Ali Hamzehpour
SID: 810100129
Telegram ID: @tempali2002

Presentation video link

Coded Merkle tree: Solving data availability attacks in blockchains

From the paper: M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, "Coded Merkle tree: Solving data availability attacks in blockchains," in Financial Cryptography and Data Security (FC), 2020. (link)

Problem Explanation

A light node in a blockchain network doesn't download the entire ledger and block tree; it only verifies some transactions. In contrast, full nodes download the entire blockchain locally and are responsible for validating new blocks. Because a light node doesn't download the block tree, it may be exposed to a type of attack known as a data availability attack. This paper aims to solve this problem with reduced time and space complexity when the light node is connected to a majority of dishonest nodes by introducing an enhanced version of the Merkle tree and an encoding system.

Data Availability Attack

A data availability attack occurs when a malicious node broadcasts the header of an invalid block (a block that contains invalid transactions) but then hides some of the block's data (those invalid transactions). As a result, full nodes can't be certain that the block is invalid. In this situation, a light node might verify a transaction within this block tree even though the block is invalid because the full nodes aren't sure yet and cannot inform the light node not to verify the transaction. To handle this attack, the light node must find a way to ensure that the data is available on its own. It can't rely on other full nodes because we've assumed the majority of connected full nodes are dishonest, and network delays can cause trust issues.

So, the problem the light node should address is:

Data Availability Problem: Upon receiving a hash commitment D, how can a light node efficiently verify that a data block B that satisfies g(B) = D is fully available to the system?

Previous Approaches

The simplest way to check if a block's data is fully available is by randomly sampling parts of it. For example, after getting the block header, the light node requests some data symbols from the block tree, and if it doesn't get a response for any of them, it can suspect a data availability attack. This approach has a problem: a single transaction is a small portion of the block tree, so we need to sample a lot of data to be statistically sure that the data is fully available.

Another idea is to use erasure coding, which extends the block data with redundancy so some parts of the block data can be revealed even when hidden. With this approach, a malicious node must hide more data symbols, allowing the light node to detect data unavailability by sampling less data.

For example, if our data contains c0, c1, c2, and c3, we can add these redundancies:
c4 = c0 + c1,
c5 = c1 + c2,
c6 = c2 + c3,
c7 = c3 + c0.

Now, to hide some of the data, the malicious node must hide at least three data blocks; otherwise, we can reveal all of the data.

In this approach, full nodes should check the block data after receiving it. If there's a contradiction in the redundant part, they should broadcast it as an incorrect-coding proof so other light nodes know the block is invalid.

Previously, a paper used 2D-RS (2-dimensional Reed-Solomon) code for the erasure coding approach, which had some disadvantages regarding time and space complexity. (source) This paper uses a different coding and data structure to solve this problem.

Light-Node-Only Verification idea(S. Cao, S. Kadhe and K. Ramchandran, "CoVer: Collaborative Light-Node-Only Verification and Data Availability for Blockchains,")(source)

Another way to handle data availability attacks is to have light nodes work together, with each one verifying a part of the block tree. Instead of every light node trying to check the entire block on its own, they split up the task. Each light node takes responsibility for a different section of the block tree, so every part gets verified without overwhelming any single node. This teamwork approach helps ensure that all data is checked and makes it harder for attackers to hide any missing or corrupted data. By sharing the workload, the network becomes more secure and efficient, allowing light nodes to maintain the integrity of the blockchain more effectively.

