A Preliminary Investigation into Storageless Blockchain

Yang Ji

April 15, 2019

Contents

Introduction

Background

- In cryptocurrency, peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger.

Background

- In cryptocurrency, peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger.
- Consensus protocol requires nodes to validate new transactions.

Background

- In cryptocurrency, peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger.
- Consensus protocol requires nodes to validate new transactions.
- e.g. To send 6 ETC from Alice to Bob requires that Alice has at least 6 ETC.

Background

- In cryptocurrency, peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger.
- Consensus protocol requires nodes to validate new transactions.
- e.g. To send 6 ETC from Alice to Bob requires that Alice has at least 6 ETC.
- Simply querying the history from adjacent nodes is infeasible and insecure due to the large size of blocks.

Background

- In cryptocurrency, peer-to-peer payment transactions are asynchronously broadcasted and recorded in an ordered ledger.
- Consensus protocol requires nodes to validate new transactions.
- e.g. To send 6 ETC from Alice to Bob requires that Alice has at least 6 ETC.
- Simply querying the history from adjacent nodes is infeasible and insecure due to the large size of blocks.
- For that reason, most cryptocurrency nodes need to locally maintain the validation state, which means downloading all past transactions/accounts.

Background

- In Bitcoin, Zcash and Komodo, validation state is a set of immutable coins called UTXO (unspent transaction output)

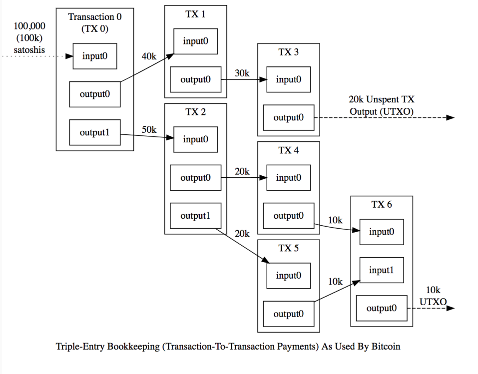


Fig. 1 : UTXO Model

Background

- In Ethereum, Nxt and Bitshares organize validation state as a set of mutable (and potential long-living) accounts.

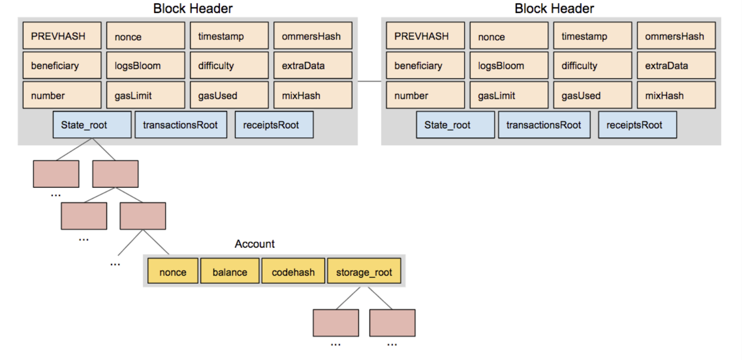


Fig. 2 : Account-based Model

Background

- Other cryptocurrencies

System	Execution State	Proof Size	Bootstrap Security
Bitcoin [21]	UTXOs	Headers + TXs	Probabilistic (heaviest chain wins)
Ethereum [9]	All accounts	Headers + All accounts	Probabilistic (heaviest chain wins)
Permissioned	$\frac{\text{Live accounts}}{\text{Shards}}$	Majority of trust set's signatures	Cryptographic if majority never compromised; none otherwise
OmniLedger [17] + Chainiac [22]	$\frac{\text{UTXOs}}{\text{Shards}}$	$\frac{\text{Headers+Certificates}}{\text{Sparseness}} + \frac{\text{UTXOs}}{\text{Shards}}$	Cryptographic with static attacker; none with adaptive attacker
Algorand [14]	UTXOs	Headers + Certificates + TXs	Cryptographic
Vault	$\frac{\text{Live accounts}}{\text{Shards}}$	$\frac{\text{Headers+Certificates}}{\text{Sparseness}} + \frac{\text{Live accounts}}{\text{Shards}}$	Cryptographic

