This post was co-authored with Nima Vaziri (@NimaVaziri). Special thanks to John Adler (@adlerjohn), Mustafa Al-Bassam (@musalbas) and Ismail Khoffi (@ismail) for their review.

HackMD mirror: Rollups on Celestia - HackMD

This document describes what rollups are and how they are used to ensure the security of applications on Celestia.

Security and Validity

Each application hosted on Celestia is associated with a state

which stands for the information remembered by the application and generated as a result of previous inputs, i.e transactions. These applications also come with state transition functions f

which are used to evolve the associated states. Given an initial state S_0

and a valid

transaction tx

, f

outputs the updated state S_1 = f(S_0,tx)

reflecting the new information included in tx

. A state is said to be valid

if it is the output of the state transition function applied on a valid initial state and a valid transaction. Genesis state of each application, i.e the state at the beginning of time, is assumed to be valid. Validity of the transactions is determined by the logic and the current state of the application.

For instance, given an initial state of account balances (A \leftarrow 2, B \leftarrow 1)

, the transaction paying 3

dollars from A

to B

would be invalid whereas a transaction paying 1

dollar from B

to A

would be valid. Thus, given the initial valid state (A \leftarrow 2, B \leftarrow 1)

and a transaction paying 1

dollar from B

to A

, the only valid next state is (A \leftarrow 3, B \leftarrow 0)

. In this example, a state with negative balances is always invalid regardless of any preceding transaction.

We say that a state S

is finalized in the view of an application user at time t

if the user started to accept S

as the current state of the application at time t

. In this context, security of the application states consists of two components:

1. Liveness:

New states reflecting valid transactions sent by honest application users are finalized in the view of all honest users within bounded time.

1. Safety:

Every state finalized in the view of an honest application user is valid and the honest application users agree on the same sequence of finalized states.

Rollups

In Celestia, applications live on rollups

that use the blockchain, hereafter called the parent chain

as a consensus and data availability layer. Rollups are side chains maintained by rollup full nodes

and rollup light clients

, collectively referred to as rollup nodes

unless stated otherwise. Rollup full nodes process the transactions received from the application users, update the application state and produce new rollup blocks which contain a commitment to the new state, e.g a state root, along with the transactions associated with the state transition. If there is a designated rollup full node responsible for block production, it is called the aggregator

. Aggregators publish rollup blocks including the state roots and the transactions as rollup-specific messages

on the parent chain. For this purpose, they send rollup-specific messages to consensus nodes

which order and store them within the parent chain. Thus, the parent chain provides consensus on the sequence of rollup-specific messages and ensures their availability.

Rollup light clients do not process the transactions received from the application users, thus cannot calculate the latest state root or verify its validity themselves. They learn about the latest state root from the parent chain and ask rollup full nodes for the state elements. Rollup full nodes reply with the queried state elements and the Merkle proofs showing that the latest state root published on the parent chain commits to these elements. Then, rollup light clients check these proofs using the state roots obtained from the parent chain.

Rollup nodes use the ordering on the rollup blocks provided by the parent chain to determine the canonical tip of the rollup. They finalize a rollup block if it is the first valid block at its height within the rollup to be published on the parent chain. As rollup full nodes process transactions, upon observing a rollup block with a new state root, they can verify the state committed by that root. Thus, rollup full nodes can finalize the first valid rollup block at its height as soon as it is published on the parent chain (Figure 2-(i)). However, this is not the case for light clients, which rely on validity proofs, fraud proofs or dispute resolution protocols to determine the validity of the rollup blocks in their view.

Rollups on Ethereum

Rollups deployed on Ethereum are monitored by EVM based smart contracts which keep track of the rollup blocks posted on the Ethereum parent chain. For their blocks to be published, rollup aggregators are required to lock funds on the contract in addition to paying transaction fees to Ethereum miners. Locked funds are released once the posted blocks are accepted

by the contract, i.e once they are finalized in the view of the consensus nodes which manage these contracts. Rollup light clients finalize blocks when they are accepted by the contract.