SPAR Components

SPAR (Sparse Fraud Protection) has four main components:

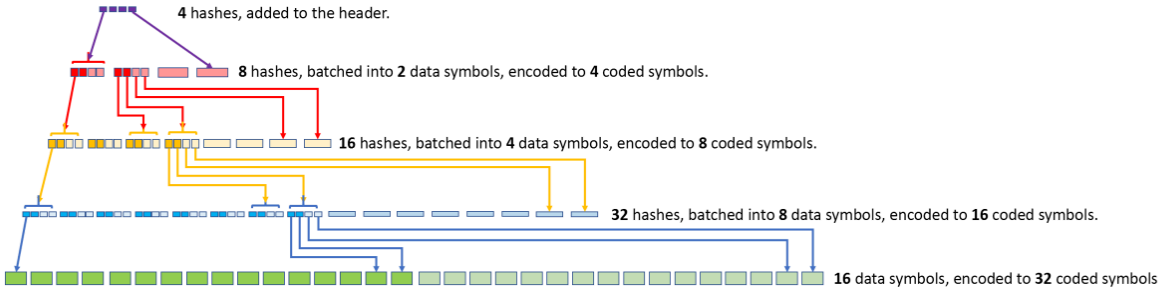
1. Coded Merkle Tree
2. Sampling Mechanism of Light Nodes
3. Hash-Aware Peeling Decoder and Incorrect-Coding Proof
4. Construction of Erasure Code

Coded Merkle Tree

A Coded Merkle Tree is similar to a normal Merkle tree but with some differences:

- Each intermediate node has q children instead of 2, so we need the hashes of q-1 siblings for the inclusion proof of a transaction.
- In each layer of the tree, we add some redundant parity data (called parity symbols), and we call the original data blocks data symbols. We have a coefficient r that shows how much the data size increases in the layer (if r = ½, the data size of the layer will be twice as large).
- This procedure continues until the current layer has only t hashes of data.
- Interleaving Batches: Batches of data are interleaved across multiple layers of the Coded Merkle Tree. This interleaving ensures that any missing data in one layer will still be partially available in another layer, improving the overall robustness against data availability attacks. This technique also helps in spreading out the redundancy, making it harder for an attacker to hide significant portions of data without detection.

The figure below shows a coded Merkle tree with k = 16, r = ½, q = 4, and t = 4:



Sampling Mechanism of Light Nodes

The sampling mechanism of light nodes in SPAR involves the random sampling of base layer coded symbols along with their Merkle proofs to determine the availability of the data. This process is designed to ensure that if the base layer is available, the entire data block is likely

available. The Coded Merkle Tree (CMT) structure allows light nodes to efficiently sample higher layers based on these base layer proofs. In a traditional Merkle tree, a Merkle proof includes all sibling hashes between the leaf and the root, providing one data symbol from each intermediate layer. However, in CMT, each layer has q−1 sibling hashes, enabling light nodes to sample s distinct base layer symbols and automatically sample s distinct symbols from every intermediate layer with high probability.

Mathematically, for the light nodes to check the availability of an intermediate layer, they need to sample about (1−r)s parity symbols from that layer. This process is managed probabilistically: if a parent layer data symbol is sampled, one of its q(1−r) child parity symbols is sampled uniformly at random with probability 1−r. This reduces the need for additional Merkle proofs for these parity symbols. The size of the response to one sampling request is given by:

$$\frac{b}{k} + [y(q-1) + yq(1-r)] \log_{qr} \frac{k}{rt}$$

Which is equal to $O(log\ b)$.

Hash-Aware Peeling Decoder and Incorrect-Coding Proof

Below is the algorithm used for peeling decoder. If a full node finds out that there's something wrong with the block data. It sends an incorrect-coding proof which is like a Merkle proof where other nodes can check the data that is wrongly coded.(so the decoding is done in $O(b)$ and incorrect-coding proof size is $O(log\ b)$)

Algorithm 1: Hash-aware peeling decoding algorithm

1 **Inputs**: the hashes of all n coded symbols and $(1-\alpha)n$ coded symbols;
2 **Initial check**: checks all the degree-0 parity equations (i.e., those whose coded symbols are all known). If any parity equation is failed, report an incorrect-coding proof and exit;
3 **while** *not all the k data symbols are recovered* **do**
4 Find a degree-1 parity equation, which only has one unknown coded symbol;
5 Recover this coded symbol and verify it with its hash. If failed, report an incorrect-coding proof and exit;
6 Check all the associated degree-0 parity equations. If any parity equation is failed, report an incorrect-coding proof and exit.
7 **end**

