LVMT: An Efficient Authenticated Storage for Blockchain

Chenxing Li¹, Sidi Mohamed Beillahi², Guang Yang¹, Ming Wu¹, Wei Xu³, Fan Long¹,²

¹Shanghai Tree-Graph Blockchain Research Institute
²University of Toronto
³Tsinghua University
Evolution of Blockchain Performance

Early blockchain systems are slow

- Bitcoin (< 30 transactions / second)
- Ethereum
- VISA (Reaches 20,000 transactions/second)

Resolved bottlenecks

- Transaction Broadcast
  - Resolved by bandwidth-efficient protocol

Resolved bottlenecks (reaches 20,000 transactions/second)

- Consensus for Transaction Order
  - Resolved by high performance consensus protocols

Next bottleneck

- Ordered transaction with consensus
  - Transaction Execution
    - Impedes by inefficient authenticated storage.

Ordered transaction with consensus

Execution Receipt
Architecture of Blockchain Execution Layer

Ordered Transactions → Execute

Stack-based Virtual Machine

Cache

- Fetch value of key 0x23
- Push the value to stack

If not exists, load the value from the storage

A Key-value storage

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH 0x23</td>
<td>[0x23]</td>
</tr>
<tr>
<td>SLOAD</td>
<td>[val]</td>
</tr>
<tr>
<td>PUSH 0x45</td>
<td></td>
</tr>
<tr>
<td>SSTORE</td>
<td></td>
</tr>
</tbody>
</table>
Architecture of Blockchain Execution Layer

Ordered Transactions → Stack-based Virtual Machine → Cache

- **Opcode** | **Stack**
  - PUSH 0x23  | [0x23]
  - SLOAD     | [val]
  - PUSH 0x45 | [val, 0x45]
  - SSTORE    | []

Set value of key 0x45

Flush all the changes at the end of block execution

A Key-value storage
Architectures of Blockchain Execution Layer

- **Ordered Transactions**
- **Stack-based Virtual Machine**
- **Cache**
- **Authenticated Data Structure**
- **Backend Key-value Database**

### Opcode Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH 0x23</td>
<td>0x23</td>
</tr>
<tr>
<td>SLOAD</td>
<td>val</td>
</tr>
<tr>
<td>PUSH 0x45</td>
<td>val, 0x45</td>
</tr>
<tr>
<td>SSTORE</td>
<td></td>
</tr>
</tbody>
</table>

- **Execute**
- **Cache**: 1 r, 1 w
- **Authenticated Data Structure**: 4 r, 8 w (By our experiments)
Authenticated Storage

Server

A key-value storage

Compute a succinct digest

Commitment

Client

Query a key

Answer (Reveal) the value with proof

Client

Verify proof with commitment

Commitment

Put in block

Read from block
The Merkle Tree

Vector Commitment protocol: Merkle Tree

Variants for blockchain system: MPT, RainBlocks, LMPTs

- When an input element changes, the nodes along the path also changes.
- Each node is a key-value pair in backend → $O(\log n)$ read-write amplification

Each node is the hash value of its children.
The Merkle Tree

Vector Commitment protocol: **AMT**

Variants for blockchain system: **LVMT (our work)**

- AMT removes the inner nodes and achieves $O(1)$ cost in maintaining commitment.
Challenges in using AMT

• Fast in complexity ≠ fast in practice
  • AMT has slow cryptographic operations.

• AMT is not scalable.
  • Max capacity of AMT = Size of public parameters.

• Proof generation is Expensive
  • Maintaining data for generating proofs is also $O(\log n)$
Challenge 1: Costly cryptographic operations

In AMT, each time a value changes as,

\[ a_i \rightarrow a_i' \]

the commitment adjusts accordingly

\[ C \rightarrow C + (a_i' - a_i) \cdot G_i. \]

- Precomputed Parameter (200 byte)
- Elliptic Curve Multiplication (92 μs)
- Big Integer Subtraction (<0.01 μs)
- Elliptic Curve Addition (0.34 μs)
Challenge 1: Costly cryptographic operations

In AMT, each time a value changes as,

\[ a_i \rightarrow a_i' \quad (\text{Assumes } a_i' - a_i = 1) \]

the commitment adjusts accordingly

\[ C \rightarrow C + 1 \cdot G_i. \]
Solution 1: Version-based database

Set \((key, val)\)

- Increase ver of key by 1
- Multi-level AMTs
- Prove the current ver of key

Store tuple \((key, ver, val, loc)\)

- Append-only Merkle trees
- Prove existence of tuple \((key, ver, val, loc)\)

Prove key
Challenge 2: AMT is not scalable

1. Determined at the setup phase

Input vector of size $2^k$ (for any $k$)

2. Precomputed parameters in size of $2^k$

3. Blockchain has 256-bit key space, but $k = 256$ is infeasible.

- AMT
- Commitment
- 254-bit value
- Vector Index
Solution 2: Use multiple-level AMT.

