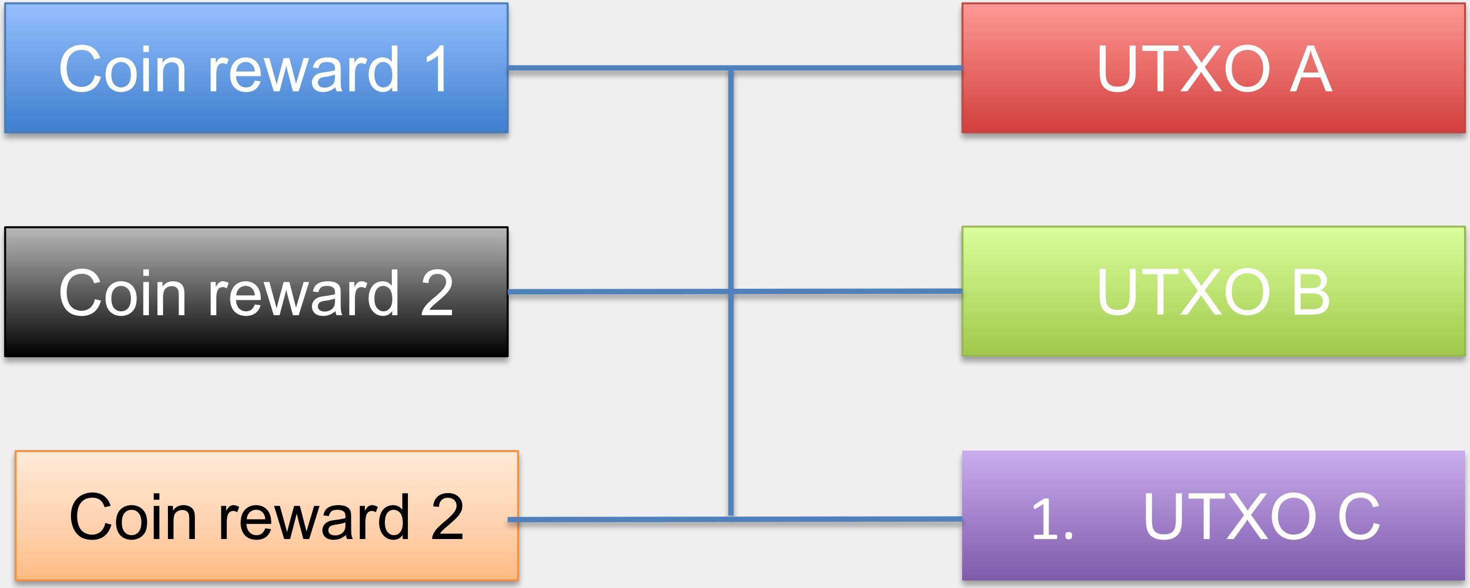
Accumulators and their applications to Mimblewimble

