This is continuation introduction of Part1: Boosting the connectivity of Ethereum Layer2s

Module Introduction

Launch Pad Contract

When users send the original intent message on the Source Chain, they interact with this contract.

Relayer Contract

This contract is used for deployment on the Orbital Station network to interface with various Layer2s, responsible for temporarily storing and relaying messages from the Source Chain.

After the Relayer listens to the message event from the Launch Pad, it needs to pass the message to the Relayer Contract within the OSN network for interfacing, transmitting the user's parameters as well as the corresponding Tx hash on the Source Chain.

Landing Pad Contract

When the message is delivered to the Destination Chain, users obtain the transmitted message by interfacing with this contract on the Destination Chain

Orbital Station Network

Maintains a decentralized network for the Relayer

contract, similar to Layer2 and compatible with EVM smart contracts, ensuring the security and decentralized characteristics of the Relayer

contract during the cross-Layer2 communication process.

The state of this network will be kept similar to that of Rollup mechanisms, and through the technology of zkevm, state data will be synchronized to the Ethereum mainnet in a relatively short period.

Arbitration Mechanism

Notation

- \pi

represents a ZKP for a specific computation, \pi(a,b,c)

represents a ZKP carrying three PIS (Public Inputs) a, b, c

.

- id_{src}

and id_{dst}

represent the IDs of the source and destination chains, respectively.

- time_{earlist}

represents the earliest time at which the user's message arrives on the Destination Chain.

- time_{latest}

represents the latest time by which the user's message is to be delivered to the Destination Chain.

- addr_{relayer}

represents the relayer address chosen by the user.

- msg

represents the message that the user wishes to send.

- calldata

represents the final data type used by users when conducting cross-chain transactions, which includes id_{src}, id_{dst}, time_{earlist}, time_{latest}, addr_{layer}, msg

- S_{c}

The state root corresponding to this transaction on the chain

network.

- addr_{sc}

the smart contract deployed on the chain

network.

addr

is the user’s address, Launch_{src}

is the LaunchPad with which the user interacts on the Src Chain, and Landing_{dst}

is the LandingPad for receiving messages on the Destination Chain.

Merkle tree verification constraints are used to indicate that the three members \mathrm{A, B, C}

are all part of the same Root Hash H

after being verified through a Merkle Proof, which can be concisely represented as follows:

\mathrm{MERKLE(A,B,C)} \equiv H

It is used for initiating a message transaction on the Source Chain, where the user’s intentions are all encapsulated in calldata^{user}_{launch}

, which includes id{src}, id_{dst}

, as well as the selected Relayer node address.

\pi^{user}*{launch} \mapsto* *pi(id*{src}, id_{dst},time_{earlist}, time_{latest},calldata^{user}*{launch}, S^{launch}*{src},H^{launch}*{src},addr^{user}*{src},addr^{relayer}*{src},func^{launch}*{src},status^{launch}*{src}) \mapsto* \ *(status^{launch}*{src} \equiv true) \land \mathrm{MERKLE}(addr^{user}*{src},func^{launch}*{src},calldata^{user}*{src}) \equiv H^{launch}*{src} \land (func^{launch}*{src} \equiv \mathrm{ADDR^{launch}*{src}}) \land (H^{launch}*{src} \in S^{launch}*{src}) \land \ (id_{src}, id_{dst},addr^{relayer}*{src} \in calldata^{user}*{launch})

The transaction initiated by the Relayer on the OSN

for message passing, which includes a list of validator addresses for this transaction on the OSN

network.

\pi^{user}*{relayer } \mapsto* *pi(S^{relayer}*{osn},
H^{relayer}*{osn},validators^{H*{realayer}}*{osn},calldata^{user}*{osn},func^{relayer}*{osn}) \mapsto* \ *(status^{relayer}*{osn} \equiv true) \land \mathrm{MERKLE}(validators^{H_{realayer}}*{osn},addr^{relayer}*{osn},func^{relayer}*{osn}, calldata^{relayer}*{osn}) \equiv H^{relayer}*{osn} \land (func^{relayer}*{osn} \equiv \mathrm{ADDR^{relayer}*{osn}}) \land(H^{relayer}*{osn} \in S^{relayer}_{osn})

The transaction relayed by the Relayer on the OSN to the Destination Chain.

\pi^{user}*{landing} \mapsto* *pi(S^{landing}*{dst},
H^{landing}*{dst},addr^{relayer}*{dst},addr^{user}*{src},addr^{landing}*{dst},func^{landing}*{dst},calldata^{relayer}*{landing},status_{landing}) \mapsto \(status^{landing}*{dst} \equiv true) \land \mathrm{MERKLE}(addr^{relayer}*{dst},func^{landing}*{dst},calldata^{relayer}*{landing}) \equiv H^{landing}*{dst} \land(func^{landing}*{dst} \equiv \mathrm{ADDR^{landing}*{dst}}) \land(H^{landing}*{dst} \in S^{landing}_{dst})

Arbitration Flow

Case1

- Anyone can initiate a challenge against any cross-chain transaction . The challenger first generates \pi^{user}_{launch}

. * Submit \pi^{user}_{launch}

to the L1 challenge entry contract.

- The contract will generate a challengeId based on the PIS.
- Submit \pi^{user}_{launch}

to the L1 challenge entry contract.

- The contract will generate a challengeId based on the PIS.

Case 2

The Relayer fabricated a message out of thin air and then transmitted it to the Destination Chain.

- The Challenger generates a \pi^{user}_{landing}

and submits it to the challenge entry contract on L1.

- Subsequently, the account with the address addr^{relayer}_{dst}

**must respond to the corresponding challenge within the time t_{relayer}

, generating the corresponding \pi^{user}_{relayer}

- If it cannot be generated, then it is determined that the relayer has acted maliciously, and the corresponding validator will be penalized.
- If it is generated, then it is considered a malicious challenge by the Challenger.
- If it cannot be generated, then it is determined that the relayer has acted maliciously, and the corresponding validator will be penalized.
- If it is generated, then it is considered a malicious challenge by the Challenger.

Income Distribution Flow

In the OSN network, the balances of all relayers and validators are maintained through the MPT data structure.The calculation rules for the Merkle path are as follows:

path = Hash(id_{chain},address)

Through this path calculation method, the corresponding revenue amount under each address can be accurately maintained. When a certain address wants to withdraw earnings on the target network, it is equivalent to initiating a cross-chain information transfer from the OSN

network. The OSN

network will locally generate an incremental id, which is then consensus-signed by the validators, transmitted to the contract on the target network, and accompanied by the corresponding Merkle Proof, ultimately to obtain the corresponding earnings.

The earnings contract needs to have constraints in place to prevent replay attacks, ensuring that the ids transmitted are monotonically increasing.

Argument Space

Why choose a optimist approach for message passing?

If the state of the Source network is synchronized to the Destination network in real-time using ZKP

- Performance cannot be met; Layer 2 networks have block times in the order of seconds, and generating proofs at a second-level frequency is currently not feasible. If proofs are conducted in batches for multiple blocks, there would be waiting times, resulting in slow message transmission performance.
- The overhead of proof verification; if there are periods when no users are utilizing the system, the continuous need for proof generation and verification would result in a significant waste of resources.

Why is ZKP chosen for the arbitration process?

Computational costs on-chain are significantly higher than those off-chain. By employing ZKP (Zero-Knowledge Proof) technology, the gas expenditure during the arbitration process can be effectively reduced. Based on the experience from the Orbiter decentralized bridge, the gas cost for validating the effectiveness of L2 transactions on L1 using Solidity is approximately XX million Gas.