Shortcutting the Bitcoin verification process for your smartphone

Pierre-Louis Roman

University of Rennes
Me, me, me

- BSc in Computer Science in 2011 from UPMC, Paris
- MSc in Distributed Systems in 2013 from UPMC, Paris
- PhD from Oct 2014 to Dec 2018 in Rennes
  Supervisors: François Taïani and Davide Frey

Research interests

Decentralized systems and algorithms, blockchains, epidemic protocols, consistency models, privacy-enhancing techs

Dietcoin\(^1\): shortcutting the Bitcoin verification process for your smartphone.
Davide Frey, Marc X. Makkes, Pierre-Louis Roman, François Taïani, Spyros Voulgaris. Research report, Mar 2018

\(^1\)Not Dietbitcoin
Lovely Brittany

Credits: wikipedia.com, dailybreizh.fr, saint-malo-plage.com, ferienhaus-bretagne.com, officiel-des-vacances.com
Distributed ledgers: abstraction

Ledger object\(^2\)
- Append-only queue of records
- Two operations: append(), read() \(\Rightarrow\) Tamper-proof ledger
- Distributed object \(\Rightarrow\) Needs consensus
- Ledger \(\sim\) Replicated State Machine that keeps the history of events

\(^2\)More in Chapter 16.7 of Michel Raynal's 2018 book
Distributed ledgers: in practice

Made of three layers

- Membership protocol
  *Who can commit*

- Consensus protocol
  *How they agree*

- Record ruleset
  *What they can commit*

Sybil-proof membership protocols
- Trusted authority
- Proof-of-Stake membership
- **Proof-of-Work membership**

Byzantine-tolerant consensus protocols
- Quorum-based (PBFT, Zyzzyva)
- Random leader election

Record ruleset
- From plain data
- To executable scripts
- Can be verifiable (for audit)
Distributed ledgers: in practice

Made of three layers

- Membership protocol
  *Who can commit*
- Consensus protocol
  *How they agree*
- Record ruleset
  *What they can commit*

Sybil-proof membership protocols

- Trusted authority
- Proof-of-Stake membership
- **Proof-of-Work membership**

Byzantine-tolerant consensus protocols

- Quorum-based (PBFT, Zyzzyva)
- Random leader election

Record ruleset

- From plain data
- To executable scripts
- Can be verifiable (for audit)
Distributed ledgers: in practice

Made of three layers

- Membership protocol
  *Who can commit*
- Consensus protocol
  *How they agree*
- Record ruleset
  *What they can commit*

Sybil-proof membership protocols

- Trusted authority
- Proof-of-Stake membership
- **Proof-of-Work membership**

Byzantine-tolerant consensus protocols

- Quorum-based (PBFT, Zyzzyva)
- Random leader election

Record ruleset

- From plain data
- To executable scripts
- Can be verifiable (for audit)
Distributed ledgers: in practice

Made of three layers

- Membership protocol  
  *Who can commit*
- Consensus protocol  
  *How they agree*
- Record ruleset  
  *What they can commit*

Sybil-proof membership protocols

- Trusted authority
- Proof-of-Stake membership
- **Proof-of-Work membership**

Byzantine-tolerant consensus protocols

- Quorum-based (PBFT, Zyzzyva)
- Random leader election

Record ruleset

- From plain data
- To executable scripts
- Can be verifiable (for audit)
### Distributed ledgers: examples

<table>
<thead>
<tr>
<th>Membership</th>
<th>Consensus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authority</td>
<td>Quorums</td>
</tr>
<tr>
<td>Proof-of-Work</td>
<td>Random leader</td>
</tr>
<tr>
<td>Proof-of-Work</td>
<td>Quorums</td>
</tr>
<tr>
<td>Proof-of-Stake</td>
<td>Random leader</td>
</tr>
<tr>
<td>Proof-of-Stake</td>
<td>Quorums</td>
</tr>
</tbody>
</table>
The Bitcoin payment system

"Be your own bank"

Payment system
- Public
- Decentralized
- Trustless
- Resilient to attackers?
- Scalable?

