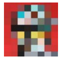
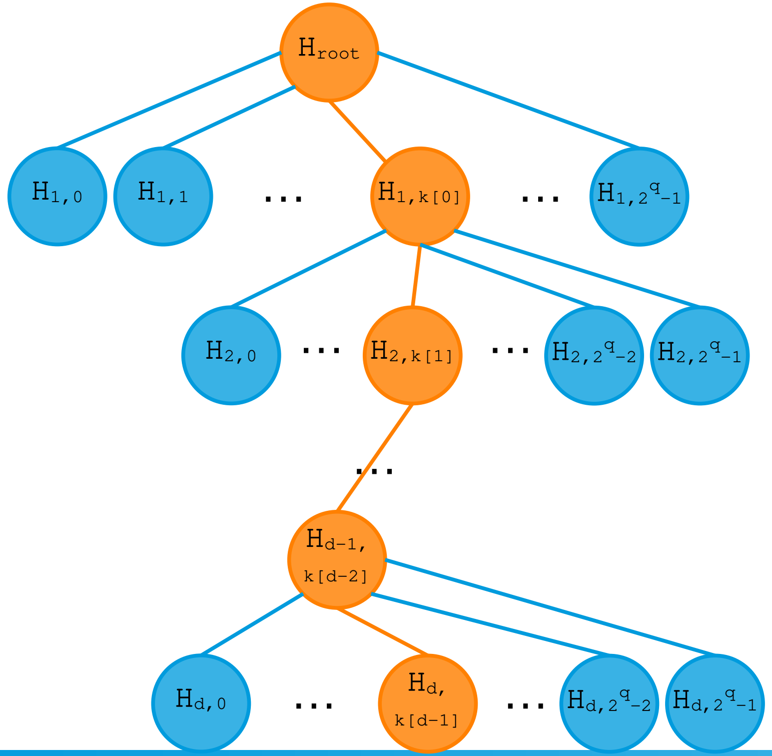
Merged proofs in Verkle tree and optimization

Ressac_No1

Acknowledgments

- My appreciation of their instruction and discussion with me:
  - EF Stateless consensus & Verkle-tree migration team
    -  Guillaume Ballet
    -  Ignacio Hagopian
    -  Josh Rudolf
  - ©g11tech g11tech
  - Milos Stankovic
  - Dankrad Feist

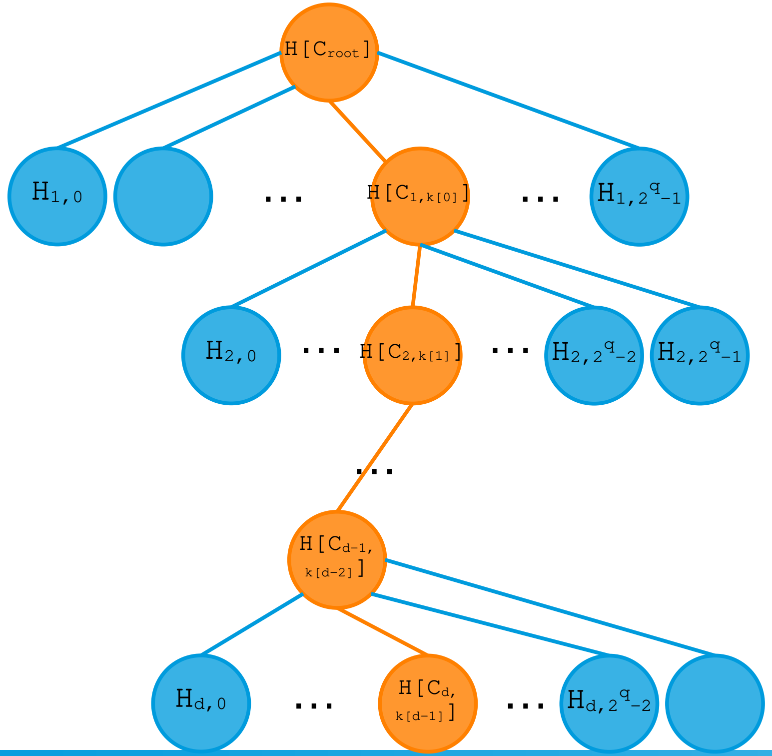
- Merkle-Patricia Trie has overhead in proof size



