A Unicorn Walks Into A Bar...
Or…. Accumulators for UTXOs

Benedikt Bünz

Joint work with:

Ben Fisch and Dan Boneh
(Formerly BPASE)
Conference: Jan. 30 - Feb 1, 2019. (three days)
Only technical submissions
Submission: October 16th, 2018
https://cyber.stanford.edu/sbc19
Videos: https://tinyurl.com/bpasevideos
Part of the Stanford Center for Blockchain Research (@CBRStanford)
UTXO Set: A Growing Problem
UTXOs

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

Look up UTXO: All transactions
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

prev: H( )
trans: H( )
utxos: H( )

prev: H( )
trans: H( )
utxos: H( )

prev: H( )
trans: H( )
utxos: H( )

TX:
Spend UTXO 426
Proof: $\pi$

Looks good
Problems with Merkle Trees

- Log(n) inclusion proof per transaction
- Inclusion proofs can hardly be aggregated
  - 600 GB naively
  - 160 GB with many optimizations
- Verification not that cheap
  - Full node sync too slow
  - Proposed for only old transactions
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):
• \( \pi = A^x \in \mathbb{G} \)
• Computed using \( \text{trapdoor}(p, q) \text{ Or } O(|S|) \)

Verify(A, x, \( \pi \))
• \( \pi^x = A \)

Exclusion(A, x)
• \( A = g^u \)
• \( a \cdot x + b \cdot u = \gcd(x, u) = 1 \)

Efficient stateless updates:
[LiLiXue07]
RSA = Trusted Setup?

N = p \times q, p, q unknown

Efficient delete needs trapdoor

You can find Ns in the wild (Ron Rivest Assumption)
Class Groups [BW88,L12]

\[ \text{CL}(\Delta) - \text{Class group of quadratic number field } \mathbb{Q}(\sqrt{\Delta}) \]

\[ \Delta = -p \text{ (a large random prime)} \]

**Properties**

- Element representation: integer pairs \((a, b)\)
  \[ |a| \approx |b| \approx \sqrt{-\Delta} \]

- Tasks believed to be hard to compute:
  
  Odd prime roots      Group order

- \(\Delta \approx 1536 \text{ bits } \Rightarrow 128 \text{ bit security} \)
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?  This work
- Aggregate/batch inclusion proofs (many at cost of one)
- Stateless deletes
- Faster (batch) verification
Aggregate Inclusion Proofs

\[ \pi_1^x = A, \pi_2^y = A \]

Shamir's Trick:
\[ a \cdot x + b \cdot y = 1 \]
\[ \pi_{1,2} = \pi_1^b \pi_2^a \]
\[ \pi_{1,2}^x = A \]

All inclusion proofs per block: 1.5kb
All inclusion proofs ever: 160GB -> 1.5kb
Stateless Deletion

Delete with trapdoor \( (A_t, x) \):
- \( A_{t+1} = A_t^x \)

Delete with inclusion proof \( (A_t, x, \pi) \)
- \( A_{t+1} = \pi \)

BatchDelete \( (A_t, x, y, \pi_1, \pi_2) \)
- Compute \( \pi_{1,2} \) s.t. \( \pi_{1,2}^{x \cdot y} = A_t \)
- \( A_{t+1} = \pi_{1,2} \)

Using knowledge of p, q

\( \pi = \frac{u}{g^x} \)

No State, no Trapdoor, asynchronous
Too slow?

- Openssl 2048 bit RSA:
  - 219 updates per second
  - Verification/Full sync would be problematic
- Class groups: No good benchmarks yet
Wesolowski Proof [Wesolowski’18]

Computation

Peggy

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

Victor

Random $\lambda$ bit prime $l$

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

$\pi = x^q$

Computes
$r = 2^T \mod l$

Checks:
$\pi^l x^r = y$
$x^{q \cdot l} x^r = x^{2^T}$
Proof of Exponentiation

\[(x, y, \alpha): x^\alpha = y\]

**Computes**

q, r s.t.

\[\alpha = q \cdot l + r \text{ and } 0 \leq r < l\]

**Victor**

Random \(\lambda\) bit prime \(l\)

**Peggy**

\[\pi = x^q\]

**Computes**

\[r = \alpha \mod l\]

**Checks:**

\[\pi^l x^r = y\]

\[x^{q \cdot l} x^r = x^\alpha\]
Proof of Exponentiation Efficiency

Direct Verification:

\( x^\alpha = y \in \mathbb{G} \)

PoE Verify:

\[ r = \alpha \mod l \]

\[ \pi^l g^r \]

Exponentiation in \( \mathbb{G} \) vs. 128 bit long-division:

5000x difference for 128 bit security
Fast Block Verification

Header:
- TXs: Spent s, new N
- BLS $\sigma$

$A_{t+\frac{1}{2}}, A_{t+1}, PoE$

Remove $s$

Add $N$

Verify $\sigma$

Verify $PoE(A^{\prod_{s \in S} s}_{t+\frac{1}{2}} = A_t)$

$PoE(A^{\prod_{n \in N} N}_{t+\frac{1}{2}} = A_{t+1})$
Performance

Macbook, Java BigInteger, JDK Hash

Merkle Tree: 26 x SHA-256:
8.5 μs

Add: \( g^x \mod N \), \(|x|=256\) bit \(|N|=3072\):
1535 μs

Verify: \( x \mod l \), \(|x|=256\) bit \(|l|=128\) bit
0.3 μs
Vector Commitments

\[ VC = \text{Commit}(a_1, a_2, a_3, \ldots, a_n) \]

\[ \pi = \text{Open}(VC, a_i, i) \]

\[ \text{Verify}(VC, \pi, a_i, i) = \{0, 1\} \]

Merkle trees are VCs not just accums!

Classical VCs: Verifier requires GBs of memory

New VCs: Zero-memory
Short IOPs (STARKs etc.)

\[ MT = \text{Commit} \left( \ldots \right) \]

\[ \pi_{i_1}, \ldots, \pi_{i_\lambda} \text{ and Merkle Paths} \]
Short IOPs (STARKs etc.)

VC = Commit( Long Proof )

\[ i_1, \ldots, i_\lambda \]

\[ \pi_{i_1}, \ldots, \pi_{i_\lambda} \] and 1 VC Opening

200kb vs. 600kb
A Unicorn Walks Into A Bar...

Accumulators, Unions, Wesolowski, IOPs, Aggregation and Blockchains
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