Integrate with Blobstream contracts

Getting started

Prerequisites

Make sure to have the following installed:

- Foundry

Installing Blobstream contracts

We will be using theIDAOracle interface to verify inclusion. So, we will install the Blobstream contracts repo as a dependency:

sh forge

install

celestiaorg/blobstream-contracts

--no-commit forge

install

celestiaorg/blobstream-contracts

--no-commit Make sure that the directory you're running this command from is an initialized git repository. If not, just initialize the repo using:

sh git

init git

init Note that the minimum Solidity compiler version for using the Blobstream contracts is0.8.19 .

Example usage

Example minimal Solidity contract for a stub ZK rollup that leverages theBlobstreamX.sol contract to check that data has been posted to Celestia:

solidity // SPDX-License-Identifier: Apache-2.0 pragma

solidity

^0.8.19 ;

import

"blobstream-contracts/IDAOracle.sol" ; import

"blobstream-contracts/DataRootTuple.sol" ; import

"blobstream-contracts/lib/tree/binary/BinaryMerkleProof.sol" ;

contract MyRollup { IDAOracle immutable blobstream; bytes32 [] public rollup_block_hashes;

constructor (IDAOracle _blobstream) { blobstream = _blobstream; }

function

submitRollupBlock (bytes32 _rollup_block_hash, bytes

calldata _zk_proof, uint256 _blobstream_nonce, DataRootTuple

calldata _tuple, BinaryMerkleProof

calldata _proof) public { // Verify that the data root tuple (analog. block header) has been // attested to by the Blobstream contract. require (blobstream. verifyAttestation (_blobstream_nonce, _tuple, _proof));

// Verify the ZKP (zero-knowledge proof). // _tuple.dataRoot is a public input, leaves (shares) are private inputs. require (

verifyZKP (_rollup_block_hash, _zk_proof, _tuple.dataRoot));

// Everything checks out, append rollup block hash to list. rollup_block_hashes. push (_rollup_block_hash); }

function

verifyZKP (bytes32 _rollup_block_hash, bytes

calldata _zk_proof, bytes32 _data_root) private

pure

returns (bool) { return

true ; } } // SPDX-License-Identifier: Apache-2.0 pragma

solidity

^0.8.19 ;

import

"blobstream-contracts/IDAOracle.sol" ; import

"blobstream-contracts/DataRootTuple.sol" ; import

"blobstream-contracts/lib/tree/binary/BinaryMerkleProof.sol" ;

contract MyRollup { IDAOracle immutable blobstream; bytes32 [] public rollup_block_hashes;

constructor (IDAOracle _blobstream) { blobstream = _blobstream; }

function

submitRollupBlock (bytes32 _rollup_block_hash, bytes

calldata _zk_proof, uint256 _blobstream_nonce, DataRootTuple

calldata _tuple, BinaryMerkleProof

calldata _proof) public { // Verify that the data root tuple (analog. block header) has been // attested to by the Blobstream contract. require (blobstream. verifyAttestation (_blobstream_nonce, _tuple, _proof));

// Verify the ZKP (zero-knowledge proof). // _tuple.dataRoot is a public input, leaves (shares) are private inputs. require (verifyZKP (_rollup_block_hash, _zk_proof, _tuple.dataRoot));

// Everything checks out, append rollup block hash to list. rollup_block_hashes. push (_rollup_block_hash); }

function

verifyZKP (bytes32 _rollup_block_hash, bytes

calldata _zk_proof, bytes32 _data_root) private

pure

returns (bool) { return

true ; } }

Data structures

EachDataRootTuple is a tuple of block height and data root. It is analogous to a Celestia block header.DataRootTuple s are relayed in batches, committed to as aDataRootTuple s root (i.e. a Merkle root ofDataRootTuple s).

TheBinaryMerkleProof is anRFC-6962 -compliant Merkle proof. SinceDataRootTuple s are Merkleized in a binary Merkle tree, verifying the inclusion of aDataRootTuple against aDataRootTuple s root requires verifying a Merkle inclusion proof.

Interface

TheIDAOracle (D ataA vailabilityO racle Interface) interface allows L2 contracts on Ethereum to query theBlobstreamX.sol contract for relayedDataRootTuple s. The single interface methodverifyAttestation verifies a Merkle inclusion proof that aDataRootTuple is included under a specific batch (indexed by batch nonce). In other words, analogously it verifies that a specific block header is included in the Celestia chain.

Querying the proof

To prove that the data was published to Celestia, check out the proof queries documentation to understand how to query the proofs from Celestia consensus nodes and make them usable in the Blobstream X verifier contract.

Verifying data inclusion for fraud proofs

A high-level overview of how a fraud-proof based L2 would interact with Blobstream can be found in the inclusion proofs documentation .

The DAVerifier library is available at blobstream-contracts/lib/verifier/DAVerifier.sol , and provides functions to verify the inclusion of individual (or multiple) shares against a DataRootTuple . The library is stateless, and allows to pass an IDAOracle interface as a parameter to verify inclusion against it.

In the DAVerifier library, we find functions that help with data inclusion verification and calculating the square size of a Celestia block. These functions work with the Blobstream X smart contract, using different proofs to check and confirm the data's availability. Let's take a closer look at these functions:

- verifySharesToDataRootTupleRoot
- : This function verifies that the shares, which were posted to Celestia, were committed to by the Blobstream X smart contract. It checks that the data root was committed to by the Blobstream X smart contract and that the shares were committed to by the rows roots.
- verifyRowRootToDataRootTupleRoot
- : This function verifies that a row/column root, from a Celestia block, was committed to by the Blobstream X smart contract. It checks that the data root was committed to by the Blobstream X smart contract and that the row root commits to the data root.
- verifyMultiRowRootsToDataRootTupleRoot
- : This function verifies that a set of rows/columns, from a Celestia block, were committed to by the Blobstream X smart contract. It checks that the data root was committed to by the Blobstream X smart contract and that the rows roots commit to the data root.
- computeSquareSizeFromRowProof
- : This function computes the Celestia block square size from a row/column root to data root binary Merkle proof. It is the user's responsibility to verify that the proof is valid and was successfully committed to using the verifyRowRootToDataRootTupleRoot()
- method.
- computeSquareSizeFromShareProof
- : This function computes the Celestia block square size from a shares to row/column root proof. It is the user's responsibility to verify that the proof is valid and that the shares were successfully committed to using the verifySharesToDataRootTupleRoot()
- method.

For an overview of a demo rollup implementation, head to the next section . [][Edit this page on GitHub] Last updated:
Previous page Overview of Blobstream Next page Integrate with Blobstream client []