Fraud Proofs

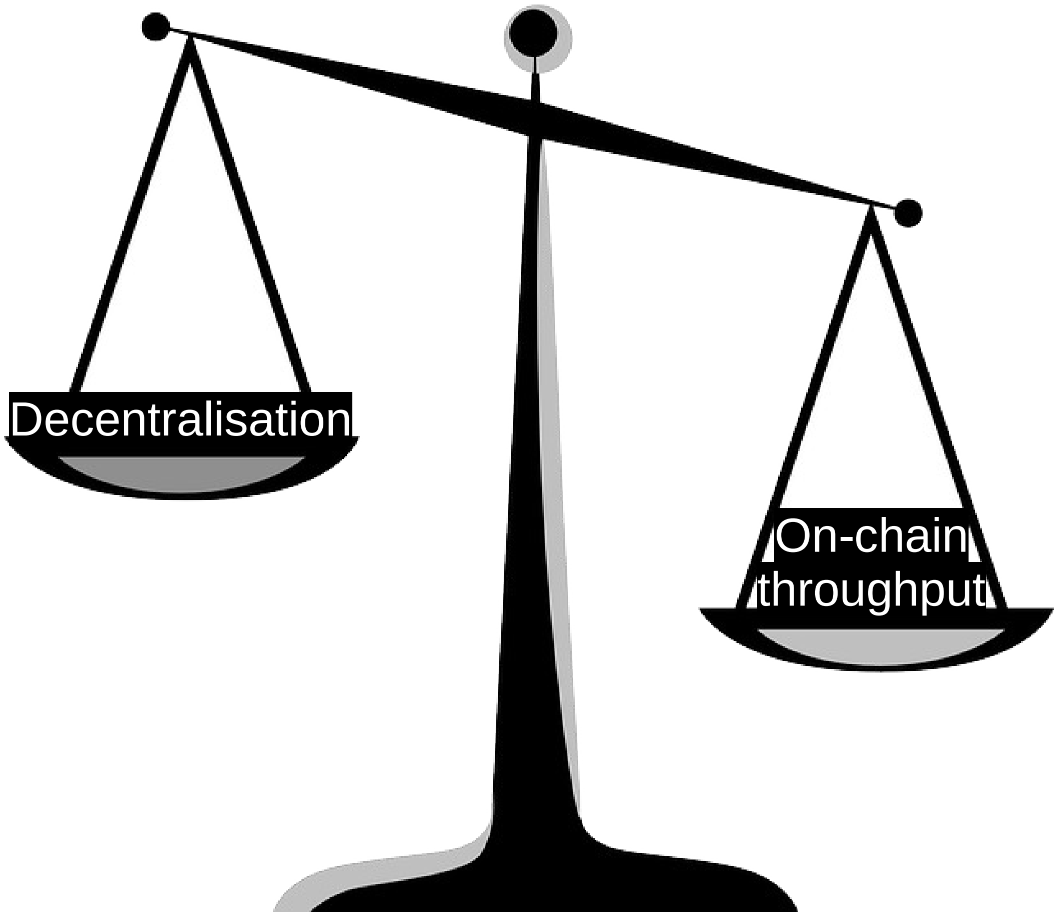
Maximising SPV Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam

6 October 2018

Motivation

- Currently there is a huge trade off between decentralisation and on-chain scalability.
- Because SPV nodes (non fully validating nodes) will accept invalid blocks.
  - They assume that the majority of the consensus is honest.



Can we reduce this tradeoff?

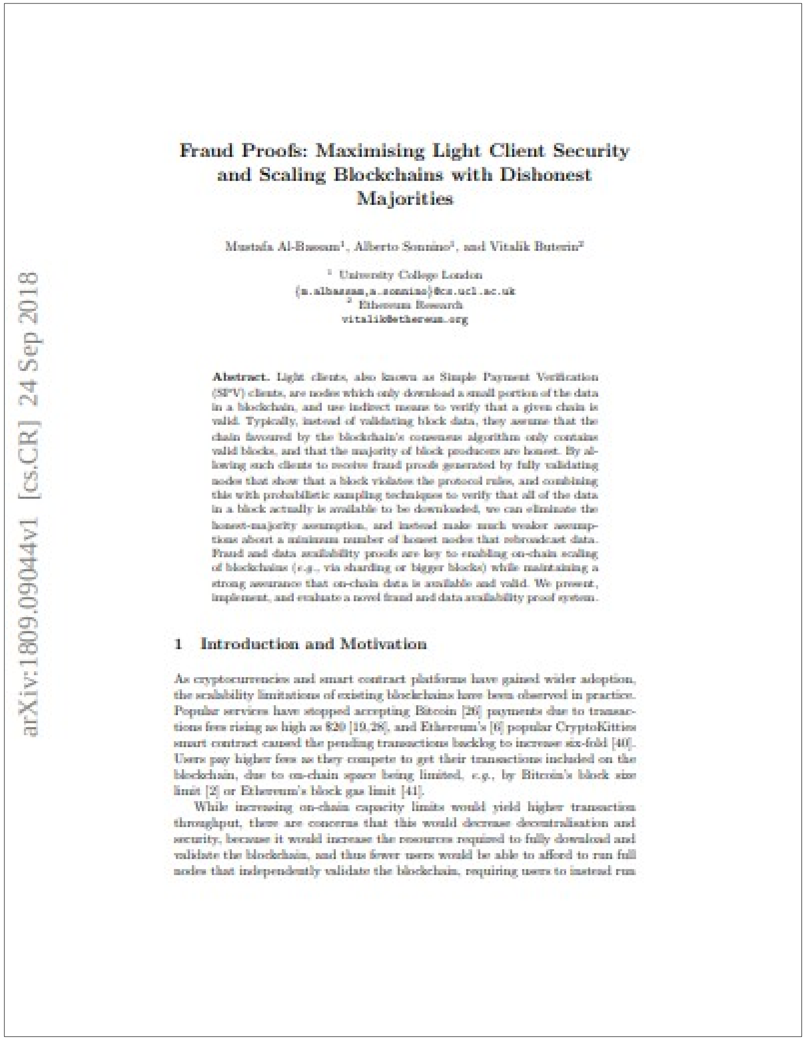
- The big question: how can we make non fully validating (SPV) nodes reject invalid blocks, so that they don't have to trust miners?
- Let's use fraud proofs and data availability proofs!

