Introduction to Blockchain

Lecture 2: Inside Bitcoin

Yury Yanovich

November 01, 2018

Table of Contents

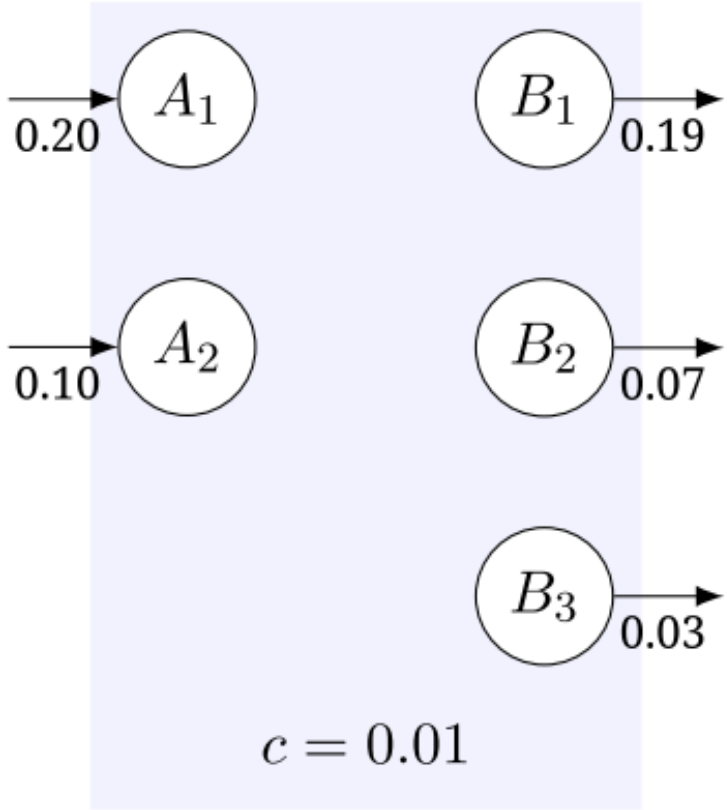
Bitcoin white paper and a bit beyond

Satoshi Nakamoto – Bitcoin: A Peer-to-Peer Electronic Cash System, pp. 1-9, 2008. Online. Available: www.bitcoin.org

White paper outline

1. Introduction
2. Transactions
3. Timestamp Server
4. Proof-of-Work
5. Network
6. Incentive
7. Reclaiming Disk Space
8. Simplified Payment Verification
9. Combining and Splitting Value
10. Privacy
11. Calculations
12. Conclusion

Transactions



- double-spend

General format of a Bitcoin transaction (inside a block)

Field	Description	Size
Version no	currently 1	4 bytes
Flag	If present, always 0001, and indicates the presence of witness data	optional 2 byte array
In-counter	positive integer VI = VarInt	1 - 9 bytes
list of inputs	the first input of the first transaction is also called "coinbase" (its content was ignored in earlier versions)	<in-counter>-many inputs
Out-counter	positive integer VI = VarInt	1 - 9 bytes
list of outputs	the outputs of the first transaction spend the mined bitcoins for the block	<out-counter>-many outputs
Witnesses	A list of witnesses, 1 for each input, omitted if flag above is missing	variable, see Segregated_Witness
lock_time	if non-zero and sequence numbers are <0xFFFFFFFF: block height or timestamp when transaction is final	4 bytes

General format (inside a block) of each input of a transaction - Txin

Field	**Description**	**Size**
Previous Transaction hash	doubled SHA256-hashed of a (previous) to-be-used transaction	32 bytes
Previous Txout-index	non negative integer indexing an output of the to-be-used transaction	4 bytes
Txin-script length	non negative integer VI = VarInt	1 - 9 bytes
Txin-script / scriptSig	Script	<in-script length> -many bytes
sequence_no	normally 0xFFFFFFFF; irrelevant unless transaction's lock_time is >0	4 bytes

General format (inside a block) of each input of a transaction - Txin

Field	**Description**	**Size**
Previous Transaction hash	doubled SHA256-hashed of a (previous) to-be-used transaction	32 bytes
Previous Txout-index	non negative integer indexing an output of the to-be-used transaction	4 bytes
Txin-script length	non negative integer VI = VarInt	1 - 9 bytes
Txin-script / scriptSig	Script	<in-script length> -many bytes
sequence_no	normally 0xFFFFFFFF; irrelevant unless transaction's lock_time is >0	4 bytes

General format (inside a block) of each output of a transaction - Txout

Field	**Description**	**Size**
value	non negative integer giving the number of Satoshis(BTC/10^8) to be transfered	8 bytes
Txout-script length	non negative integer	1 - 9 bytes VI = VarInt
Txout-script / scriptPubKey	Script	<out-script length> -many bytes

Script

Bitcoin uses a scripting system for transactions. Forth-like, Script is simple, stack-based, and processed from left to right. It is intentionally not Turing-complete, with no loops.
A script is essentially a list of instructions recorded with each transaction that describe how the next person wanting to spend the Bitcoins being transferred can gain access to them. The script for a typical Bitcoin transfer to destination Bitcoin address D simply encumbers future spending of the bitcoins with two things: the spender must provide

- a public key that, when hashed, yields destination address D embedded in the script, and
- a signature to prove ownership of the private key corresponding to the public key just provided.

A transaction is valid if nothing in the combined script triggers failure and the top stack item is True (non-zero) when the script exits.

Script-2

De facto, Bitcoin script is defined by the code run by the network to check the validity of blocks.
Opcodes (Script words) examples

- Constants: OP_PUSHDATA1. The next byte contains the number of bytes to be pushed onto the stack.
- Flow control: OP_NOTIF. If the top stack value is False, the statements are executed. The top stack value is removed.
- Stack: OP_DEPTH. Puts the number of stack items onto the stack.

Types of Transaction-1: Pay-to-PubkeyHash

scriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG scriptSig: <sig> <pubKey>

Stack	Script	Description
Empty.	<sig><pubKey>OP_DUP OP_HASH160 <pubKeyHash>OP_EQUALVERIFY OP_CHECKSIG	scriptSig and scriptPubKey are combined.
<sig><pubKey>	OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Constants are added to the stack.
<sig><pubKey> <pubKey>	OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is duplicated.
<sig><pubKey> <pubHashA>	<pubKeyHash>OP_EQUALVERIFY OP_CHECKSIG	Top stack item is hashed.
<sig><pubKey> <pubHashA> <pubKeyHash>	OP_EQUALVERIFY OP_CHECKSIG	Constant added.
<sig><pubKey>	OP_CHECKSIG	Equality is checked between the top two stack items.
true	Empty.	Signature is checked for top two stack items.

A Bitcoin address is only a hash, so the sender can't provide a full public key in scriptPubKey. When redeeming coins that have been sent to a Bitcoin address, the recipient provides both the signature and the public key. The script verifies that the provided public key does hash to the hash in scriptPubKey, and then it also checks the signature against the public key.

