Blockchains & Cryptocurrencies

Anonymity in Cryptocurrencies II

Instructors: Matthew Green & Abhishek Jain
Johns Hopkins University - Spring 2019

Housekeeping

- 2nd reminder: we have a midterm on 3/13!
  - This will cover all readings and lectures up through 3/11 (next Monday)
- A1 grades are out
- Assignment 2 due 3/11 end of day
- Note: readings listed post-Spring Break are currently out of whack and will be updated soon

News?

News?

By **JEN WIECZNER** March 5, 2019

Tens of millions of dollars worth of Ethereum frozen in 2017 may finally be recovered, thanks to a new upgrade in the cryptocurrency technology.

More than 500,000 Ethereum—now worth more than $62 million—stored in wallets provided by Parity, a blockchain software company, were essentially lost when a hacker exploited a software bug that froze the funds. But last week's Ethereum hard fork, known as Constantinople (which includes the so-called CREATE2 upgrade), paves the way to getting those funds back, Parity CEO Jutta Steiner says on *Fortune's* latest episode of "Balancing the Ledger."

News?

Matt Suiche Follow

Nov 7, 2017 · 3 min read

As you may have read, Parity issued a security advisory today to inform its users and developers about a bug that got "*accidentally*" triggered which resulted in freezing more than $280M worth of ETH, including $90M belonging to Parity's Founder & Ethereum former core developer: Gavin Woods.

Tuur Demeester

News?

The Bug

A user named devops199 claimed he triggered the bug “accidentally” and reported it through a GitHub ticket.

anyone can kill your contract #6995

Open **devops199** opened this issue a day ago · 12 comments

devops199 commented a day ago • edited

I accidentally killed it.

https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4

News?

Transaction #1

In the above transaction, the user initialized the owner to himself (`0xae7168deb525862f4fee37d987a971b385b96952`) of the Parity library using the `initWallet()` function which is the function that was originally exploited on July 19th. Assigning an owner to the library directly enabled the user to convert the library into a regular multi-sig wallet.

```
// throw unless the contract is not yet initialized.
modifier only_uninitialized { if (m_numOwners > 0) throw; _; }

 // constructor - just pass on the owner array to the multiowned and
 // the limit to daylimit
function initWallet(address[] _owners, uint _required, uint
_daylimit) only_uninitialized {
    initDaylimit(_daylimit);
    initMultiowned(_owners, _required);
 }
```

News?

Transaction #2

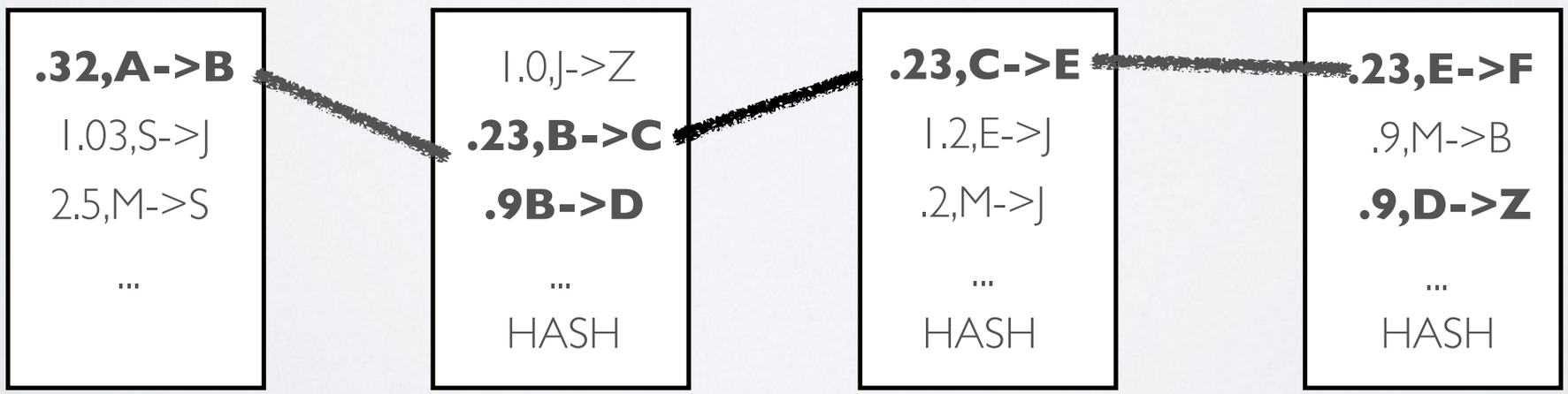
After initializing himself as owner of the library (now regular multi-sig wallet), the user was able to call the `kill()` routine which resulted in paralyzing all the wallets that were dependent of this third party library. This affects all the wallets created after 20th July, since they can't use this library anymore.

```
// kills the contract sending everything to `_to`.
function kill(address _to) onlymanyowners(sha3(msg.data)) external {
    suicide(_to);
 }
```