Read the full paper (33 pages)

- Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

  Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin

- https://arxiv.org/abs/1809.09044



arXiv:1809.09044v1 [cs.CR] 24 Sep 2018

Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam[1], Alberto Sonnino[1], and Vitalik Buterin[2]

[1] University College London
{m.albassam,a.sonnino}@cs.ucl.ac.uk
[2] Ethereum Research
vitalik@ethereum.org

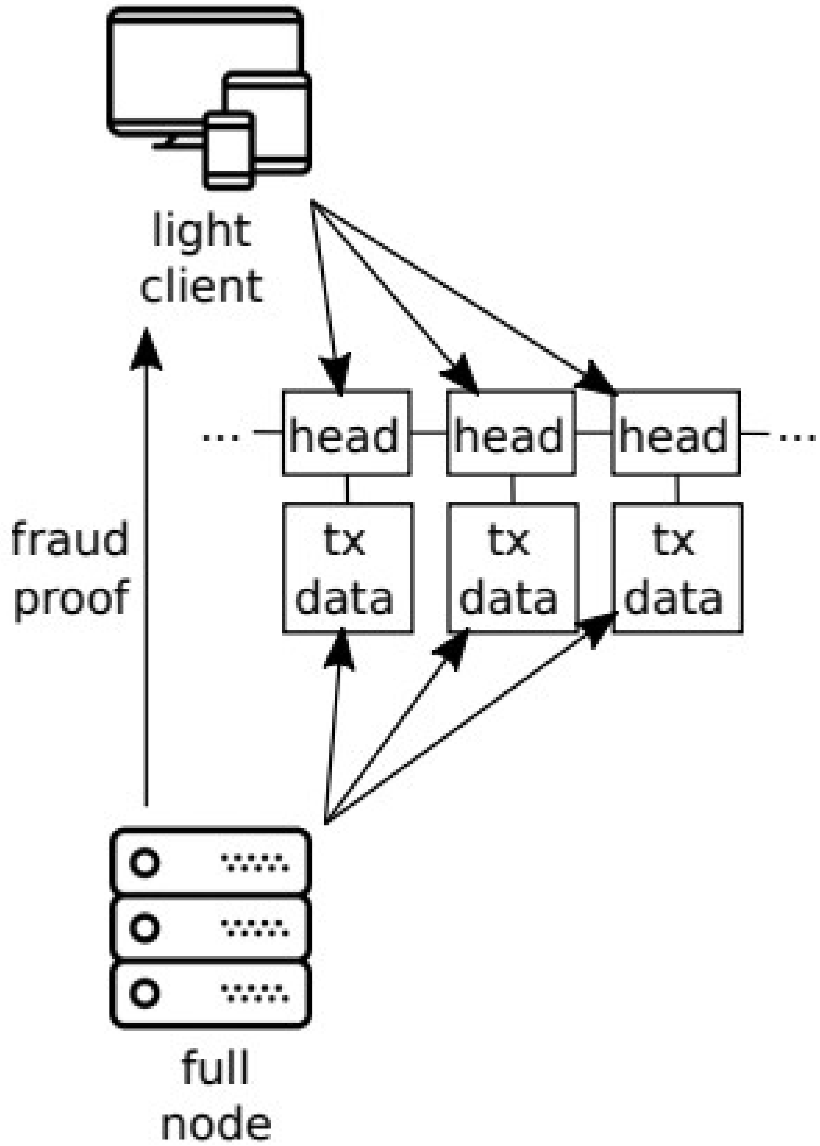
Abstract. Light clients, also known as Simple Payment Verification (SPV) clients, are nodes which only download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid. Typically, instead of validating block data, they assume that the chain favoured by the blockchain's consensus algorithm only contains valid blocks, and that the majority of block producers are honest. By allowing such clients to receive fraud proofs generated by fully validating nodes that show that a block violates the protocol rules, and combining this with probabilistic sampling techniques to verify that all of the data in a block actually is available to be downloaded, we can eliminate the honest-majority assumption, and instead make much weaker assumptions about a minimum number of honest nodes that rebroadcast data. Fraud and data availability proofs are key to enabling on-chain scaling of blockchains (*e.g.*, via sharding or bigger blocks) while maintaining a strong assurance that on-chain data is available and valid. We present, implement, and evaluate a novel fraud and data availability proof system.

1 Introduction and Motivation

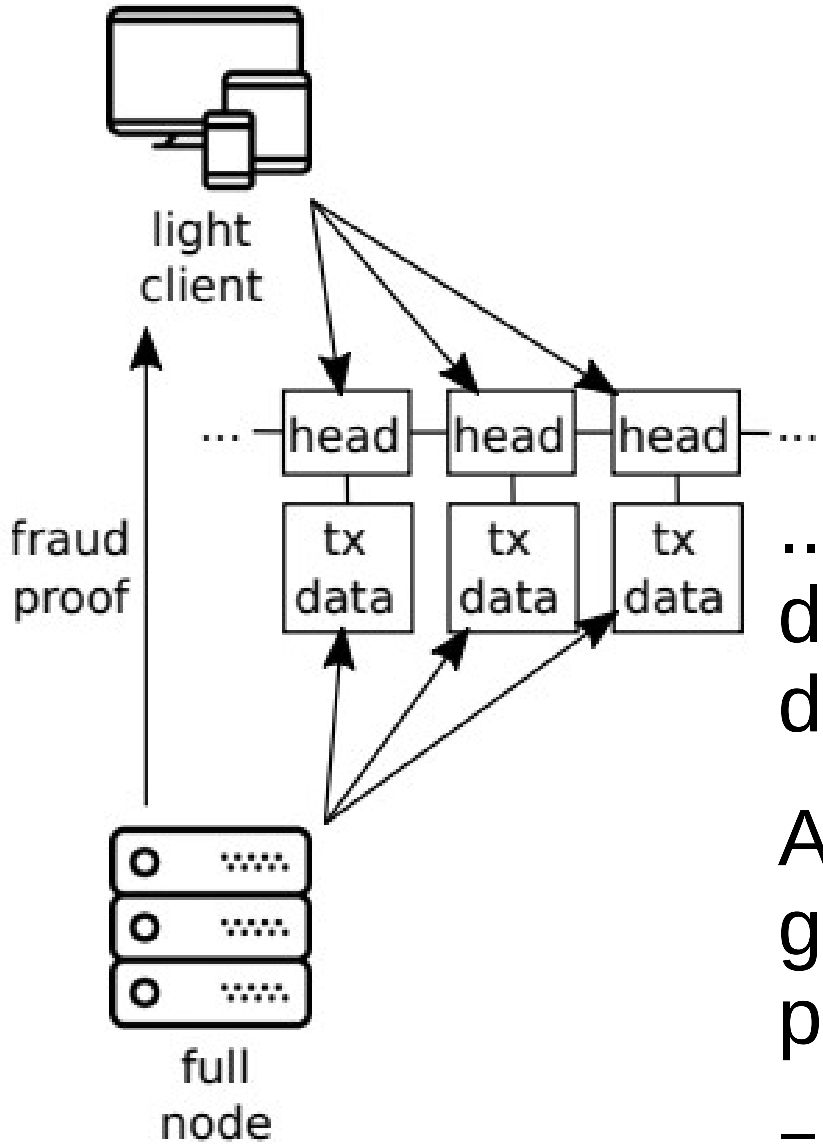
As cryptocurrencies and smart contract platforms have gained wider adoption, the scalability limitations of existing blockchains have been observed in practice. Popular services have stopped accepting Bitcoin [26] payments due to transactions fees rising as high as $20 [19,28], and Ethereum's [6] popular CryptoKitties smart contract caused the pending transactions backlog to increase six-fold [40]. Users pay higher fees as they compete to get their transactions included on the blockchain, due to on-chain space being limited, *e.g.*, by Bitcoin's block size limit [2] or Ethereum's block gas limit [41].

While increasing on-chain capacity limits would yield higher transaction throughput, there are concerns that this would decrease decentralisation and security, because it would increase the resources required to fully download and validate the blockchain, and thus fewer users would be able to afford to run full nodes that independently validate the blockchain, requiring users to instead run

Fraud proofs: the basic idea



Fraud proofs: the basic idea



...but what if the miner doesn't release the tx data?