Types of Transaction-2: Pay-to-Script-Hash (BIP 0016)

scriptPubKey: OP_HASH160 <scriptHash> OP_EQUAL
scriptSig: ..signatures... <serialized script>
For example, m-of-n multi-signature transaction:
scriptSig: 0 <sig1> ... <script> script: OP_m <pubKey1> ... OP_n OP_CHECKMULTISIG

Stack	Script	Description
Empty.	0 <sig1><sig2>OP_2 <pubKey1><pubKey2> <pubKey3>OP_3 OP_CHECKMULTISIG	Only the scriptSig is used.
0 <sig1><sig2>OP_2 <pubKey1> <pubKey2><pubKey3>OP_3	OP_CHECKMULTISIG	Constants are added to the stack.
true	Empty	Signatures validated in the order of the keys in the script.

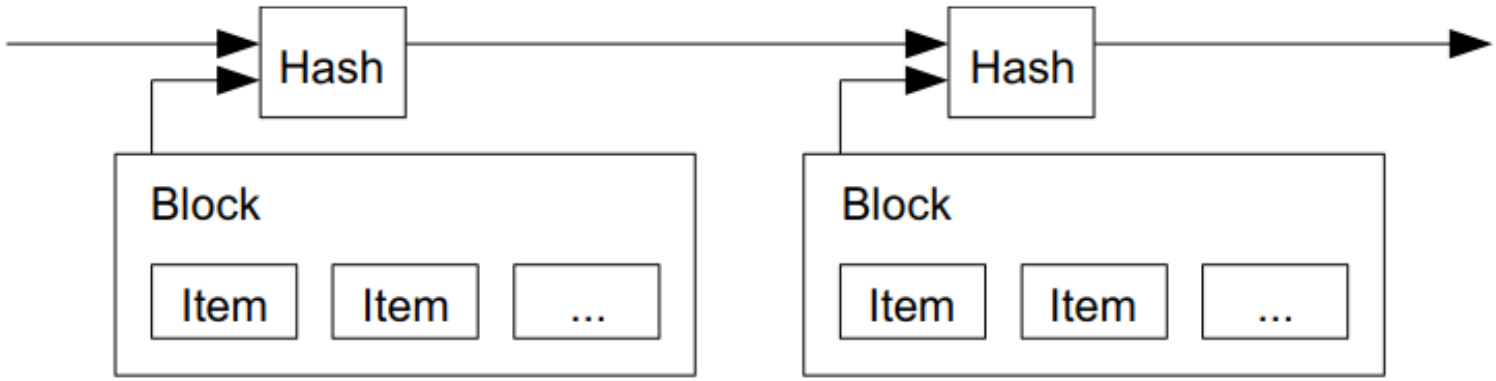
P2SH addresses were created with the motivation of moving "the responsibility for supplying the conditions to redeem a transaction from the sender of the funds to the redeemer. They allow the sender to fund an arbitrary transaction, no matter how complicated, using a 20-byte hash"1. Pay-to-Pubkey-hash addresses are similarly a 20-byte hash of the public key.
Pay-to-script-hash provides a means for complicated transactions, unlike the Pay-to-pubkey-hash, which has a specific definition for scriptPubKey, and scriptSig. The specification places no limitations on the script, and hence absolutely any contract can be funded using these addresses.

Types of Transaction-3: Generation

The Coinbase transaction, or Generation transaction, is a special transaction in the Bitcoin protocol that differs from a standard transaction as it creates coins from nothing.

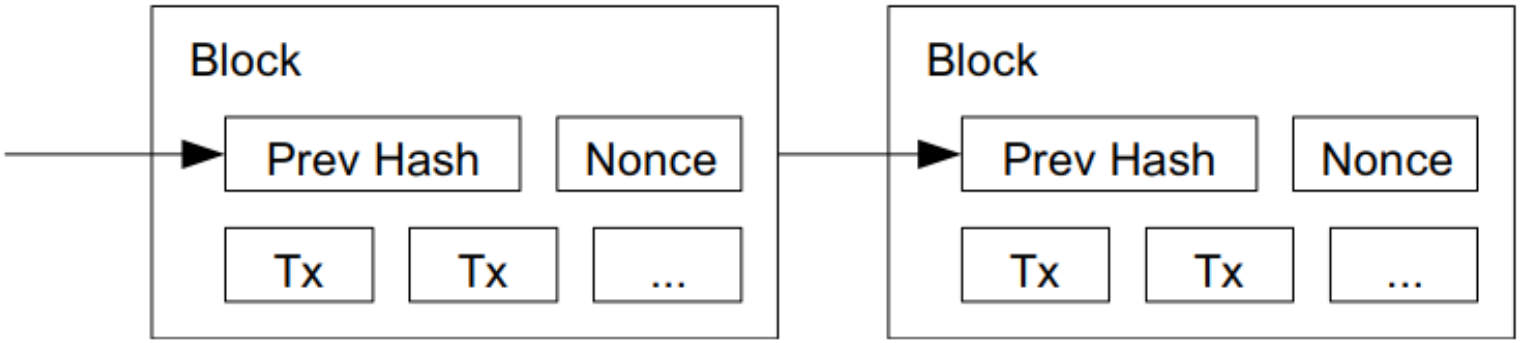
- It is the reward that miner gets for successfully mining a block.
- reward dependents on the number of blocks from the genesis block and the number of fees included in the transactions of the block.
- The transaction hash is special alongside the output index where all the bits are set to o and 1 respectively. The coinbase data size is the size of the next field or the coinbase data field.
- The coinbase data field can contain information such as the extra nonce, merged mining information and a list of the addresses of the mining pool where the funds are to be sent. The sequence number – all bits are set to 1.
- Once a block has been mined and either sent to one address or the miners who participated in pool the block reward has to be confirmed by 100 blocks – or roughly six hours.

Timestamp Server



The solution we propose begins with a timestamp server. A timestamp server works by taking a hash of a block of items to be timestamped and widely publishing the hash, such as in a newspaper or Usenet post. The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash. Each timestamp includes the previous timestamp in its hash, forming a chain, with each additional timestamp reinforcing the ones before it.

Proof-of-Work



- one-CPU-one-vote
- longest chain = greatest proof-of-work effort invested

Network

1. New transactions are broadcast to all nodes.
2. Each node collects new transactions into a block.
3. Each node works on finding a difficult proof-of-work for its block.
4. When a node finds a proof-of-work, it broadcasts the block to all nodes.
5. Nodes accept the block only if all transactions in it are valid and not already spent.
6. Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Network: node's data structures

Transactions

- **transaction pool:** an unordered collection of transactions that are not in blocks in the main chain, but for which we have input transactions
- **orphan transactions:** transactions that can't go into the pool due to one or more missing input transactions.

Blocks

- **blocks in the main branch:** the transactions in these blocks are considered at least tentatively confirmed
- **blocks on side branches off the main branch:** these blocks have at least tentatively lost the race to be in the main branch
- **orphan blocks:** these are blocks which don't link into the main branch, normally because of a missing predecessor or nth-level predecessor.

