Monero

Sebastian Kung

November 19, 2018

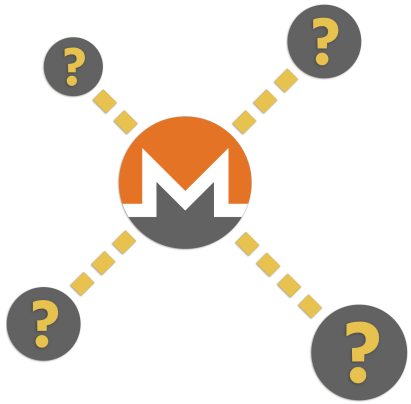
Overview

- What is Monero?
- Elliptic Curve Crypto Primer
- Ring Signatures
- Confidential Transactions
- Stealth Addresses

What is Monero?

Secure

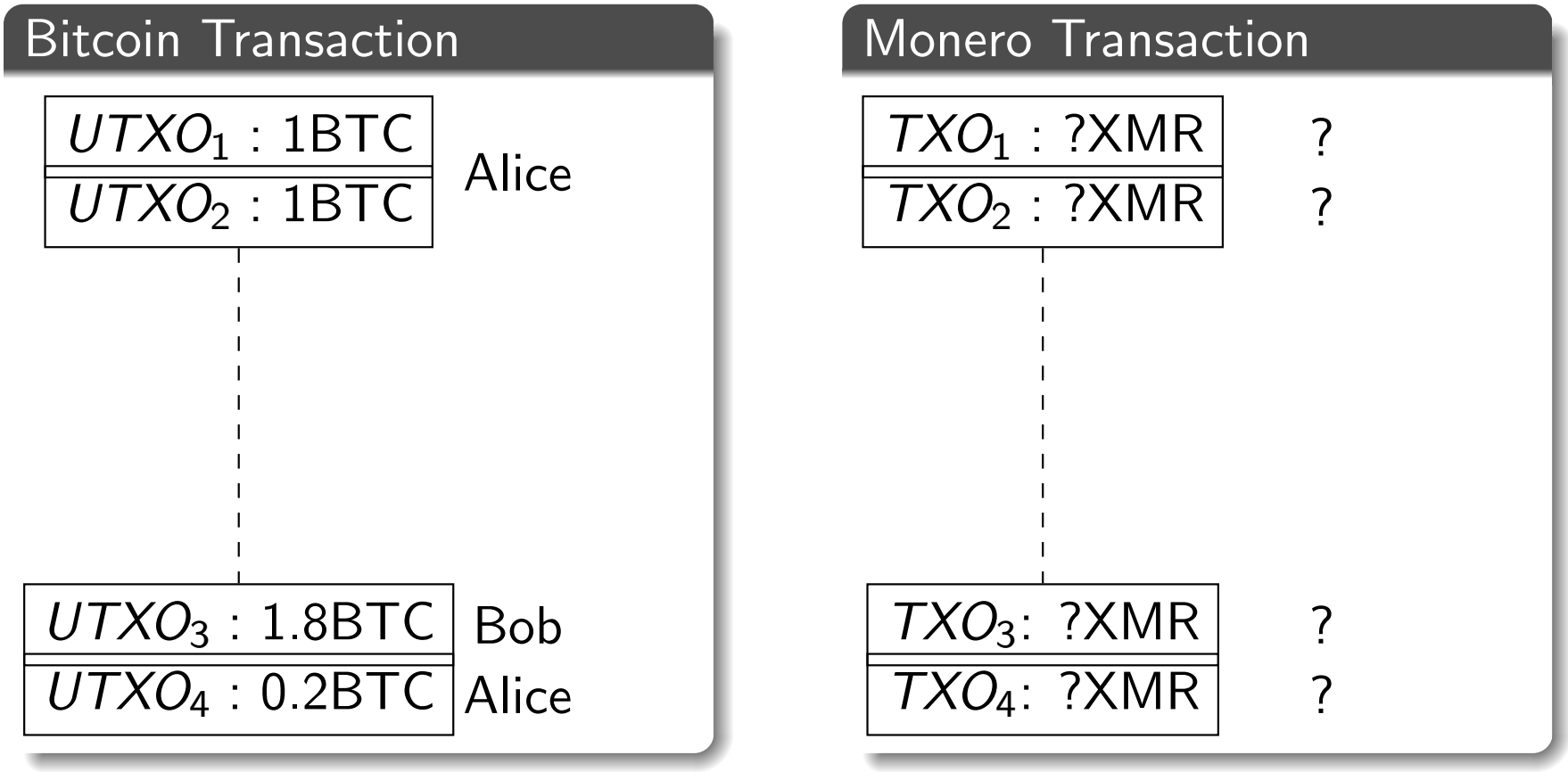
Untraceable



Private

Fungible

Unlinkable Transactions



Other Differences to Bitcoin

- No Scripting
- Cryptonight Proof of Work Algorithm
- LMDB
- Fluffy Blocks

Cryptonight

- 'ASIC-hard' hashing algorithm
- Continuously shuffling and AES-encrypting a 2MB scratchpad gives a certain memory constraint and high amount of branching
- Makes it difficult to implement in an IC