Fig. 3 : Comparison of different cryptocurrency models

Motivation

- Maintaining ledger state is cumbersome from the perspectives of storage and bootstrapping
  - Large size (Bitcoin is around 150 GB and Ethereum has exceeded 400 GB)
  - Data storage size is linear with block number and could grow substantially in the coming years
  - Slow disk I/O operations (LevelDB or RocksDB)
  - DoS attack (adversarially-crafted transactions that need massive of disk accesses)
  - Increase the possibility of centralization in blockchains.
- Several people including Peter Todd and Mike Hearn also talked about storageless clients for Bitcoin in 2013.
- It thus appears that we need to compress the ledger history or allow miners to validate pending transactions without storing all past blocks.

Difficulty

- Currently the maximum size of the UTXO/Account set is unbounded as there is no consensus rule that limits growth.
- State set growth is driven by a number of factors, including the following fact.
  - For Bitcoin, there are massive of merge inputs, lost coins and dust outputs.
  - For Ethereum, smart contract inadvertently created many zero-balance accounts (account for around 38%).
- Based on these, a naïve way is to prune and clean 'useless' dust coins/accounts.
- However, there is little incentive to carry it out for many reasons:
  - Dust coins can't be economically spent and have other use cases.
  - We can't delete zero accounts because nodes need to keep track of the sequence number ("nonce") [Woo+].
- Redesigning index structures and storage mode has limited improvement.

Analysis of Existing Works

Possible Solutions

- Method 1: Storage Rent

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].
  - **Drawbacks:** Security issues; Incentive mechanism;

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].
  - **Drawbacks:** Security issues; Incentive mechanism;
- Method 3: Merging Transactions

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].
  - **Drawbacks:** Security issues; Incentive mechanism;
- Method 3: Merging Transactions
  - Record fewer transactions using payment channels e.g. lightning network [PD16]

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].
  - **Drawbacks:** Security issues; Incentive mechanism;
- Method 3: Merging Transactions
  - Record fewer transactions using payment channels e.g. lightning network [PD16]
  - **Drawbacks:** Only support pairwise payment relationship;

Possible Solutions

- Method 1: Storage Rent
  - Miners could know and update ledger state by outsourcing local database to third party or sharing common files with other miners in IPFS system.
  - **Drawbacks:** The operation cost is very high; less competitive; low I/O operations.
- Method 2: Sharding
  - Partition the whole blockchain/storage into many shardings like Zilliqa [Tea+17] and Omniledger [KJG+18].
  - **Drawbacks:** Security issues; Incentive mechanism;
- Method 3: Merging Transactions
  - Record fewer transactions using payment channels e.g. lightning network [PD16]
  - **Drawbacks:** Only support pairwise payment relationship;
- Method 4: Compacting Blocks [Poe16]

Stateless Blockchain

Stateless Blockchain

- This design concept of Stateless Blockchain is referred to Peter Todd's blog post [Tod].

Stateless Blockchain

- This design concept of Stateless Blockchain is referred to Peter Todd's blog post [Tod].
- Nodes might participate in transaction validation without storing the entire state of the ledger.

Stateless Blockchain

- This design concept of Stateless Blockchain is referred to Peter Todd's blog post [Tod].
- Nodes might participate in transaction validation without storing the entire state of the ledger.
- Transaction would include membership proofs for all its input.

Stateless Blockchain

- This design concept of Stateless Blockchain is referred to Peter Todd's blog post [Tod].
- Nodes might participate in transaction validation without storing the entire state of the ledger.
- Transaction would include membership proofs for all its input.
- A node would only need to store the current state and verify transactions by checking membership proofs against accumulator state.