Benedikt Bünz

Joint work with Ben Fisch, Dan Boneh

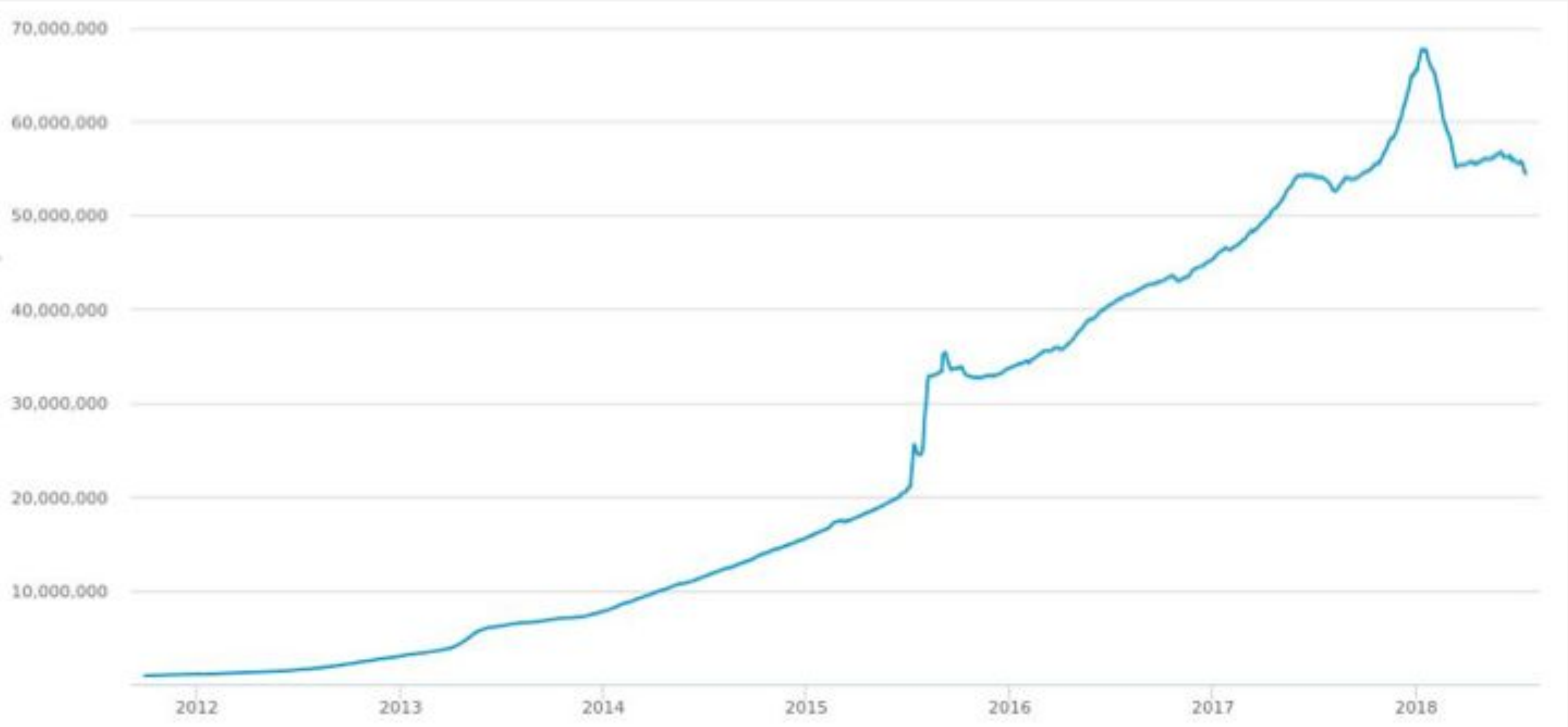
Grincon, January 2019

Mimblewimble

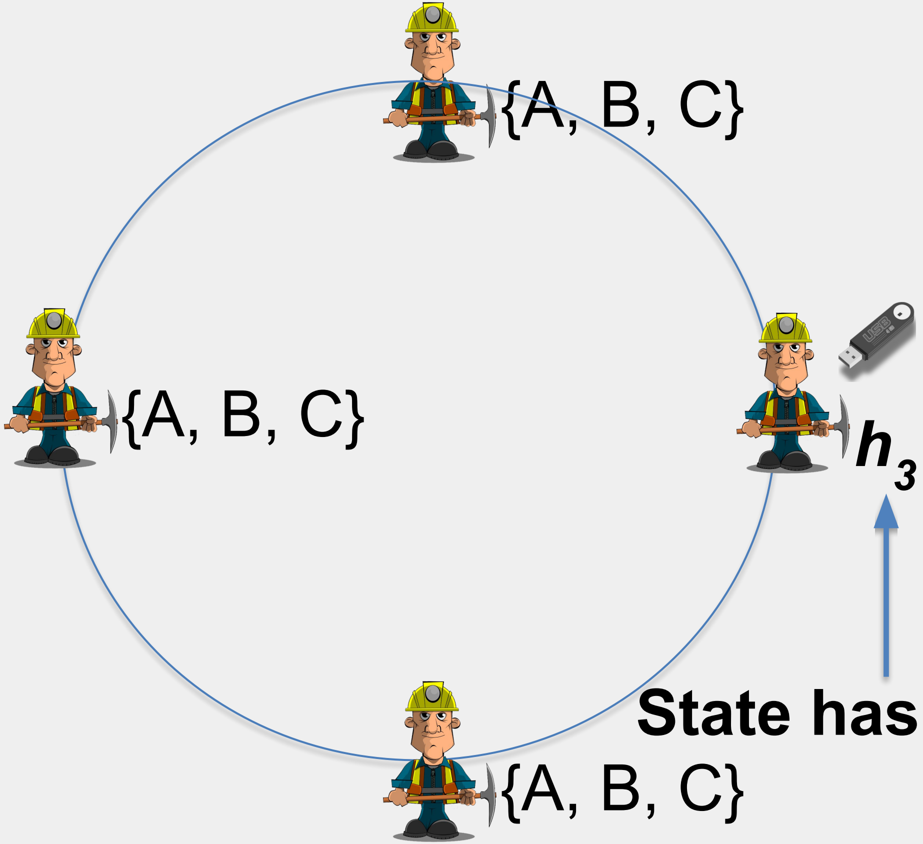


State is (almost) self verifying

Bitcoin State: A Growing Problem



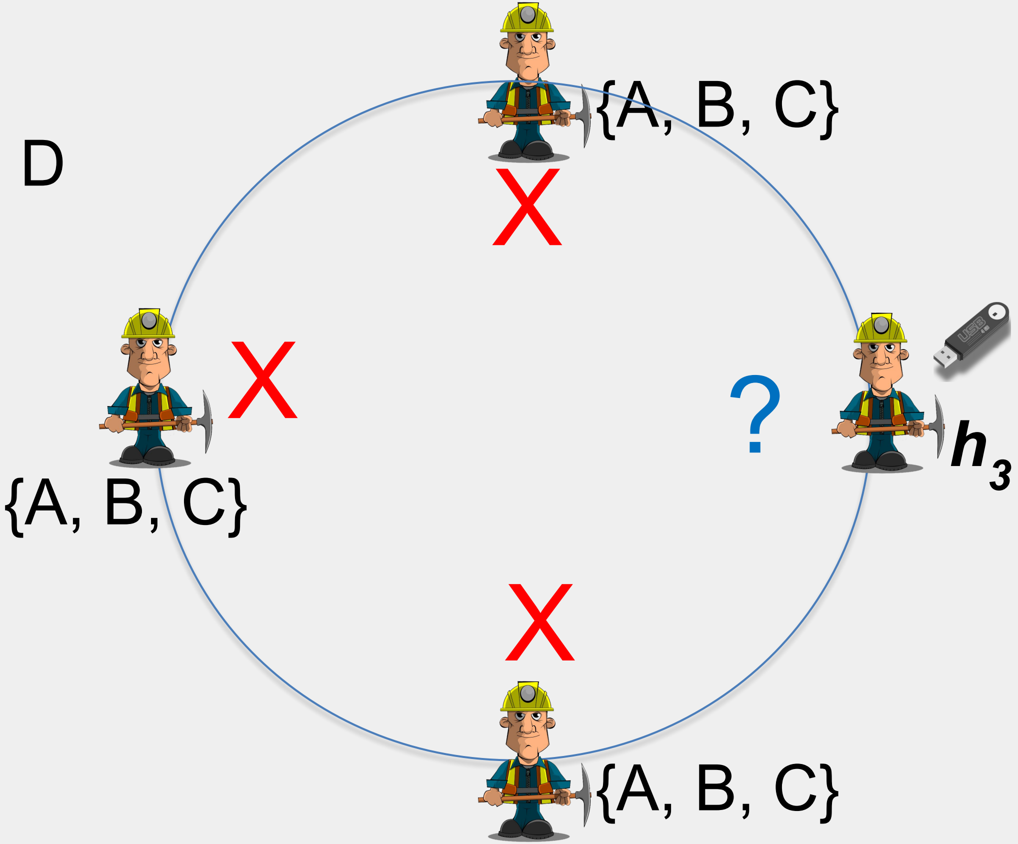
Stateless verification



Verify other nodes agree on same sequence of updates?