Construction of Erasure Code

The construction of erasure codes in SPAR uses an (n,k) format, where n is the total number of coded symbols and k is the number of original data symbols. The erasure code relies on an n × (n−k) parity check matrix H, where each column represents a parity equation. The goal is to ensure that any valid codeword C satisfies $CH = 0$. A key aspect of this construction is the stopping set, a subset of rows in H that influences the undecidable ratio α, which is the fraction of symbols that need to be hidden to prevent decoding.

SPAR constructs H using a permutation matrix J, partitioned into n × (n−k) slices, each a c × d. The entries of H are based on whether these slices contain an odd number of 1s. This ensures that H has a maximum row weight of c and a column weight of d. This method probabilistically ensures a stopping ratio of at least α^*, although determining the exact stopping ratio is NP-hard.

What Each node does in this structure

- Block producer (full node): Generates the Coded Merkle Tree (CMT) and broadcasts its root to all nodes, as well as the entire original block to the full nodes and it responds to sample requests from light nodes to ensure data availability verification .
- Light node: Upon receiving a new CMT root, it makes separate, anonymous, and intermittent sampling requests to full nodes claiming block availability. It Broadcasts received samples to all connected full nodes and assumes the block is available if all samples are received. If samples are not fully received within a fixed time, it "pends" the block. Rejects blocks upon receiving incorrect-coding or bad-code proofs, updating the erasure code of the failed layer if needed.
- Other full nodes: Attempt to recover the data block by downloading it from other full nodes and collecting coded symbols forwarded by light nodes. They use a hash-aware peeling decoder to decode the tree from top to bottom. They also Rebroadcast valid samples and send proofs and rejection notices for detected incorrect or bad codes. Once fully decoded and verified, they declare block availability and respond to sample requests from light nodes.

Performance Analysis

After explaining the structure and how nodes behave in it, the paper explains the performance analysis of this mechanism. They first prove this theorem:

Theorem 1. *In SPAR, a block producer cannot cause the soundness and agreement to fail with a probability lower than*

$$P_f \leq \max\left\{(1-\alpha_{\min})^s, \quad 2^{\max_i\left[H(\alpha_i)n_i - ms\log\frac{1}{1-\alpha_i}\right]}\right\}.$$

Here n_i and α_i are the number of symbols and undecodable ratio on the ith layer of CMT, $\alpha_{\min} \triangleq \min_i \alpha_i$, and s is the number of coded symbols each light node samples from the base layer.

Where the soundness and agreement definitions are as below:

- soundness: If a light node has determined that a data block is fully available, then at least one honest full node will be able to fully recover this data block within a constant delay.
- agreement: If a light node has determined that a data block is fully available, then all the other light nodes in the system will determine that the data block is fully available within a constant delay.

They also explain the complexity of time and size of operations which the result is the figure below:

	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	$O(1)$	$O(1)$	$O(\log b)$	$O(b)$

They also indicate that how choice of parameters can change α^* but SPAR finally settles at a good code.

Final Parts

In the end, it's said that a library in Rust and Python was written for this mechanism and it was tested in a test blockchain and the results were much better than previous approaches. They also state that adding this structure to bitcoin needs minimum changes and no extra bandwidth consumption. An honest block producer only needs to broadcast the original data block as usual and attach the CMT root in the block header. This is sufficient for other full nodes to reproduce the CMT and offer sampling services for light nodes.

