Introduction to Blobstream rollups

Blobstream is the first data availability solution for EVM chains that securely scales with the number of users. It allows rollups to post their data on Celestia while proving their availability in the rollup settlement contract.

This document will outline a few ways to build optimistic or zk-rollups that post their data to Celestia and use Blobstream to prove that data's availability.

Concepts

Share commitment

Theshare commitment is a commitment over the data contained in theMsgPayForBlobs transaction . This commitment allowsproving that the corresponding data exists on Celestia efficiently.

Share commitment: Proof details

To prove that the data corresponding to a share commitment was posted to Celestia using Blobstream, the following proofs need to be verified:

1. share inclusion proof to the share commitment
2. : meaning creating two merkle proofs:1. share merkle proof up to thesubtree root
3.
    1. corresponding to that share
4.
    1. subtree root merkle proof to theshare commitment
5. share commitment inclusion proof to the data root tuple root
6. : meaning four merkle proofs:1. subtree roots merkle proofs to the share commitment
7.
    1. : to make sure the subtree roots are valid
8.
    1. subtree roots merkle proofs up to the row roots
9.
    1. : to prove that the subtree roots belong to a set of rows in the Celestia block
10.
    1. row roots proofs to the data root
11.
    1. : to prove that those rows belong to the Celestia Block
12.
    1. data root tuple proof to the data root tuple
13.
    1. : to prove that the Celestia block referenced by its height and data root, was committed to by Blobstream.

More details on the share commitment inclusion proof can be found in thecommitment scheme docs and also thedata square layout .

If all of these proofs are valid, then you successfully managed to prove that the data corresponding to that share commitment has been posted to Celestia.

NOTE

Generating/verifying share commitment proofs is still not supported. It still needs tooling to generate the proofs on the node side, and verifying them on the Solidity side which will be built in the upcoming months.

Share commitment: Compact proofs

There is a way to have compact proofs, when using share commitments, unlike the ones defined above; that allow less costly inclusion proofs. These require the ability to parse the protobuf encoded PFBs.

In fact, if the rollup project has a way to parse the protobuf encoded PFB, either in a smart contract or a zk-circuit, they will be able to create compact proofs of the rollup data.

These proofs will work as follows:

- Parsing the PFB and taking out the share commitment
- Comparing the PFB commitment to the one saved in the rollup contract
- Proving inclusion of the PFB to the data root tuple root. This will be a compact proof since we will only be proving two shares regardless of the size of the rollup data.

More details on compact proofs can be found inADR-011 .

Share commitment: Pros

The pros of referencing rollup data using a share commitment:

- Using the same commitment that exists on the PFB, without having to find another way of referencing the rollup data.
- If the team has access to protobuf parsing, it allows for compact proof, but the parsing costs need to be investigated.

Share commitment: Cons

- Large/expensive proofs in the case of having no way to parse the protobuf PFB encoding.
- In the optimistic rollups construction, defined below, this requires waiting for the Celestia block to be committed to by Blobstream before saving updating the settlement contract. This might require waiting for a few hours, depending on the batches size on each chain, to finally submit the rollup update.

Given these limitations, an alternative design will be discussed in the next section.

Sequence of spans

An alternative way of referencing rollup data in the rollup settlement contract is using a sequence of spans.

A sequence of spans is a data pointer that allows pointing to the rollup data inside a Celestia square using its location inside the square. It can be defined using the following information:

- height
- : The height of the Celestia block containing the rollup data.
- startIndex
- : The index of the first share containing the rollup data.
- dataLen
- : The number of shares containing the rollup data.

ThestartIndex and thedataLen can be queried from Celestia after the corresponding transaction gets included in a block and committed to the chain. An example of how to query them can be found in theverify command. TheTxShareRange returns the start and end share of the data referenced by a transaction hash.

NOTE

If the rollup data is submitted in multiple blocks, the above sequence of spans can be generalized to include multiple blocks. For simplicity, we will stick with the data only submitted to a single Celestia block.

Sequence of spans: Proof details

Using sequence of spans is different from using the share commitment because we're referencing a location in the square, and not actual data commitment. So, the proof types and their generation are different.

Sequence of spans: Proving unavailable data

By construction, if the sequence of spans refers to a certain location in the square, that location is the data. This location can be in the reserved namespaces, the parity bytes, etc. What matters is that it's part of the square. So to prove that the sequence of spans is invalid, i.e., refers to data that is not available on Celestia, it is necessary and sufficient to show that the sequence of spans doesn't belong to the Celestia block, i.e., the span is out of bounds.

We could create this proof via generating a binaryMerkle proof of any row/column to the Celestia data root. This proof will provide thetotal which is the number of rows/columns in the extended data square. This can be used to calculate the square size. ThecomputeSquareSizeFromRowProof method in theDAVerifier library allows calculating the square size from a row proof or a share proof.

Then, we will use that information to check if the provided share index, in the header, is out of the square size bounds. In order words, we will check if thestartIndex and thestartIndex + dataLen are included in the range[0, 4*square_size] .