Made possible thanks to its blockchain
The Bitcoin blockchain

Blockchain
- Instance of a distributed ledger
- Mostly for decentralized ledgers
- Backward linked list by storing $H(\text{Block}_{k-1})$ in Block$_k$
- Blocks are commits

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- Trustless: fully verifiable
- Records financial transactions

Operations and roles
- append(block) by miners by solving Proofs-of-Work
- read() by full nodes
The Bitcoin blockchain

Blockchain
- Instance of a distributed ledger
- Mostly for decentralized ledgers
- Backward linked list by storing $H(\text{Block}_{k-1})$ in Block$_k$
- Blocks are commits

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- **Trustless: fully verifiable**
- Records financial transactions

Operations and roles
- append(block) by miners by solving Proofs-of-Work
- read() by full nodes
The Bitcoin blockchain

Blockchain
- Instance of a distributed ledger
- Mostly for decentralized ledgers
- Backward linked list by storing $H(\text{Block}_{k-1})$ in $\text{Block}_k$
- Blocks are commits

Bitcoin blockchain
- Public: anyone can create a block
- Secured: Proof-of-Work on blocks
- **Trustless: fully verifiable**
- Records financial transactions

Operations and roles
- append(block) by miners by solving Proofs-of-Work
- read() by full nodes
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- \( \sum \text{inputs} \geq \sum \text{outputs} \)

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum \text{inputs} \geq \sum \text{outputs}$

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum \text{inputs} \geq \sum \text{outputs}$

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum \text{inputs} \geq \sum \text{outputs}$

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum$ inputs $\geq \sum$ outputs

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum \text{inputs} \geq \sum \text{outputs}$

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- $\sum \text{inputs} \geq \sum \text{outputs}$

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying transactions

A transaction is valid if:

- Its inputs refer to valid past outputs, with correct signatures
- Its inputs are not yet spent
- \( \sum \text{inputs} \geq \sum \text{outputs} \)

Nodes must store the UTXO set (set of Unspent Transaction Outputs)
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 1, 2, 3 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 1, 2, 3, 4, 5 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 3, 4, 5 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 3, 4, 5, 6, 7, 8 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 6, 7, 8 \}
Verifying blocks

Update the UTXO set on valid blocks

UTXO set = \{ 6, 7, 8 \}
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty $\neq$ Correctness

180 GiB
Full node
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Verification process

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness
Rapidly growing UTXO set
Scalability and security problem

- Light nodes don’t verify the correctness of blocks
- Building the UTXO set requires a lot of bandwidth and time
- (and it’s getting worse)

WHAT IF

a light node receives a block with fake transactions?
Intuition

Make the UTXO set queriable by light nodes

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

- Diet nodes consume more bandwidth than light nodes
Intuition

Make the UTXO set queriable by light nodes

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

- Diet nodes consume more bandwidth than light nodes
Intuition

Make the UTXO set queriable by light nodes

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

Diet nodes consume more bandwidth than light nodes

What if a diet node receives a fake UTXO set?
Hash of the UTXO set (strawman)

- It works!

- A node has to download the entire UTXO set, even for small queries
Hash of the UTXO set (strawman)

- It works!
- A node has to download the entire UTXO set, even for small queries
Merkle tree: efficient proof of inclusion

Download
Reconstruct
Merkle tree: efficient proof of inclusion

H_{ABCD}

H_{ABCDEF}

Download
Reconstruct

H_{EFGH}

H_{AB}

H_{CD}

H_{EF}

H_{GH}

H_{A}

H_{B}

H_{C}

H_{D}

H_{E}

H_{F}

H_{G}

H_{H}

A

B

C

D

E

F

G

H

Download
Reconstruct
Merkle tree: efficient proof of inclusion
Merkle tree: efficient proof of inclusion

H_{ABCDEFGH}

H_{ABCD}

H_{EF}

H_{GH}

H_{AB}

H_{CD}

H_{EF}

H_{GH}

Download
Reconstruct
Merkle tree: efficient proof of inclusion
Merkle tree: efficient proof of inclusion
Merkle tree of UTXO set (2012 post from Andrew Miller)

- Precise queries, no superfluous UTXOs sent
- Full node storage overhead: 3 GiB
  \[ |\text{UTXOs}| = 50 \text{M} \Rightarrow |\text{hashes}| = 100 \text{M} \Rightarrow \text{tree size} = 3 \text{GiB} \]
- Lots of leaf insertions and deletions
Merkle tree of UTXO set (2012 post from Andrew Miller)

- Precise queries, no superfluous UTXOs sent
- Full node storage overhead: 3 GiB
  \[ |\text{UTXOs}| = 50 \text{ M} \Rightarrow |\text{hashes}| = 100 \text{ M} \Rightarrow \text{tree size} = 3 \text{ GiB} \]
- Lots of leaf insertions and deletions
Merkle tree of sharded UTXO set (Dietcoin)

- Full node storage overhead: 128 MiB \((k = 2^2 \Rightarrow |\text{hashes}| = 2^{22})\)
- Diet node bandwidth consumption: 12.8 MiB of query per block
  \[10000 \text{ shards} \times (0.64 \text{ KiB} + 22 \times 32 \text{ B}) = 12.8 \text{ MiB}\]
- Parameterized trade-off \(k\): bandwidth consumption vs storage overhead
- Stable tree: no insertion, no deletion \(\Rightarrow\) Enable subchain verification
Full node storage overhead: 128 MiB \((k = 22 \Rightarrow |\text{hashes}| = 2^{22})\)