Review stuff (from last time)

Bitcoin privacy

- **Disadvantage: Bitcoin is not very anonymous**
  - All these transactions are visible to the world, your neighbors, etc.
  - If people can link you to your address, you're screwed
  - How does one address this?



Chaum (CRYPTO '83)

Payer

$\mathbf{SN}_\sigma$

(Blind) signature

sk

Bank

pk

Merchant

Redeem/
verify not previously
spent

Bitcoin privacy solutions (early)

- **Mixes/Laundry services**
  - Create a centralized server, many people send coins
  - Mixer shuffles those together, sends the right amount back to each user (less a fee), thus unlinking the sources of transactions
    - Risk 1: Mixer can "exit" and steal your cash
    - Risk 2: Mixer keeps track of the sources/destinations
    - Risk 3: Low volume of mixing can make tracing easy

Bitcoin privacy solutions (early)

- **CoinJoin**
  - Proposed by Maxwell; variants even earlier by "killerstorm" on BitcoinTalk*
    - Solves the "scamming mixer" problem
- Idea: each transaction has multiple inputs and outputs
  - Have a mixer author one single transaction that consumes N equal-value inputs, produces N outputs
  - All N parties then sign it

Decentralizing E-Cash

- The challenge:
  - Without a central bank we have no more **secret keys** which rules out digital signatures
- We need a fundamentally different ingredient.

CryptoNote & RingCT

- 2012: CryptoNote ("Nicolas van Saberhagen")
  - Originally launched as part of the ByteCoin currency
  - Anonymous creator, did a pre-mine
  - Was forked multiple times into many different currencies, including bitmonero -> Monero
  - Protocol ideas later improved into RingCT, which hid amounts as well as inputs (used in Monero today)

CryptoNote idea

- I want to make a transaction with (e.g.,) one input
  - But I don't want to reveal <u>which</u> transaction is my input
  - Standard Bitcoin transactions do reveal this, and it leads to privacy problems
- I could mix with other people (e.g., CoinJoin) but they would have to participate with me online, and that's annoying

Ingredient: ring signatures

- Normal signature: sign with *sk*, verify with *pk*
- Ring signature:
  - Sign with my secret key + N-1 other people's public keys (Signer does not have to know the other secret keys!)
  - Verifier verifies with all *N* public keys (she must know them)
  - **Privacy**: verifier does not learn which signer actually made the signature! (It could be any of the key owners!)
  - **Unforgeability:** Verifier knows one of them did sign it