NOTE

The square size is the number of rows of the original square. For the data root, we will use a binary Merkle proof to prove its inclusion in a data root tuple root that was committed to by the Blobstream smart contract. More on this in thedata root inclusion proofs section .

Sequence of spans: Proving inclusion of some data

The difference between using a share commitment and a sequence of spans is that when using a share commitment, an extra merkle proof is needed to prove inclusion of the share to the share commitment. However, in the case of a sequence of spans, only the usual inclusion proof of a share to the data root tuple root is needed. The inclusion of the share to the sequence of spans is gotten using the same proof.

In fact, proving that a share is part of the sequence of spans, i.e., part of the rollup data is done as follows:

1. Prove that the data root tuple is committed to by the Blobstream smart contract:
2. To prove the data root is committed to by the Blobstream smart contract, we will need to provide a Merkle proof of the data root tuple to a data root tuple root. This can be created using thedata_root_inclusion_proof
3. query. More on this can be found in thedata root inclusion proofs documentation
4. .
5. Verify inclusion proof of the data to Celestia data root:
6. To prove that the data is part of the data root, we will need to provide two proofs: a namespace Merkle proof of the data to a row root. This could be done via proving the shares that contain the data to the row root using a namespace Merkle proof. And, a binary Merkle proof of the row root to the data root.
7. These proofs can be generated using theProveShares
8. query.
9. More details on these proofs can be found in the transaction inclusion proofdocumentation
10. .
11. Prove that the data is in the sequence spans:
12. To prove that the data is part of the rollup sequence of spans, we take the authenticated share proofs in step (2) and use the shares begin/end key to define the shares' positions in the row.
13. Then, we use the row proof to get the row index in the extended Celestia square and get the index of the share in row major order:
14. solidity
15. uint256
16. shareIndexInRow
17. =
18. shareProof.shareProofs[
19. 0
20.].beginKey;
21. uint256
22. shareIndexInRowMajorOrder
23. =
24. shareIndexInRow
25. +
26. shareProof.rowProofs[
27. 0
28.].numLeaves
29. *
30. shareProof.rowProofs[
31. 0
32.].key;
33. uint256
34. shareIndexInRow
35. =
36. shareProof.shareProofs[
37. 0
38.].beginKey;
39. uint256
40. shareIndexInRowMajorOrder
41. =
42. shareIndexInRow
43. +
44. shareProof.rowProofs[
45. 0
46.].numLeaves
47. *
48. shareProof.rowProofs[
49. 0
50.].key;

Finally, we can compare the computed index with the sequence of spans, and be sure that the data/shares is part of the rollup data.

Sequence of spans: Pros

- Using a sequence of spans instead of the share commitment allows for simpler proofs

Sequence of spans: Cons

None

Optimistic rollups

One type of rollups that can be built with Blobstream is optimistic rollups. An optimistic rollup is a rollup that commits optimistically to a set of blocks, and allows the other parties to verify that the blocks are valid, and if they're not, they can create fraud proofs to signal that.

Celestia allows optimistic rollups to post their data on its DA layer, and to prove that the data is available using Blobstream.

To build an optimistic rollup that uses Celestia as a DA layer, the following constructions can be inspired by.

Optimistic rollups that use a sequence of spans

Optimistic rollups can post their data in Celestia, then in the rollup settlement contract, they can reference optimistically that data using a sequence of spans . Then, rollup full nodes can verify if that data is valid. If not, they can trigger a fraud proof.

When using a sequence of spans, triggering the data availability fraud proofs, which are different from the state transitions fraud proofs (left for the rollup to define), goes back to the following cases:

- Proving that the rollup data is unavailable: this goes back to proving that the sequence of spans is out of bounds
- .
- Proving an invalid state transition: this goes back to:* Proving that the rollup data is available and is part of the sequence of spans
-
    - .
-
    - Parsing that data and verifying the invalid state transition: this is the rollup logic fraud proofs, and it's left to the rollup to define.

Optimistic rollups that use a sequence of spans: Pros

- Not needing to verify anything at the moment of submitting the commitments to the rollup settlement contracts
- The fraud proofs are simple and can be reduced to a single share: if, for example, a single transaction in the rollup data that was posted to Celestia is faulty, only the shares containing that transaction, which can be as minimal as a single share, need to be proven on chain and verified.

Optimistic rollups that use a sequence of spans: Cons

None

Optimistic rollups that use a sequence of spans: Example

An example optimistic rollup that uses sequence of spans to reference its data can be found in the RollupInclusionProofs . It portrays the different possible data availability proofs, constructs them and shows how to verify them.

Also, more details on querying these kinds of proofs can be found in the proof queries documentation.

Optimistic rollups that use share commitments

Another way to build a rollup is to replace the sequence of spans with a height and a share commitment. Then, users/rollup full nodes will be able to query that data and validate it. If the rollup data is not valid, they can create a fraud proof.