  - In a MPT of width `d` and branch width 2^q
  - The proof of the value of key `k[0:d]` consists of `d` hashing computations, each convinces that a node along the root-leaf path has the correct hash encoded from its all children
  - Thus, the hashes of all the nodes along the path and their siblings (except the root) must be contained in the proof
  - Total proof size $O(2^q d)$

From Merkle to Verkle

- Verkle tree reduces the proof size by avoiding sibling hashes



- A node's hash is derived from the commitment of the polynomial with children's hashes as evaluations at point 0, 1, …, 2^q-1
- The proof only contains those hashes of nodes along the root-leaf path and correctness of every evaluation: $H[C_{i+1,k[i]}]=f_{i,k[i-1]}(k[i])$ in which $f_{i,k[i-1]}$ is committed as $C_{i,k[i-1]}$

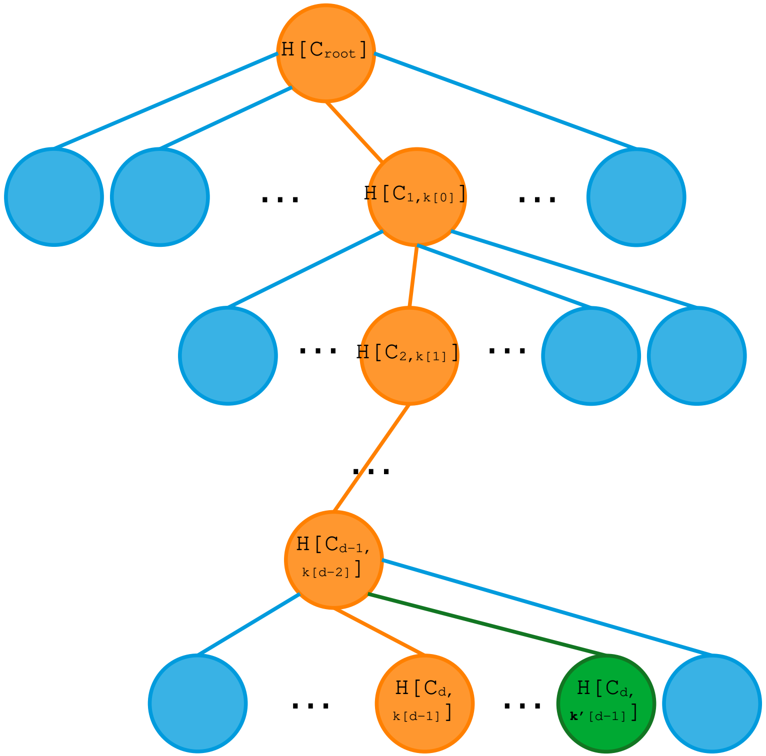
Verkle proof of one node

- Given a key $k[0:d]$, along its access path in the Verkle tree, it must prove for each node at depth $i \in [0:d-1]$ that $H[C_{i+1}, k[i]] = f_{i,k[i-1]}(k[i])$, in which $f_{i,k[i-1]}$ is committed as $C_{i,k[i-1]}$
- Polynomial commitment scheme: ~~KZG ($O(1)$ proof size, $O(1)$ pairing check in verification)~~ Inner Product Arguments ($O(\log n)$ proof size, $O(\log n)$ group operations in verification)
- KZG is discarded because of the compulsory trusted setup
- Pedersen+IPA, n is the degree of the committed polynomial, equal to branch width (2^q) in Verkle tree proof.