CryptoNote idea

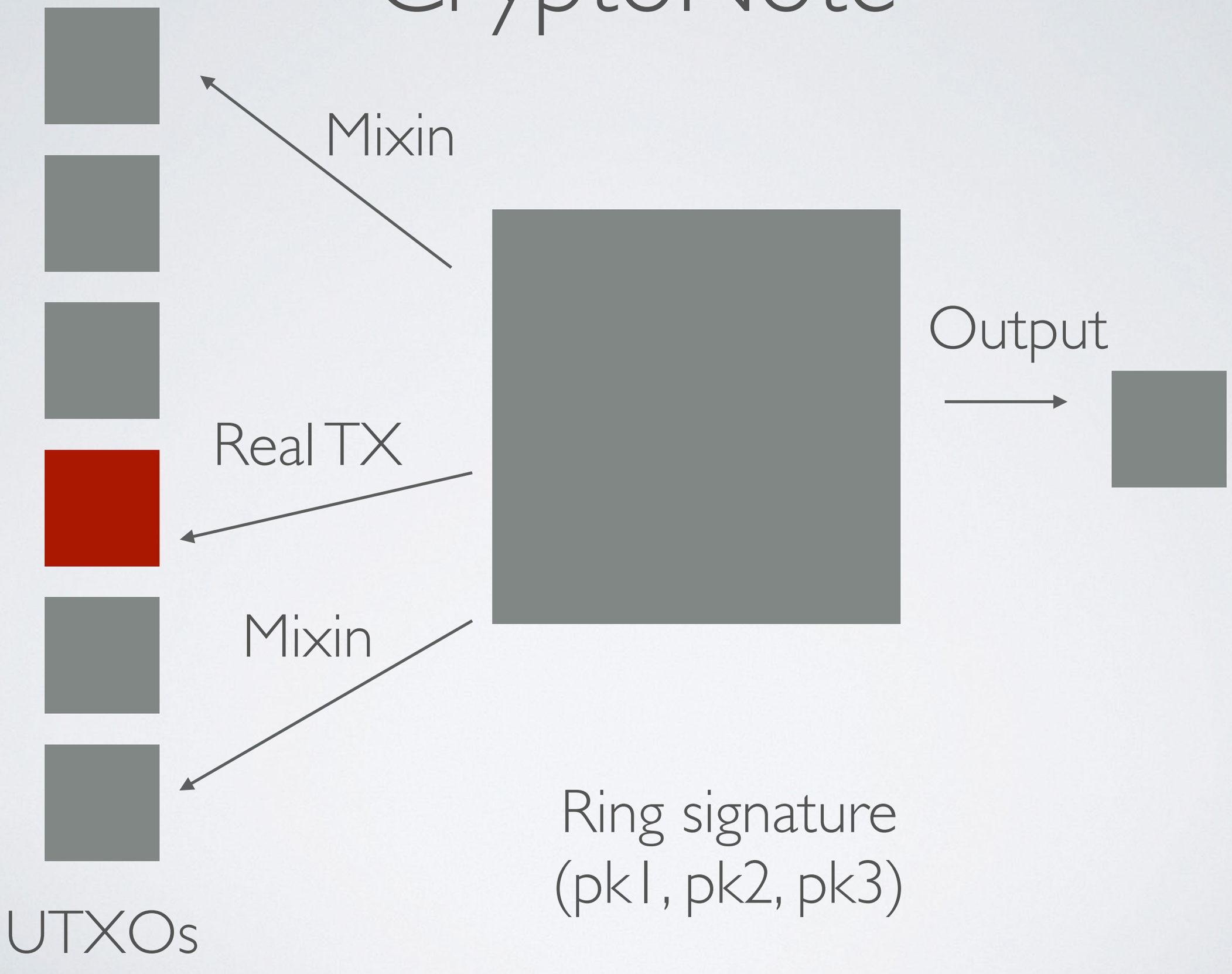
- Make all transactions the same value (e.g., 1 ByteCoin)
- Make all addresses single-use (auto-generated)
- Assume (for simplicity) that spender has one "real" input

1. Identify N-1 unrelated "cover" transactions from the UTXO set, get those public keys ("mixins")

2. Make a ring signature on her transaction, using her secret key and the N-1 public keys for the mixing

3. Post signature plus a "key image" (function of the real secret key) to prevent the real transaction being spent twice

CryptoNote



CryptoNote Limitations

- CryptoNote ring signatures grow as O(N) where N is number of inputs (including Mixins)
  - Ditto signing time and verification time
  - In practice this limits Mixin number to something modestly small (1-7)
- Original CryptoNote required all input transactions be the same value, requiring multiple "denominations" (RingCT fixes this)
- Surprisingly advanced crypto, surprisingly advanced code