Network: process transaction-1

1. Check syntactic correctness
2. Make sure neither in or out lists are empty
3. Size in bytes <= MAX_BLOCK_SIZE
4. Each output value, as well as the total, must be in legal money range
5. Make sure none of the inputs have hash=0, n=-1 (coinbase transactions)
6. Check that nLockTime <= INT_MAX (nLockTime must not exceed 31 bits, as some clients will interpret it incorrectly), size in bytes >= 100 (a valid transaction requires at least 100 bytes. If it's any less, the transaction is not valid), and sig opcount <= 2 (the number of signature operands in the signature for standard transactions will never exceed two)
7. Reject "nonstandard" transactions: scriptSig doing anything other than pushing numbers on the stack, or scriptPubkey not matching the two usual forms (this is not a hard requirement on clients)

Network: process transaction-2

8. Reject if we already have matching tx in the pool, or in a block in the main branch
9. For each input, if the referenced output exists in any other tx in the pool, reject this transaction (not a hard requirement on clients).
10. For each input, look in the main branch and the transaction pool to find the referenced output transaction. If the output transaction is missing for any input, this will be an orphan transaction. Add to the orphan transactions, if a matching transaction is not in there already.
11. For each input, if the referenced output transaction is coinbase (i.e. only 1 input, with hash=0, n=-1), it must have at least COINBASE_MATURITY (100) confirmations; else reject this transaction
12. For each input, if the referenced output does not exist (e.g. never existed or has already been spent), reject this transaction (protection against double-spending).

Network: process transaction-3

13. Using the referenced output transactions to get input values, check that each input value, as well as the sum, are in legal money range
14. Reject if the sum of input values $<$ sum of output values
15. Reject if transaction fee (defined as sum of input values minus sum of output values) would be too low to get into an empty block
16. Verify the scriptPubKey accepts for each input; reject if any are bad
17. Add to transaction pool (when the transaction is accepted into the memory pool, an additional check is made to ensure that the coinbase value does not exceed the transaction fees plus the expected BTC value)
18. "Add to wallet if mine"
19. Relay transaction to peers
20. For each orphan transaction that uses this one as one of its inputs, run all these steps (including this one) recursively on that orphan

Network: process block

Specific points

- Verify Merkle hash
- Reject if timestamp is the median time of the last 11 blocks or before
- For certain old blocks (i.e. on initial block download) check that hash matches known values
- Main brank, side chain and orphan blocks are supported.

Network and Reclaiming Disk Space

Network

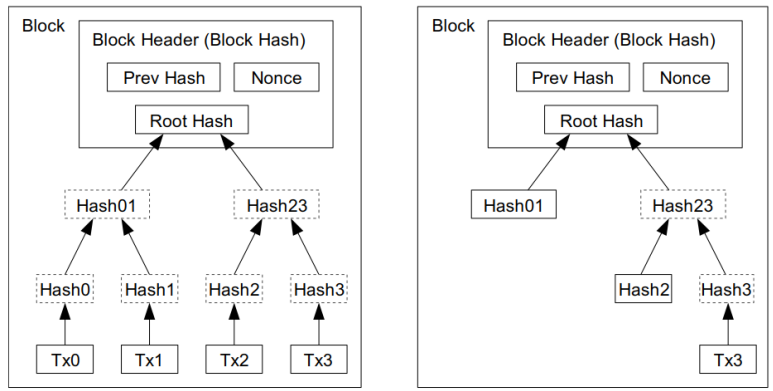
- the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block
- can also be funded with transaction fees

Network and Reclaiming Disk Space

Network

- the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block
- can also be funded with transaction fees

Reclaiming Disk Space

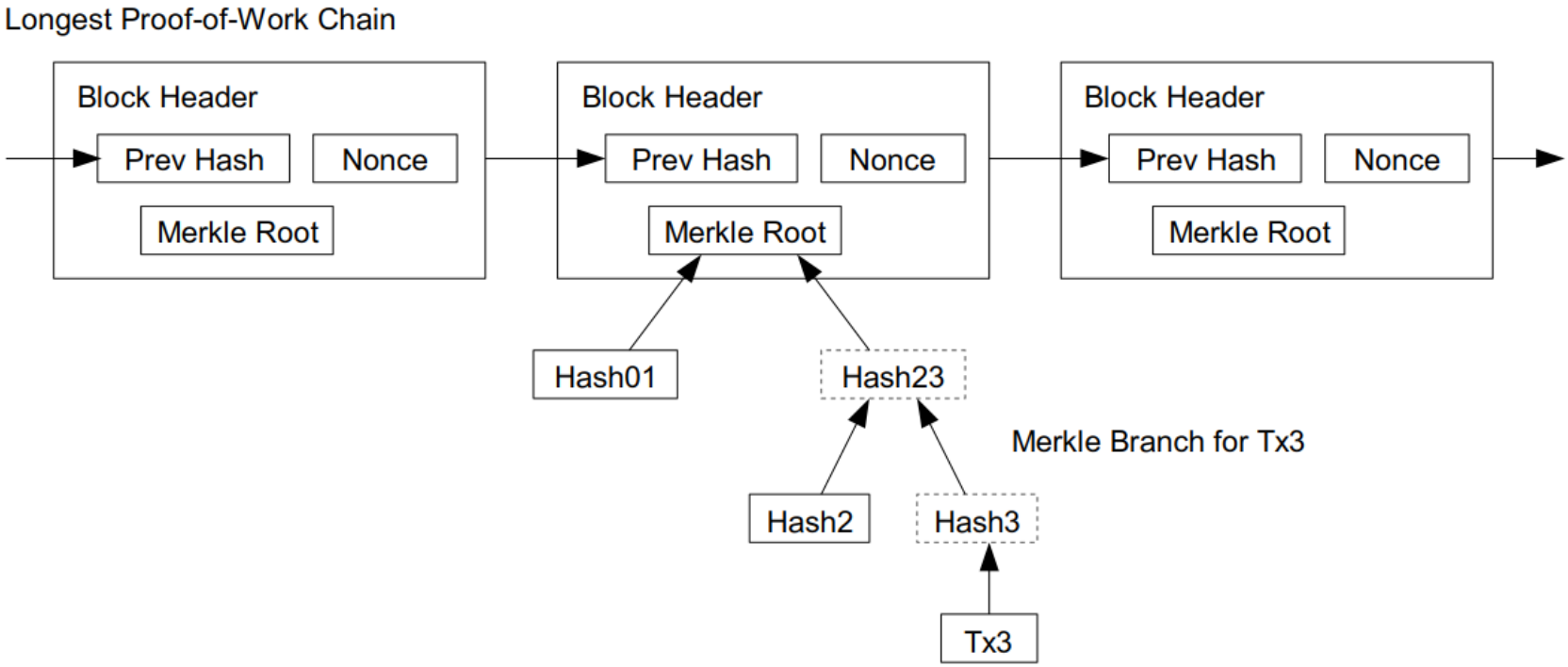


Transactions Hashed in a Merkle Tree

After Pruning Tx0-2 from the Block

A block header $\sim$ 80 bytes. If we suppose blocks are generated every 10 minutes, 80 bytes * 6 * 24 * 365 = 4.2MB per year. With computer systems typically selling with 2GB of RAM as of 2008, and Moore's Law predicting current growth of 1.2GB per year, storage should not be a problem even if the block headers must be kept in memory.

