Validium Reference

Verify data availability on Ethereum

In order to verify data availability on Ethereum it is necessary to first submit data to Avail as a data submission transaction. Data submitted this way will be included in Avail blocks, but not interpreted or executed in any way. The submission can be done usingPolkadot-JS which is a collection of tools for communication with chains based on Substrate (which is now part of the Polkadot SDK).

Complete example can be found ongithub(opens in a new tab) .

Example of sending data to Avail:

async

function

submitData (availApi , data , account) { let submit =

await

availApi . tx . dataAvailability .submitData (data); return

await

sendTx (availApi , account , submit); } FunctionsubmitData receivesavailApi api instance,data that will be submitted, and theaccount which is sending the transaction. In order to create account it is necessary to createkeyring pair for the account that wants to send the data. This can be done withkeyring.addFromUri(secret) which creates keyring pair via suri (the secret can be a hex string, mnemonic phrase or a string). After creating keyring pair, it is possible to submit data in a transaction to the Avail network withavailApi.tx.dataAvailability.submitData(data); . Once the transaction is included in an Avail block, it is possible to initiate the dispatch of the data root by creating a dispatch transactionavailApi.tx.daBridge.tryDispatchDataRoot(destinationDomain, bridgeRouterEthAddress, header); with the parameters:

destinationDomain Destination domain 1000.

bridgeRouterEthAddress Address of the main data availability router contract deployed on Sepolia network for Goldberg (0x305222c4DdB86FfA9fa9Aa0A479705577E3c4d33), for Kate0xbD824890A51ed8bda53F51F27303b14EFfEbC152 .

header Provided from the block when data is submitted.

async

function

dispatchDataRoot (availApi , blockHash , account) { const

header

=

await

availApi . rpc . chain .getHeader (blockHash); let tx =

await

availApi . tx . daBridge .tryDispatchDataRoot (process . env . DESTINATION_DOMAIN , process . env . DA_BRIDGE_ADDRESS , header ,); return

await

sendTx (availApi , account , tx); } Example of submitting data to Avail and dispatching the data root usingPolkadot-JS .

Environment variables:

AVAIL_RPC= # avail network websocket url SURI= # mnemonic DA_BRIDGE_ADDRESS= # main da bridge contract address deployed to Sepolia test network in format (Kate) 0x0000000000000000000000bD824890A51ed8bda53F51F27303b14EFfEbC152 (Goldberg) 0x0000000000000000000000305222c4DdB86FfA9fa9Aa0A479705577E3c4d33 DESTINATION_DOMAIN= # destination domain is 1000 DATA= # data sending to avail Dispatch Data Root Javascript Example import { ApiPromise , Keyring ,

```
WsProvider } from
'@polkadot/api' ; import
*
as dotenv from
'dotenv' ;
dotenv .config ();
/* * Creates api instance. * * @param url websocket address/ async
function
createApi (url) { const
provider
=
new
WsProvider (url); return
ApiPromise .create ({ provider , rpc : { kate : { queryDataProof : { description :
'Generate the data proof for the given index' , params : [ { name :
'transaction_index' , type :
'u32' , } , { name :
'at' , type :
'Hash' , isOptional :
true , } , ] , type :
'DataProof' , } , } , } , types : { AppId :
'Compact' , DataLookupIndexItem : { appId :
'AppId' , start :
'Compact' , } , DataLookup : { size :
'Compact' , index :
'Vec' , } , KateCommitment : { rows :
'Compact' , cols :
'Compact' , dataRoot :
'H256' , commitment :
'Vec' , } , V1HeaderExtension : { commitment :
'KateCommitment' , appLookup :
'DataLookup' , } , VTHeaderExtension : { newField :
'Vec' , commitment :
'KateCommitment' , appLookup :
'DataLookup' , } , HeaderExtension : { _enum : { V1 :
'V1HeaderExtension' , VTest :
'VTHeaderExtension' , } , } , DaHeader : { parentHash :
```

'Hash' , number :

'Compact' , stateRoot :

'Hash' , extrinsicsRoot :

'Hash' , digest :

'Digest' , extension :

'HeaderExtension' , } , Header :