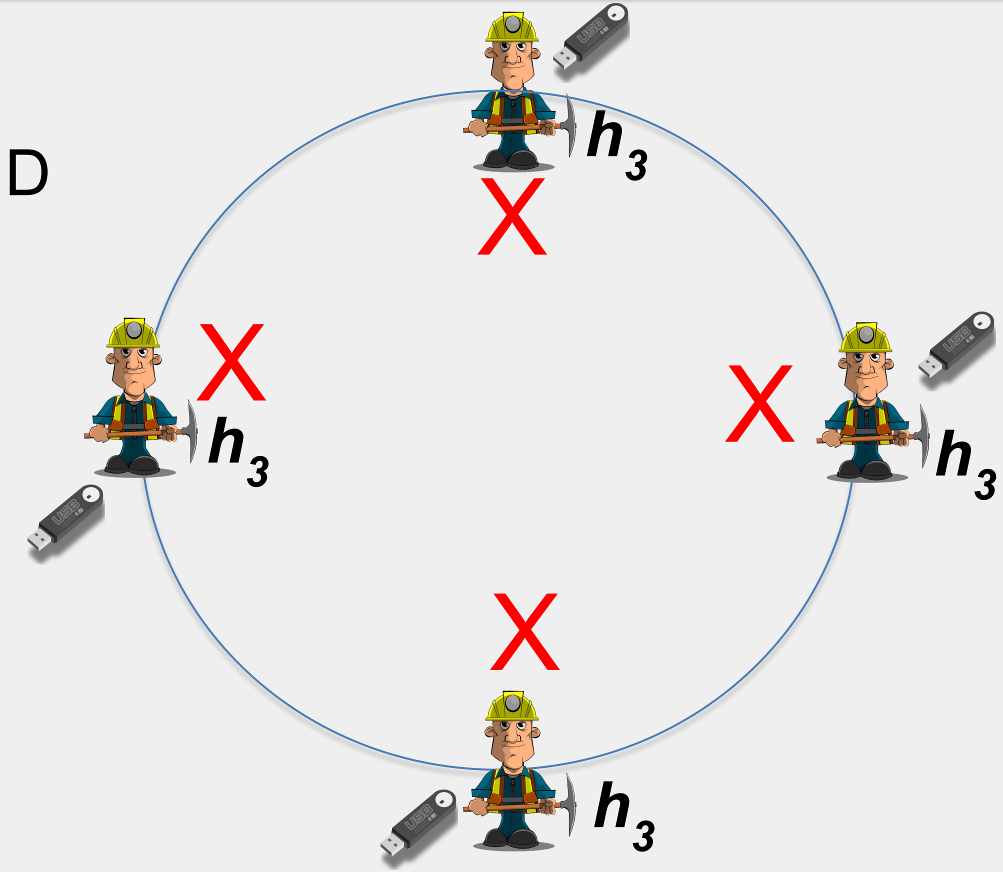
State hash/commitment

Stateless verification

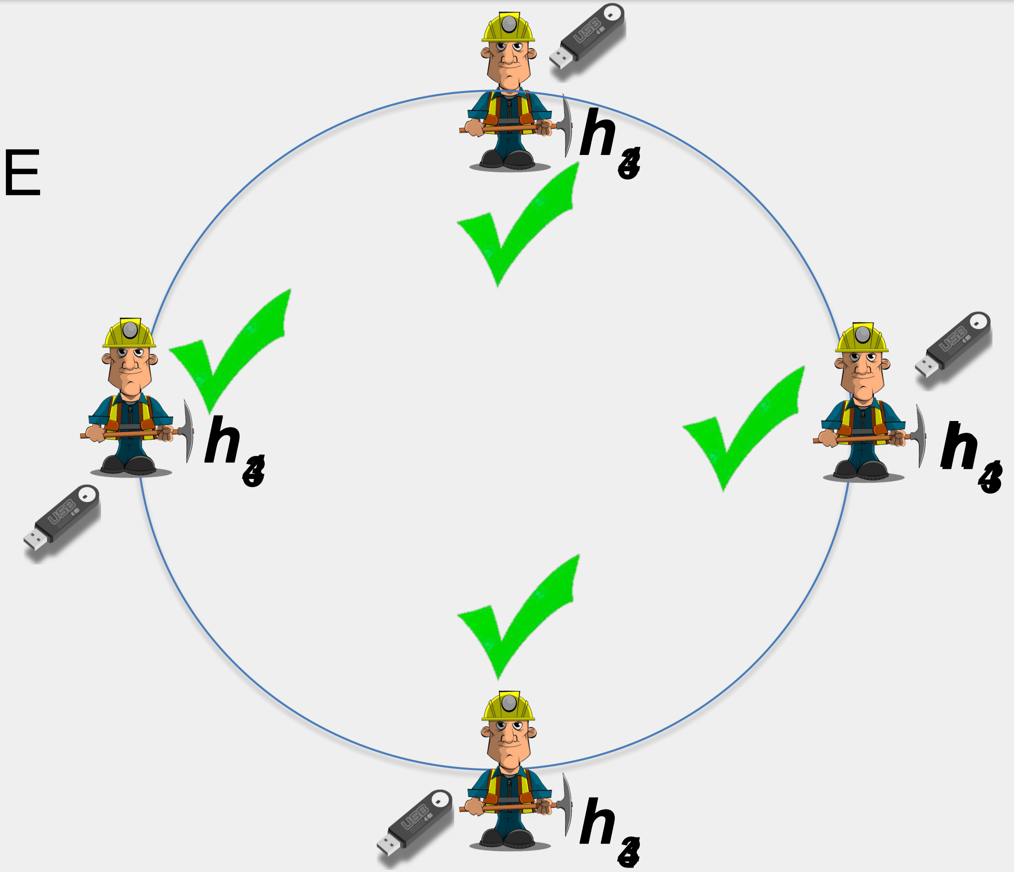


Stateless node can't participate in consensus with rules…

Stateless consensus



Stateless consensus



How can nodes check validity of update?