Now we explain two more ideas which try to improve coded Merkle trees:

Graph Coded Merkle Tree idea(D. Mitra, L. Tauz and L. Dolecek, "Graph Coded Merkle Tree: Mitigating Data Availability Attacks in Blockchain Systems Using Informed Design of Polar Factor Graphs,")(source)

Traditional coded Merkle trees, like those using LDPC and 2-D Reed Solomon (2D-RS) codes, struggle with large blockchain blocks. LDPC codes, for instance, have trouble maintaining reliable performance with long codes, making them less practical for sizable blocks. The Graph Coded Merkle Tree (GCMT) addresses this by using polar encoding graphs, known for their efficient encoding and decoding. GCMT employs a special algorithm called Sampling Efficient Freezing to design the encoding graph and another to prune it, ensuring efficiency and scalability. This method improves data availability detection, reduces communication costs, and decreases the size of fraud proofs compared to LDPC and 2D-RS codes. It also maintains reliable performance with large code lengths, similar to 2D-RS codes, and keeps decoding complexity manageable, making it a practical solution for large blockchain applications.

Asynchronous Coded Data idea(Sheng, P.Y.; Xue, B.W.; Kannan, S.; Viswanath, P. ACeD: Scalable data availability oracle. In Proceedings of the International Conference on Financial Cryptography and Data Security)(source)

Asynchronous Coded Data (ACeD) ensures data availability in blockchain networks efficiently and scalably. Instead of making every node store or verify the entire data block, ACeD splits the data into smaller pieces and encodes it so that each of the N nodes only needs to receive O(1/N) of the data. This distribution guarantees that the data can be reconstructed even if some parts are missing or corrupted. By combining this method with Merkle tree commitments, ACeD ensures that the data is both verifiable and reconstructable in a tamper-proof way, tackling issues of scalability and robustness against worst-case corruption scenarios.

NOTES

Citation: M. Yu, S. Sahraei, S. Li, S. Avestimehr, S. Kannan, and P. Viswanath, “Coded merkle tree: Solving data availability attacks in blockchains,” in Financial Cryptography and Data Security (FC), 2020.
URL: arXiv:1910.01247v2
Keywords: Blockchain, Data Availability, Merkle Tree, Erasure Codes, Light Nodes, Fraud Proofs
General Subject: Blockchain Technology
Specific Subject: Data Availability and Security in Blockchain Systems

Hypothesis: Coded Merkle Tree (CMT) can provide a scalable and secure solution to data availability attacks in blockchains, ensuring light nodes can verify data availability with minimal resources.

Methodology: The authors propose the CMT, constructed using sparse erasure codes applied iteratively to layers of a Merkle tree. They implement a modular library in Rust and Python, and integrate it into the Parity Bitcoin client to demonstrate efficacy.

Results: The implementation of CMT significantly improves data availability verification efficiency, with a constant-cost proof size and reduced decoding complexity. The method enables light nodes to ensure data availability by downloading a minimal portion of the block data.

Summary of Key Points:

- Coded Merkle Tree (CMT) uses sparse erasure codes to create a compact proof for data availability.
- Light nodes can verify the availability of data blocks with minimal resource usage.
- The method provides robust protection against data availability attacks, even with a majority of malicious nodes.
- The implementation shows substantial performance improvements over existing methods.

Context: This work builds on and improves previous research on fraud proofs and data availability in blockchains, such as the use of 2D Reed-Solomon codes. It provides a more efficient and secure solution, addressing the critical need for scalable light node operations.

Significance: The CMT solution is highly significant for enhancing blockchain scalability and security, particularly for light nodes. It ensures that these nodes can reliably verify data availability without being compromised by malicious attacks, which is crucial for the widespread adoption of blockchain technology.

Important Figures and/or Tables:

- **Figure 1**: Connection model between full and light nodes (Page 4)
- **Table 1**: Comparison of download costs and decoding complexity for different coding methods (Page 3)

Cited References to Follow Up On:

- Original proposal of fraud proofs and data availability solutions using 2D Reed-Solomon codes.
- New ideas mentioned in the report.