Motivation:

In a previous post, we discussed that we cannot reduce the size of the data availability header (DAH) without increasing the fraud proof size. The conclusion was that the data availability header has to be available, but the light node does not necessarily have to download the DAH.

We will look into addressing two bottlenecks that light nodes experience:

- Bandwidth usage of syncing headers
- Storage of the header

DAHs can be big. If the square size of the original data square is continuously 128 * 128 shares, then the resulting DAH will be 90 * 256 * 2 = 46080 bytes large. In 4 weeks with a 12 second blockime, this would result in 46080×5×60×24×7×4 = 9.3 GB of headers being downloaded and stored for a light node. This doesn't count the Tendermintheader itself, which can be significant in size.

Make it available

Instead of downloading the DAH, we will just make it available. We do it in the following construction:

Today, we create a Merkle tree over the DAH, resulting in the DataRoot.

In the new construction, we will first erasure code the DAH in one dimension. We will build the binary Merkle tree on the new points generated.

[

908×333 12.6 KB

](https://forum.celestia.org/uploads/default/original/2X/5/5a51ce368414d662863754393190c75c1cbfc307.png)

The new construction will look like the diagram above. You could build a tree over the [Rows|Cols|ECs] as is, but that would require more samples. For once, the light nodes would not sample the parity data naturally as the parity data does not commit to anything. Additionally, the sampling protocol only goes over the row roots, which means that only ¼ of the DAH would be sampleable and not the whole range, making it very easy for an attacker to make ½+1 unavailable.

For that reason we merge the different parts of the DAH togther so that every sample contains parts of the whole DAH. In this construction one leaf of the binary merkle tree has one row root, one column root and 2 parity roots.

When we randomly sample a share in the square we will sample it over a row root, which means that we are sampling the DAH and the square at the same time. The current 16-sample default will give us a 1 - 16^(0.5) = 0.99998 probability of the DAH being available.

How does it change the security model?

First, if the erasure coding of the DAH is wrong, we can provide a fraud-proof that it is the wrong encoding by providing half of the nodes from the DAH. This fraud-proof trumps all other fraud proofs, so if there is malicious behavior in other places, this is the only one that has to be provided.

If a data-withholding attack occurs, the full nodes have to reconstruct the square. If they need to construct a BEFP (Bad Encoding Fraud Proof), they might also have to reconstruct the DAH. Therefore, there should be a protocol similar to that for block construction for the DAH. Most likely, they do not have to fetch these explicitly but can reconstruct the DAH while fetching samples for the square.

Now let's assume that we have the case that was described inthis post , which is, in short, a scenario where 2 rows and columns are not sampleable. For simplicity, let's take the above diagram and say that COL2/COL3 and ROW2/ROW3 cannot be sampled. Now, there was some bad erasure coding beneath ROW3. As ROW3 cannot be sampled, we cannot guarantee a fraud proof size of sqrt(n) of the square as we would need a bigger preimage to compare against.

Now, we can use the sampled DAH to reconstruct it so that the worst-case fraud-proof size turns into DAH + 1 row. Previously, the majority of light nodes would be fooled if a constant number of rows or columns were hidden. With the DAH being erasure coded, the majority will not be fooled, and we can guarantee the fraud-proof size upper bound.

Square size is now an unknown

Some logic in the light node is dependent on the square size.

Without the DAH, how could we derive the square's size as a light node? We can get the square's size with the first sample.

The first sample gives us a proof to a node in the DAH and a share in the square. We can derive the size from both. We can assume this is correct and reject anything that does not follow that size.

Next, we must figure out how to sample without having the size. In the next version ofdata availability sampling, the request will happen against an index. This means that a square size will be required.

A light node should sample randomly against the maximum square size. To maintain the randomness property, a full node can provide samples modulo the requested index. One might observe a collision in the samples, where two samples in the big square hit the same sample in the smaller one. It is a requirement that the samples are non-repeating. After the first sample is received, the light node knows the square size, which means it can adjust the sampling index when collisions happen.

If the light node does not sample before a query occurs, any proof to the DataRoot will reveal the size of the square, and it can be initialized that way as well.

Another option would be to commit the square size inside the tendermint header. That would reduce the complexity of the methods mentioned above and would be a less breaking change to the celestia-node software and instead a consensus-breaking change celestia-app.

Requirments Equilibrium

In this table, we show how much light nodes have to download in bytes given both constructions. This includes the samples themselves, the DAH, and the inclusion proofs to the samples but excludes the tendermint header:

[

image

736×370 13.4 KB

](https://forum.celestia.org/uploads/default/original/2X/a/a008ba22289604a496c445e6101af856246090bc.png)

The table shows that after an average EDS of 32*32, switching to the new construction for light nodes would be beneficial.

Shoutout

Thank you, @walldiss, for the discussions that led to this construction and@adlerjohn, for the review. Further reading is Coded Merkle Tree: Solving Data Availability Attacks in Blockchainswhich follows a similar idea.