LMDB

- Database to store blockchain
- Quite stable, corruption is seldom
- Stores Key/Value pairs as byte strings
- Purely transactional. No log required as with Berkeley DB

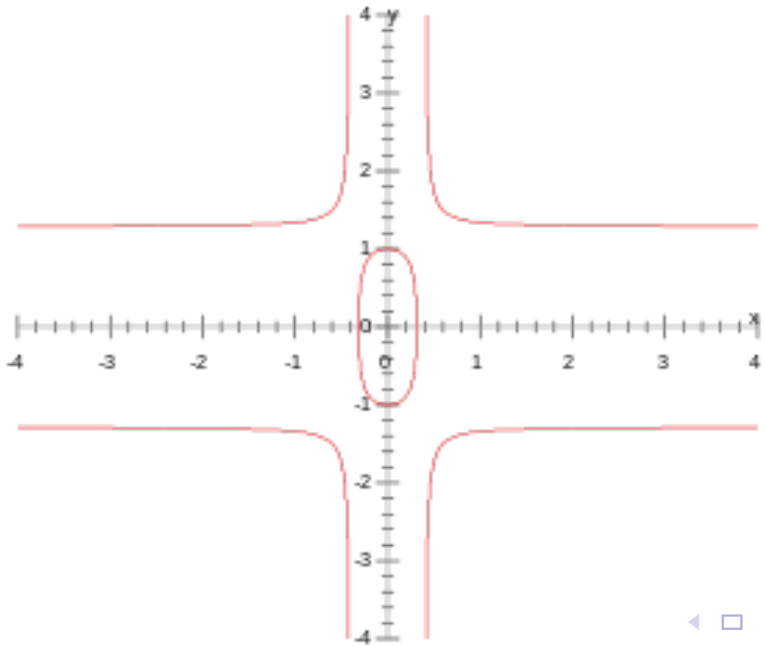
Fluffy Blocks

- Nodes only flood Block Headers and transaction IDs first
- Full block content on demand

Elliptic Curve Crypto Primer

- Monero uses the elliptic curve ED25519 with the KECCAK hashing algorithm
- 5 fundamental crypto 'building blocks' needed to define all operations, 1 assumption on hardness

⚠These are broad simplifications and do not take any underlying nuances into account



Elliptic Curve Crypto Primer

- A public key K is a point on the elliptic curve
- A private key k is a scalar value
- Their relationship is $kG = K$, where G is the unique, publicly known generator point
- Operations on the curve are distributive: $(k + k')G = kG + k'G$
- We can define two different methods for hashing a value, $\mathcal{H}_n(m)$ is a scalar value and $\mathcal{H}_p(m)$ is a point on the curve for arbitrary m
- It is hard to calculate k given G and K

Quick Warmup: Diffie-Hellman

- Produce shared secret S with key pairs (k_A, K_A) and (k_B, K_B)
- Exchange public keys, then
- $S = k_A K_B = k_A k_B G = k_B k_A G = k_B K_A = S$

Schnorr Signatures - Signing

- A and B have key pairs (k_A, K_A) and (k_B, K_B)
- A wants to sign a message m
- A generates random scalar α, computes αG
- Generate a challenge $c = \mathcal{H}(m, \alpha G)$
- Define a response $r := \alpha - ck_A$
- Publish (c, r, K_A, m)

Schnorr Signatures - Verification

- B now has to verify if r is the correct response to the challenge c
- $c' = \mathcal{H}(m, rG + cK_A)$
- If $c = c'$, the Sig is correct

Schnorr Signatures - Explanation

- Goal: Show why $c = c'$
- Remember: $r = \alpha - ck_A$; $c' = \mathcal{H}(m, rG + cK_A)$
- $c' = \mathcal{H}(m, rG + cK_A) = \mathcal{H}(m, (\alpha - ck_A)G + cK_A) =$
  $\mathcal{H}(m, \alpha G - ck_A G + cK_A) = \mathcal{H}(m, \alpha G - cK_A + cK_A) = \mathcal{H}(m, \alpha G) = c$

