Data Availability

#

Data Availability

A major part of being a rollup is that the entirety of the L2 state can be reconstructed from the data that the zkSync network (L2) submits to Ethereum (L1).

Instead of submitting the data of each transaction, however, zkSync submits how the state of the blockchain changes, this change is called thestate diff. This approach allows the transactions that change the same storage slots to be very cheap, since these transactions don't incur additional data costs.

Besides the state diff we alsopost additionalopen in new windowdata to L1, such as the L2->L1 messages, the L2->L1 logs, the bytecodes of the deployed smart contracts.

We alsocompressopen in new windowall the data that we send to L1, to reduce the costs of posting it.

By posting all the data to L1, we canreconstructopen in new windowthe state of the chain from the data on L1. This is a key security property of the rollup.

Info

If the chain chooses not to post this data, it becomes a validium. This makes transactions there much cheaper, but less secure. Because we use state diffs to post data, we can combine the rollup and validium features, by separating storage slots that need to post data from the ones that don't. This construction combines the benefits of rollups and validiums, and it is called azkPorteropen in new window.

#

Recreating L2 State From L1 Pubdata

zkSync has developed a tool to help users validate L2 state data or “pubdata” submitted to L1. Here’s the basic flow of the tool:

#

Basic Flow

1. First, we need to filter all of the transactions to the L1 zkSync contract for only thecommitBlocks
2. transactions where the proposed block has been referenced by a correspondingexecuteBlocks
3. call (the reason for this is that a committed or even proven block can be reverted but an executed one cannot).
4. Once we have all the committed blocks that have been executed, we then will pull the transaction input and the relevant fields
5.
    1. Kinds of pubdata we’ll pull from transaction data:* L2 to L1 Logs
6.
    1.
        - L2 to L1 Messages
7.
    1.
        - Published Bytecodes
8.
    1.
        - Compressed State Diffs

The 2 main fields needed for state reconstruction are theState Diffs andContract Bytecodes .

#

State Diffs

State diffs are key value pairs that represent the 32 byte value stored at a particular storage slot for a particular address.

naive way: (storage_slot, address, value) actual: (derived_key, value) compressed: (derived_key or enumeration index, compressed_value) * derived_key * is thehash(storage_slot || address) * and * enumeration index * is the short hand ID that can be substituted forderived_key * but only after its first write. The deeper meaning is that an enumeration key is the leaf index in our storage Merkle tree.

#

Contract Bytecodes

At a high level, contract bytecode is chunked into opcodes (which have a size of 8 bytes), assigned a 2 byte index, and the newly formed byte sequence (indexes) are verified and sent to L1.

This process is split into 2 different parts: (1)the server side operatoropen in new windowhandling the compression and (2)the system contractopen in new windowverifying that the compression is correct before sending to L1.

The compressed bytecode makes it way up throughfactoryDeps and the hash of uncompressed bytecode is stored on theAccountStorage contract so the hash of the uncompressed bytecode will be part of the state diffs

[ ]Edit this pageopen in new windowLast update: Contributors: [[Ramon "9Tails" Canales ,][albicodes]]

Next Recreating State form L1