A: Then we can't generate a fraud proof. 😱

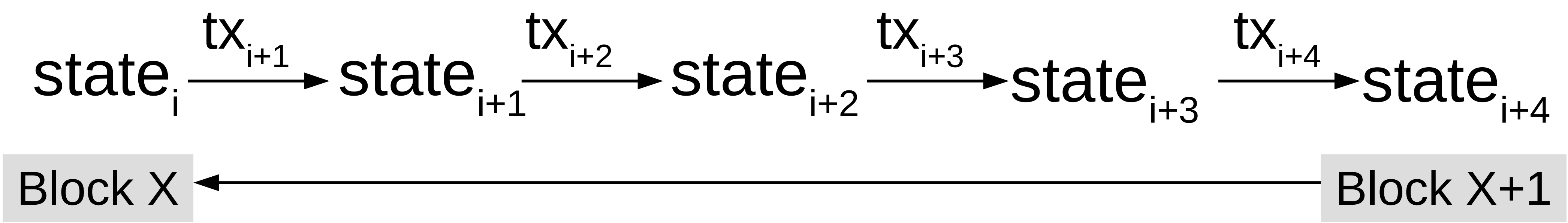
= We also need a way to guarantee data availability!

Earlier discussion on fraud proofs

- The original Bitcoin whitepaper briefly mentions “alerts”; which are messages that full nodes can send to SPV clients to alert them the block is invalid. (Vulnerable to DoS.)
- G. Maxwell and P. Todd have done some work on “compact fraud proofs”. Early proposals require a different fraud proof for each way to violate the rules. We improve on this.
- G. Maxwell has discussed on IRC using erasure coding for data availability with a scheme using a “designated source” with PoW rate-limiting (and no way to deal with incorrectly generated codes?)

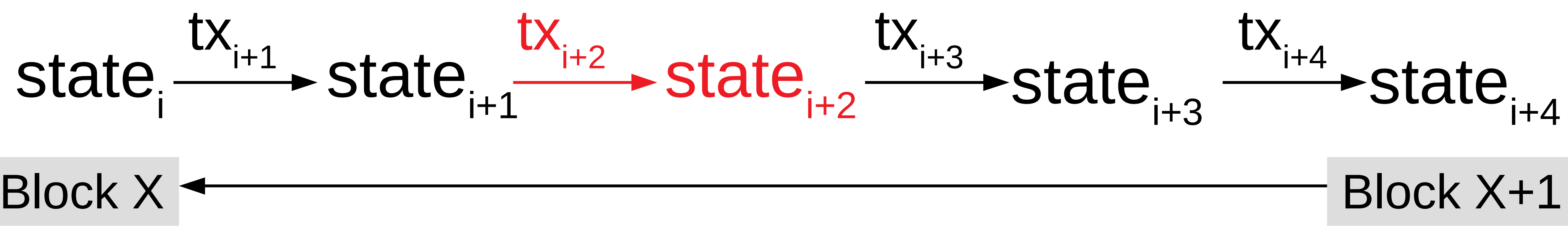
Generalising the blockchain as a state transition system

- Each transaction reads and modifies the state of the blockchain.
  - transition(state, tx) = state' or *error*



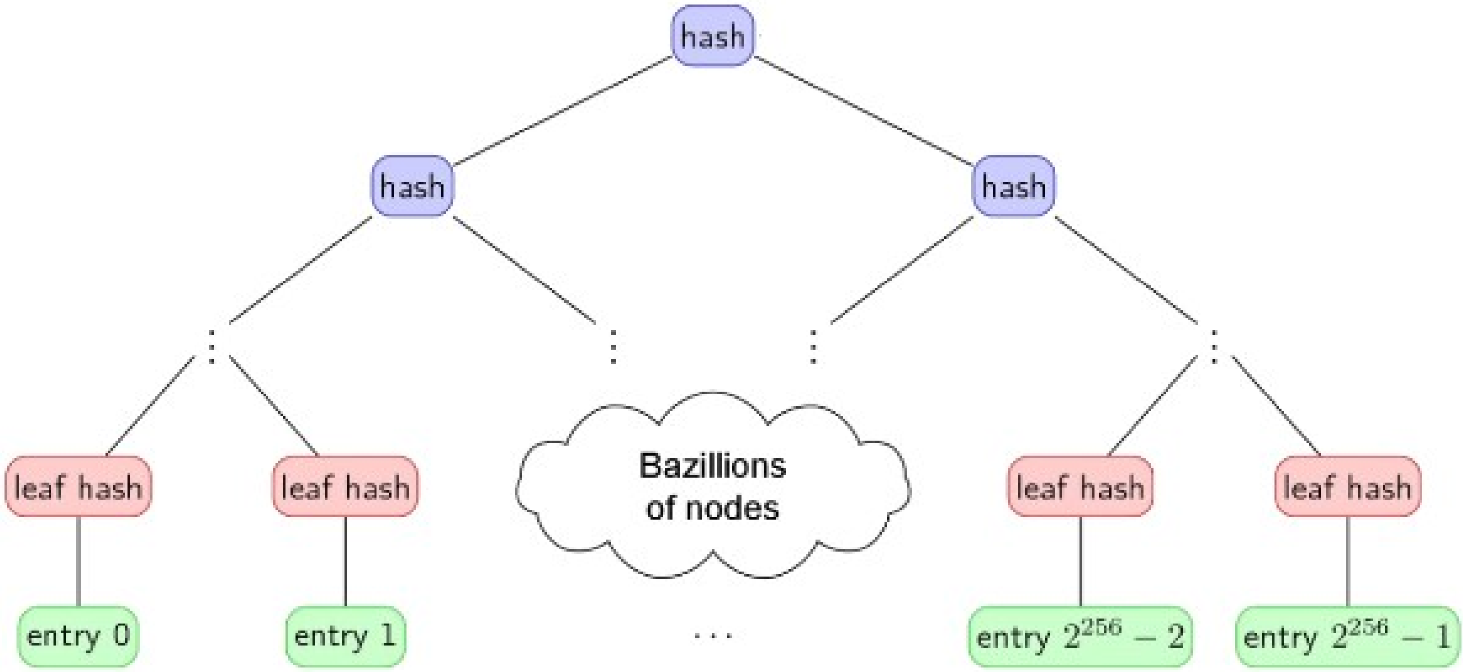
Generalising the blockchain as a state transition system

- Each transaction reads and modifies the state of the blockchain.
  - transition(state, tx) = state' or *error*
- But what if one of the transitions is errorneous?
  - We need a way to prove this.