Ring Signatures

- Goal: Hide transaction origin
- LSAG, Linkable Spontaneous Anonymous Group Signatures
- Linkable: Detect signature re-use (double spend detection)
- Spontaneous: A signature can be produced at any time, no collaboration needed
- Anonymous: Unclear which ring member actually signed

Ring Signatures - Signature

- Let $\mathcal{R}$ be the set of distinct public keys: $\mathcal{R} = \{K_1, K_2, K_3, ..., K_n\}$
- Let u be the index number of the user owned key pair in the ring
- Calculate key image: $\tilde{K} = k_u \mathcal{H}_p(K_u)$
- Generate scalar α and random scalars r_i for $i \in 1, 2, ..., u-1, u+1, ..., n$
- Calculate $c_{u+1} = \mathcal{H}_n(m, \alpha G, \alpha \mathcal{H}_p(K_u))$
- Then calculate iteratively up to $u-1$: $c_{i+1} = \mathcal{H}_n(m, r_i G + c_i K_i, r_i \mathcal{H}_p(K_i) + c_i \tilde{K})$
- Define $r_u = \alpha - c_u k_u$
- Complete sig: $\sigma(m) = \{c_1, r_1, r_2, ..., r_n\}$, extra: $\tilde{K}$, $\mathcal{R}$, m

Ring Signatures - Verification

- For $i = 1, 2, ..., n$ iteratively compute:

$$c'_{i+1} = \mathcal{H}_n(\mathcal{R}, \tilde{K}, m, r_i G + c_i K_i, r_i \mathcal{H}_p(K_i) + c_i \tilde{K})$$

- If $c'_1 = c_1$ then the signature is valid
- In principle like a Schnorr Sig, but with multiple keys

