Why Multisig Is Important – And How To Do It Right

Presented by Luke Parker
@kayabaNerve

Who is this dude?

- Kayaba, or kayabaNerve, online
- Monero ecosystem developer
- Did the first Monero atomic swap
- Amateur cryptographer

Multisignature Wallets

- Multisignature wallets come in two forms.
- n-of-n: Every participant must participate in creating the signature.
- t-of-n: t (the threshold) signers must participate to create the signature.

Why would someone use a multisig wallet?

- Classical uses:
  - Security
  - Redundancy
- Modern uses:
  - Atomic swaps
  - Decentralized custody by protocols
    - Payment channels

Atomic Swaps

- The Monero atomic swap protocol starts by locking Bitcoin, then locking the Monero into a 2-2 multisig.
- This means that both parties' private keys must be used to move the Monero, with Bitcoin scripts forcing the counterparty to reveal their private key.

Decentralized Exchanges

- Decentralized exchanges can operate a t-of-n multisig to hold funds.
- This means that funds are secure so long as t signers are honest.
- This makes the security and decentralization of funds equivalent to the network behind the DEX’s.

Quick Maths

- Elliptic curve cryptography, the backbone of modern cryptocurrencies, follows mathematical rules.
- There are private keys (“scalars”, which are very large numbers) and public keys (“points”).
- Public keys can be added and subtracted, and multiplied against private keys, yet not divided nor multiplied against other public keys.
- A private key can be converted to a public key by multiplying it against the “basepoint” (G).
- $xG = X$ = Public Key of x
- The distributive property holds. $xG + yG = (x + y)G$.
- A n-of-n multisig wallet can be created by summing the public keys of all participants: $X + Y$.
- Its private key would be the sum of the private keys of all participants: $x + y$.

Insecure Key Aggregation

- The described scheme, naive sums, is vulnerable to “Rogue-Key Attacks”.
- Alice creates a private key x, with a public key of X, and sends it to Bob.
- Bob creates a private key of y. They then send a public key of Y - X.
- X + Y - X == Y. Bob’s public key is now the multisig’s public key, giving them full control of it.

Secure Key Aggregation

- The simplest way is to require Bob to prove they know the private key of their claimed public key.
- This is known as a Proof of Possession.
- They published a public key of $Y - X$ which has a private key of $y - x$.
- They know y, yet x is Alice's private key, which they do not know.
- They are accordingly unable to produce a signature for their claimed public key.

Signing

- The simplest signature scheme possible is a Schnorr Signature.
- With a private key, a, select a nonce, r. Calculate the challenge (c) as hash(R, A, message). Calculate the solution (s) as r + ca. Publish the signature, R and s.
- The signature is verifiable by multiplying both sides by G, obtaining the form R + cA == sG. While r and a are private, R and A are public.
- While anyone can select an r and calculate an S for any key, you cannot get s from S, making this unforgeable without knowledge of a.
- If you know r, you can recover the private key from s. That's why it must be private and cannot be reused (as reuse enables recovery of r via a system of equations).

MuSig

- MuSig is a 2-round protocol for producing Schnorr signatures.
- Every participant selects their own nonce, sharing R.
- The signature’s nonce is the sum of the participants’ nonces.
- Once every participant knows the signature’s R, calculate the solution for their nonce and private key.
- The signature’s solution is the sum of the participants’ solutions.
- MuSig was written for usage in Bitcoin and was widely planned for adoption.

Insecure MuSig

- This is the current way we refer to MuSig.
- It was broken.
- It took 2 years.
- Writing secure multisignature algorithms is really hard.

Drijver’s Attack

- The birthday problem asks the question, for n people in a room, what is the probability at least two will share a birthday. Just 23 people is enough to make it more likely than not.
- This is the basis of a generalized problem in cryptography asking for a given hash function, what is the probability of a collision.
- Wagner published an efficient algorithm to discover solutions for this class of problems.
- Drijver proved this was successfully applicable to MuSig.
- By having multiple signing sessions run in parallel, a malicious signer can cause the group to sign for a message it didn’t even know of.
- Just 9 concurrent signing sessions is sufficient to enable this attack.

Drijver's Attack (continued)

- For a malicious signer to forge a signature for message m', they must find a challenge ($c' = hash(R', A, m')$) that is the sum of n other, honest, challenges ($\sum^{n}_{i=1} c_i = hash(R_i, A, m_i)$).

- By manipulating other sessions in real time, it is possible to find a set which sums as needed, even though it isn't feasible to find a single hash which collides (as doing so would mean breaking the hash function).

MuSig2

- One year after Drijver et. al published their work, MuSig2 was published.
- Instead of publishing R, participants publish R_1, R_2.
- A “binding factor” is generated by hashing all of the nonces with the message being signed.
- Each participant’s R is defined as $R_1 + (R_2 * \text{binding factor})$.
- By hashing the selected nonces into a factor for the actual nonce, a malicious signer cannot successfully manipulate them, as any attempt to do so changes the actual nonce in use.

How does all of this relate to Monero?

- CLSAG, the algorithm currently used for ring signatures, produces multiple Schnorr-like signatures.
- This makes the concepts and algorithms behind Schnorr multisignatures applicable to Monero.
- Monero also introduces its own complexities as part of being a privacy coin.

The Burning Bug

- Every output is to a (theoretically) unique public key.
- When you spend an input in Monero, you provide the key image for that public key.
- The key image prevents you from spending an output multiple times, while not revealing which output you’re spending.
- Senders can create output keys which are not actually unique, leaving you with multiple outputs when only one is spendable.
- This was realized years ago and Monero does handle it properly.

The Burning Bug and Multisig

- With Monero’s multisig algorithm, there is a leader who provides a variety of values, from Bulletproofs to the output keys.
- This leader could reuse an output key.
- This could burn all the funds in the multisig.
- :(

How do we prevent these attacks?

- Minimize the amount of data any one participant controls.
- Reduce the amount of data sent around. If a participant can locally create a value, let them. This guarantees it was created as expected.
- If a value must be unique, create it via unique values. For Monero specifically, key images are guaranteed to be unique for any given chain, so hashing them into the output key forces the output key to be unique.

The Burning Bug Multisig Fix

- Output keys were re-defined as the output of a hash function, which the leader provides entropy for.
- By having all parties locally generate them, it minimizes transmitted data while ensuring proper generation.
- The leader’s reduction to entropy piped into a hash function removes their ability to manipulate the output and maintains privacy.
- This hash function is also seeded with key images to ensure the output keys are unique.

Where We Are Now

- Monero’s multisig has, unfortunately, been insecure for a while.
- We have a PR available, authored by perfect-daemon and UkoeHB, which implements the binomial nonce scheme from MuSig2, along with other fixes (such as the one for the burning bug).
- This has been subject to extensive review to ensure we properly fix this and don’t once again put the community at risk.
- This is currently undergoing an audit provided by RINO.

The Future of Multisig

- Monero uses a custom key generation algorithm, and signing algorithm, which is $t = n - 1$ optimized.
- This makes it great for arbitrated escrow situations, such as Haveno, yet slow for larger use cases.
- FROST, a protocol adjacent to MuSig2 yet t-of-n instead of n-of-n, also offers a 2-round protocol while being highly performant, and is my current advocacy.

Final Comments

- Multisig is critical not only as a offering for security, yet also to expand the use cases of Monero, notably around payments and interoperability.
- I actually have a FROST-based Monero multisig library in Rust, which I published shortly before this talk.
- Do not use it.
- Seriously. Don’t use it. It’s not even reviewed.
- If you’re a Rust developer interested in such work, reach out, as *I am hiring*.

Questions?