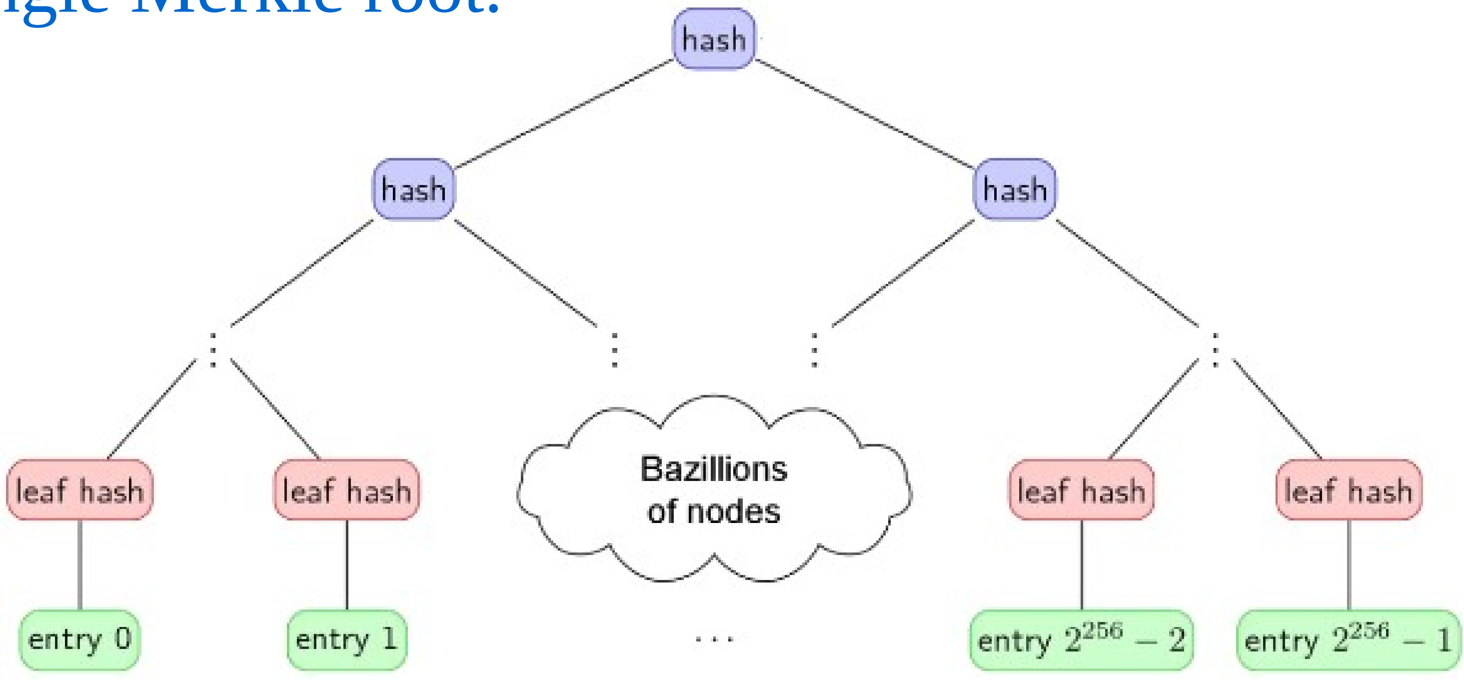
Representing the entire state as a Merkle root

- We can store the state as a key-value store.
  - Key: UTXO ID. Value: 1 if unspent, 0 otherwise.
- We can use a Sparse Merkle tree to do this.
  - A tree with 2^{256} leaves (every possible SHA-256 hash).



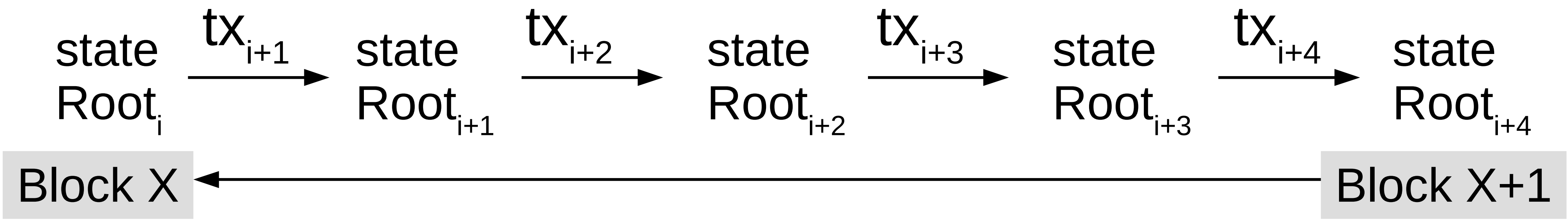
Representing the entire state as a Merkle root

- Merkle proofs are still O(log(n)) as most branches will only contain leaves with default values (0).
- To access key K in the tree, access the *hash(K)*th item.
- We can thus represent the entire state of the blockchain as a single Merkle root.



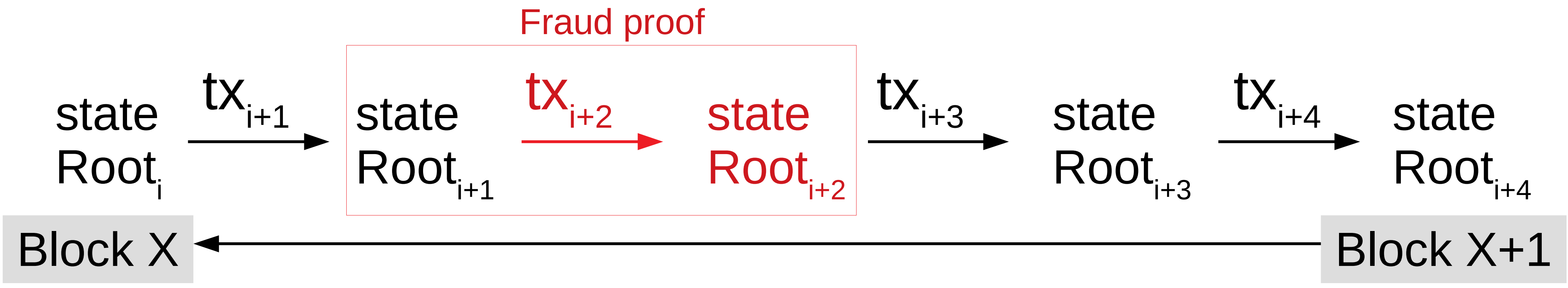
Generalising the blockchain as a state transition system

- Each transaction changes the state root of the blockchain.
  - transitionRoot(stateRoot, tx, witnesses) = stateRoot' or *error*
  - The witnesses of a transaction are Merkle proofs of all the parts of the state the transactions accesses.
- **Execution trace:** we need to include the post-state root of transactions in the block. (e.g. every few transactions)



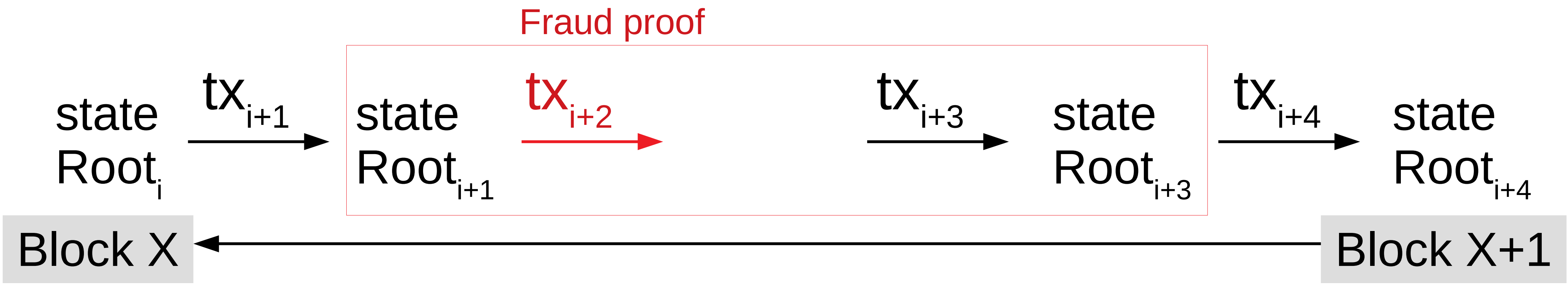
Generalising the blockchain as a state transition system

- The fraud proofs consisters of the transaction and its pre-state root, post-state root, witnesses (+ Merkle proofs).
- If the Merkle proofs are valid, and $rootTransition(stateRoot_{i+1}, tx_{i+1}, witnesses_{i+1}) \mathrel{!=} stateRoot_{i+2}$ then the fraud proof is valid.