'DaHeader' , CheckAppIdExtra : { appId :

'AppId' , } , CheckAppIdTypes : {} , CheckAppId : { extra :

'CheckAppIdExtra' , types :

'CheckAppIdTypes' , } , DataProof : { root :

'H256' , proof :

'Vec' , numberOfLeaves :

'Compact' , leafIndex :

'Compact' , leaf :

'H256' , } , Cell : { row :

'u32' , col :

'u32' , } , } , signedExtensions : { CheckAppId : { extrinsic : { appId :

'AppId' , } , payload : {} , } , } , }); }

/* * *Sends transaction to Avail. * * @param api instance of the api * @param account sending the transaction * @param tx transaction* / async

function

sendTx (api , account , tx) { return

new

Promise (async (resolve) => { try { const

res

=

await

tx .signAndSend (account , (result) => { if (result . status .isReady) { console .log (Txn has been sent to the mempool); } if (result . status .isInBlock) { console .log (Tx hash: { result .txHash } is in block { result . status .asInBlock } ,); res (); resolve (result); } }); } catch (e) { console .log (e); process .exit (1); } }); }

/** * Submitting data to Avail as a transaction. * * @param availApi api instance * @param data payload to send * @param account that is sending transaction * @returns

{Promise} */ async

function

submitData (availApi , data , account) { let submit =

await

availApi . tx . dataAvailability .submitData (data); return

await

sendTx (availApi , account , submit); }

/** * Sending dispatch data root transaction. * * @param availApi api instance * @param blockHash hash of the block * @param account sending transaction * @returns

{Promise} */ async

function

dispatchDataRoot (availApi , blockHash , account) { const

destinationDomain

=

process . env . DESTINATION_DOMAIN ; const

bridgeRouterEthAddress

=

process . env . DA_BRIDGE_ADDRESS ; const

header

=

await

availApi . rpc . chain .getHeader (blockHash); console .log (Block Number: { header .number }); console .log (State Root: { header .stateRoot }); let tx =

await

availApi . tx . daBridge .tryDispatchDataRoot (destinationDomain , bridgeRouterEthAddress , header ,); return

await

sendTx (availApi , account , tx); }

/** * Returns data root for the particular block. * * @param availApi api instance * @param blockHash hash of the block * @returns

{Promise<*()[]>*} / async

function

getDataRoot (availApi , blockHash) { const

header

=

JSON .parse (await

availApi . rpc . chain .getHeader (blockHash)); return [header . extension . v1 . commitment .dataRoot ,

header .number]; }

(async

function

dataRootDispatch () { const

availApi

=

await

createApi (process . env . AVAIL_RPC); const

keyring

=

new

Keyring ({ type :

'sr25519' }); const

account

=

keyring .addFromMnemonic (process . env . SURI); console .log ('Submitting data to Avail...');

let result =

await

submitData (availApi ,

process . env . DATA , account); const

txIndex

=

JSON .parse (result .events[0].phase).applyExtrinsic; const

blockHash

=

result . status .asInBlock; console .log (Transaction: { result .txHash } . Block hash: { blockHash } . Transaction index: { txIndex } . ,);

console .log ('Triggering Home...'); result =

await

dispatchDataRoot (availApi , blockHash , account); console .log (Sent txn on Avail. Txn Hash: { result .txHash } .); let [root , blockNum] =

await

getDataRoot (availApi , blockHash); console .log ('Data Root:'

+ root +

' and Block number: '

+ blockNum);

await

availApi .disconnect (); })() .then (() => { console .log ('Done'); }) .catch ((err) => { console .error (err); process .exit (1); });
Dispatching data root will trigger an optimistic bridge which will bridge the data root to the Ethereum network. Since the bridge is optimistic, it is necessary to wait for 30 minutes before the data root is available on Ethereum.