Consensus and State

State Update + Proof

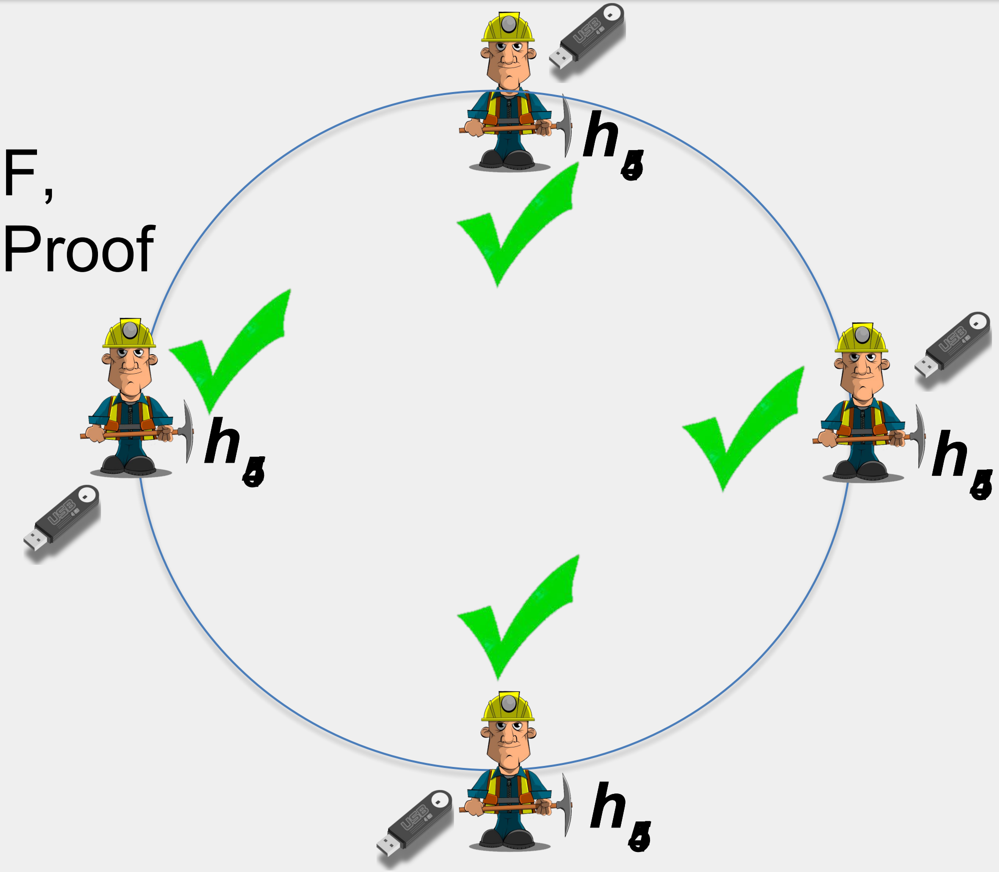
Consensus

Checks proof and updates state

State storage

Provides proofs

Stateless consensus



How is the proof generated? Does it require state?

Who generates the proof?

Stateless consensus

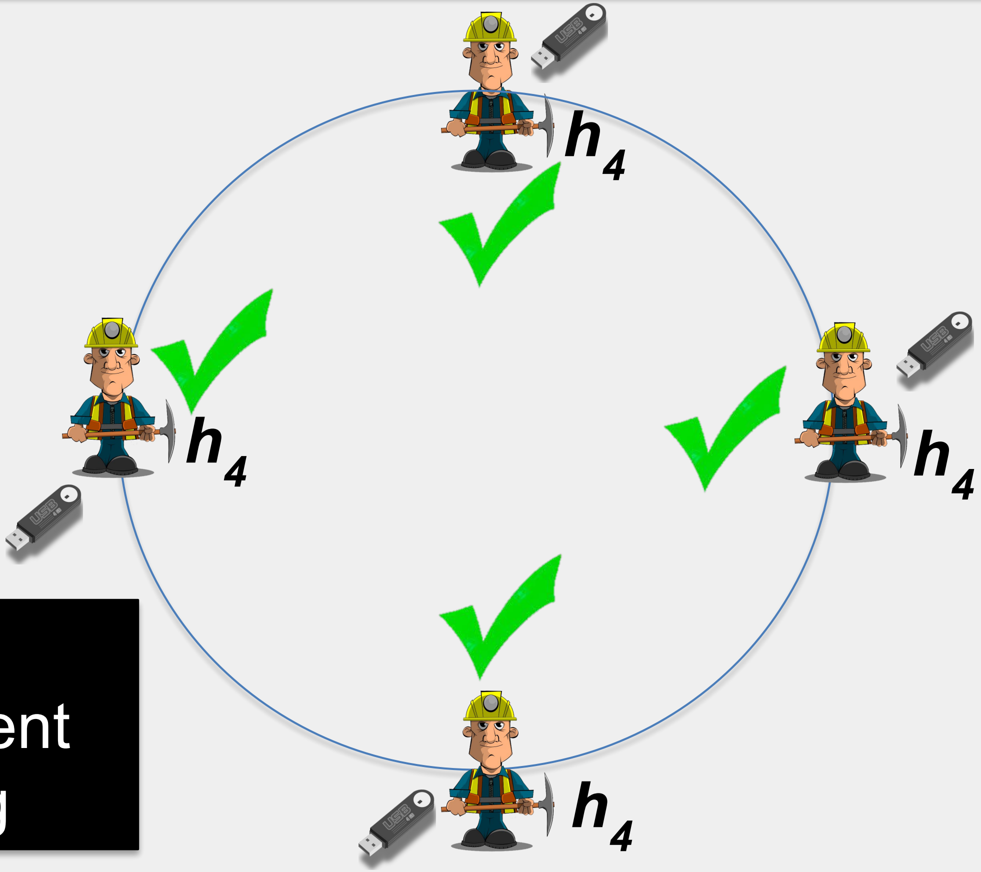
Accounts

Balances

Update

Proof

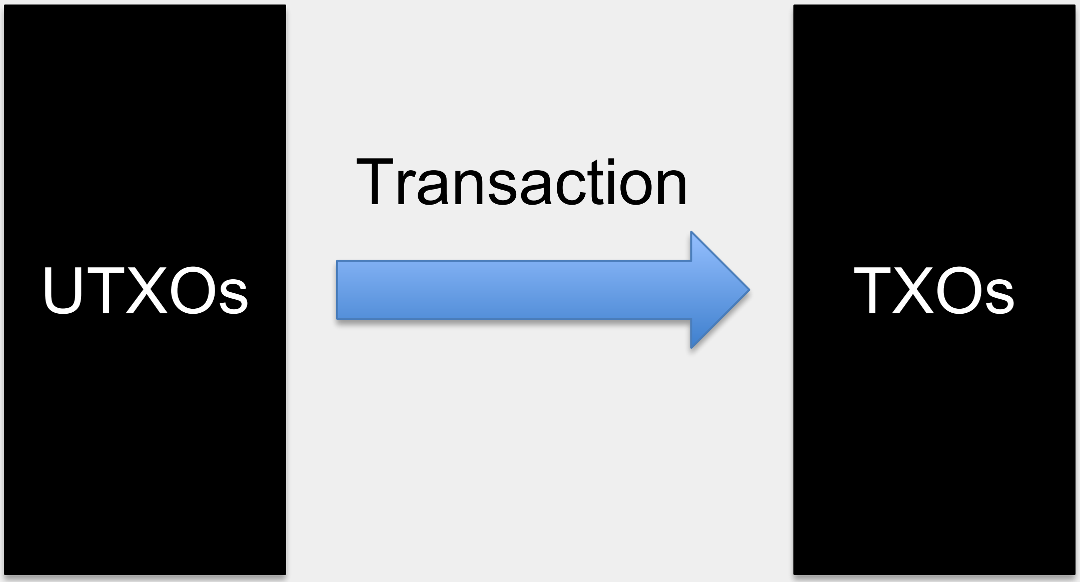
Vector commitment opening



Head contains vector commitment to state

Bitcoin/Mimblewimble UTXOs

Unspent transaction outputs (Coins)