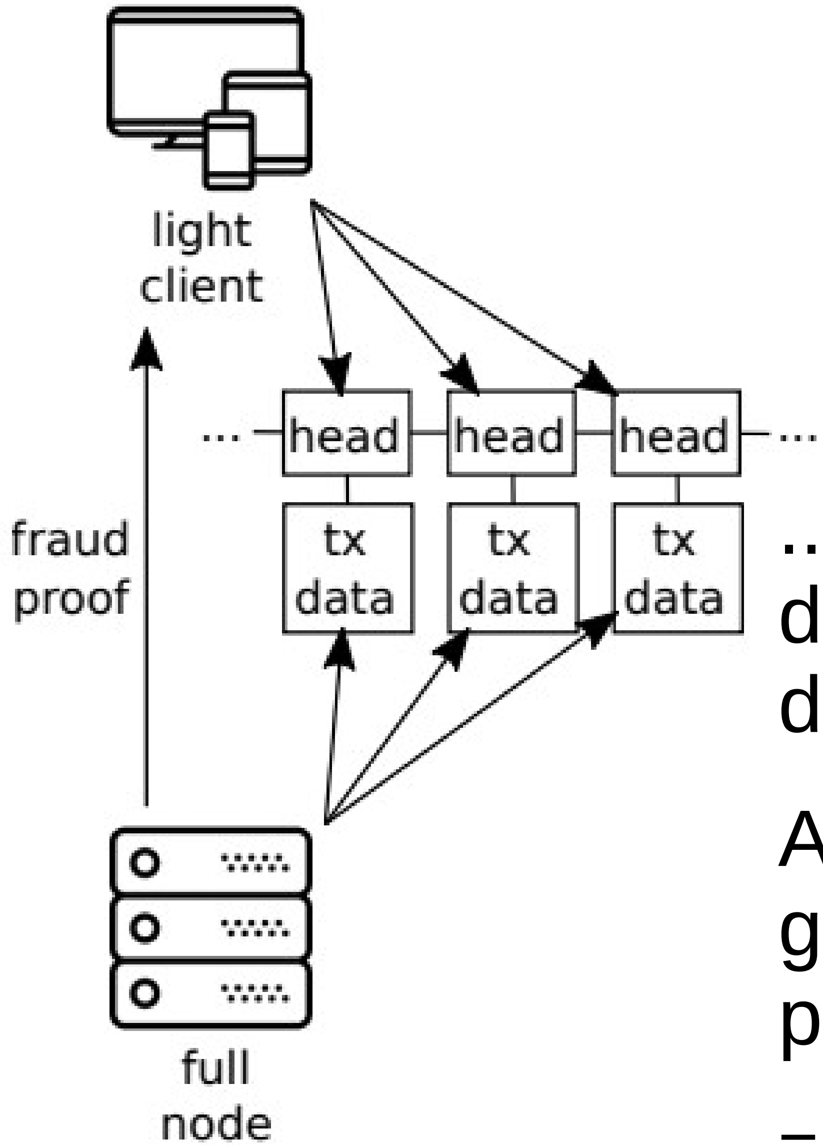


Generalising the blockchain as a state transition system

- You don’t have to include the state root after every transaction – this saves block space but the fraud proof gets bigger.



The data availability problem



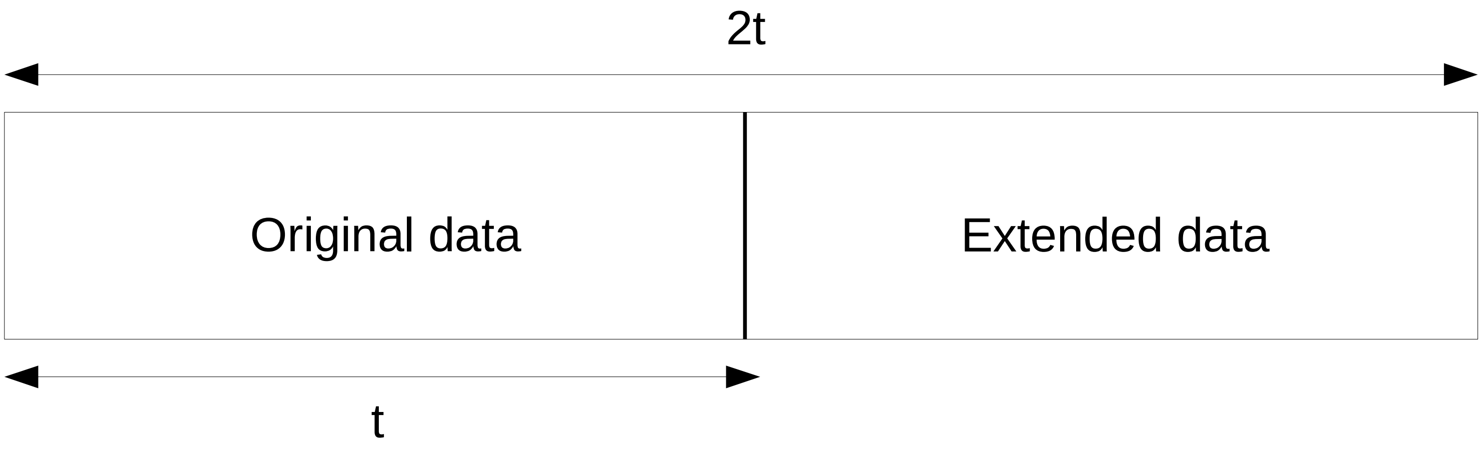
...but what if the miner doesn't release the tx data?

A: Then we can't generate a fraud proof. 😱

= We also need a way to guarantee data availability!