Ring Signatures - Example with 3 ring members

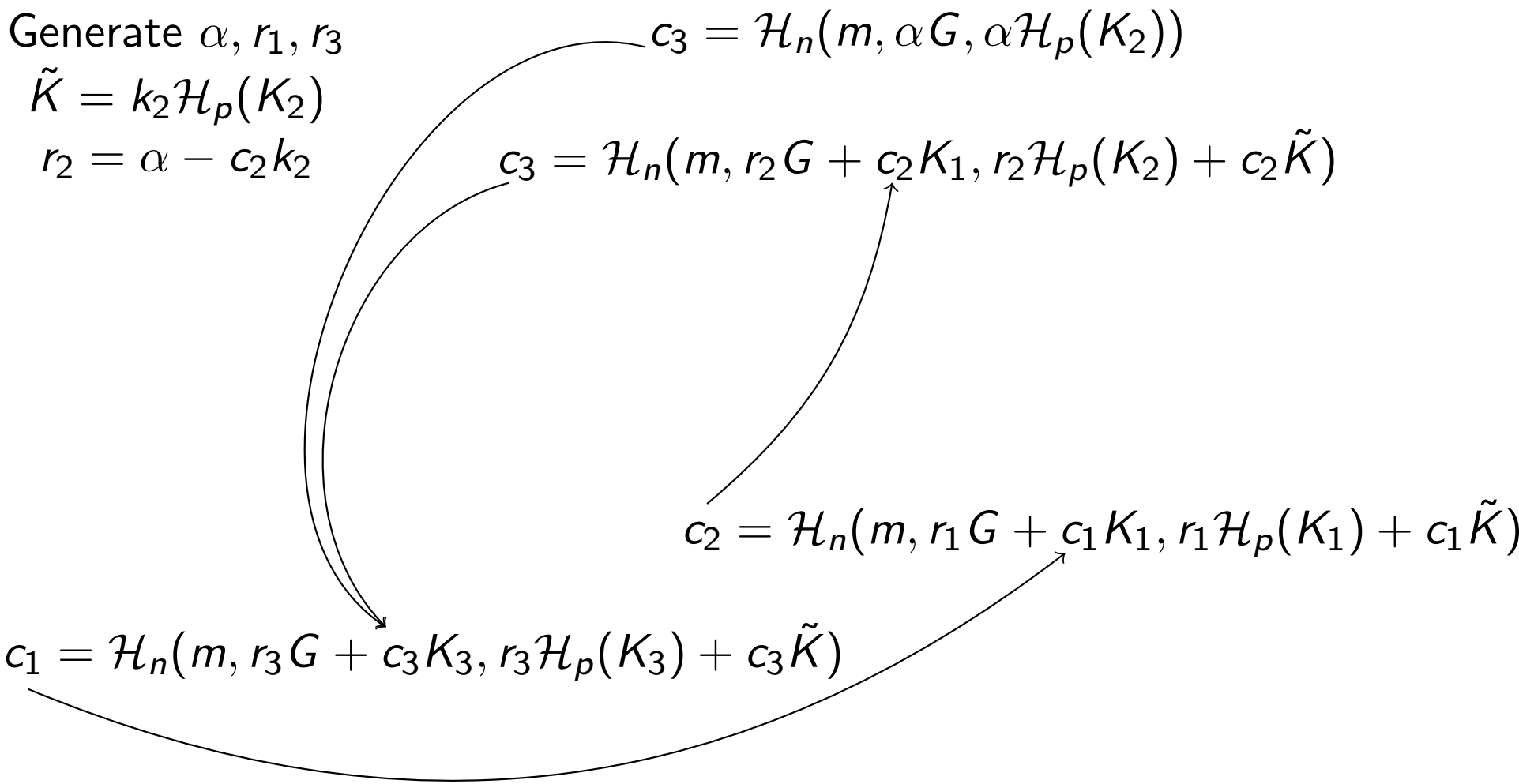
$R = \{K_1, K_2, K_3\}$

Signer: (k_2, K_2)

Generate α, r_1, r_3

$\tilde{K} = k_2 \mathcal{H}_p(K_2)$

$r_2 = \alpha - c_2 k_2$



Confidential Transactions - Pedersen Commitments

- Goal: 'Encrypt' transaction amounts
- Cryptographic Commitments allow you to commit to a specific value
- Pedersen Commitments: Preserve values under addition
- Assuming $a = b$:

$$aG - bG = (a - b)G = 0$$

$$C_a - C_b = C_{a-b} = 0$$

Confidential Transactions

- If we use plain amounts, we can construct lookup tables
- Solution: Add 'blinding factors' x and y: $x - y = 0$
- Use a new generator point $H = \gamma G$ (with unknown scalar γ) for the amounts
- In Monero's case $H = \mathcal{H}_p(G)$

$$(aH + xG) - (bH + yG) = xG - yG = 0$$

$$C_a^x - C_b^y = C_{a-b}^{x-y} = 0$$

Confidential Transactions - Monero

- If we sign a transaction with multiple ring members and have commitments that subtract to zero we can detect which commitment in the signing ring is constructed by us
- Solution, do not use $x{=}y$:

$$(aH + xG) - (bH + yG) = zG$$

$$C_a^x - C_b^y = C^{x-y} = C^z = zG$$

- Sender only needs to prove knowledge of z with a signature

Confidential Transactions - Usage

- Raw amount and blinding factors are encrypted with Diffie-Hellman
- The 'zero commitment' zG is treated as an extra member in the Ring of transaction public keys
- Network only needs to check if the sender knows z by checking its signature
- If the sender forges the amounts, he wont be able to produce a valid signature for zG. Observe for forged amounts $a - b = c$:

$$aH + xG - bH + yG = zG + cH$$

$$C_a^x - C_b^y = C^z + C_c$$

- Which can't be checked, since the network only checks signatures with G

Monero Addresses

- Users of Monero have two key pairs
- View key K^v and spend key K^s
- A monero address is the checksummed concatenation of the public 'view key' and public 'spend key'

One Time Addresses

- Recipient with pairs (k_B^v, k_B^s) and (K_B^v, K_B^s)
- Sender generates random scalar r
- Then calculates one-time key:

$$K^\circ = \mathcal{H}_n(rK_B^v)G + K_B^s$$

- Sender adds K° and rG to the transaction data
- Recipient receives the transaction and calculates $k_B^v rG = rK_B^v$
- When he sees that $K^\circ - \mathcal{H}_n(rK_B^v)G = K_B^s$, he knows the transaction is addresses to him

Developing

- Written in c++
- Standard rpc interface
- All functionality exposed in separate libraries
- Bindings available in java, python, javascript...

Contributing

- Github
- Forum Funding System
- Monero Research Lab
- Kovri

References I

- *Cryptonight hash function*, https://cryptonote.org/cns/cns008.txt, Last Accessed: 2018-11-17.
- *Ring confidential transactions*, https://lab.getmonero.org/pubs/MRL-0005.pdf, Last Accessed: 2018-11-17.
- *An efficient implementation of monero subaddresses*, https://lab.getmonero.org/pubs/MRL-0006.pdf, Last Accessed: 2018-11-17.
- *Thring signatures and their applications to spender-ambiguous digital curencies*, https://eprint.iacr.org/2018/774.pdf, Last Accessed: 2018-11-17.

References II


Zero to monero, https://www.getmonero.org/library/Zero-to-Monero-1-0-0.pdf, Last Accessed: 2018-11-17.

[Zer] [Thr] [Rin] [Sub] [Cry]