Pedersen+IPA multiproof

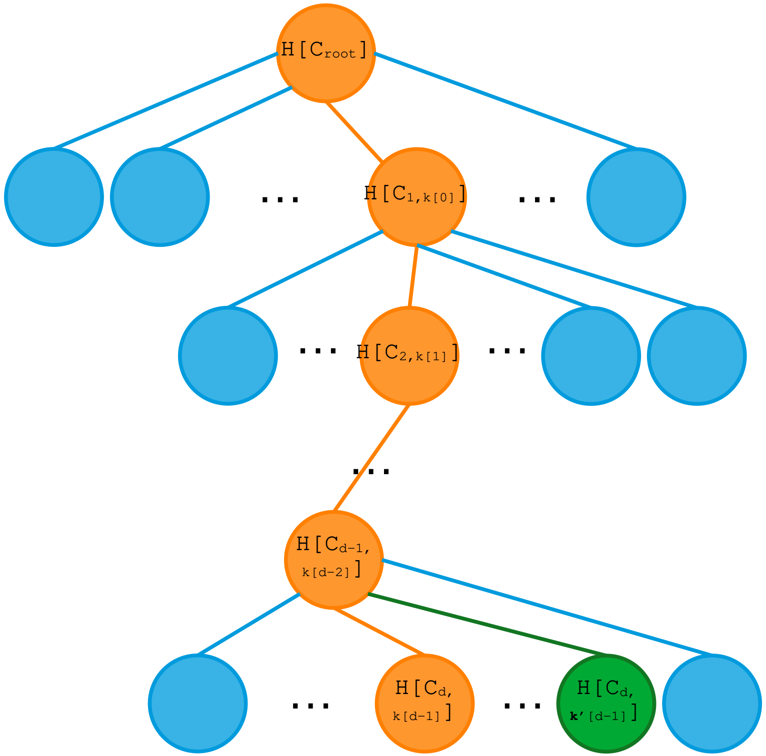
- Pedersen commitment is additive homomorphic
  - An n-dimension vector x is committed as $C_x = \sum_{i=0}^{n-1} x_i g_i$
  - In which g is an n-dimension vector of group elements, as the basis
- Hence, the opening of multiple commitments can be merged
  - To show $f_0(z_0)=y_0$, $f_1(z_1)=y_1$, …, $f_{d-1}(z_{d-1})=y_{d-1}$
  - Enough to show $\sum_{j=0}^{d-1} a_j f_j(z_j) = \sum_{j=0}^{d-1} a_j y_j$ in case all a_j are random
  - Commit the linear combined $2^q d$-dimension vector and open it via IPA
  - $O(q \log d)$ proof size and verification time for a SLOAD access

Merged proof: shared commitments



- In multiple accesses of the same leaf node, naturally, the proof remains the same
- In accesses of multiple nodes? Some ancestors are shared
- Example: two keys `k` and `k'` differs only at the last letter

Merged proof: shared commitments



One proof: d commitments involved

Two proofs: d+1 commitments involved

- In multiple accesses of the same leaf node, naturally, the proof remains the same
- In accesses of multiple nodes? Some ancestors are shared
- Example: two keys `k` and `k'` differs only at the last letter

Stem and suffix

- In the Verkle tree scheme applied to Ethereum upgrade, q=8, each edge matches a byte
- A key's last byte is its **suffix** and the left is its **stem**
- Basic form of proofs of storage slot operations is up to stems, and the commitments of sibling leaf nodes (representing the same stem but different suffices) are opened together
- Amortized gas cost is much lower to access multiple keys with shared stems (EIP-4762)