Set \((key, val)\)

1. Increase version numbers in \(\star\), \(B\) and \(A\) by 1 and update commitments.
2. Add the following tuples to the Merkle trees.
   
   \[
   (key, \star, val, (3,2)) \\
   ((2,1), B, B) \\
   (2, A, A)
   \]

Prove key

1. Prove the version numbers with respect to the AMT commitment:
   
   \[
   \star \rightarrow B \rightarrow A \rightarrow Rt
   \]
2. Prove the existence of the left three tuples in Merkle trees to demonstrate the commitments at specified version numbers.
   
   \[
   \star \rightarrow B \rightarrow A \rightarrow Rt
   \]
Challenge 3: Maintaining proof data incurs significant costs

Nodes *not serving clients* only need to maintain commitment in $O(1)$ time.

Nodes *serving clients* maintain auxiliary information for generating proof in $O(\log n)$ time.
Solution 3: Proof Sharding

Commitment node
Don’t maintain proof and do not serve users

Auxiliary information for generating proofs
Input vector (Version numbers)
Sub-AMTs

RPC provider
Maintain proofs with a cluster to serve users
Modular Authenticated Storage Benchmark Tool

Execution Task

Authenticated Data Structure

Key-Value Database

- 1m 10m 100m: Random task on various ledger size (in million)
- fresh: Randomly access new keys only
- real: Real Ethereum trace

- LVMT-r: Only maintains commitment
- LVMT#: #: the fraction of proof shards
- MPT, RAIN, LMPTs: Baselines
Throughput on micro-benchmarks
Throughput on a Blockchain Node

Throughput for Simple Transactions

<table>
<thead>
<tr>
<th>Number of Initialized Keys</th>
<th>RAW</th>
<th>LVMT-r</th>
<th>LVMT64</th>
<th>LVMT16</th>
<th>RAIN</th>
<th>LMPTs</th>
<th>MPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m</td>
<td>36</td>
<td>31</td>
<td>28</td>
<td>25</td>
<td>22</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3m</td>
<td>36</td>
<td>29</td>
<td>27</td>
<td>24</td>
<td>18</td>
<td>17</td>
<td>11</td>
</tr>
<tr>
<td>5m</td>
<td>35</td>
<td>29</td>
<td>27</td>
<td>23</td>
<td>17</td>
<td>16</td>
<td>11</td>
</tr>
</tbody>
</table>
Time Usage Breakdown

Time Usage Breakdown for Simple Transactions

- **Execution Engine**
- **Authenticated Structure**
- **Backend**

<table>
<thead>
<tr>
<th>Authenticated Storage Systems</th>
<th>Time (us)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAW</td>
<td>16</td>
</tr>
<tr>
<td>LVMT-r</td>
<td>16</td>
</tr>
<tr>
<td>LVMT64</td>
<td>16</td>
</tr>
<tr>
<td>LVMT16</td>
<td>18</td>
</tr>
<tr>
<td>RAIN</td>
<td>19</td>
</tr>
<tr>
<td>MPT</td>
<td>42</td>
</tr>
</tbody>
</table>

- RAW: 16 us
- LVMT-r: 16 us
- LVMT64: 16 us
- LVMT16: 18 us
- RAIN: 19 us
- MPT: 42 us
Conclusion

• LVMT utilizes the superior vector commitment protocol AMT, offering higher optimization potential.

• Through the version-based design, multi-level AMT, and proof sharding, LVMT addresses challenges effectively.

• LVMT enhances the execution throughput of a blockchain system by up to 2.7x.
Thank you and see you in Q&A

Email: lylcx2007@gmail.com
Github: https://github.com/ChenxingLi/authenticated-storage-benchmarks
https://github.com/Conflux-Chain/conflux-rust/tree/asb-e2e