Diet node bandwidth consumption: 12.8 MiB of query per block

\[10000 \text{ shards} \times (0.64 \text{ KiB} + 22 \times 32 \text{ B}) = 12.8 \text{ MiB}\]

Parameterized trade-off \(k\): bandwidth consumption vs storage overhead

Stable tree: no insertion, no deletion \(\Rightarrow\) Enable subchain verification
Codependency of blocks and the UTXO set
Subchain verification

Trust a block, verify all the next ones
Shift the trust from the genesis block to any block

A diet node verifying the UTXO Merkle root in block $B_k$
- Queries the UTXO Merkle root in $B_{k-1}$ ($B_{k-1}$ is trusted)
- Queries UTXO shards for transaction inputs and outputs of $B_k$
- Verifies transactions in $B_k$, updating its local copies of UTXO shards
- Recomputes its UTXO Merkle root, check it against the one in $B_k$
- Repeat for the next block

Effectively shortcuts the verification process
Subchain verification

Trust a block, verify all the next ones
Shift the trust from the genesis block to any block

A diet node verifying the UTXO Merkle root in block $B_k$

- Queries the UTXO Merkle root in $B_{k-1}$ ($B_{k-1}$ is trusted)
- Queries UTXO shards for transaction inputs and outputs of $B_k$
- Verifies transactions in $B_k$, updating its local copies of UTXO shards
- Recomputes its UTXO Merkle root, check it against the one in $B_k$
- Repeat for the next block

Effectively shortcuts the verification process
Subchain verification

Trust a block, verify all the next ones
Shift the trust from the genesis block to any block

A diet node verifying the UTXO Merkle root in block $B_k$

- Queries the UTXO Merkle root in $B_{k-1}$ ($B_{k-1}$ is trusted)
- Queries UTXO shards for transaction inputs and outputs of $B_k$
- Verifies transactions in $B_k$, updating its local copies of UTXO shards
- Recomputes its UTXO Merkle root, check it against the one in $B_k$
- Repeat for the next block

Effectively shortcuts the verification process
Adaptive number of UTXO shards

- Global parameter: cap on the average shard size $M$ (e.g., 1 KiB)
- Local rule: split all shards in two when shard sizes are on average above $M$
- Global effect: one layer added to the UTXO Merkle tree

Updating all the tree prevents subchain verification (may happen rarely)
Evaluation goal

Metrics

- Bandwidth consumption
- Storage consumption
- Light node battery consumption

Evaluation setup

- Use a historical dataset
  - Replay the Bitcoin chain
- Use popular code
  - Extend the bitcoinj library with Dietcoin features
- Light node in a realistic environment
  - Hide the complexity in the server
Evaluation goal

Metrics
- Bandwidth consumption
- Storage consumption
- Light node battery consumption

Evaluation setup
- Use a historical dataset
  
  *Replay the Bitcoin chain*

- Use popular code
  
  *Extend the bitcoinj library with Dietcoin features*

- Light node in a realistic environment
  
  *Hide the complexity in the server*
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test protocol improvements
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test protocol improvements
Evaluation scheme

- Bitcoin server providing original blockchain
- Dietcoin proxy providing modified Bitcoin data to the phone
- Phone using an oblivious diet node

Reusable Bitcoin replayer to test protocol improvements
Summary

- Diet nodes can verify the correctness of blocks and subchains
- Diet nodes shortcut the verification process

Future work

- Evaluation
- Decoupled storage → DHT
- Shard compression
- Combine with Non-interactive Proofs of Proof-of-Work (NiPoPoWs)?
- Combine with Ethereum Casper?
Perspectives on the field

Distributed ledgers
- Scalability of layer 1
- Privacy in public ledgers
- Correctness of scripts in records

Decentralized coordination
- Is deterministic consensus with an unbounded and dynamic number of nodes possible? *Under which synchrony assumptions?*
- Apart from Proof-of-Work, are there any other Sybil-proof open-membership protocols?
- Can we design a Sybil-proof open-membership protocol that is fair and deterministic?
Perspectives on the field

Distributed ledgers

- Scalability of layer 1
- Privacy in public ledgers
- Correctness of scripts in records

Decentralized coordination

- Is deterministic consensus with an unbounded and dynamic number of nodes possible? **Under which synchrony assumptions?**
- Apart from Proof-of-Work, are there any other Sybil-proof open-membership protocols?
- Can we design a Sybil-proof open-membership protocol that is fair and deterministic?