Merged proof: rebuilding the witness tree

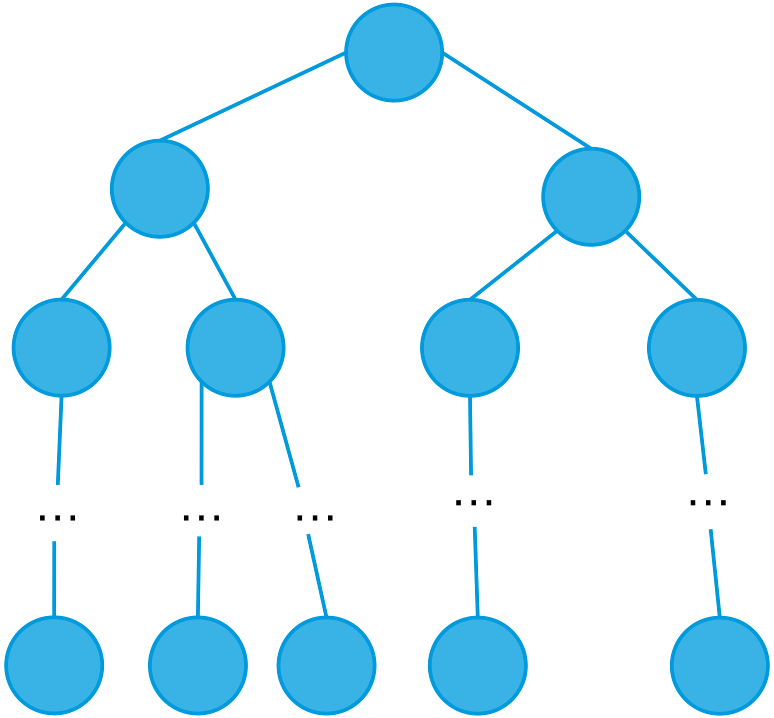
- In practice, it is the block builder to determine the accessed (including updated) storage slots caused by the txns in the block
- Clients are stateless, only trusted block builders own the world states, including codes and stored values in the Verkle tree
- Rebuilding the “witness tree” of all accessed nodes and their values and commitments, with updates as the result of SSTOREs
- Then producing proofs of EVM execution transcripts and storage operations based on the Verkle tree storage

But how to deal with SSTORE?

- A SSTORE in a block modifies the polynomial and commitment of every node along the root-leaf path
- Because the root commitment is also updated, old committed values in the "partial witness tree" seem to expire after a SSTORE

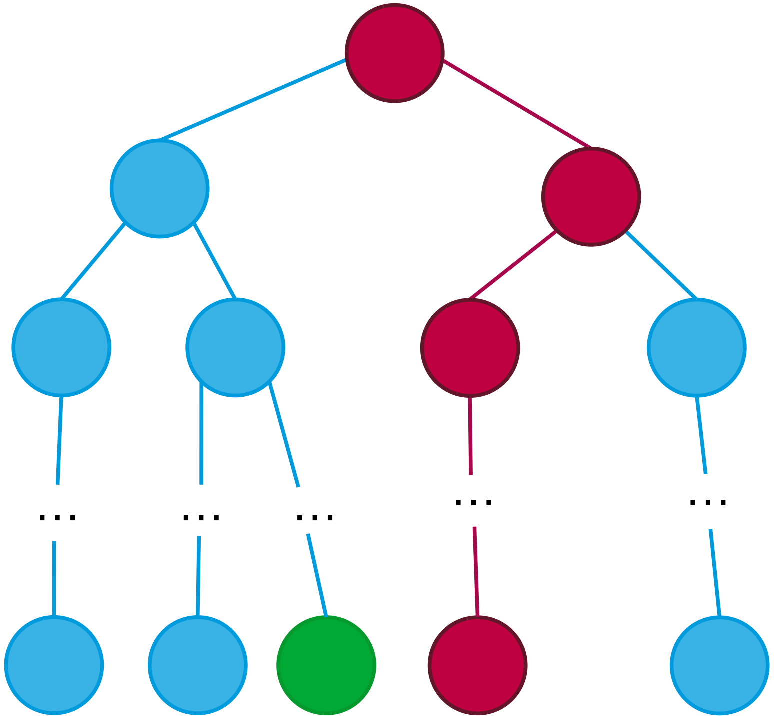
- But, really?