ZK Rollups (Validity Proofs)

[

figure1

794×509 12.5 KB

](https://forum.celestia.org/uploads/default/original/1X/168bced1b8183d0e3e319f7e3cf89c43e9b7e929.png)

Figure 1: Consensus in a ZK rollup using validity proofs. Invalid rollup blocks are shown with red edges and those published on the parent chain are shown with double arrows. Correct validity proofs are represented by the certificates. Rollup blocks accepted and rejected by the on-chain contract are shaded in green and red respectively. (If there is no on-chain contract, blocks finalized in the view of the rollup nodes are shaded in green and those rejected by the rollup nodes are shaded in red.) Contract accepts b_{j+1}

and b_{j+4}

as they are the first valid blocks to be published on the parent chain, their validity attested by the attached proofs. They are then finalized in the view of both rollup full nodes and light clients. Since b_{j+3}

is invalid and does not come with a correct validity proof, it is rejected by the contract. As b_{j+2}

is not the first valid block at its height and did not come with an attached validity proof, it is also rejected. (If there is no on-chain contract, it is directly the rollup nodes that reject b_{j+2}

and b_{j+3}

upon seeing them published without correct validity proofs. In this case, b_{j+1}

and b_{j+4}

are finalized in the view of all rollup nodes as soon as they are published with correct validity proofs.)

In ZK rollups, the aggregator submits blocks to the contract with validity proofs attesting to the validity of the state roots included in them. Published blocks are accepted by the contract once it verifies the validity proofs (Figure 2). If the proof is incorrect, contract rejects

the block and slashes the aggregator's funds.

Optimistic Rollups

[

figure2i

773×491 9.99 KB

](https://forum.celestia.org/uploads/default/original/1X/434b52008727b5c5e71febca378a4afd7e7f7da8.png)

Figure 2-(i): Consensus in an optimistic rollup using fraud proofs or dispute resolution protocols. Invalid rollup blocks are shown with red edges and those published on the parent chain are shown with double arrows. Rollup blocks that were finalized by the rollup full nodes after being published on the parent chain are shaded in blue. Since b_{j+1}

and b_{j+4}

are the first valid blocks at their respective heights within the rollup, they are finalized by the rollup full nodes. Rollup light clients initially do not know which blocks are valid or invalid, thus they optimistically believe that b_{j+1}

and b_{j+3}

are the first valid blocks at their heights.

[

figure2ii

968×572 16.1 KB

](https://forum.celestia.org/uploads/default/original/1X/147a918549edef08b1796fa47119d164b6ef0dac.png)

Figure 2-(ii): Consensus in an optimistic rollup using fraud proofs or dispute resolution protocols after dispute time window. Rollup blocks accepted and rejected by the contract on the parent chain are shaded in green and red respectively. (If there is no on-chain contract, rollup blocks finalized in the view of the rollup light clients are shaded in green and those rejected by the light clients are shaded in red.) Dispute time window has passed after the blocks b_j

to b_{j+4}

were published on the parent chain. A fraud proof or dispute resolution protocol has notified the on-chain contract that b_{j+3}

is invalid. Thus, b_{j+3}

is rejected by the contract, which is now also convinced that b_{j+1}

and b_{j+4}

are the first valid rollup blocks at their height and should be accepted. In this case, b_{j+2}

is rejected as it is not the first valid block at its height. As a result, rollup light clients have now learned that the correct tip of the rollup is b_{j+4} rather than b_{j+3}

, finalizing b_{j+1}

and b_{j+4}

. (If there is no on-chain contract, rollup light clients finalize b_{j+1}

and b_{j+4}

after the dispute time window passes and they are convinced that these are the first valid blocks at their heights. In this case, b_{j+2}

and b_{j+3}

are simply rejected by the rollup light clients.)

