Coded Merkle Tree: Solving Data Availability Attacks in Blockchains

Authors and Affiliations: Mingchao Yu, Saeid Sahraei, Songze Li, Salman Avestimehr, Sreeram Kannan, Pramod Viswanath from University of Southern California, Trifecta Blockchain, University of Washington Seattle, University of Illinois at Urbana-Champaign.

Presented by: Ali Hamzehpour

Background

- **Full Nodes**: Full nodes store the entire blockchain, verify all transactions, and provide high security by checking the validity of all blocks and transactions.

- **Light Nodes**: Light nodes verify transactions using Merkle proofs without downloading the entire blockchain, which poses challenges for data availability verification.

- **Merkle Trees**: Merkle trees are used in blockchains to ensure data integrity and allow efficient verification of data.

Data Availability Attack

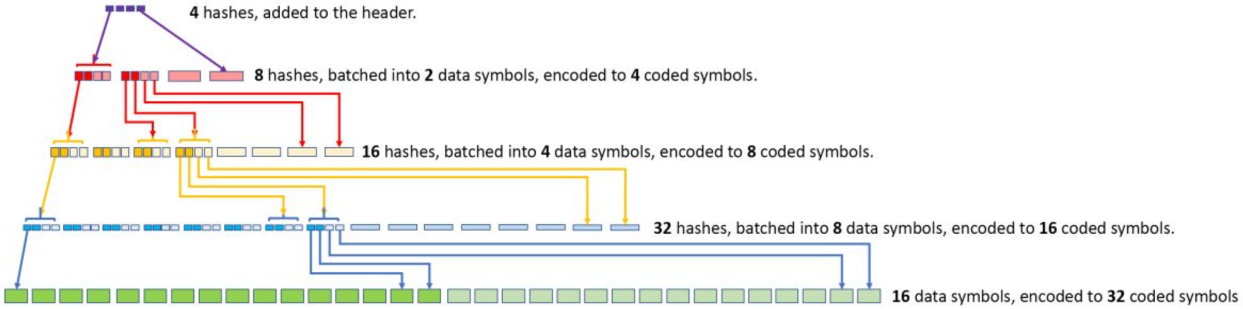
- **Data Availability Attack**: A malicious actor withholds parts of the data block, making it impossible for nodes to verify the block's integrity.
- **Challenge**: Light nodes need an efficient way to ensure data availability without downloading the entire block
- **Simple Solution**:
  - Idea: Sample some random transactions to verify the availability.
  - Problem: The size of hidden part is much smaller than the block size!

Idea: Adding Redundant Data

- **Redundant Data**: Adding redundancy involves creating additional data that can be used to reconstruct the original data if parts are missing or corrupted
- **Erasure Coding**: This technique divides data into multiple pieces and adds extra encoded pieces, allowing recovery even if some pieces are lost.
- **Benefit**: Enhances data availability by allowing nodes to reconstruct the full data from partial data.
- **Previous Approaches**:
    - Used 2D-RS (2-dimensional Reed-Solomon) code for the erasure coding
    - Problem: Time and size complexity

Solution: Coded Merkle Tree

- The structure is like a Classic Merkle Tree but have some additions:
  - Each layer is extended by parity coded data.
  - Each node has q-1 siblings.
  - This procedure continues until we reach t hashes in a layer.
- Benefit: Sampling a transaction will cause sampling more intermediate nodes.



Mechanism

01 Full nodes use Hash-Aware Peeling Decoder Algorithm and create incorrect-coding proofs when needed.

02 Construction of Erasure Code is done with paity checks equation(A form of LDPC)

What Each Node Should Do

- **Producers**:
  - **Data Creation**: Split the original data block into smaller pieces.
  - **Erasure Coding**: Encode the data pieces using sparse erasure codes.
  - **Merkle Tree Construction**: Create the coded Merkle tree and broadcast the root hash.
- **Full Nodes**:
  - **Storage**: Store the entire blockchain, including the coded Merkle tree.
  - **Verification**: Verify the integrity of blocks and the correctness of the CMT construction.
  - **Data Availability**: Provide data to light nodes upon request.
- **Light Nodes**:
  - **Verification**: Verify data availability by downloading a small sample of encoded data.
  - **Proof Generation**: Generate or get the compact proofs to confirm data availability and correctness.

Implementation and Results

- **Library**: The CMT is implemented in a modular library available in Rust and Python
- **Efficiency**: Requires fewer samples and smaller proofs compared to traditional methods, making it more efficient.
- Adding this structure to bitcoin needs minimum changes and no extra bandwidth consumption

	hash commitment size (bytes)	# of samples to gain certain confidence about data availability	incorrect-coding proof size (bytes)	decoding complexity
Uncoded	$O(1)$	$O(b)$	-	-
1D-RS	$O(1)$	$O(1)$	$O(b \log b)$	$O(b^2)$
2D-RS [9]	$O(\sqrt{b})$	$O(1)$	$O(\sqrt{b} \log k)$	$O(b^{1.5})$
SPAR	$O(1)$	$O(1)$	$O(\log b)$	$O(b)$

Thank you for your time 😊