Stateless Blockchain

- This design concept of Stateless Blockchain is referred to Peter Todd's blog post [Tod].
- Nodes might participate in transaction validation without storing the entire state of the ledger.
- Transaction would include membership proofs for all its input.
- A node would only need to store the current state and verify transactions by checking membership proofs against accumulator state.
- Many optimized schemes are put forward in the forum, such as Merkle Mountain tree [TO] and asynchronous accumulator [RY16] for dual accumulator.

Stateless Blockchain

D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, "Vault: Fast bootstrapping for the algorand cryptocurrency,", NDSS, 2019

Vault Techniques

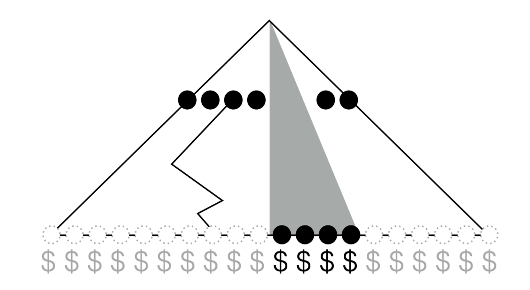
- **Vault:** Reducing the cost of storage and bootstrapping without weakening security guarantees.

Approach	Challenge	Vault's Solution
Reduce state transmitted: Garbage collection	Transaction replay attacks	Force transactions to expire
Reduce state transmitted: Shard state	Small shards lose security	Adaptive Merkle Tree sharding
Reduce size of state proof: Compress history	Attacker tampers with history	Succinct certificates

Vault: Forcing Transactions to Expire

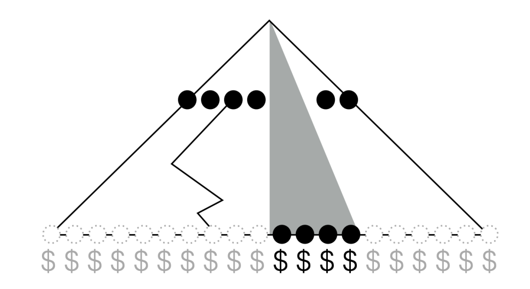
- All transactions contain the fields $t_{issuance}$ and t_{expiry}.
- We define $0 \leq t_{expiry} - t_{issuance} \leq t_{max}$
- The choice of T_{max} affects two considerations.
  - The block number of transactions to detect double spend.
  - Expiration mechanism requires the issuer to reissue expired transactions.
- Support off-chain payment channels e.g. Lightning Network.

Vault: Sharding Balance Storage



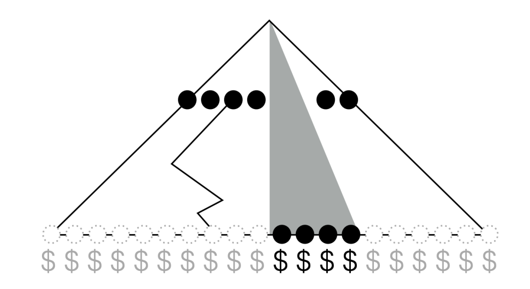
- **Shard Witness:** Transactions include Merkle witness for source and destination accounts.

Vault: Sharding Balance Storage



- **Shard Witness:** Transactions include Merkle witness for source and destination accounts.
- **Updating Witness:** The witness could be updated without requiring the issuer to resign the entire transaction.

Vault: Sharding Balance Storage



- **Shard Witness:** Transactions include Merkle witness for source and destination accounts.
- **Updating Witness:** The witness could be updated without requiring the issuer to resign the entire transaction.
- **Adaptive Sharding:** Truncating witness.