The first difference between the sequence of spans construction and the share commitment construction is having to verify that the provided share commitment is part of the Celestia block, referenced by its height in the moment of submitting the rollup commitments to the settlement contract. This is necessary to make sure that the commitment is part of Celestia. Otherwise, rollup sequencers can commit to random share commitments and there won't be a way to prove they're invalid.

The second difference is the proof types. In the case of a fraud proof, the proofs outlined in the proofs details of share commitment section would need to be verified to be sure that the share containing the invalid state transition is part of the rollup data. Alternatively, the rollup settlement contract would need to have a library to parse protobuf encoded PFBs, as explained in the compact proofs of share commitment section, to have less expensive proofs. The cost of parsing the protobuf is not included in this analysis and needs to be investigated separately.

Optimistic rollups that use share commitments: Pros

- Using the same share commitment as the one saved in Celestia which gives access to existing tooling

Optimistic rollups that use share commitments: Cons

- The proofs are expensive in the base case. And if the settlement contract is able to parse the PFBs, thorough investigations of the cost of that would need to be done.

Zk-rollups

Zk-rollups, aka validity rollups, can also use Celestia as a DA and Blobstream to verify that the data was posted. However, the submission process is different from the above constructions, since there are no fraud proofs, and everything should be verified when submitting the commitment to the settlement contract.

Similar to the optimistic case, the rollup settlement contract can reference the rollup data using either the sequence of spans approach or the share commitments. We will discuss both in this section.

Zk-rollups that use sequence of spans

When submitting the commitments to the rollup settlement contract, this latter will need to verify the following:

1. Zk-proof of the state transitions, which is left for the rollup to define.
2. Verify that the sequence of spans isvalid
3. , i.e., is part of the Celestia block referenced by its height, as described in theproof details
4. section.
5. Zk-proof of the rollup data to the data root. The verification process of this should accept a commitment as input so that the settlement contract makes sure it's the correct value that's being saved. The commitment can be the data root and the sequence of spans. And, when the rollup data is proven inside the circuit to the data root, the used data root is asserted to be the input one. Similarly, the data's location is asserted to be the same as the input sequence of spans. These arguments are the ones used in the sequence of spans verification in (2).

Once these are valid, the settlement contract can be sure that the rollup data was posted to Celestia, and the sequence of spans references it correctly.

Zk-rollups that use sequence of spans: Pros

- The inclusion proof inside the zk-circuit is a simple proof that uses traditional merkle tree. In the case of using share commitment, as will be explained below, additional libraries that can be expensive to prove are required.

Zk-rollups that use sequence of spans: Cons

None

Zk-rollups that use share commitments

To use share commitments to reference rollup data in the zk-rollup settlement contract, the zk-circuits need to be able to deserialize protobuf encoded messages. Alternatively, more involved merkle proofs will need to be verified.

Protobuf deserialization inside a zk-circuit

One way of using the share commitment to reference the rollup data in zk-rollups is via using a protobuf deserialization library inside the zk-circuit. And the verification would proceed as follows:

1. Zk-proof of the state transitions, which is left to the rollup team to define.
2. Verify that the share commitment is valid using the proofs laid out in theproof details of share commitment
3. section.

4. The zk-proof verifier would take as argument the data root and the share commitment. Then, inside the circuit, the protobuf encoded PFB transaction will be deserialized and then verify the following:

5. The deserialized share commitment is the same as the one provided as input

6. The circuit will prove the inclusion of the PFB to the data root, then assert that the data root is the same as the one provided as input.

If the above conditions are valid, the rollup settlement contract can be sure that the rollup data was posted to Celestia and is correctly referenced.

Zk-rollups that use share commitments: Pros

None

Zk-rollups that use share commitments: Cons

- This approach requires having access to a protobuf decoder inside a zk-circuit which is not straightforward to have. Also, the relative costs will need to be investigated.

Heavy merkle proofs usage

Similar toProtobuf deserialization inside a zk-circuit , the zk-circuit will proceed to the verification of the availability of the data. The difference is that instead of parsing the encoded protobuf, the proofs defined under theshare commitment proof details section will need to be verified inside the zk-circuit as follows:

1. Zk-proof of the state transitions, which is left to the rollup team to define.
2. Verify that the share commitment is valid using the proofs laid out in theshare commitment proof details
3. section.

4. The zk-proof verifier would take as argument the data root and the share commitment. Then, inside the circuit:

5. It will verify that the input share commitment corresponds to the rollup data.

6. Verify that the input data root commits to that share commitment. Check theshare commitment proof details
7. for more details

Once these proofs are valid, the rollup settlement contract can be sure that the rollup data was posted to Celestia and is correctly referenced.

heavy merkle proofs usage: Pros

None

heavy merkle proofs usage: Cons

- More heavy usage of merkle proofs inside and outside the zk-circuit.

Conclusion

Given the above details, using the sequence of spans is the better solution in the general case as explained in theoptimistic rollups that uses a sequence of spans andzk-rollups that use sequence of spans sections. The proof sizes are small and allow for greater flexibility. However, if the rollup team has different requirements, then the other designs can be explored. [][Edit this page on GitHub] Last updated: Previous page Sovereign SDK Next page Dymension []