UTXOs

Miners agree on UTXO set S

$\boldsymbol{h} = \boldsymbol{Commit}(S)$

Accumulator

Transaction

UTXOs
TXOs
Proof

Proof

$\boldsymbol{h} = \boldsymbol{Commit}(S)$
UTXOs $\in$ S

Accumulators [Bd94, CL02]

Short Membership Witness

$$A_{i+1} \leftarrow \textbf{Add}(A_i, \mathrm{x})$$

$$\pi = \textbf{InclusionProof}\ (A_{i+1}, \mathrm{x})$$

$$\mathrm{Verify}(\pi, A_{i+1}, x) = \{0{,}1\}$$

Examples:

Merkle trees

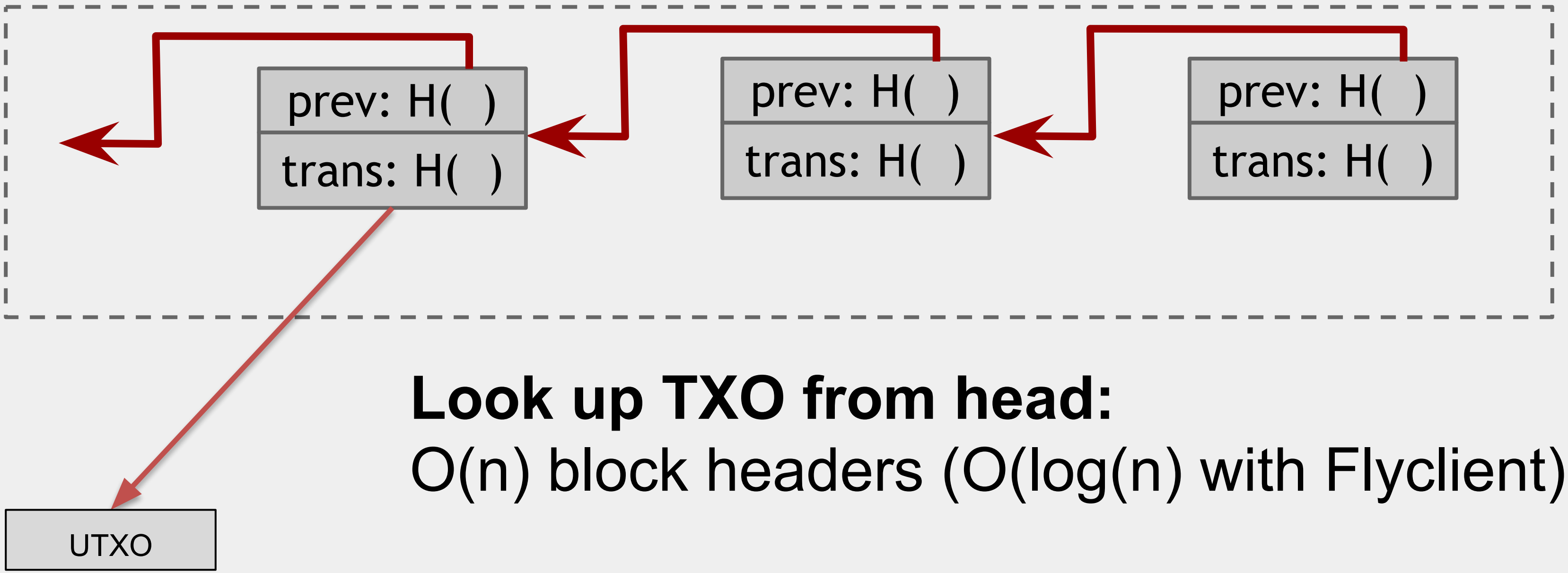
RSA Accumulators [CL02]

Pairing-based accumulators [NG05]

Constant size

Bitcoin UTXOs

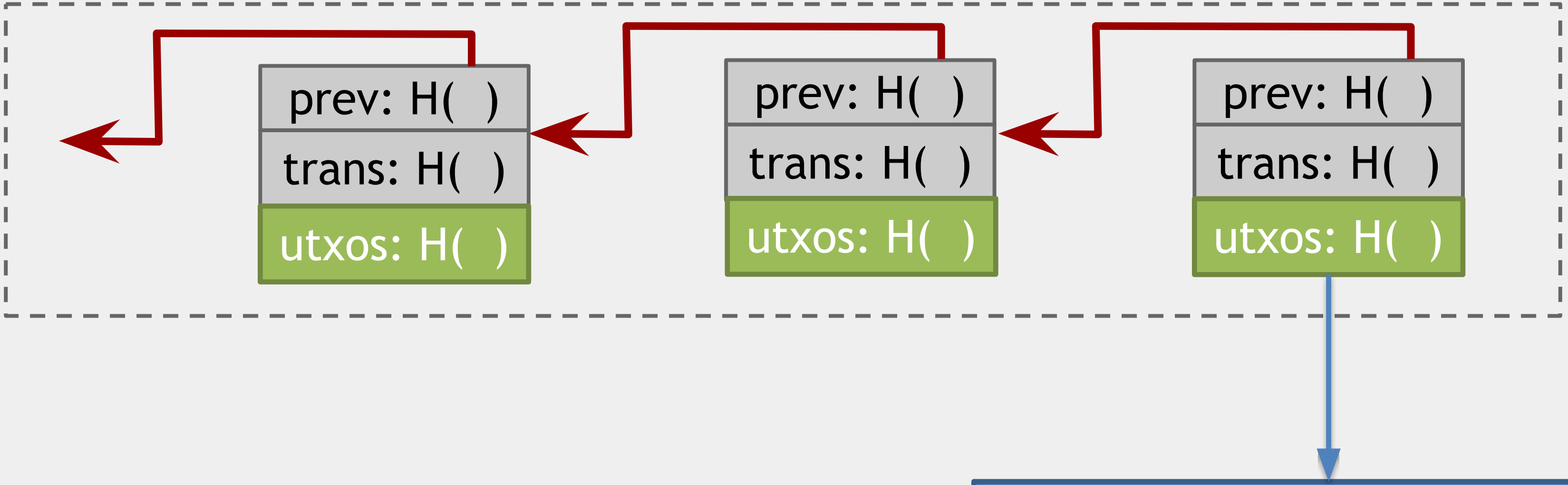
Unspent transaction outputs



Look up TXO from head:
O(n) block headers (O(log(n) with Flyclient)