Simplified Payment Verification



Combining and Splitting Value

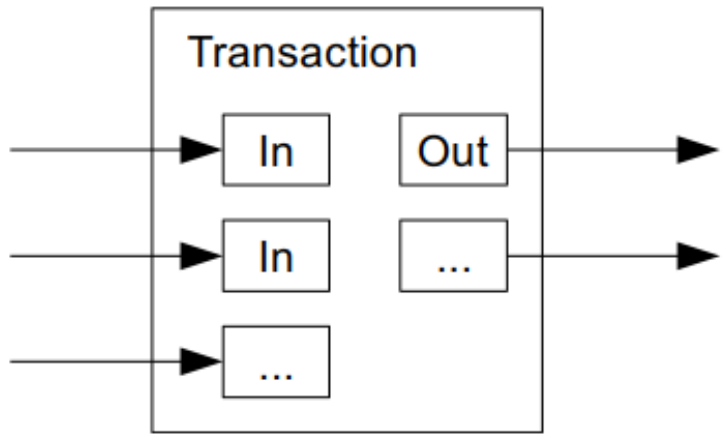
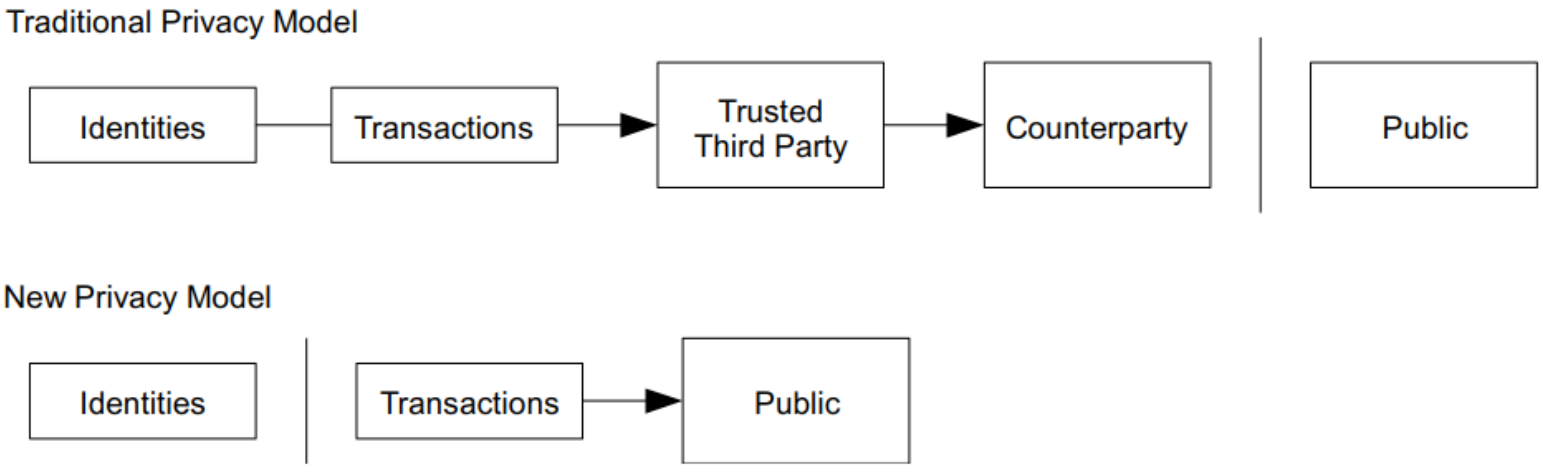


Table of Contents

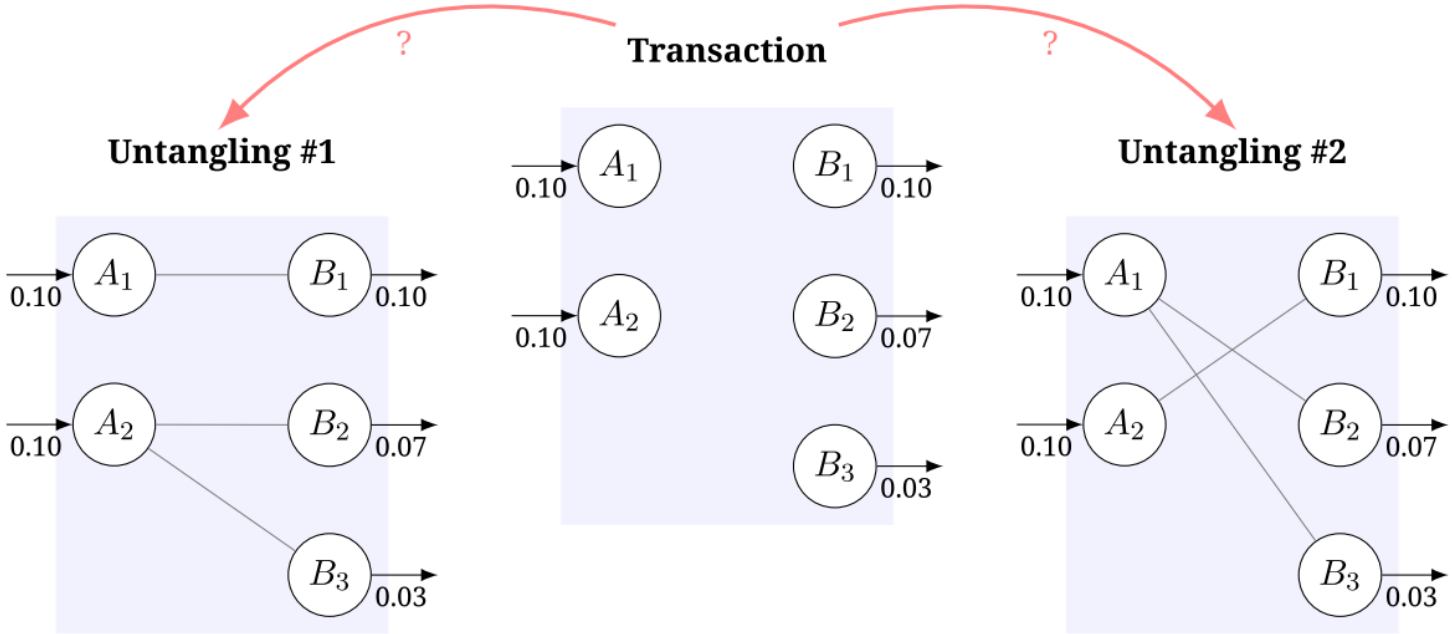
Privacy


Traditional Privacy Model
Identities
Transactions
Trusted
Third Party
Counterparty
Public
New Privacy Model
Identities
Transactions
Public

