Authors: Daniel Ivanov, George Spasov

Abstract

The following is a comparison table examining the data structure, hashing and compression algorithms and the available APIs for retrieving witness

nodes for executing Merkle Inclusions Proofs against a Rollup's state.

The research has been conducted as part of Wisp's ongoing work to enable trustless cross-rollup communication.

Comparison

Rollup

Data Structure

Hashing Algorithm

Compression & State on L1

Data Availability & Proofs API

Comment

Arbitrum

MPT

keccak

The state root is too obfuscated with the compression algorithm. Hard to derive.

Blockchain RPC: eth_getProof

Bedrock (Op/Base)

MPT

keccak

1. mapping of blockNumber→outputRoot 2. Algorithm for verifying blockHash using outputRoot

Blockchain RPC: eth_getProof

Polygon zkEVM

Sparse Merkle Tree

Poseidon

1. mapping of batchNumber→stateRoot 2. Blockchain RPC endpoint for mapping batchNumber to blockNumber

Not Supported

Witness nodes data for Merkle Inclusion Proof is not freely available

Scroll

MPT

Poseidon

Contract not open-sourced and not verified. #1 and #2

Blockchain RPC: eth_getProof

is present in the node's codebase, but not exposed in public RPC endpoints

Witness nodes data for Merkle Inclusion Proof can be accessed if a private node is used

Taiko

MPT

keccak

mapping of number → blockHash

Blockchain RPC: eth_getProof

zkSync Era

Sparse Merkle Tree

Blake2

mapping of number → blockHash

Not supported

Witness nodes data for Merkle Inclusion Proof is not freely available

StarkNet

MPT

Poseidon

mapping of number → state root hash

Blockchain RPC: pathfinder_getProof

Linea (ConsenSys zkEVM)

Sparse Merkle Tree

MiMC

Contract not open-sourced and not verified

Not Supported. The team will introduce a new RPC API

ZKP Verification (Groth16)

Groth16 verification requires ecadd

, ecmul

and ecpairing

precompiles to be supported. The following table shows whether a rollup is “ready” to execute ZKP verifications or not.

Rollup

ECADD

ECMUL

ECPAIRING

Comment

Arbitrum

Yes

Yes

Yes

Bedrock (OP/Base)

Yes

Yes

Yes

Polygon zkEVM

No

No

No

WIP

Scroll

Partially

Partially

Partially

Calls to precompiles are trusted. Execution of those precompiles is not part of the validity proof.

Taiko

Partially

Partially

Partially

Calls to precompiles are trusted. Execution of those precompiles is not part of the validity proof.

zkSync Era

No

No

No

WIP

Linea

Partially

Partially

Partially

Calls to precompiles are trusted. Execution of those precompiles is not part of the validity proof.

Disclaimer:

Although the majority of the data described above has been verified by the corresponding teams, some properties can be erroneous. If that is the case, comment on the error and the tables will be updated.

Conclusions

- Optimistic rollups tend to have more complex compression algorithms that obfuscate the state root

of the L2 network. One possible reason is due to their maturity. Since they have been for a while now, it is evident that they are putting a lot of effort into optimising L1 gas costs to reduce the L2 TX costs. It is important to note though that too much compression leads to obfuscation thus harder for users to prove the Rollup’s state

- Most of the zkEVMs are not providing the necessary tools and APIs for users to prove a contract’s state on the Rollup. It is expected that as those rollups mature, the required APIs will be supported.

- When it comes to precompiles support, Optimistic rollups support the verification of ZKPs, however, zkEVMs either decide not to support the precompiles at all or support them partially by enabling execution even though it is not part of the validity proofs.