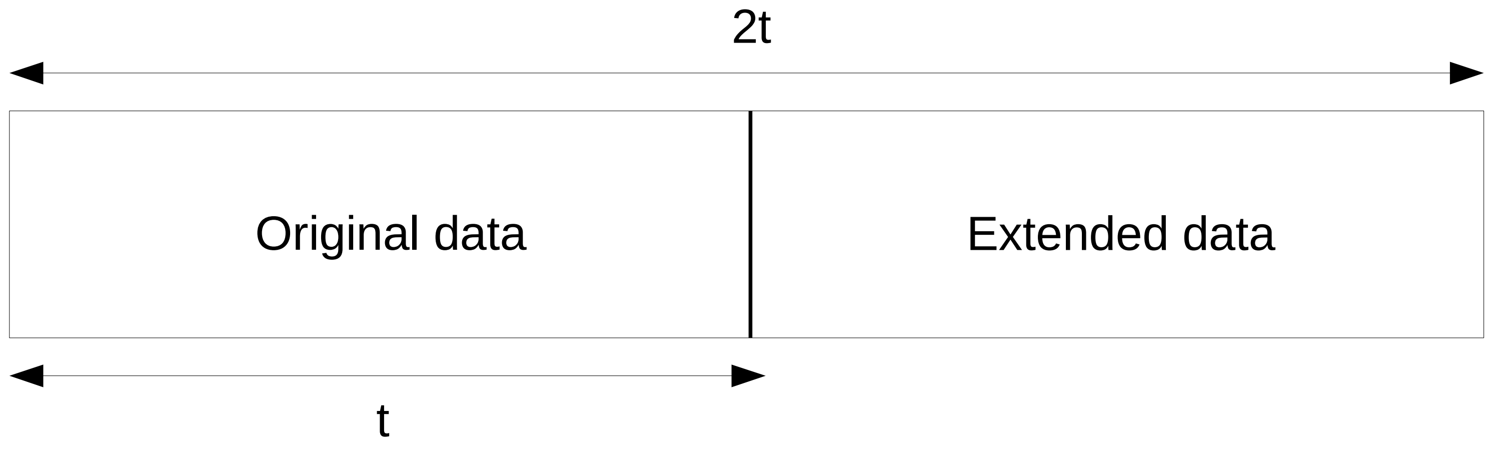
Erasure coding

- Using erasure coding, you can extend data *t* pieces long to *2t* pieces, such that you can recover the whole data from any *t* pieces.



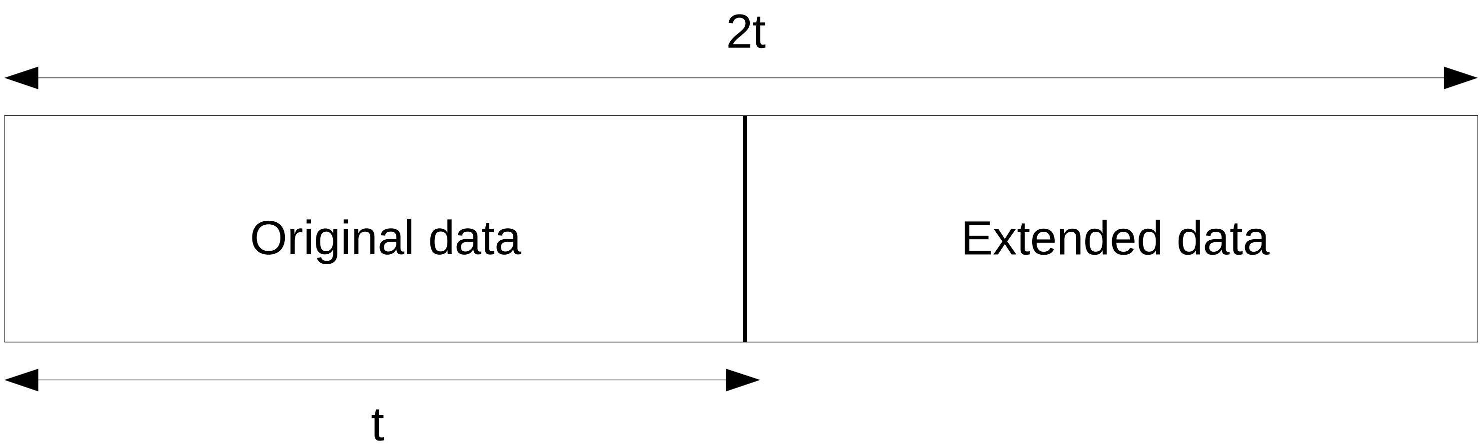
Naive data availability scheme

- We can require miners to commit to the Merkle root of the erasure coded verison of the block data. In order for a miner to hide any piece of the block, they must hide 50%.
- Thus clients can randomly sample parts of the block (with replacement), and if 50% is hidden, then there is a 2^{-s} chance of landing on an unavailable piece, then the block is rejected.
- Clients gossip pieces to full nodes for recovery.



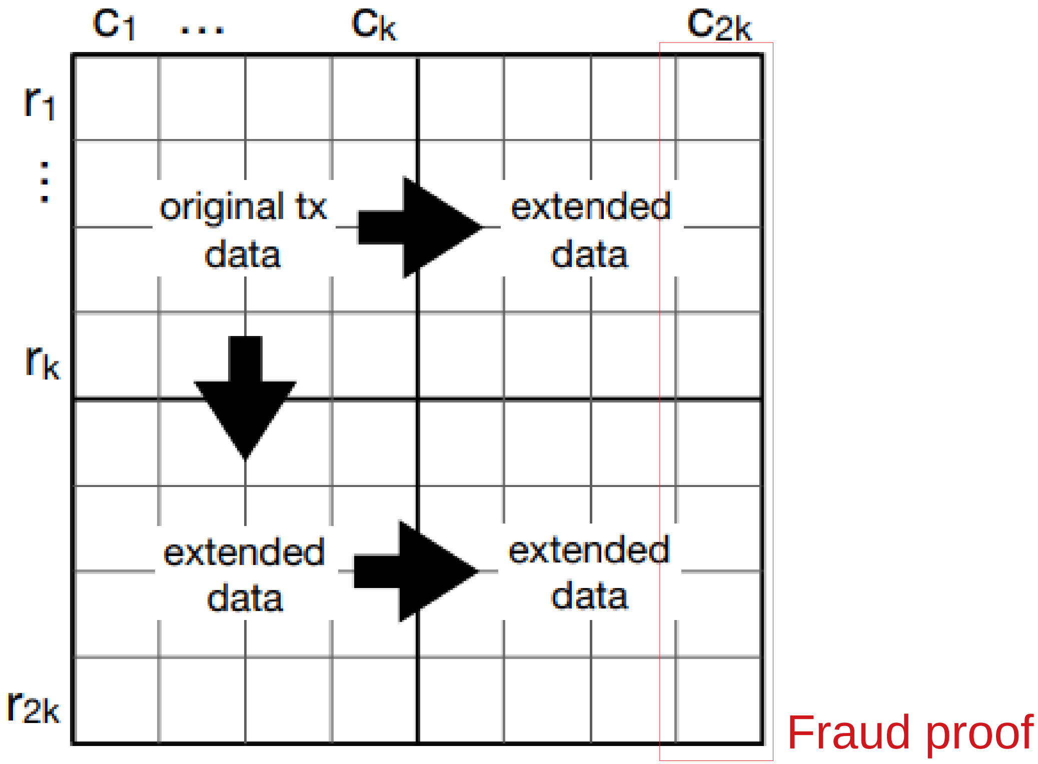
Naive data availability scheme

- Problem: what if the miner incorrectly generates the erasure code?
  - In order to prove this, the fraud proof consists of the entire block, as clients will need to download and regenerate the erasure code to check if it’s correct.
  - That’s O(blocksize). Back to square one!