Bitcoin and mixing

- shared send and shared coin mixers

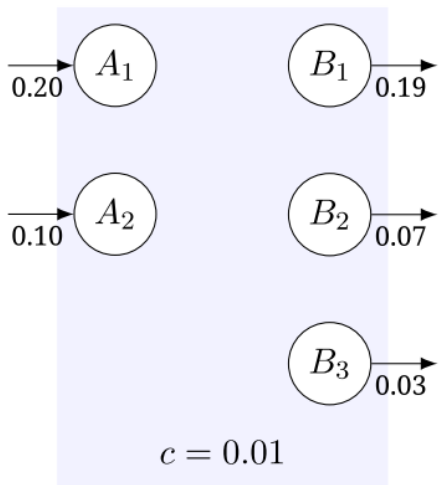
`http://bitfury.com/content/5-white-papers-research/bitfury_whitepaper_shared_send_untangling_in_bitcoin_8_24_2016.pdf`



Transaction for address clustering

For the purpose of transaction analysis, a (Bitcoin) transaction is viewed as an ordered triplet $t = (\mathcal{A}, \mathcal{B}, c)$, consisting of:

- The finite multiset of transaction inputs $\mathcal{A}$, where each input $(a_i, A_i) \in \mathcal{A}$ is an ordered pair of the address A_i and the value of the input $a_i > 0$.
- The finite multiset of transaction outputs $\mathcal{B}$, where each output $(b_j, B_j) \in \mathcal{B}$ is an ordered pair of the address B_j and the value of the output $b_j \geq 0$.
- The transaction fee $c = \sum_{(a_i,\cdot)\in\mathcal{A}} a_i - \sum_{(b_j,\cdot)\in\mathcal{B}} b_j \geq 0$.



Bitcoin Clustering Problem

Assumptions

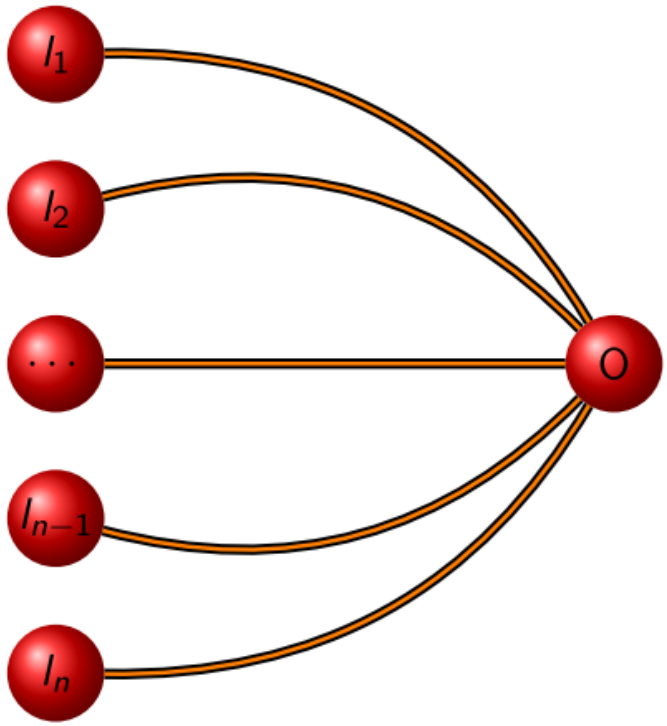
- Each Bitcoin address is controlled by a single real-world entity. Thus, we will ignore those sufficiently rare cases in which a multi-signature address is used for joint ownership of bitcoins, and not for multi-factor authentication.
- A single entity may control more than one address.

Goal: Bitcoin clustering algorithm

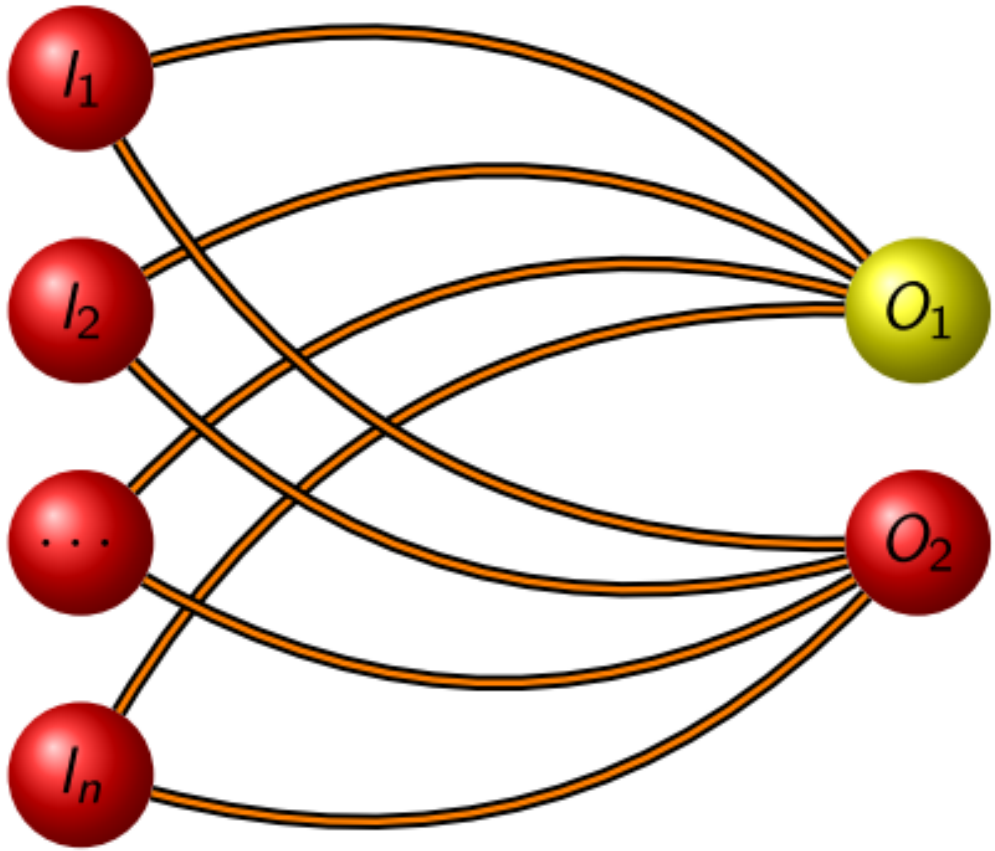
- minimal number of clusters
- all the addresses in each cluster are controlled by the same user.

Common Spending

If two or more addresses are inputs of the same transaction with one output, then all these addresses are controlled by the same user.



One-time Change



One-time Change (2)

We say that the transaction $t = (\mathcal{A}, \mathcal{B}, c)$ satisfies the condition of a one-time change if the following conditions hold.