After successfully bridging data root to the main data availability attestation contract on Ethereum, it is possible to prove that data is available on the Avail network by submitting a Merkle proof to the verification contract. Fetching proof from Avail can be done via RPC callkate_queryDataProof for exampleavailApi.rpc.kate.queryDataProof(transactionIndex, hashBlock); wheretransactionIndex is index of the transaction in the block andhashBlock which is a hash of the block in which the data is included. This RPC endpoint returnsDataProof object that can be used to prove on Ethereum that data is available on the Avail network. Example:

async

function

getProof (availApi , hashBlock , transactionIndex) { const

dataProof

=

await

availApi . rpc . kate .queryDataProof (transactionIndex , hashBlock ,); return dataProof; } Returned data:

DataProof: { root: 'H256', proof: 'Vec', numberOfLeaves: 'Compact', leafIndex: 'Compact', leaf: 'H256' } root Root hash of generated merkle tree.

proof Merkle proof items (does not contain the leaf hash, nor the root).

numberOfLeaves Number of leaves in the original tree.

leafIndex Index of the leaf the proof is for (starts from 0).

leaf Leaf for which is the proof.

EXAMPLE

Example of Verification Contract // SPDX-License-Identifier: Apache-2.0 // Modified from https://github.com/QEDK/solidity-misc/blob/master/contracts/Merkle.sol pragma

solidity ^0.8.21;

import

"@openzeppelin/contracts/access/Ownable.sol" ; // or for foundry: // import "openzeppelin-contracts/contracts/access/Ownable.sol";

interface IDataAvailabilityRouter { function

roots (uint32 blockNumber) external

view

returns (bytes32 root); }

contract

ValidiumContract

is

Ownable { IDataAvailabilityRouter private router;

function

setRouter (IDataAvailabilityRouter

_router) public

virtual

onlyOwner { router = _router; }

function

checkDataRootMembership (uint32 blockNumber , bytes32 [] calldata proof , uint256 width ,

// number of leaves uint256 index , bytes32 leaf) public

view

virtual

returns (bool isMember) { bytes32 rootHash = router. roots (blockNumber); // if root hash is 0, block does not have a root (yet) require (rootHash !=

bytes32 (0) ,

"INVALID_ROOT"); assembly ("memory-safe") { if proof.length { let end :=

add (proof.offset ,

shl (5 , proof.length)) let i := proof.offset

for {} 1 {} { let leafSlot :=

shl (5 ,

and (0x1 , index)) if

eq (add (index ,

1) , width) { leafSlot :=

0x20 } mstore (leafSlot , leaf) mstore (xor (leafSlot ,

32) ,

calldataload (i)) leaf :=

keccak256 (0 ,

64) index :=

shr (1 , index) i :=

add (i ,

32) width :=

add (shr (1 ,

sub (width ,

1)) ,

1) if

iszero (lt (i , end)) { break } } } // checks if the calculated root matches the expected root isMember :=

eq (leaf , rootHash) } } } By submitting proof to the verification contract it is possible to verify that data is available on Avail. Merkle proof is a list of hashes that can be used to prove that given leaf is a member of the Merkle tree. Example of submitting a proof to the verification contract deployed on Sepolia network for Kate (0xA06386C65B1f56De57CE6aB9CeEB2552fa811529) and Goldberg (0x67044689F7e274a4aC7b818FDea64Cb4604c6875) can be queried by calling data root membership functionasync function checkProof(sepoliaApi, blockNumber, proof, numberOfLeaves, leafIndex, leafHash); where

sepoliaApi Sepolia network api instance.

blockNumber Avail block number.

proof Merkle proof for the leaf.

numberOfLeaves Number of leaves in the original tree.

leafIndex Index of the leaf in the Merkle tree.

leafHash Hash of the leaf in the Merkle tree.

This will call deployed contracts functionverificationContract.checkDataRootMembership(blockNumber, proof, numberOfLeaves, leafIndex, leafHash) and returntrue orfalse depending on the provided proof.

