Cross-Chain Transfer Protocol

Definition

1. A standard solution for cross-chain transfers between blockchain systems.
2. Operational specifications for cross-chain asset transfers between blockchain systems.
3. A standardized interface and data structure for building decentralized services for cross-chain asset transfers.
4. A common solution for building decentralized services for cross-chain asset existence proof.

Goal

The purpose of this protocol is to solve the problems of:

1. The purpose of this protocol is to solve the problems of data disconnection and low interoperability caused by the incompatibility of different blockchain protocols.
2. The purpose of this protocol is to solve the problems of low efficiency high cost of cross-chain asset transfer due to the common solution and standardized interface.

Application Scenario

This protocol is applicable to:

1. Cross-chain asset transfers between blockchain systems.
2. Cross-chain asset exchanges between blockchain systems.
3. Finish existence proof between blockchain systems.

Asset cross-chain transfer data protocol

1 . Leaf node construction method

- **Interface Description**

  Encoding cross-chain asset transfer data.

- **Input Data**

Variables	Types	Data Description	
		Description	Example
amount	Number	Amount of assets transferred by cross-chain	100
address	String	The string of the 'receive' chain address	“28Y8JA1i2cN6oHvdv7EkEd3Y”
unique_id	Byte Array	The unique identification of a single cross-chain transfer	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee
num_range	Number	The range of assets expressed in bytes.	num_range(int64) = 64 num_range(int32) = 32 num_range(int16) = 16
bigendian	BOOLEAN	Code endianness, Big – Endian Set	–

- **Calculation**

$$
\begin{aligned}
&\text{leaf\_data} = \text{sha}(\\
&\quad \text{sha}(\text{encode}(\text{amount}, \text{bigendian})), \\
&\quad \text{sha}(\text{encode}(\text{address}, \text{UTF8})), \\
&\quad \text{unique\_id})
\end{aligned}
$$

- **Output Data**

Variables	Types	Data Description	
		Description	Example
leaf_data	Byte array	Encoded leaf node data	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee

2 . Merkle Tree Construction Method

- **Interface description**

  According to the leaf node data, generate a Merkle Tree, get the root node data.

- **Calculation**

$$parent = sha(left, right)$$

  - Copy the last node data if there are odd leaf nodes, right = left
  - Generate a new set of nodes, repeat the 'generate parent', and finally get the root node, root.

- **Output Data**

Variables	Types	Data Description	
		Description	Example
root	Byte array	Root data	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee

3 . Merkle Path Generation Method

- **Interface description**

  For a leaf node in Merkle Tree, calculates all the path nodes required from the leaf node to the root node.

- **Calculation**

  - Get adjacent nodes

$$merkel\_path[i].is\_left = index \% 2 == 1$$
$$merkel\_path[i].node = merkel\_path[i].is\_left\ ?$$
$$tree\_nodes[index - 1] : tree\_nodes[index + 1]$$

  - Update to the parent node and repeat the above process until the root

$$index = index(parent)$$

- **Output Data**

Variables		Types	Data Description	
			Description	Example
merkle_path_node	node	Byte array	Node data	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee
	is_left	BOOLEAN	Left set node	–
merkle_path		merkle_path_node array	Merkle path data for cross-chain verification	–

4 . Merkle Tree Root Data Record

- **Interface description**

  Record the Merkle Tree root data generated by the cross-chain transfer initiator system.

- **Input Data**

Variables	Types	Data Description	
		Description	Example
root	Byte array	Node data	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee

5 . Cross-Chain Transfer Verification Method

- **Interface description**

  Verify the existence of cross-chain transfer data.

- **Input Data**

Variables	Types	Data Description	
		Description	Example
amount	Number	Amount of assets transferred by cross-chain	100
address	String	The string of the 'receive' chain address	“28Y8JA1i2cN6oHvdv7EkEd3Y”

unique_id	Byte array	The unique identification of a single cross-chain transfer	0x3b35bfb78fbdfe8681898a2a02ad5c0755d910767bad93047d5e4cf2dc79beee
num_range	Number	The range of assets expressed in bytes.	num_range(int64) = 64 num_range(int32) = 32 num_range(int16) = 16
bigendian	BOOLEAN	Code endianness, Big – Endian Set	–
merkle_path	merkle_path_node array	Merkle path data for cross-chain verification	–

- **Calculation**
  - How to generate Leaf node

$$\begin{aligned}&\text{leaf\_data} = \text{sha}(\\&\quad \text{sha}(\text{encode}(\text{amount}, \text{bigendian})),\\&\quad \text{sha}(\text{encode}(\text{address}, \text{UTF8})),\\&\quad \text{unique\_id})\end{aligned}$$

  - Merkle Proof – Use the leaf_data generated by the above calculation and the external input Merkle_path for iterative calculation.

$$\begin{aligned}&\text{cal} = \text{merkle\_path[i].is\_left} ?\\&\text{sha}(\text{merkle\_path[i].node}, \text{cal}) : \text{sha}(\text{cal}, \text{merkle\_path[i].node}),\end{aligned}$$

- Verify that the calculation results are consistent with the root node data

$$require(cal == root)$$

- **Output data**

Variables	Types	Data Description	
		Description	Example
result	BOOLEAN	Is cross-chain Transfer Verification Successful	-