1. $\#Addr(\mathcal{B}) = 2$, i.e. the transaction t has exactly two outputs.
2. $\#Addr(\mathcal{A}) \neq 2$, i.e. the number of t inputs is not equal to two. If $\#Addr(\mathcal{A}) = \#Addr(\mathcal{B}) = 2$ the transaction is most likely shared send mixer.
3. Both outputs of transaction t, B_1 and B_2, are not self-change addresses, i.e. $B_1, B_2 \notin Addr(\mathcal{A})$.
4. One output of the transaction B_1 did not exist before transaction t and decimal representation of the value b_1 has more than 4 digits after the dot.
5. The other output of the transaction B_2 was previously part of the Bitcoin network and has not been OTC addressed in previous transactions.

The OTC output and all the inputs of the transaction are controlled by the same user.

Off-Chain information

Much public information can be found on the Internet (off-chain information).
If the Bitcoin address is mentioned in the same data frame with the tag (key phrase-entity, for example, company name or username), then it is said say that the address has such a tag.

Tag collection

- passive approach: web crawling of public forums and user profiles (for example, *Bitcointalk.com*, *Twitter* and *Reddit*) and Darknet markets (for example, *Silkroad*, *The Hub Marketplace* and *Alphabay*)
- active approach: manual analysis of Bitcoin companies and data actualization procedures (*Satoshi Bones* casino uses *1change* and *1bones* prefixes and *BTC-E* exchange uses *1eEUR* and *1eUSD* prefixes, etc.).

Bitcoin Organization Types

We distinguish six categories of Bitcoin organizations

- mining pools (pools)
- exchanges
- Darknet markets (dnm)
- mixers
- gambling
- other services (services).

Unique clean tags per category

services	gambling	mixer	dnm	exchange	pool
57	80	3	16	98	52

Let us call $L = \{\{a_i, a_j\}\}$ the set of negative pairs, where addresses a_i and a_j in each pair have either different tags from the same category or tags from a forbidden pair of categories.

Bitcoin Organization Types

	services	gambling	mixer	dnm	exchange	pool
services	165,970	66,387	0	0	441	186
gambling	66,387	11,9857	0	0	0	9
mixer	0	0	0	0	1,703	1
dnm	0	0	0	0	13	18
exchange	441	0	1,703	13	606,357	198,551
pool	186	9	1	18	198,551	1,561

Table: The number of addresses with distinct clean tags in respective pairs of categories.

	services	gambling	mixer	dnm	exchange	pool
services	F	A	F	F	F	A
gambling	A	F	F	F	F	A
mixer	F	F	F	F	F	F
dnm	F	F	F	F	F	F
exchange	F	F	F	F	F	A
pool	A	A	F	F	A	F

Table: Table shows if the appearance of addresses having two distinct clean tags in the same cluster should be forbidden. "F" stays for forbidden and "A" is for allowed.

Probabilistic model

We note that both CS and OTC heuristics and the set of negative pairs L may contain erroneous information.

Different types of observations (which we *treat as an independent* to make it computationally solvable):

- In the event that all the addresses $Addr_H(t)$ for some $t \in T_H$ indeed belong to the same user is true with probability p.
- In the event that two addresses $\{a_i, a_j\} \in L$ are controlled by the same user is true with probability q. In other words, the information about the negative association between any pair of addresses in L is validated by the probality $1 - q$.

Probabilistic model (2)

Let the likelihood $\mathbb{P}(A, T_H, L \mid p, q)$ be a function of the clustering A, transactions T_H and negative pairs L:

$$\begin{aligned}
&\mathbb{P}(A, T_H, L \mid p, q) = \\
&= \prod_{t \in T_H} p^{\mathbb{I}\left(Addr_H(t) \subset Cl(A)\right)} \times (1-p)^{\mathbb{I}\left(Addr_H(t) \not\subset Cl(A)\right)} \\
&\times \prod_{\{a,a'\} \in L} (1-q)^{\mathbb{I}(\{a,a'\} \not\subset Cl(A))} \times q^{\mathbb{I}(\{a,a'\} \subset Cl(A))},
\end{aligned}$$

where for some set of Bitcoin addresses S the notation $S \subset Cl(A)$ means, that there exists a cluster A_l such that $S \subseteq A_l$.

Probabilistic model (3)

Finally, the log-likelihood reads as

$$\ln \mathbb{P}(A, T_H, L \mid p, q) =$$
$$= \sum_{t \in T_H} \mathbb{I}\big(Addr_H(t) \subset Cl(A)\big) \ln(1-p) + \sum_{t \in T_H} \mathbb{I}\big(Addr_H(t) \not\subset Cl(A)\big) \ln(p)$$
$$+ \sum_{\{a,a'\} \in L} \mathbb{I}\big(\{a, a'\} \not\subset Cl(A)\big) \ln(1-q) + \sum_{\{a,a'\} \in L} \mathbb{I}\big(\{a, a'\} \subset Cl(A)\big) \ln(q).$$

Let Δ_{A_m} be the number of negative pairs in cluster A_m. Then, if we merge all the clusters corresponding to $Addr_H(t_j)$, the change of the log-likelihood is equal to

$$\Delta_{\mathbb{P}}(t_j, A, L \mid p, q) = \ln\left(\frac{p}{1-p}\right) + \left(\Delta_{\hat{A}_j} - \sum_{i=1}^{m_j} \Delta_{A_{k_i}}\right) \ln\left(\frac{q}{1-q}\right).$$

Experimental Setup

- Bitcoin blockchain data from 3^d January of 2009 to 9^{th} March of 2017: $211,789,876$ transactions which cover $244,030,115$ unique addresses.
- CS heuristic condition is satisfied for $8,161,086$ transactions with $28,416,034$ unique addresses.
- OTC heuristic condition holds for $35,844,487$ OTC transactions with $69,520,194$ unique addresses.
- Both conditions give a total of $44,005,573$ covered transactions with 95.250.167 unique addresses (the overlap is $2,686,061$ addresses).

Category	Number of tags	Number of common tags (size)	Examples of common tags
services	33	5 ($>$ 100K)	*Bitpay.com, Xapo.com*
gambling	34	6 ($>$ 50K)	*999Dice.com, primedice.com*
mixer	3	1 ($>$ 100K)	*BitcoinFog*
dnm	14	5 ($>$ 100K)	*SilkRoad Marketplace*
exchange	64	12 ($>$ 100K)	*BTC-e.com, Bittrex.com*
pool	15	2 ($>$ 50K)	*BTCChina, Hashnest.com*

Table: Tags of the biggest cluster in case of clustering without constraints ($26,694,671$ addresses).

Algorithms

- greedy (**historical**) approach: go retrospectively through all the transactions in the Bitcoin network, which satisfy one of the heuristics
- *greedy additive clustering* (**add**): first step, to perform clustering with $q \to 0$, where the clusters are not allowed to contain any negative pairs. In the second step, we once again go through Bitcoin transactions historically and optimize likelihood.