RingCT Extension

- Extends this idea to support variable-value transactions
- Add to the transaction a "hidden" transaction value (using encryption and commitments) commitment is less than or equal to input transaction(s)
  - Encrypting destination and value is called "stealth transaction"
- Use zero knowledge proofs (upcoming) to prove that total output transaction value < input values

Ingredient: Commitments

- Like a digital "envelope": allows you to commit to a message value, without revealing what it is
  - C = Commit(message; randomness)
- **Hiding**: given a commitment, can't see what message it is, until I "open" the commitment and reveal it to you
- **Binding**: giving you a commitment "binds" me to a specific message/value. I can't change my mind when I open it.

Hash commitments

- Example: hash commitments
  - Pick some random "salt" (e.g., 256 bits) **r**
  - Compute C = Hash(message || r)
  - To open: reveal (message, r), verifier checks hash

Hash commitments

- Example: hash commitments
  - Pick some random "salt" (e.g., 256 bits) **r**
  - Compute C = Hash(message || r)
- To open: reveal (message, r), verifier checks hash
- **Hiding:** For many common hash functions, it is hard to "invert" C to get back (message, r) so this is hiding — *but what if we didn't add randomness?*
- **Binding:** Finding another pair (message, r) that hashes to C would require finding a collision in the hash function

Pedersen Commitments

- We need a cyclic group. $G = \langle g \rangle$ where it is hard to find *x* given (g, g^x) — AKA the "discrete log problem" (DLP) is hard
  - E.g., G can be a subgroup of a finite field $\{1, \ldots, p-1\}$ where exponentiation/multiplication are modulo *p*

- We also need two public generators: g, h
  such that nobody knows the discrete log of g resp. h (vice versa)

- Commitment to message: pick random $r \in \{0, \ldots, groupOrder - 1\}$, compute: $C = g^m \cdot h^r$

- To open the commitment, simply reveal (m, r)

Pedersen Commitments

- Why is this secure?
  - **Hiding:** If g, h are generators, then h^r is a random element of the group, so. $C = g^m \cdot h^r$ is too

Pedersen Commitments

- Why is this secure?
  - **Hiding:** If g, h are generators, then h^r is a random element of the group, so. $C = g^m \cdot h^r$ is too
  - **Binding:** Let q be the group order. Let $h = g^x$ for some unknown *x*. Assume an attacker can find (m, r) != (m', r') such that $g^m h^r = g^{m'} h^{r'}$. Then it holds that:

$$g^m g^{xr} = g^{m'} g^{xr'}$$

and thus,

$$m + xr = m' + xr' \ mod \ q$$

We can solve for *x*, which means solving the DLP!

Confidential Transactions

- Pedersen commitments are <u>additively homomorphic:</u>

  - Commit to "m1": $C_1 = g^{m_1} h^{r_1}$
    Commit to "m2": $C_2 = g^{m_2} h^{r_2}$

  - Now multiply the two commitments together:

$$\begin{aligned} C_3 &= C_1 \cdot C_2 \\ &= g^{m_1} h^{r_1} \cdot g^{m_2} h^{r_2} \\ &= g^{m_1+m_2} h^{r_1+r_2} \end{aligned}$$

Notice that C3 is a commitment to the <u>sum</u> m1+m2 (under randomness r1+r2)

Confidential Transactions

- Introduced by Maxwell
  - Does not provide privacy for the identity of the input transactions, can be combined with CoinJoin or ringsides
  - Does allow you to hide the value of input transactions
- Basic idea: use a Pedersen commitment to each transaction value, rather than revealing this in cleartext $C = g^v h^r$
- Do a CoinJoin, and use additive property of Pedersen commitments to sum the values and then subtract each output commitment (board)