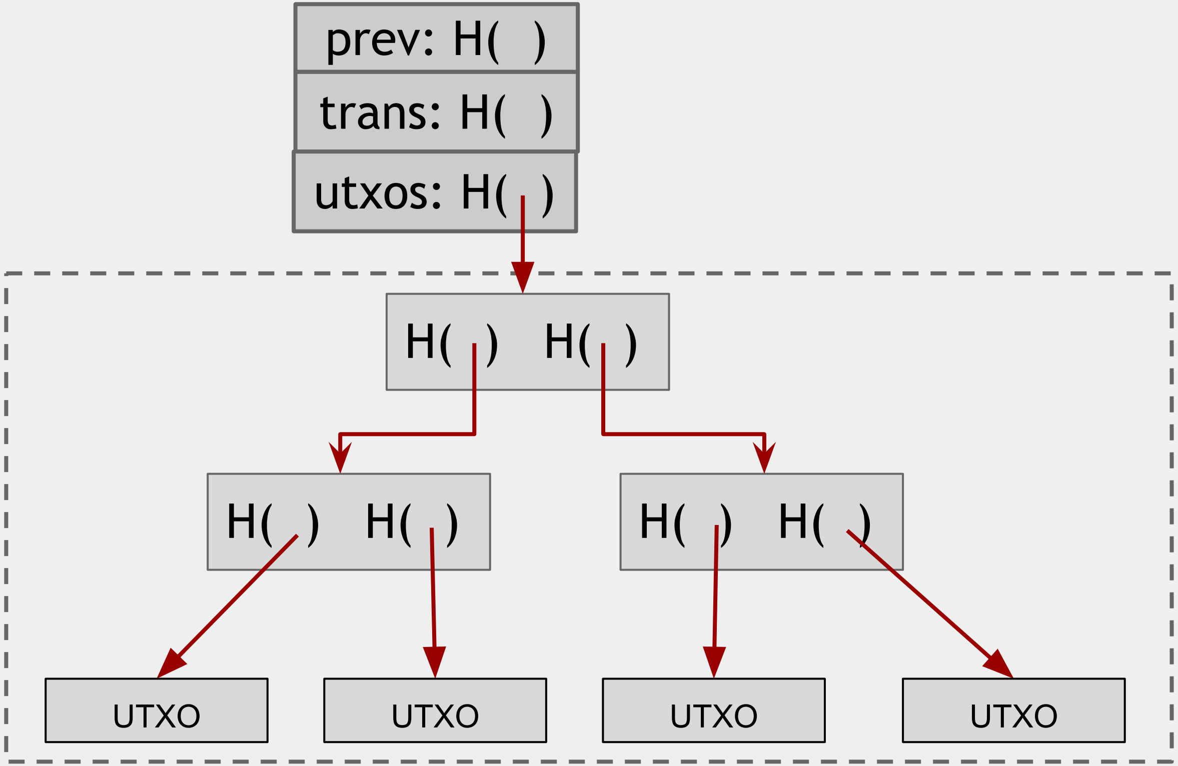
Look up UTXO: All transactions

UTXO Commitments [TMA13]



Consensus ensures:
All UTXO committed here

Merkle Trees



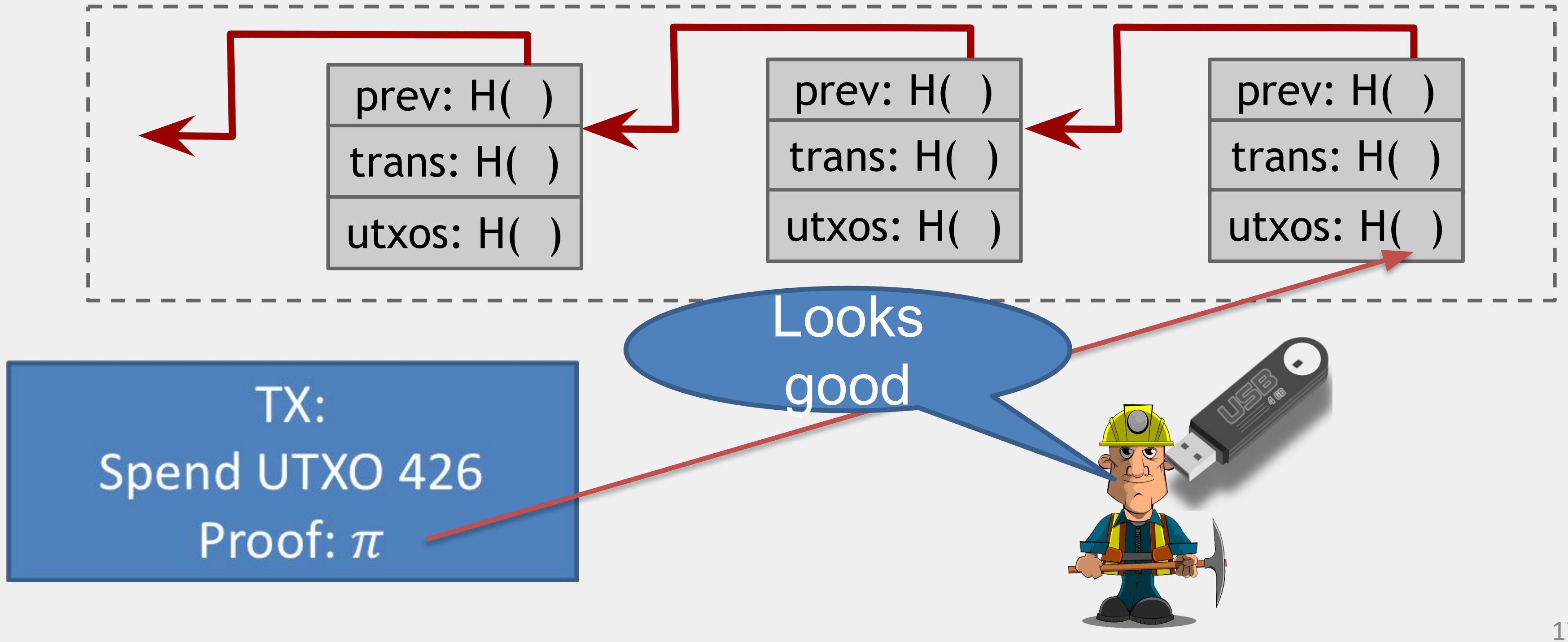
Inclusion: O(log(n))