Maximum Cluster Size

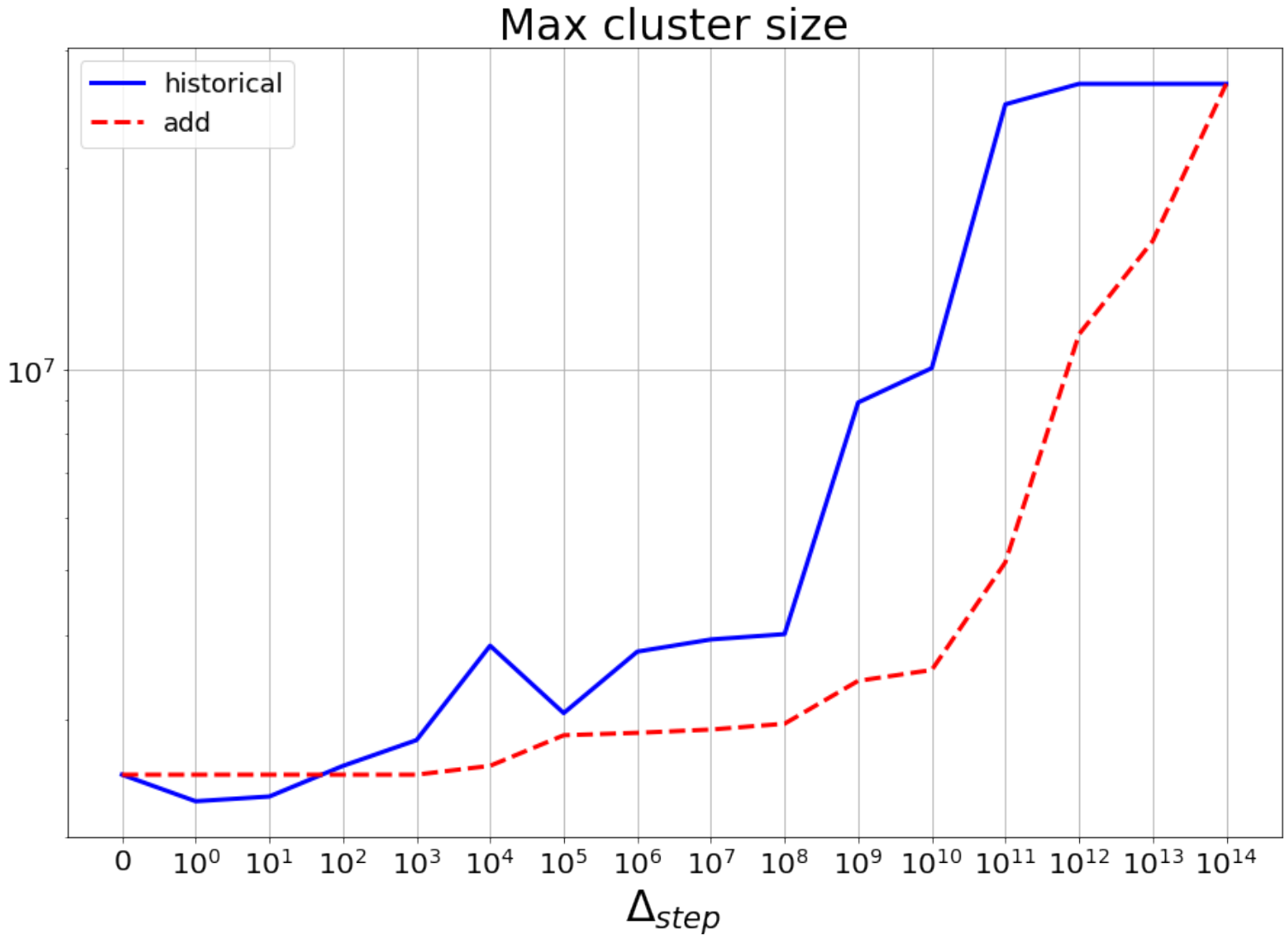


Figure: The size of the biggest cluster, where Δ_{step} is allowed to increase the number of negative pairs per transaction.

Negative Links

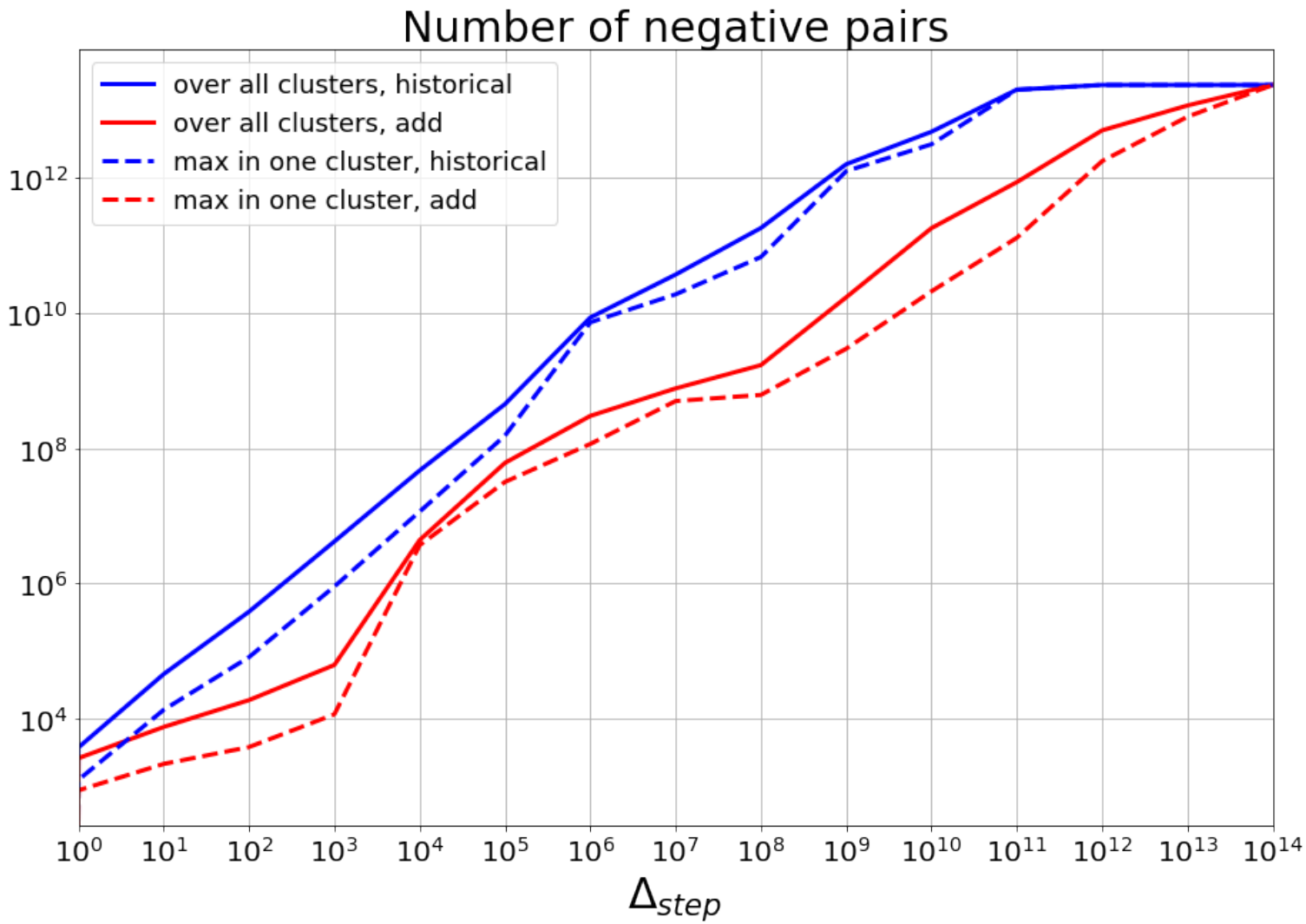


Figure: Number of negative pairs in whole clustering and the maximum in one cluster, where Δ_{step} is allowed to increase the number of negative pairs per transaction.