Vault: Compressing History

- Skipping Blocks

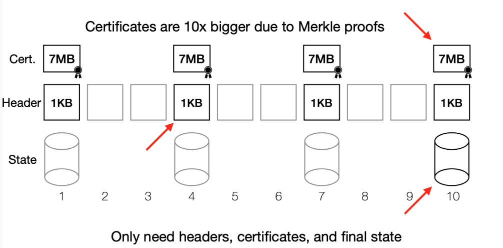


Fig. 4 : An illustration for skipping blocks

Vault: Compressing History

- Skipping Blocks

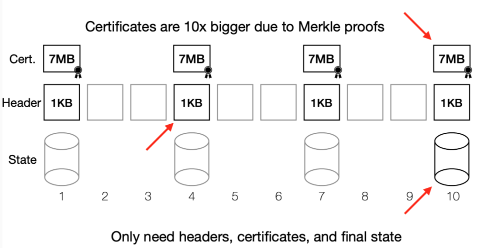


Fig. 4 : An illustration for skipping blocks

- Shrinking Certificates

Stateless Blockchain

A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018

EDRAX Architecture

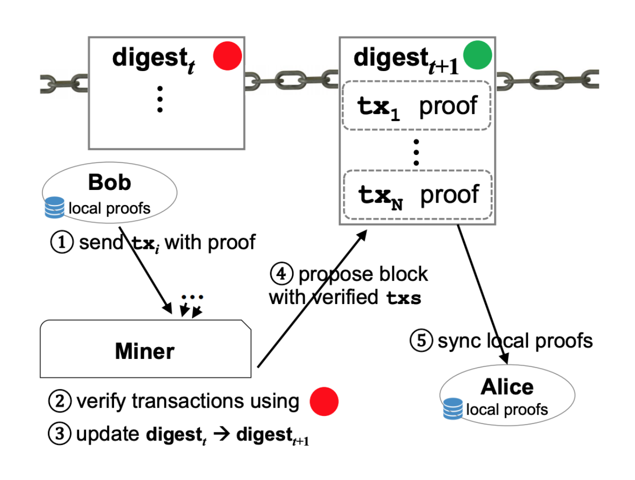


Fig. 5 : Edrax System Architecture

General Idea

- Represent UTXO as a sparse Merkle tree.

General Idea

- Represent UTXO as a sparse Merkle tree.
- For account-based model, we can use MHT to represent the mapping of $pk \leftrightarrow balance$

General Idea

- Represent UTXO as a sparse Merkle tree.
- For account-based model, we can use MHT to represent the mapping of $pk \leftrightarrow balance$
- Store the MHT root in the block as digest

General Idea

- Represent UTXO as a sparse Merkle tree.
- For account-based model, we can use MHT to represent the mapping of $pk \leftrightarrow balance$
- Store the MHT root in the block as digest
- However, during `SPEND`, we need the Merkle proofs of both sender and receiver

General Idea

- Represent UTXO as a sparse Merkle tree.
- For account-based model, we can use MHT to represent the mapping of $pk \leftrightarrow balance$
- Store the MHT root in the block as digest
- However, during `SPEND`, we need the Merkle proofs of both sender and receiver
- Instead, we can use vector commitment to represent account-balance mapping.

Edrax in account-based model

- Edrax requires an one-time setup phase.
- Each block stores a validation digest
  - A vector commitment digest whose mapping is $i \leftrightarrow h(\mathrm{PK})||balance$
  - A counter cnt_t which indicates the number of existing accounts
- Each client has a public key in the form $\mathrm{PK} = pk||i||upk_i$
- Clients store vector commitment proof π w.r.t. their index i
- The index i is assigned when client submits an `INIT` transaction $[\mathtt{INIT}, pk]$
  - Miner verifies transaction signature
  - Client is assigned to $i \leftarrow cnt_t$
  - Miner updates $cnt_{t+1} \leftarrow cnt_t + 1$ and a new digest $\mathrm{UpdateDigest}(dig, cnt, h(\mathrm{PK})||0, upk_{cnt})$
  - Clients synchronize proofs accordingly using UpdateProof