Exclusion: O(log(n))[1]

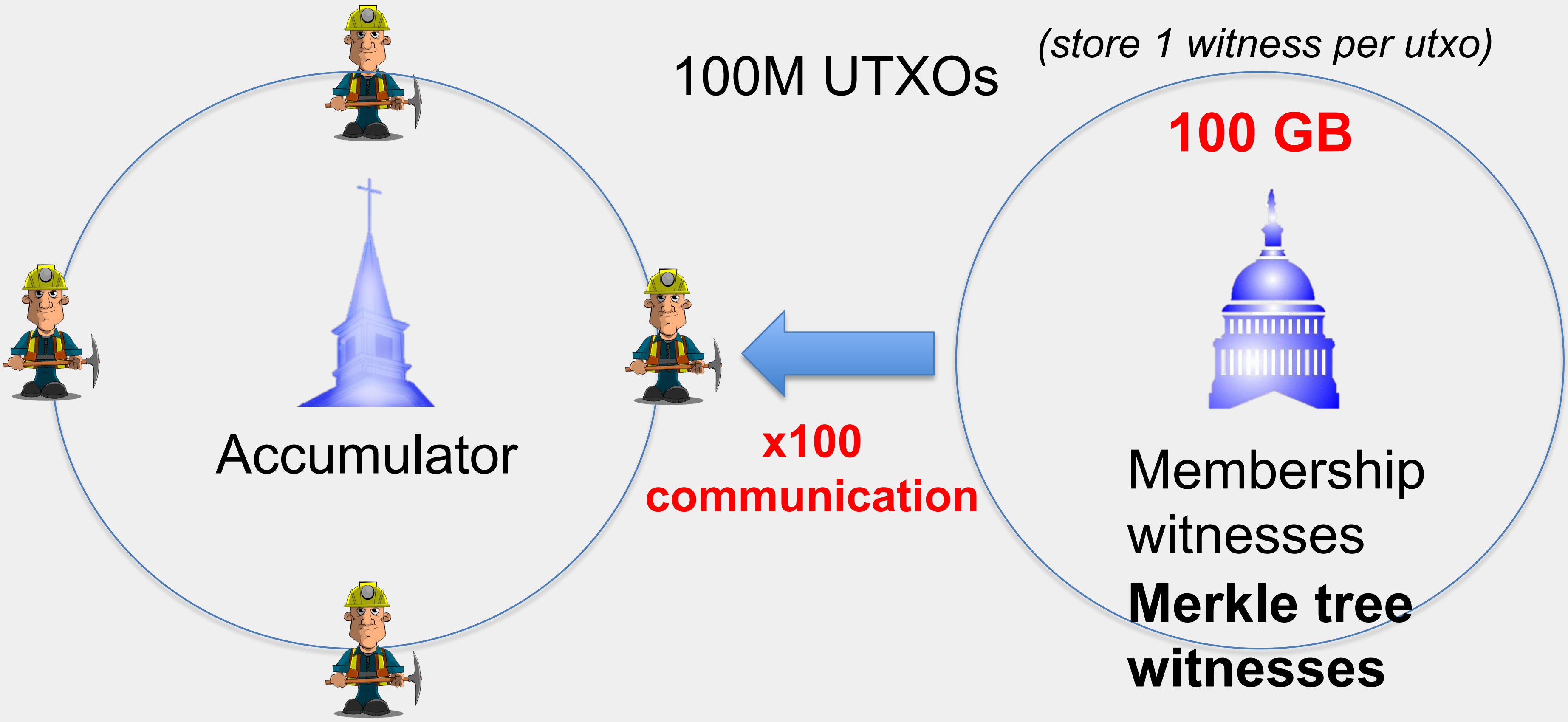
Update: O(log(n))

[1] If sorted

Stateless Full Nodes/Mining



Membership Witnesses



Membership Witnesses

Desiderata

- **Short** membership witness per item
- **Efficiently** updatable (storage + computation)
- **Efficient** verification
- **Aggregate witnesses?**
- **Batch generate & verify?**

Problems with Merkle Trees

- Log(n) inclusion proof per transaction
- Inclusion proofs can hardly be aggregated
  - 600 GB naïvely
  - 160 GB with many optimizations
- Verification not that cheap
  - Full node sync too slow
  - Proposed for only old transactions

RSA Accumulators [CL02, LiLiXue07]

Setup:

- Choose N=pq where p, q are secret primes
- H: Hash function to primes in [0, 2^λ]
- $A_0 = g \in Z_N$ *(initial state)*

Add(A_i, x)

- $A_{i+1} = A_i^{H(x)}$

Del(A_i, x)

- $A_{i+1} = A_i^{1/H(x)}$

State after set S added:

$$u = \prod_{s \in S} s$$
$$A_t = g^u$$

Accumulator Proofs

InclusionProof(A,x):

- $\pi = A^{\frac{1}{x}} \in \mathbb{G}$
- Computed using trapdoor(p,q) Or $O(|S|)$

Verify$(\mathrm{A}, \mathrm{x}, \pi)$

- $\pi^x = A$

Exclusion(A, x)

- $A = g^u$
- $a \cdot x + b \cdot u = \gcd(x, u)$=1
- $\pi = (g^a, b) \Rightarrow$ Verify $\pi^x \cdot A^b = g$

Efficient stateless updates to (non)-membership witnesses: [LiLiXue07]

RSA Accumulator State of Art

Positives

- Constant size inclusion proofs ($\approx$ 3000 bits)
  Better than Merkle tree for set size > 4000
- Dynamic stateless adds (can add elements w/o knowing set)
- Decentralized storage (no need for full node storage)
  - *Users maintain their own UTXOs and membership proofs*

Room for improvement? **New work**

- Aggregate/batch inclusion proofs (many at cost of one)
- Trapdoor free (no trusted setup)
- Stateless deletes
- Faster (batch) verification

Batching Techniques for Accumulators with Applications to IOPs and Blockchains

Joint work with: Ben Fisch and Dan Boneh

https://eprint.iacr.org/2018/1188

Aggregate Membership Witnesses

$$\pi_1^x = A, \pi_2^y = A$$

Shamir's Trick:

$$a \cdot x + b \cdot y = 1$$

$$\pi_{1,2} = \pi_1^b \pi_2^a$$

$$\pi_{1,2}^{x \cdot y} = A$$

All membership witnesses per transaction block: ~3000 bits

RSA = Trusted Setup?