In optimistic rollups, there is a dispute time window that spans from the time an aggregator submits a rollup block to the contract to the time that rollup block becomes finalized

(Figure 2 (ii)). This time delay exists to give verifiers

a chance to check whether the submission was valid, and if it is not to challenge the submission. This challenge protocol has two variations: Fraud Proofs and Interactive Dispute Resolution.

Fraud Proofs

In certain optimistic rollups, full nodes called verifiers

use fraud proofs

to warn the on-chain contract that an aggregator's state root proposed on the parent chain is invalid. Upon receiving a fraud proof for a proposed rollup block, contract verifies the the proof and rejects

the block, slashing the funds of the aggregator.

Interactive Dispute Resolution Protocols

In certain optimistic rollups, if verifiers

observe an invalid block published on the parent chain, they engage in an interactive dispute resolution protocol

with its producer. During the protocol, verifiers and the block producer communicate off-chain until they converge on a single instruction within the challenged block, outcome of which is the source of disagreement. Then, both the verifiers and the challenged block producer create state roots for the states preceding and following that instruction. These state roots and the instruction are provided to the on-chain contract which acts as the arbitreur. If the post state root provided by the challenged block producer is invalid, contract rejects the disputed block and slashes the funds of its producer. Conversely, if the post state root provided by the verifiers is invalid, their funds locked on the contract are slashed.

Rollups on Celestia

In Celestia, the parent chain does not interpret rollup-specific messages and does not host any contract that manages the rollups. It is used by the rollup nodes solely as an ordering and data availability service. Rollup-specific messages published on the parent chain are selected using a fee-market, where consensus nodes responsible for proposing parent chain blocks publish any

message that is paid for.

Recall that on Ethereum, balances of the rollup full nodes and light clients were monitored by smart contracts on the parent chain. In Celestia, these balances are maintained as part of the rollup's application state

.

ZK rollups (Validity Proofs)

In ZK rollups on Celestia, blocks are published on the parent chain with attached validity proofs. Upon observing a validity proof on the parent chain, rollup nodes verify the proof and finalize the associated block (Figure 1). If the proof is incorrect, rollup nodes reject the block and propose a penalty transaction

with the incorrect validity proof acting as evidence of the aggregator's malicious behavior. Funds of the malicious aggregator is slashed by this transaction in the view of the rollup nodes once the state reflecting the transaction is finalized in their view.

Optimistic Rollups

Fraud Proofs

In certain optimistic rollups on Celestia, verifiers could create and broadcast fraud proofs to warn rollup light clients that a published block has an invalid state root. Rollup light clients either receive these fraud proofs via the peer-to-peer network of the rollup or obtain them from the parent chain once they are published. Upon observing a rollup block published on the parent chain, rollup light clients wait for a dispute time window

before they finalize the block (Figure 2-(ii)). If the block is invalid, it is the responsibility of the verifiers to notify the light clients via fraud proofs before this period expires.

After receiving a fraud proof for an invalid block, rollup nodes reject the block and propose a penalty transaction

with the fraud proof acting as evidence of the aggregator's malicious behavior. Funds of the malicious aggregator are slashed by this transaction in the view of the rollup nodes once the state reflecting the transaction is finalized in their view.

Interactive Dispute Resolution Protocols

If verifiers

observe an invalid block published on the parent chain, they can also choose to engage in an interactive dispute resolution protocol

with its producer. Protocol proceeds in the same manner off-chain as in Ethereum until the verifiers and the challenged block producer create state roots for the states preceding and following the disputed instruction. Then, the instruction and the state roots are published on the parent chain for the sake of the rollup light clients which, after observing this data, determine for themselves whether the challenged block is invalid. Just like the case for fraud proofs, light clients wait for a dispute time window

before they accept a published block (Figure 2-(ii)). If the block is invalid, it is the responsibility of the verifiers to challenge the block producer via a dispute resolution protocol and notify the light clients before this period expires.