SPEND Transaction

- To spend δ to client with public key $\mathrm{PK}_b = pk_b || j || upk_j$
- Client sends transaction $[\mathrm{PK}_a, \mathrm{PK}_b, v, \pi_i, v']$ with signature *sig*
- Miner verifies:
  - *sig* is valid
  - $v \leq v'$
  - $\mathrm{Verify}(dig_t, i, h(\mathrm{PK}_a)||v', \pi_i, vrk)$ passes
- Miner updates digest
  - $dig' \leftarrow \mathrm{UpdateDigest}(dig_t, i, -v, upk_i)$
  - $dig_{t+1} \leftarrow \mathrm{UpdateDigest}(dig', j, v, upk_j)$
- Clients synchronize proofs accordingly using UpdateProof

Discussion

- Either dynamic accumulator or vector commitment provides a short binding commitment to a set of elements together with short membership/non-membership proof.

Discussion

- Either dynamic accumulator or vector commitment provides a short binding commitment to a set of elements together with short membership/non-membership proof.
- In essence, stateless blockchain reduces the storage burden for performing transaction validation by sacrificing communication overhead.

Discussion

- Either dynamic accumulator or vector commitment provides a short binding commitment to a set of elements together with short membership/non-membership proof.
- In essence, stateless blockchain reduces the storage burden for performing transaction validation by sacrificing communication overhead.
- Possible optimizations:

Discussion

- Either dynamic accumulator or vector commitment provides a short binding commitment to a set of elements together with short membership/non-membership proof.
- In essence, stateless blockchain reduces the storage burden for performing transaction validation by sacrificing communication overhead.
- Possible optimizations:
  - Shorten membership proof size

Discussion

- Either dynamic accumulator or vector commitment provides a short binding commitment to a set of elements together with short membership/non-membership proof.
- In essence, stateless blockchain reduces the storage burden for performing transaction validation by sacrificing communication overhead.
- Possible optimizations:
  - Shorten membership proof size
  - aggregate membership proofs for a batch of transactions [BBF18]

Possible Works and Directions

- More efficient authenticated data structure instead of sparse merkle tree

Possible Works and Directions

- More efficient authenticated data structure instead of sparse merkle tree
- Local proof synchronization by miner

Possible Works and Directions

- More efficient authenticated data structure instead of sparse merkle tree
- Local proof synchronization by miner
- Introducing proof-serving nodes

Possible Works and Directions

- More efficient authenticated data structure instead of sparse merkle tree
- Local proof synchronization by miner
- Introducing proof-serving nodes
- Supporting smart contracts in the stateless setting

Possible Works and Directions

- More efficient authenticated data structure instead of sparse merkle tree
- Local proof synchronization by miner
- Introducing proof-serving nodes
- Supporting smart contracts in the stateless setting
- Adding privacy to account-based model

References

[BBF18] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to iops and stateless blockchains," Cryptology ePrint Archive, Report 2018/1188, Tech. Rep, Tech. Rep., 2018.

[CPZ18] A. Chepurnoy, C. Papamanthou, and Y. Zhang, "Edrax: A cryptocurrency with stateless transaction validation," Cryptology ePrint Archive, Report 2018/968, Tech. Rep., 2018.

[KJG+18] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 583–598.

[LSGZ19] D. Leung, A. Suhl, Y. Gilad, and N. Zeldovich, "Vault: Fast bootstrapping for the algorand cryptocurrency,", NDSS, 2019.

[PD16] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2016.

[Poe16] A. Poelstra, *Mimblewimble*, 2016.

[RY16] L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed pki," in *International Conference on Security and Cryptography for Networks*, Springer, 2016, pp. 292–309.

References

[Tea+17] Z. Team *et al.*, *The zilliqa technical whitepaper*, 2017.

[TO] P. Todd and OpenTimestamps, *Merkle mountain tree*, `https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md`, Accessed: 2019-04-15.

[Tod] P. Todd, *Making utxo set growth irrelevant with low- latency delayed txo commitments*, `https://petertodd.org/2016/delayed-txo-commitments`, Accessed: 2019-04-15.

[Woo+] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger,",

Thanks

Questions?