Part of old multiproofs still available



Partial witness tree before SSTORE

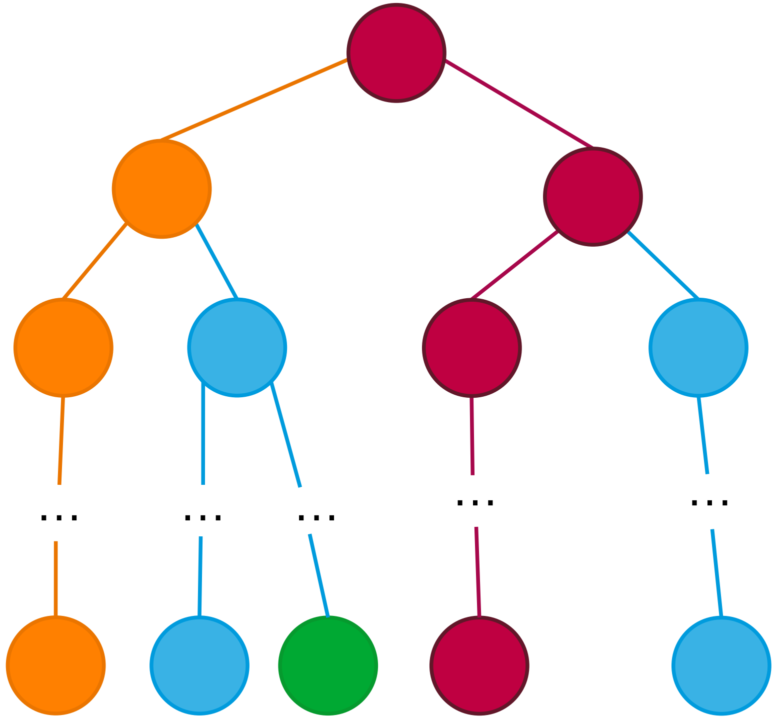
Part of old multiproofs still available



Partial witness tree after SSTORE

- In the multiproof of the green leaf node, all the ancestors other than the root remains the same committed polynomials after a SSTORE at the red leaf node
- This part is still available for proofs after the SSTORE

Slices of multiproof



- Consider a SSTORE at the orange leaf followed by a SSTORE at the red leaf
- The proof of the green leaf can be described as three slices (root, orange one at depth 1, the other nodes)
- A slice is linearly combined into a multiproof and reusable

Optimized redesigning: proofs of SSTORE?

- It seems ultimate to optimize a Verkle proof that makes use of multiproofs and slices
- Bottleneck: a Verkle proof of SSTORE must involve $O(d)$ polynomial commitments, indicating all the nodes along a root-leaf path
- **Our goal: to redesign the Eth storage system to reach both client statelessness and $o(d)$ proof size and verification time for a SSTORE (may be amortized)**
- Idea: to record events and proofs of SSTORE?

Witness of SSTORE

- The block builder can organize all the SSTORE (and other storage update) events in a witness vector/tree/other ADS
- An element of the witness of SSTORE contains the targeted storage slot key and value, as well as its timestamp
- To prove a SLOAD
  - If updated by a SSTORE in the witness, prove its value correctness and its timestamp before this SLOAD with no other SSTORE at the same slot in between
  - If not updated by any SSTORE in the witness, produce a proof from the world state

Frequent updated slots

- Observation: a few storage slots are frequently updated, within one block, or even within one txn
- E.g. counters, nonces and account balances
- Should be experiments to confirm this observation

- Shorter proofs of SSTORE and SLOAD events in building a block, and update in the world state only once per slot per block

Optional: storage slots with small values

- This optimization idea was proposed in a post by Milos
- MPT and the adopted Verkle tree scheme distinguish zero and non-zero storage slots
- What will happen if all storage slots with small values (less than a threshold `U`) are stored separately from the entire tree? In this case, `U` reserved storage sets are enabled to hold all slots with value 0, 1, …, `U−1`
- Unsure if it actually improves performance for any `U`

Thanks for listening!

Questions welcome