If the dispute resolution data published on the parent chain indicate that a block is invalid, rollup nodes reject the block and propose a penalty transaction

with the instruction and the state roots acting as evidence of malicious behavior by the block producer. Funds of the block producer are slashed by this transaction in the view of the rollup nodes once the state reflecting the transaction is finalized in their view.

Security of Rollups

States finalized by rollup full nodes and light clients satisfy safety and liveness as defined in Section Security and Validity under the following assumptions:

For liveness;

1. RU-Liveness-1:

Rollup has an honest aggregator or block producer.

1. RU-Liveness-2:

Rollup-specific messages, e.g rollup blocks and associated transactions, are published on the parent chain within a bounded time after they are created. This is necessary for a new rollup state to become finalized in the view of all nodes in a timely manner.

For the safety of optimistic rollups;

1. ORU-Safety-1:

There exist verifiers, which challenge invalid state roots appearing on the parent chain via fraud proofs or dispute resolution protocols. Fraud proofs and the messages exchanged within the dispute resolution protocol are created within bounded time.

1. ORU-Safety-2:

Fraud proofs or messages within dispute resolution protocols are delivered to the rollup light clients over the peer-to-peer network and processed before the dispute time window

expires in their view.

1. ORU-Safety-3:

Fraud proofs or messages related to dispute resolution protocols are published on the parent chain in the view of the rollup light clients and processed before the dispute time window

expires in their view.

For rollup light clients connected to the full nodes via a peer-to-peer network, assumption ORU-Safety-2

is sufficient for the safety of the states finalized in their view. However, some of the light clients might not be part of the same peer-to-peer network as the verifiers. In this case, parent chain acts as an intermediary for transmitting messages to these light clients, implying the necessity of assumption ORU-Safety-3

.

For the safety of ZK rollups;

1. ZKRU-Safety-1:

Validity proofs satisfy completeness: For each valid state, there always exists a validity proof that can be verified in short timeframe.

1. ZKRU-Safety-1:

Validity proofs satisfy soundness: There cannot exist a verifiable validity proof attesting to an invalid state update.

Namespaces and Namespaced Merkle Trees

Figure 3: Namespaced Merkle Tree. Leaves are augmented with the respective namespaces and each internal node contains the minimum and maximum namespaces contained under them.

When rollup-specific messages are published on the parent chain in Celestia, they are stored within NMTs, where each rollup or application is associated with a unique namespace

(Figure 3). NMTs enable rollup nodes, e.g. light clients, to retrieve all rollup-specific messages they query without downloading and parsing the entire parent chain. They also allow the consensus nodes to prove that they have returned the complete set of messages for a namespace included in a parent chain block upon query.

Each leaf node in NMTs is augmented with the namespace of the data held at that node and the leaf nodes are sorted according to their namespaces. Internal nodes keep track of the minimum and maximum namespaces found among the leaf nodes under them with the identifiers minNs

and maxNs

. In this context, the namespaced

hash of each node can be expressed as $\mathrm{minNs}||\mathrm{maxNs}||\mathrm{hash}(x)$

, where x

is either the concatenation of the two namespaced hashes under an internal node or the data stored at a leaf. For leaf nodes, $\mathrm{minNs}=\mathrm{maxNs}$

equals the namespace of the data at that leaf. To ensure that the leafs are correctly sorted, NMTs require maxNs

of a left child to be less than or equal to minNs

of a right child at each internal node.

When rollup nodes query for the namespace of their rollup, consensus nodes return the Merkle proofs $\mathrm{proof}_0,..,\mathrm{proof}_n$

for the leaves with that namespace along with the index e

of the earliest such leaf. By checking the maxNs

of the left sibling nodes found in proof_0

, clients can ensure that there are no leafs with indices smaller than e

that have the rollup’s namespace. Similarly, by checking the maxNs

of the right sibling nodes found in proof_n

, clients can ensure that there are no leafs with indices larger than $e+n$

that have the rollup’s namespace. Thus, the rollup nodes can indeed ensure that they have received every rollup-specific message upon asking for the data with the associated namespace.