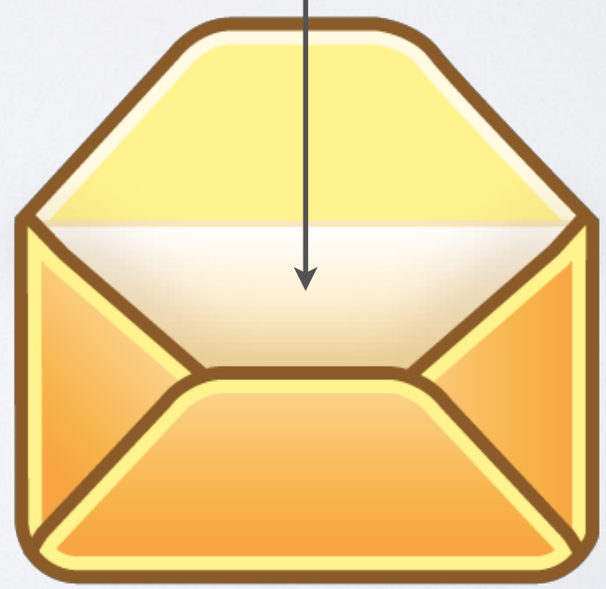
Zerocoin (MGGR14)

- Proposed as an extension to Bitcoin in 2014
  - Requires changes to the Bitcoin consensus protocol!
- I can take Bitcoin from my wallet
- Turn them into ‘Zerocoins’
- Where they get ‘mixed up’ with many other users’ coins
- I can redeem them to a new fresh Wallet

Zerocoin

- Zerocoins are just numbers
  - Each is a digital commitment to a random serial number
  - Anyone can make one!

8238482734710I2983



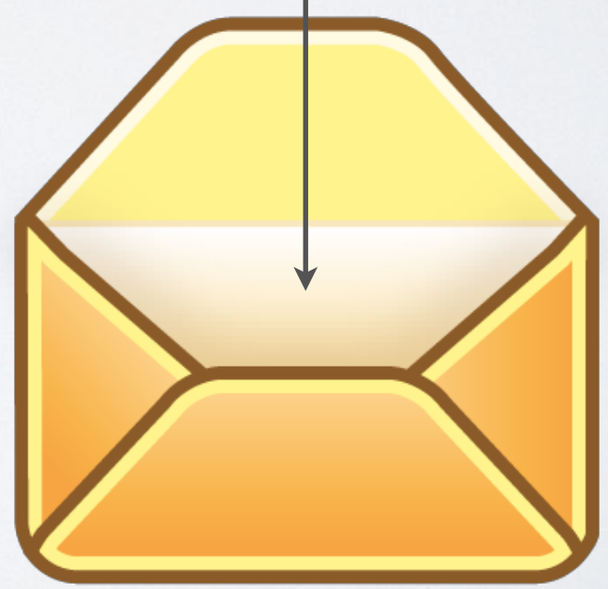
Minting Zerocoin

- Zerocoins are just numbers
  - Each is a digital commitment to a random serial number *SN*
  - Anyone can make one!

