has 2-torsion; (iii) if both a and ad are non-square, then $\mathcal{E}(a, d)$ is complete; (iv) and if $\frac{A+2}{aB}$ is a square, then $\mathcal{M}(A, B)$ is isomorphic to $\mathcal{E}(a, d)$.

There exists a diagram like the following where ϕ, ψ are isogenies. If $\frac{A+2}{a'B}$ is a square and $a' = \pm 1$ and $d' = a'\frac{A-2}{A+2}$, then η is an isomorphism.

$$
\begin{tikzcd}
\mathcal{J}(a^2, a-2d) \arrow[r, leftrightarrow] \arrow[dd, "\phi"] & \mathcal{J}((a')^2, -a'\frac{a'+d'}{a'-d'}) \arrow[d, "\psi"] \\
& \mathcal{M}(A, B) \arrow[d, "\eta"] \\
\mathcal{E}(a, d) & \mathcal{E}(a', d')
\end{tikzcd}
$$

These can be written explicitly. Picking $a' = -a, d' = \frac{ad}{a-d}$ forces the top morphism to be equality (when $a \neq d$).

Theorem: Let $H \subseteq G$ be a normal subgroup and $f : G \to G'$ be a group homomorphism. Then the naturally induced map $\overline{f} : \frac{G}{H} \to \frac{f(G)}{f(H)} \leq \frac{G'}{f(H)}$ is a group homomorphism. Furthermore, if $\ker(\phi) \leq H$ then this is a monomorphism.

Theorem: Let $H \subseteq G$ be a normal subgroup and $f : G \to G'$ be a group homomorphism. Then the naturally induced map $\overline{f} : \frac{G}{H} \to \frac{f(G)}{f(H)} \leq \frac{G'}{f(H)}$ is a group homomorphism. Furthermore, if $\ker(\phi) \leq H$ then this is a monomorphism.

Application: $\phi^* : \mathcal{J} \to \mathcal{E}(a', d')$ as $\phi^* = \eta \circ \psi$, $\ker(\phi) \subseteq \mathcal{J}[2]$ and $\ker(\phi') \subseteq \mathcal{J}[2]$. Now we have some isomorphisms to handle:

$$
\begin{tikzcd}
 & \frac{\mathcal{J}}{\mathcal{J}[2]} \arrow[dl, leftrightarrow, "\cong"'] \arrow[dr, leftrightarrow, "\cong"] & \\
\frac{2\mathcal{E}(a,d)}{\mathcal{E}(a,d)[2]} & & \frac{2\mathcal{E}(a',d')}{\mathcal{E}(a',d')[2]}
\end{tikzcd}
$$

In fact: with cofactor 8, $\frac{2\mathcal{E}}{\mathcal{E}[4]}$ is a prime order group for either $\mathcal{E}$.

So now “all” that remains is to figure out how to encode isomorphism classes of elements in these three groups consistently as bitstrings... and to determine how to go about equality testing.

Ristretto V: Encoding

So now "all" that remains is to figure out how to encode isomorphism classes of elements in these three groups consistently as bitstrings... and to determine how to go about equality testing.

This encoding and equality testing across curves is where the real meat-and-potatoes of Ristretto is.

Additive notation: elements of $\frac{\mathcal{J}}{\mathcal{J}[2]}$ are cosets of the form $(s, t) + \mathcal{J}[2]$.

Additive notation: elements of $\frac{\mathcal{J}}{\mathcal{J}[2]}$ are cosets of the form $(s, t) + \mathcal{J}[2]$. The s value of one of the representative from this coset is canonically selected and encoded into bits.

Additive notation: elements of $\frac{\mathcal{J}}{\mathcal{J}[2]}$ are cosets of the form $(s, t) + \mathcal{J}[2]$. The s value of one of the representative from this coset is canonically selected and encoded into bits. This encoding is then "transported" along the isogenies to $\mathcal{E}$.

Additive notation: elements of $\frac{\mathcal{J}}{\mathcal{J}[2]}$ are cosets of the form $(s,t)+\mathcal{J}[2]$. The s value of one of the representative from this coset is canonically selected and encoded into bits. This encoding is then "transported" along the isogenies to $\mathcal{E}$.

A representative from the coset $(x,y)+\mathcal{E}(a,d)[4]$ can be selected canonically modulo $\mathcal{E}(a,d)[2]$, providing a lifting from $\frac{\mathcal{E}(a,d)}{\mathcal{E}(a,d)[4]}$ to $\frac{\mathcal{E}(a,d)}{\mathcal{E}(a,d)[2]}$ ("torquing").

Additive notation: elements of $\frac{\mathcal{J}}{\mathcal{J}[2]}$ are cosets of the form $(s,t)+\mathcal{J}[2]$. The s value of one of the representative from this coset is canonically selected and encoded into bits. This encoding is then "transported" along the isogenies to $\mathcal{E}$.

A representative from the coset $(x,y)+\mathcal{E}(a,d)[4]$ can be selected canonically modulo $\mathcal{E}(a,d)[2]$, providing a lifting from $\frac{\mathcal{E}(a,d)}{\mathcal{E}(a,d)[4]}$ to $\frac{\mathcal{E}(a,d)}{\mathcal{E}(a,d)[2]}$ ("torquing").

Going deeper into encoding and decoding, equality testing, and hash-to-point functions would be a more intricate deep-dive than this talk. Further details can be found at `https://ristretto.group`.