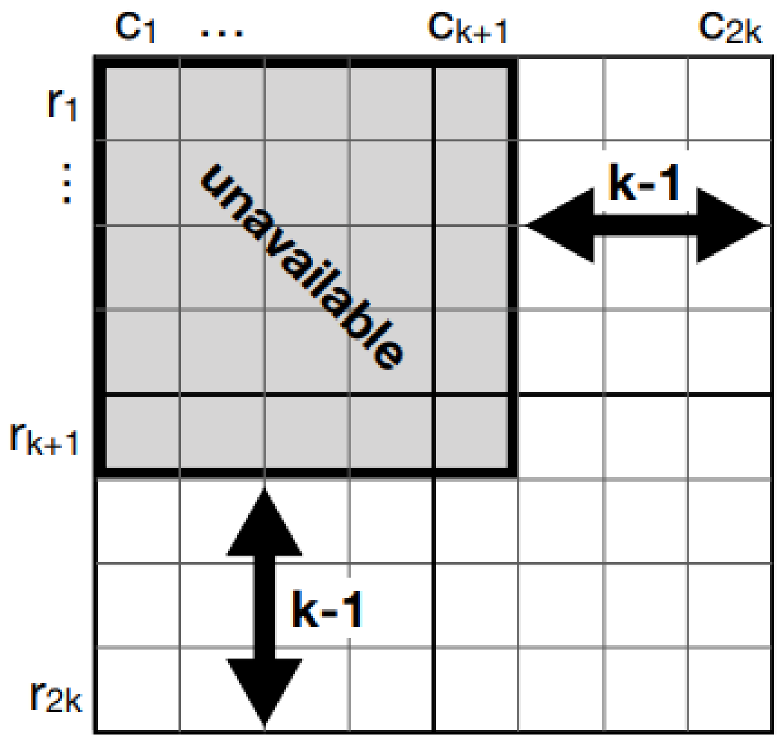
Multidimensional coding

- We can use multidimensional coding to fix this.
- If any row or column is incorrectly generate, a fraud proof that the code is incorrectly generated is limited to that specific row or column. That's O(sqrt(blocksize)).



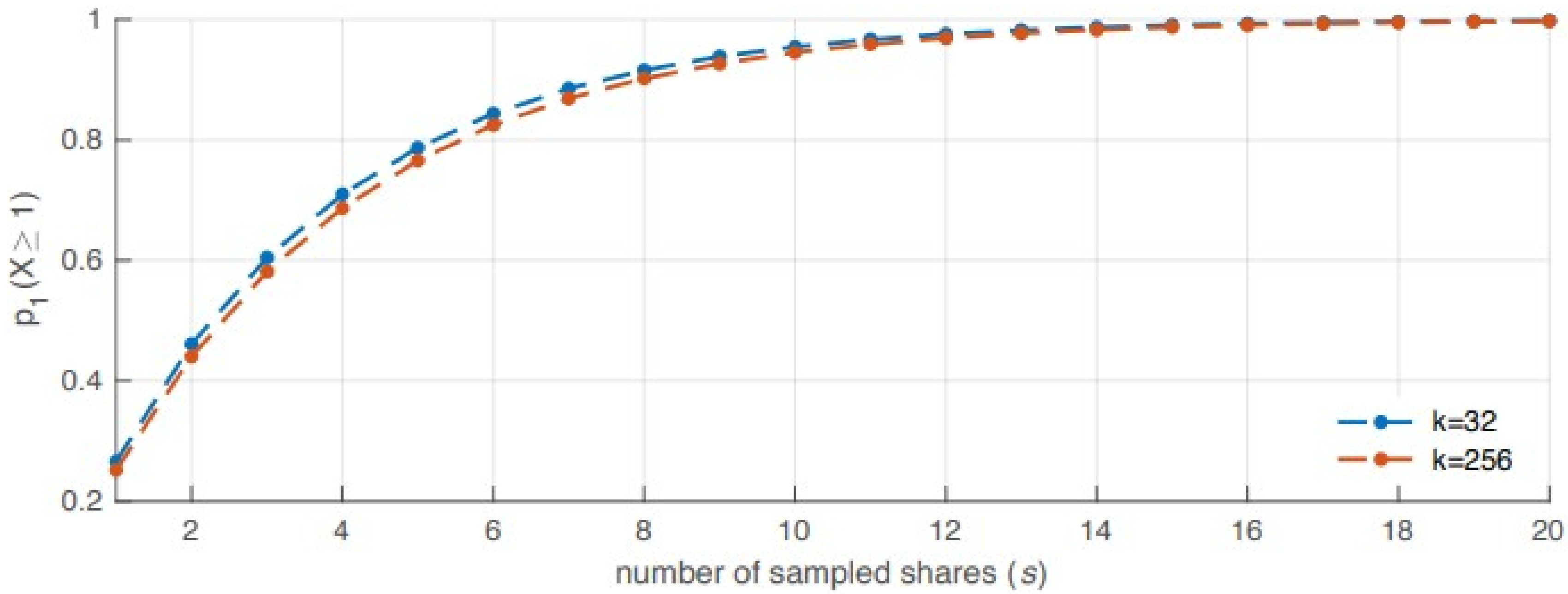
Multidimensional coding

- Miner has to hide roughly 25% of the square to hide any pieces.



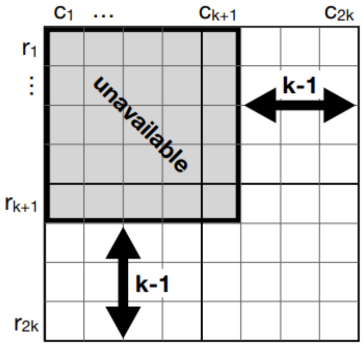
Probability analysis

- What is the probability of a client landing on at least one available piece if the miner has hidden ~25% of the square (if sampling without replacement)?
- 60% after 3 samplings; 99% after 15 samplings.



Selective releasing of pieces

- What if a miner only releases pieces as clients ask for them?
  - Then the miner can always pass the sampling challenge of the first couple of hundred to thousand clients.
  - The exact number of how many clients can be fooled depends on how many samples they make each (s) and how wide is the square (k).



$p_e(Z \geq \gamma)$	$s = 2$	$s = 5$	$s = 10$	$s = 20$	$s = 50$
$k = 16$	692	277	138	69	28
$k = 32$	2805	1,122	561	280	112
$k = 64$	11,289	4,516	2,258	1,129	451
$k = 128$	>40,000	~18,000	~9,000	~4,500	1,811

Preventing selective releasing of pieces

- We can prevent this by assuming an enhanced network model.
  - Clients send requests anonymously.
  - The order in which requests are received by the network are uniformly random (i.e. client’s sampled are interleaved). For example, using a mix net.
- This would mean that a miner would have same probability of fooling all clients, including the first ones to ask for samples.
  - Because requests are unlinkable to clients; sampling challenges cannot be satifisfied on a per-client basis.

Block validity security assumptions comparison

- What additional security assumptions are necessary to only accept valid blocks?