EXAMPLE OF GETTING THE PROOF AND CHECKING IT WITH VERIFICATION CONTRACT USINGPOLKADOT-JS ANDETHERS.JS . Environment variables:

AVAIL_RPC= # avail websocket address INFURA_KEY= # rpc provider key if needed VALIDIUM_ADDRESS= # address of the verification contract, one such is deployed on Sepolia network for Kate 0xA06386C65B1f56De57CE6aB9CeEB2552fa811529 or Goldberg 0x67044689F7e274a4aC7b818FDea64Cb4604c6875 VALIDIYM_ABI_PATH= # path to abi file e.g. abi/ValidiumContract.json BLOCK_NUMBER= # number of the block for which to get Merkle proof BLOCK_HASH= # hash of the block for which to get Merkle proof TRANSACTION_INDEX= # index of the transaction in the block Submit Proof Example import { ethers } from

'ethers' ; import

*

as dotenv from

'dotenv' ; import { hexlify } from

'ethers/lib/utils.js' ; import { readFileSync } from

'fs' ; import { ApiPromise , WsProvider } from

'@polkadot/api' ;

dotenv .config ();

/** * Creates api instance. * * @param url websocket address * @returns

{Promise} */ async

function

createApi (url) { const

provider

=

new

WsProvider (url);

// Create the API and wait until ready return

ApiPromise .create ({ provider , rpc : { kate : { queryDataProof : { description :

'Generate the data proof for the given index' , params : [{ name :

'data_index' , type :

'u32' , } , { name :

'at' , type :

'Hash' , isOptional :

true , } ,] , type :

'DataProof' , } , } , } , types : { DataProof : { root :

'H256' , proof :

'Vec' , numberOfLeaves :

'Compact' , leafIndex :

'Compact' , leaf :

'H256' , } , } , }); }

/** * Returns Merkle proof for the particular data. * * @param availApi Api instance * @param hashBlock Hash of the block * @param transactionIndex Index of the transaction in the block * @returns

{Promise<*>} / async

function

getProof (availApi , hashBlock , transactionIndex) { const

daHeader

=

await

availApi . rpc . kate .queryDataProof (transactionIndex , hashBlock ,); console .log (`Fetched proof from Avail for txn index { transactionIndex } inside block { hashBlock } ,); return daHeader; }

/** * Checks if the provided Merkle proof is valid by checking on Ethereum deployed validation contract. * * @param sepoliaApi Sepolia network api instance * @param blockNumber Avail block number * @param proof Merkle proof for the leaf * @param numberOfLeaves Number of leaves in the original tree * @param leafIndex Index of the leaf in the Merkle tree * @param leafHash Hash of the leaf in the Merkle tree * @returns

{Promise<>} / async

function

checkProof (sepoliaApi , blockNumber , proof , numberOfLeaves , leafIndex , leafHash ,) { const

abi

=

JSON .parse (readFileSync (process . env . VALIDIYM_ABI_PATH) .toString () ,); const

verificationContract

=

new

ethers .Contract (process . env . VALIDIUM_ADDRESS , abi , sepoliaApi ,); return

await

verificationContract .checkDataRootMembership (BigInt (blockNumber) , proof , BigInt (numberOfLeaves) , BigInt (leafIndex) , leafHash ,); }

(async

function

submitProof () { // connect to Sepolia through Infura but can be used any other available provider const

sepoliaApi

=

new

ethers . providers . InfuraProvider .getWebSocketProvider ('sepolia' , process . env . INFURA_KEY ,); const

availApi

=

await

createApi (process . env . AVAIL_RPC);

console .log (Getting proof for transaction index { process . env . TRANSACTION_INDEX } block number { process . env . BLOCK_NUMBER } and block hash { process . env . BLOCK_HASH } ,); const

daHeader

=

await

getProof (availApi , process . env . BLOCK_HASH , process . env . TRANSACTION_INDEX ,);

console .log (Data Root: { hexlify (daHeader .root) }); console .log (Proof: { daHeader .proof }); console .log (Leaf to prove: { hexlify (daHeader .leaf) }); console .log (Leaf index : { daHeader .leafIndex }); console .log (Number of leaves: { daHeader .numberOfLeaves });

const

isDataAccepted

=

await

checkProof (sepoliaApi , process . env . BLOCK_NUMBER , daHeader .proof , daHeader .numberOfLeaves , daHeader .leafIndex , daHeader .leaf ,); console .log ('Data is: '

+ (isDataAccepted ?

'available'

:

'not available')); await

availApi .disconnect (); await

sepoliaApi .destroy (); })() .then ((() => { console .log ('Done'); }) .catch ((err) => { console .error (err); process .exit (1); });

Madara Starknet Sovereign Rollups