$$C = Commit(SN; r)$$

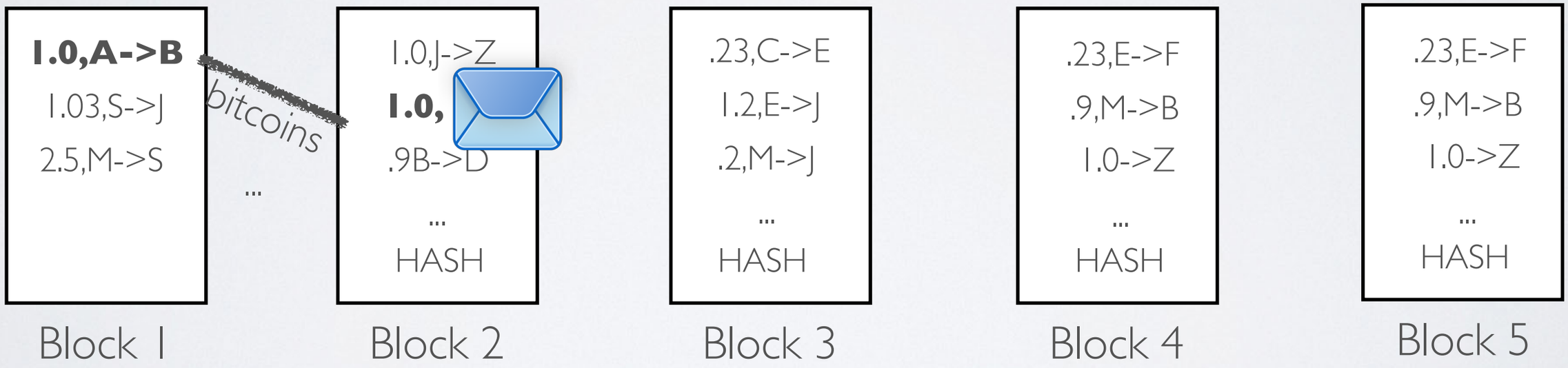
$$C = g^{SN} h^r \ mod \ p$$

8238482734710I2983



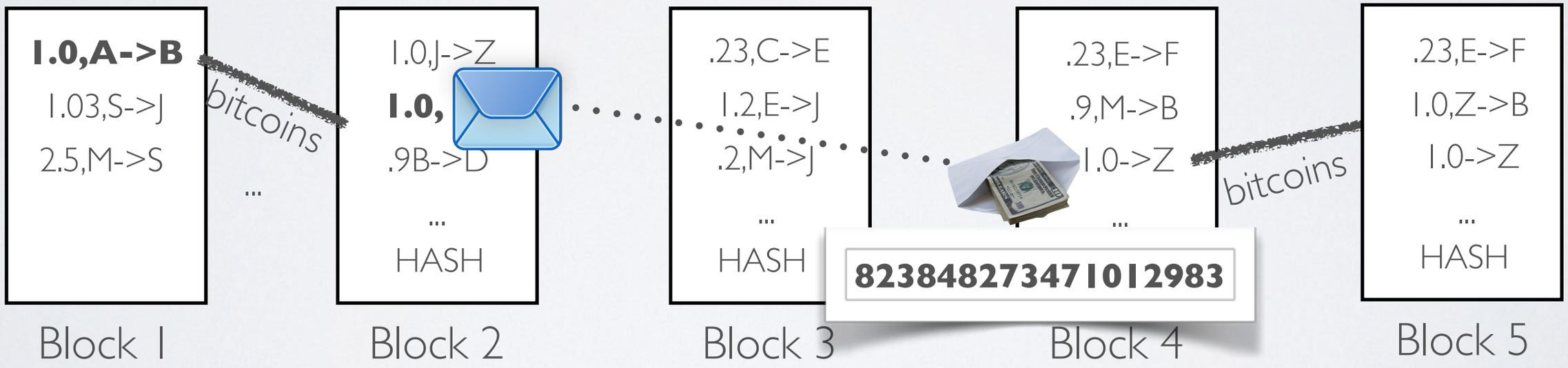
Minting Zerocoin

- Zerocoins are just numbers
  - They have value once you write them into a valid transaction on the blockchain
  - Valid: has inputs totaling some value e.g., 1 bitcoin



