Motivation

The Celestia protocol, which is based on the Lazy Ledger whitepaper and Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities always had in mind to create trust minimized light clients. The protocol aims to achieve this by using fraud proofs and having a one honest full-node assumption. At the time of developing the protocol, this approach seemed reasonable, and the implementation of bad encoding fraud proofs (BEFPs) proved the concept to be sound and scalable.

To have full trust minimized light clients, Celestia needs to have a fraud proof for every protocol rule that can be broken. Many of these rules can be fraud proven by the same fraud proof.

Five years have passed since the Lazy Ledger whitepaper, and in the meantime, ZK technology has advanced and matured so much that we should revisit the original decision.

Instead of having to implement a new fraud proof for every rule, Celestia can move to ZK proofs. This will allow us to have a fully trust minimized light client and simplify the protocol design.

Fraud Proof Types

The current specification on fraud proofs outlines 3 possible fraud proofs.

- Bad encoding fraud proofs
- Blob inclusion fraud proofs
- State fraud proofs

The next subsections will outline which fraud is captured by each fraud proof type.

Bad encoding fraud proofs

- Bad erasure encoding of a row or column
- Mismatch of a row root of shares proven against the row root and the same shares proven against a column root.
- Bad namespace merkle tree generation
- Wrong namespaces in the shares

Blob Inclusion fraud proofs

- The length of the blob specified in the PayForBlob transaction does not match the length specified in the first share of the blob
- The actual length of the blob does not match the length specified
- Index of the blob does not match the index specified
- Blob length would go outside of the max square size
- Blob is not included even though the PFB exists
- Bad encoding of PFB metadata in PFB transactions

State fraud proofs

- Wrong intermediate state root after a certain amount of transactions
- Wrong compact share encoding
- Wrong cosmos-sdk transaction encoding

Possible fraud not captured by those types of fraud proofs

Note: These types of fraud were addressed in previous specification drafts but shelved in favor of shipping first and completing non-BE fraud proofs later. Specifically since partial nodes would be cheap to run as the state machine is so simple.

- Wrong amount of padding shares using the blob share commitment rules

- Wrong transaction index incrementation
- Too many or too few intermediate state roots
- Wrong use of a reserved namespace, that might be reserved but not should not be occupied
- 2 PayForBlob transactions pointing to the same blob
- Wrong amount of fees paid by the user to the protocol

The Move

Moving from fraud proofs to ZK proofs, every fraud must be either eliminated by removing that feature or have its correctness proven with a ZK proof. The scope of a fraud proof can be reduced by proving the correctness of a subset that a fraud proof is supposed to capture using a ZK proof. An intermediate step is to have most of the fraud captured by a ZK proof, with only a subset not captured, thus limiting the scope of the fraud proof step by step.

Compact fraud proofs exist, so light clients don`t have to download the whole block to verify that fraud has happened. This requires Celestia to generate metadata to be able to convey enough information to generate a compact fraud proof. If the intention of moving to ZK proofs is accepted, it will allow us to remove the metadata and simplify Celestia.

The removal of the metadata includes but is not limited to:

- Index in the PFB
- Intermediate State Roots Namespace
- Index of Transaction (Still up to debate)
- The PFB (share) commitment

The pfb commitment existed for compact fraud proofs, so a full node can create a compact inclusion proof of a blob without needing the whole blob when fraud-proving a malicious PFB transaction. That is not needed anymore. This means we can get rid of the concept of sub-tree roots and PFB commitments. This opens up the design space of the commitment scheme blobs could use. The commitment could also go over the bolb data itself and not the shares, making it share encoding independent. That way, Celestia could ingest data into namespaces while keeping the commitment creation deterministic before the transaction submission.

Finally, the blob share commitment rules could be removed, as we no longer need the concepts of subtree roots, which completely removes all intra-blob padding.

Backwards compatibility and security considerations

Moving from non-existent fraud proofs for specific validity rules to be validated by ZK proofs does not break the client. Moving erasure encoding validity from fraud proofs to ZK proofs will break the client and light node operators that rely on the fraud proofs. This particular migration should be done with caution.

The removal of metadata should not have an impact on any current users of Celestia. Breaking consensus for this is worth it as opposed to breaking consensus otherwise to enable single round fraud proofs or being able to fraud proof wrong fee payments in a compact way.

The security guarantees for light node operators do not decrease if we move from non-existent fraud proofs to ZK proofs. Depending on the system used to generate the ZK proofs, additional trust assumptions might be added like a tested setup. How a ZK proof is generated should be discussed separately for each validity rule that we are discussing.

Conclusion

In conclusion, rewriting the app in Rust and running it throughSP1 or Risc0 seems easier than creating and supporting all fraud proof types.