N=p*q, p,q unknown

Efficient delete needs trapdoor

You can find Ns in the wild (Ron Rivest Assumption)

Class Groups [BW88,L12]

$\mathrm{CL}(\Delta)$ – Class group of quadratic number field $\mathbb{Q}(\sqrt{\Delta})$

$\Delta = -p$ (a large random prime)

Properties

- Element representation: integer pairs (a, b)

$$|a| \approx |b| \approx \sqrt{-\Delta}$$

No trusted setup

- Tasks believed to be hard to compute:

Odd prime roots **Group order**

- $\Delta \approx 1536\ bits \Rightarrow$ 128 bit security

Stateless Deletion

Delete with trapdoor(A_t, x):

- $A_{t+1} = A_t^{\frac{1}{x}}$

Using knowledge of p, q

Delete with inclusion proof($\mathrm{A_t, x}, \pi$)

- $A_{t+1} = \pi$;

$$\pi = \mathrm{g}^{\frac{u}{x}}$$

BatchDelete($\mathrm{A_t, x, y}, \pi_1, \pi_2$)

- Compute $\pi_{1,2}$ s.t. $\pi_{1,2}^{x \cdot \mathrm{y}} = A_t$
- $A_{t+1} = \pi_{1,2}$

No State,
no Trapdoor,
asynchronous

Verification of Witnesses Too slow?

- Java Big Integer Microbenchmark:
  - 600 exponentiations per second (256 bit exponents)
  - Verification/Full sync would be problematic

  (On my laptop)

- Class groups: No good benchmarks yet

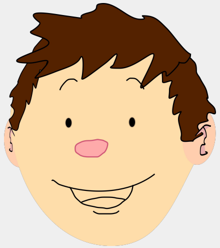
Wesolowski Proof [Wesolowski'18]

$(x, y, T): x^{2^T} = y$

Peggy

$\xrightarrow{\text{y}}$

$\xleftarrow{\text{Random } \lambda \text{ bit prime } \ell}$



Victor

Computes
q,r s.t.
$2^T = q \cdot \ell + r$ and
$0 \leq r < \ell$

$\xrightarrow{\pi = x^q}$

Computes
$r = 2^T \bmod \ell$
Checks:
$\pi^\ell x^r = y$
$x^{q \cdot \ell} x^r = x^{2^T}$

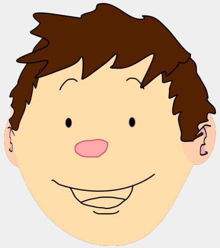
log(T) mults mod ℓ

Proof of Exponentiation (PoE)

$(x, y, {\color{red}\alpha}): x^{\color{red}\alpha} = y$

Peggy

y



Victor

$(x, y, {\color{red}\alpha}): x^{\color{red}\alpha} = y$

Computes
q,r s.t.
${\color{red}\alpha} = q \cdot \ell + r$ and
$0 \leq r < \ell$

$\pi = x^q$

Computes
$r = {\color{red}\alpha} \bmod \ell$
Checks:
$\pi^\ell x^r = y$
$x^{q \cdot \ell} x^r = x^{\color{red}\alpha}$

$\log(\alpha)$ mults mod ℓ

PoE Efficiency

Both linear in bitlength

$(x, y, \alpha): x^{\alpha} = y$

Direct Verification:
$x^{\alpha} = y \in \mathbb{G}$

PoE Verify:
$r = \alpha \bmod l$
$\pi^{l} g^{r}$

Much faster

Exponentiation in $\mathbb{G}$ vs. 128 bit long-division:
5000x difference for 128 bit security

PoE Assumption

Adaptive Root assumption:
For all efficient adversaries A
Given $u \in G \xleftarrow{\$} A(\lambda), \ell \xleftarrow{\$} Primes(\lambda)$
A will produce $w \in G$
$w^{\ell} = u$
With only negligible probability

G can't have any element of known order other than 1
-> We need to work over $Z_N^{+} := Z_N/\{\pm 1\}$
Classgroups seemingly satisfies this property

Fast Block Verification

Header:

TXs: Spent S, new N

Signatures σ

A'_t, A_{t+1}, PoE

A_t → BatchDelete → A'_t

A'_t → BatchAdd → A_{t+1}

Verify σ
Verify PoE for BatchDel
Verify PoE for BatchAdd

Fast Full Sync verification

Mimblewimble:
UTXOs
Signatures σ
Range proofs π
A

$UTXOs$ → BatchAdd → A

Batch verify σ

Batch verify π

Verify PoE for BatchAdd

Performance

Macbook, Java BigInteger, JDK Hash

Merkle Tree: 26 x SHA-256:
8.5 μs >100,000/s

Add: g^x mod N, |x|=256 bit |N|=3072:
1535 μs >600/s

Verify: x mod l, |x|=256 bit |l|=128 bit
0.3 μs+50 μs for primality checking
~20,000/s

Classgroups?

Takeaway Points

Shifting work from miners to users

Distribute the storage (load balanced blockchain)

Consensus

State storage

References

- CL02: Camenisch Lysanskaya 2002 Dynamic Accumulators
- LiLiXue07: Li, Li, Xue 2007 Universal Accumulators
- CF: Catalone Fiore: Vector Commitments
- Todd: https://petertodd.org/2016/delayed-txo-commitments#further-work
- MMR: https://github.com/opentimestamps/opentimestamps-server/blob/master/doc/merkle-mountain-range.md
- UTXO: https://bitcointalk.org/index.php?topic=101734.0
- BW88: Buchmann and Williams