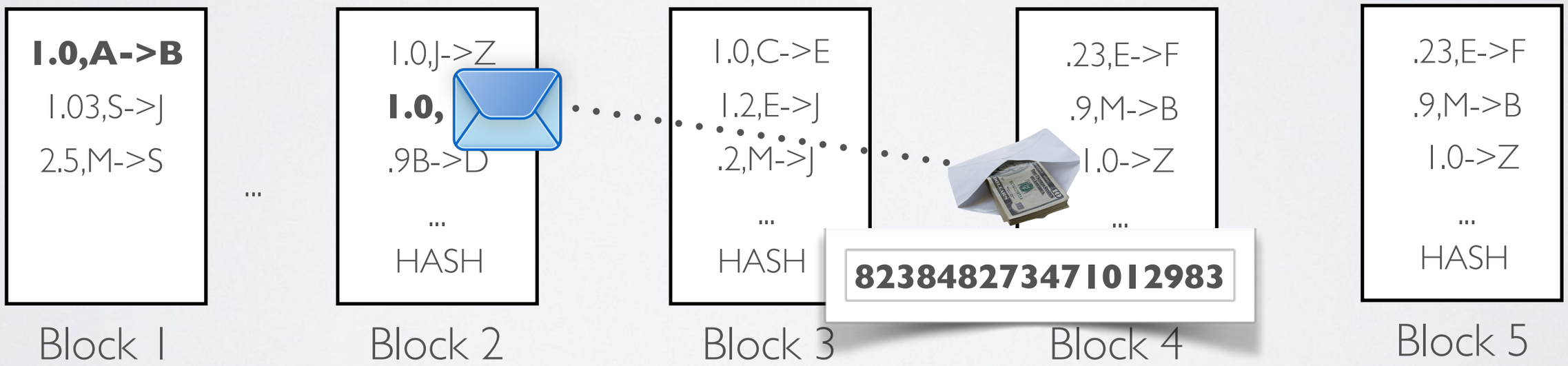
Redeeming Zerocoin

- You can redeem zerocoins back into bitcoins
  - Reveal the serial number &
    Prove that it corresponds to some Zerocoin on the chain
  - In exchange you get one bitcoin (if SN is not already used)



Spending Zerocoin

- Why is spending anonymous?
  - It's all in the way we 'prove' we have a Zerocoin
  - This is done using a zero knowledge proof



Spending Zerocoin

- Zero knowledge [Goldwasser, Micali 1980s, and beyond]
  - Prove a statement without revealing any other information
  - Here we prove that:
    (a) there exists a Zerocoin in the block chain
    (b) we just revealed the actual serial number inside of it
- Revealing the serial number prevents double spending
- The trick is doing this efficiently!

Spending Zerocoin

- Zero knowledge [Goldwasser, Micali 1980s, and beyond]
  - Prove a statement without revealing any other information (other than that a statement is true)

Spending Zerocoin

- Zero knowledge [Goldwasser, Micali 1980s, and beyond]
  - Prove a statement without revealing any other information
  - Here we prove that:
    (a) there exists a Zerocoin (commitment) in the block chain
    (b) the thing we revealed is the opening to that commitment
  - Revealing the serial number prevents double spending
  - The trick is doing this efficiently!

Spending Zerocoin

- Possible proof statement (not efficient, see CryptoNote):
  - Public values: list of Zerocoin commitments $C_1, C_2, \ldots, C_N$
    Revealed serial number *SN*
    - Prove you know a coin *C* and randomness *r* such that:

$$C = C_1 \ \vee \ C = C_2 \ \vee \ \ldots \ \vee \ C = C_N$$
$$\wedge \ C = Commit(SN; r)$$

- Problem: using standard techniques, this ZK proof has cost/size O(*N*)

Spending Zerocoin

- Zerocoin (actual protocol)
  - Use an efficient RSA one-way accumulator
  - Accumulate $C_1, C_2, \ldots, C_N$ to produce a short value A
  - Then prove knowledge of a short witness s.t. $C \in inputs(A)$
  - And prove knowledge that C opens to the serial number

Requires a DDL proof (**~25kb**)
for each spend. In the block chain.

Spending Zerocoin

- Zerocoin (actual protocol)
  - Use an efficient RSA one-way accumulator
  - Accumulate $C_1, C_2, \ldots, C_N$ to produce a short value A
  - Then prove knowledge of a short witness s.t. $C \in inputs(A)$
  - And prove knowledge that C opens to the serial number

Requires a DDL proof (**~25kb**)
for each spend. In the block chain.

Anonymity set comparison

- Anonymity set in CoinJoin:
  - **M**: where **M** is number of inputs in the transaction (bounded by TX size)
- Anonymity set in ByteCoin/RingCT:
  - **N**: where **N** is the number of inputs allowed in a transaction (bounded by TX size, 7-11 historically)
- Anonymity set in Zerocoin:
  - **P**: where **P** is number of total Zerocoins minted on the blockchain thus far* (independent of TX size)

Next time

- So far we've given a very high level view of Zerocoin, CryptoNote/RingCT
  - Next time we'll talk about Zerocash (Zcash) and MimbleWimble
- As well as other online techniques like Tumblebit