Full nodes	SPV clients	SPV clients + fraud proofs
	+ 51% of consensus is honest	+ at least one honest full node in connected network graph + maximum network delay to receive proofs (e.g. 5 mins) + minimum number of light clients (few hundred)

Implementation

- 2D Reed-Solomon Merkle tree data availability scheme: https://github.com/musalbas/rsmt2d
- Fraud proofs prototype: https://github.com/asonnino/fraudproofs-prototype
- Sparse Merkle tree library with support for state transition witness verification: https://github.com/musalbas/smt

Performance: space/bandwidth

Object	Space complexity	250KB block	1MB block
State fraud proof	$O(p + p\log(d) + w\log(s) + w)$	14,090b	14,410b
Availability fraud proof	$O(d^{0.5} + d^{0.5}\log(d^{0.5}))$	5,120b	12,288b
Single sample response	$O(shareSize + \log(d))$	320b	368b
Header	$O(1)$	128b	128b
Header + axis roots	$O(d^{0.5})$	2,176b	4,224b

Table 2: Worst case space complexity and illustrative sizes for various objects for 250KB and 1MB blocks. p represents the number of transactions in a period, w represents the number of witnesses for those transactions, d is short for dataLength, and s is the number of key-value pairs in the state tree. For the illustrative sizes, we assume that a period consists of 10 transactions, the average transaction size is 225 bytes, and that conservatively there are 2^{30} non-default nodes in the state tree.

Performance: computation

Action	Time complexity	250KB block	1MB block
[G] State fraud proof	$O(b + p\log(d) + w)$	289.78 ms	981.88 ms
[V] State fraud proof	$O(p + p\log(d) + w)$	1.50 ms	1.50 ms
[G] Availability fraud proof	$O(d^2 + d^{0.5}\log(d^{0.5}))$	7.96ms	50.88ms
[V] Availability fraud proof	$O(d + d^{0.5}\log(d^{0.5}))$	0.05ms	0.19ms
[G] Single sample response	$O(log(d^{0.5}))$	< 0.00001ms	< 0.00001ms
[V] Single sample response	$O(log(d^{0.5}))$	< 0.00001ms	< 0.00001ms

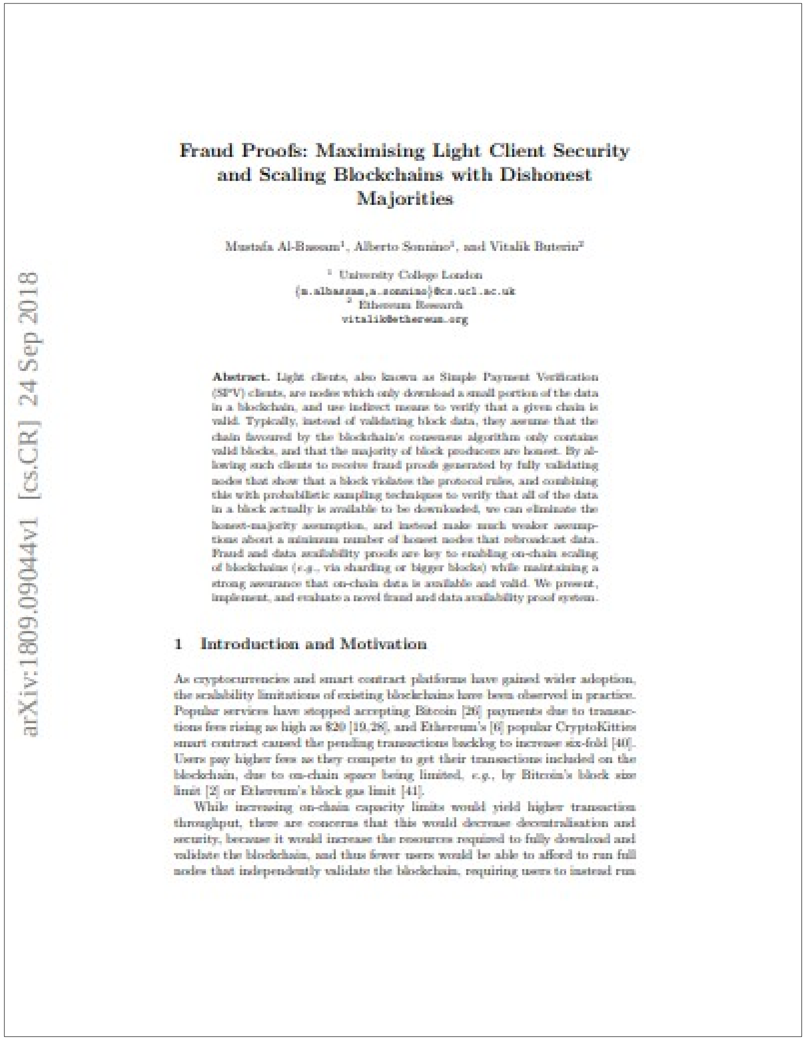
Table 3: Worst case time complexity and benchmarks for various actions for 250KB and 1MB blocks (mean over 10 repeats), where [G] means generate and [V] means verify. p represents the number of transactions in a period, b represents the number of transactions in the block, w represents the number of witnesses for those transactions, d is short for dataLength, and s is the number of key-value pairs in the state tree. For the benchmarks, we assume that a period consists of 10 transactions, the average transaction size is 225 bytes, and each transaction writes to one key in the state tree.

Read the full paper (33 pages)

- Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

  Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin

- https://arxiv.org/abs/1809.09044



arXiv:1809.09044v1 [cs.CR] 24 Sep 2018

Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities

Mustafa Al-Bassam[1], Alberto Sonnino[1], and Vitalik Buterin[2]

[1] University College London
{m.albassam,a.sonnino}@cs.ucl.ac.uk
[2] Ethereum Research
vitalik@ethereum.org

Abstract. Light clients, also known as Simple Payment Verification (SPV) clients, are nodes which only download a small portion of the data in a blockchain, and use indirect means to verify that a given chain is valid. Typically, instead of validating block data, they assume that the chain favoured by the blockchain's consensus algorithm only contains valid blocks, and that the majority of block producers are honest. By allowing such clients to receive fraud proofs generated by fully validating nodes that show that a block violates the protocol rules, and combining this with probabilistic sampling techniques to verify that all of the data in a block actually is available to be downloaded, we can eliminate the honest-majority assumption, and instead make much weaker assumptions about a minimum number of honest nodes that rebroadcast data. Fraud and data availability proofs are key to enabling on-chain scaling of blockchains (*e.g.*, via sharding or bigger blocks) while maintaining a strong assurance that on-chain data is available and valid. We present, implement, and evaluate a novel fraud and data availability proof system.

1 Introduction and Motivation

As cryptocurrencies and smart contract platforms have gained wider adoption, the scalability limitations of existing blockchains have been observed in practice. Popular services have stopped accepting Bitcoin [26] payments due to transactions fees rising as high as $20 [19,28], and Ethereum's [6] popular CryptoKitties smart contract caused the pending transactions backlog to increase six-fold [40]. Users pay higher fees as they compete to get their transactions included on the blockchain, due to on-chain space being limited, *e.g.*, by Bitcoin's block size limit [2] or Ethereum's block gas limit [41].

While increasing on-chain capacity limits would yield higher transaction throughput, there are concerns that this would decrease decentralisation and security, because it would increase the resources required to fully download and validate the blockchain, and thus fewer users would be able to afford to run full nodes that independently validate the blockchain, requiring users to instead run

Thank you

- Questions?
- Twitter: @musalbas
- Email: mus@musalbas.com