Calculations

- p = probability an honest node finds the next block
- q = probability the attacker finds the next block
- q_z = probability the attacker will ever catch up from z blocks behind
- z blocks.

$$q_z = \begin{cases} 1 & \text{if } p \leq q \\ (q/p)^z & \text{if } p > q \end{cases}$$

$$\lambda = z\frac{q}{p}$$

$$\sum_{k=0}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} \cdot \begin{cases} (q/p)^{(z-k)} & \text{if } k \leq z \\ 1 & \text{if } k > z \end{cases}$$

$$1 - \sum_{k=0}^{z} \frac{\lambda^k e^{-\lambda}}{k!} \left(1 - (q/p)^{(z-k)}\right)$$

```
q=0.3
z=0     P=1.0000000
z=5     P=0.1773523
z=10    P=0.0416605
z=15    P=0.0101008
z=20    P=0.0024804
z=25    P=0.0006132
z=30    P=0.0001522
z=35    P=0.0000379
z=40    P=0.0000095
```

Table of Contents

Bitcoin improvements: forks

- **soft fork:** backward capability
  - SegWit: August 2017.
- **hard fork:** change to a protocol that renders older versions invalid
  - November 2017: Segwit2x didn't happen in Bitcoin network -> Bitcoin Cash
  - June 2016: Ethereum Classic and The DAO.
- **source code fork**
  - October 2011: Litecoin
    - decreased block generation time (2.5 minutes)
    - increased maximum number of coins
    - different hashing algorithm (scrypt, instead of SHA-256).

Bitcoin hard forks

- Segwit2x – Bitcoin Cash
- Bitcoin Gold

Comparison BTC/BTG/BCH	BITCOIN BTC	BITCOIN GOLD BTG	BITCOIN CASH BCH
Supply	21 Million	**21 Million**	21 Million
PoW algorithm	SHA256	**Equihash**	SHA256
Mining Hardware	ASIC	**GPU**	ASIC
Block Interval	10 Minutes	**10 Minutes**	10 S - 2 H
Block size (actual)	1M (2-4M)	**1M (2-4M)**	8M (8M)
Difficulty adjustment	2 Weeks	**Every block**	2 Weeks + EDA
Segwit	✓	✓	⊖
Replay protection	●	✓	✓
Unique address format	●	✓	⊖

Bitcoin improvements protocols-1: Hierarchical Deterministic Wallets

- BIP32, BIP39 and BIP44

64536 luggage	65133 mixed	65346 nurse	65543 path	16156 prevent	16353 recipe	11666 rug
64531 lumber	65134 mixture	65341 nut	65544 patient	16151 price	16354 record	11661 rule
64532 lunar	65135 mobile	65342 oak	65545 patrol	16152 pride	16355 recycle	11662 run
64533 lunch	65146 model	65343 obey	65556 pattern	16153 primary	16466 reduce	11663 runway
64534 luxury	65141 modify	65344 object	65551 pause	16154 print	16461 reflect	11664 rural
64535 lyrics	65142 mom	65345 oblige	65552 pave	16155 priority	16462 reform	11665 sad
64546 machine	65143 moment	65356 obscure	65553 payment	16266 prison	16463 refuse	11616 saddle
64541 mad	65144 monitor	65351 observe	65554 peace	16261 private	16464 region	11611 sadness
64542 magic	65145 monkey	65352 obtain	65555 peanut	16262 prize	16465 regret	11612 safe
64543 magnet	65156 monster	65353 obvious	16666 pear	16263 problem	16416 regular	11613 sail
64544 maid	65151 month	65354 occur	16661 peasant	16264 process	16411 reject	11614 salad
64545 mail	65152 moon	65355 ocean	16662 pelican	16265 produce	16412 relax	11615 salmon
64556 main	65153 moral	65466 october	16663 pen	16216 profit	16413 release	11626 salon
64551 major	65154 more	65461 odor	16664 penalty	16211 program	16414 relief	11621 salt
64552 make	65155 morning	65462 off	16665 pencil	16212 project	16415 rely	11622 salute
64553 mammal	65266 mosquito	65463 offer	16616 people	16213 promote	16426 remain	11623 same
64554 man	65261 mother	65464 office	16611 pepper	16214 proof	16421 remember	11624 sample
64555 manage	65262 motion	65465 often	16612 perfect	16215 property	16422 remind	11625 sand
65666 mandate	65263 motor	65416 oil	16613 permit	16226 prosper	16423 remove	11636 satisfy
65661 mango	65264 mountain	65411 okay	16614 person	16221 protect	16424 render	11631 satoshi
65662 mansion	65265 mouse	65412 old	16615 pet	16222 proud	16425 renew	11632 sauce

Bitcoin improvements protocols-2: SegWit

Segregated Witness

- **Transaction malleability:** While transactions are signed, the signature does not currently cover all the data in a transaction that is hashed to create the transaction hash. Thus, while uncommon, it is possible for a node on the network to change a transaction you send in such a way that the hash is invalidated. Note that this just changes the hash; the output of the transaction remains the same and the bitcoins will go to their intended recipient. However this does mean that, for instance, it is not safe to accept a chain of unconfirmed transactions under any circumstance because the later transactions will depend on the hashes of the previous transactions, and those hashes can be changed until they are confirmed in a block (and potentially even after a confirmation if the block chain is reorganized). In addition, clients must always actively scan for transactions to them; assuming a txout exists because the client created it previously is unsafe.
- **SegWit** defines a new structure called a witness that is committed to blocks separately from the transaction merkle tree. This structure contains data required to check transaction validity but is not required to determine transaction effects. In particular, signatures and redeem scripts are moved into this new structure, which does not count towards the traditional 1 MB block size limit. Instead, a new weight parameter is defined, and blocks are allowed to have at most 4 million weight units (WU). A byte in the original 1 MB zone of the block weighs 4 WU, but a byte in a witness structure only weighs 1 WU, allowing blocks that are technically larger than 1 MB without a hardforking change.

Bitcoin improvements protocols-2: SegWit (2)

Segregated Witness profit

- protection from transaction malleability
- increase block capacity
- useful for anchoring and lightning
- optimizes CHECKSIG, CHECKMULTISIG ($O(n)$ instead of $O(n^2)$ to check n signatures per transaction).

Thank you for your attention!