hypeRoot Project Documentation

1. Introduction

hypeRoot is a cross-chain platform that bridges Rootstock, a Bitcoin sidechain compatible with Ethereum, and Hyperledger Fabric, one of the most widely adopted private blockchain frameworks. With hypeRoot, enterprises can lock RBTC tokens in a smart contract on the Rootstock network and, in return, they are issued equivalent Fabric tokens on their internal Hyperledger Fabric network. These Fabric tokens can then be unlocked for RBTC or transferred within the Fabric network as needed. hypeRoot empowers enterprises to customize chaincode functions within their Fabric network, enhancing their operations by integrating the benefits of both public and private blockchains.

2. System Components

hypeRoot consists of several integral components, each developed with a specific technology stack, and note that each of these components includes additional verification and security checks to maintain the integrity and security of the platform:

2.1 Frontend JavaScript Application

The frontend application for hypeRoot is developed in JavaScript. It allows users to interact with the system in a user-friendly manner. This application uses Ethers.js, a library that enables interactions with the Ethereum-compatible RSK network, to perform tasks such as signing and sending transactions. The JavaScript code is divided into two main parts: Helper functions and User Interaction Methods:

2.1.1 Helper Functions

- **solditySign(_message):** This function signs a message using the private key set in the server environment. The message is hashed, and the hash is sent to the server via a POST request to the '/solditySign' endpoint. The server signs the

hashed message and returns the signature.

- **connect():** This function connects the frontend application with the user's

  MetaMask wallet. If the user does not have MetaMask installed, the function throws an error.

2.1.2 User Interaction Methods

- **lockTokens(amount):** This method allows users to lock RBTC tokens on the Rootstock network. It sends a transaction to the smart contract on the RSK network, locking the amount of tokens specified by the user. Simultaneously, it sends a POST request to the '/lock' endpoint of the backend server, notifying it of the locked tokens.
- **unlockTokens(amount):** This method allows users to unlock their previously locked RBTC tokens. It sends a POST request to the '/unlock' endpoint of the backend server, supplying the user's address, the amount of tokens to be unlocked, and a signature proving the user's intent to unlock the tokens.
- **transferTokens(to, amountReceived):** This method allows users to transfer their Fabric tokens within the Fabric network. It sends a POST request to the '/transfer' endpoint of the backend server, providing the recipient's address, the amount of tokens to be transferred, and a signature proving the user's intent to transfer the tokens.
- **getTransactionData():** This method fetches the transaction data of the user. It sends a POST request to the '/transactionData' endpoint of the backend server, providing the user's address. The server returns the transaction data, which is then displayed to the user.
- **recoverTokens():** This method allows users to recover their tokens. It fetches the user's transaction data, calculates the user's current balance based on these transactions, and sends a transaction to the smart contract on the RSK network to recover the user's tokens.
- **joinFabric():** This method allows users to join the Fabric network. It sends a POST request to the '/join' endpoint of the backend server, providing a signed message as proof of the user's intent to join the Fabric network.

2.2 Backend RESTful API Server

The backend server is developed using Node.js and Express.js. It provides a RESTful API for the frontend application to interact with the Hyperledger Fabric network. The

server includes the following helper functions and endpoints:

2.2.1 Helper Functions

- **enrollUser(userId):** Enrolls a user with the provided userId into the Org1MSP organization. This is achieved by communicating with the Fabric CA to register the user and enroll the user, which results in the generation of the user's certificates. This function is crucial in onboarding new users into the Fabric network.
- **getConnectedGateway(username):** Establishes a connection to the Hyperledger Fabric network for the provided username. This function builds a gateway object using the connection profile, identity, and discovery options. The gateway object is then used to connect to the Fabric network. This function forms the foundation for all interactions between the application and the Fabric network.
- **verifyTransaction(txObject):** Verifies a transaction object by ensuring the transaction data matches the actual transaction data on the RSK network. The transaction object includes data such as the transaction hash, from and to addresses, value, and data. This function is critical in maintaining the integrity and validity of the transactions that the application processes.
- **verifyTransfer(to, amount, messageReceived):** Verifies a transfer transaction. It checks that the recipient and the amount in the received message match the provided recipient and amount. This function ensures that the details of a transfer transaction haven't been tampered with.

2.2.2 API Endpoints

- **Joining User Endpoint (/join):** This endpoint handles user join requests. It verifies the request by checking the signature of the message, enrolls the user using the enrollUser function, and initiates the user's identity on the Fabric network by invoking the 'InitIdentity' chaincode function. The user's address and the timestamp are passed as arguments to the chaincode function.
- **Locking Tokens Endpoint (/lock):** This endpoint processes requests to lock RBTC tokens. It verifies the transaction using the verifyTransaction function and issues equivalent tokens on the Fabric network by invoking the 'IssueTokens' chaincode function. The user's address, the value of the tokens, and the timestamp are passed as arguments to the chaincode function.
- **Unlocking Tokens Endpoint (/unlock):** This endpoint manages requests to

unlock Fabric tokens. It verifies the transaction by checking the signature of the value, and dissolves the tokens from the Fabric network by invoking the 'DissolveTokens' chaincode function. The user's address, the value of the tokens, the signature, and the timestamp are passed as arguments to the chaincode function.

- **Transferring Tokens Endpoint (/transfer):** This endpoint handles requests to transfer Fabric tokens from one user to another. It verifies the transaction using the verifyTransfer function and carries out the transfer operation on the Fabric network by invoking the 'TransferTokens' chaincode function. The sender's address, the recipient's address, the amount of tokens, the timestamp, the message, and the signature are passed as arguments to the chaincode function.
- **Getting Transaction Data Endpoint (/transactionData):** This endpoint responds to requests for retrieving the transaction data of a user. It invokes the 'CheckTransactionData' chaincode function to fetch the user's transaction data from the Fabric network.
- **Checking User Identity Endpoint (/check):** This endpoint verifies the existence of a user's identity on the network. It invokes the 'IdentityExists' chaincode function to check if the user's identity exists on the Fabric network.
- **Signing Message Endpoint (/solidity Sign):** This endpoint signs a message using Solidity's keccak256 hashing function. It uses the private key set in the environment to sign the message hash received in the request.
- **Generating New Signer Endpoint (/newSigner):** This endpoint generates a new signer which includes a new private key and public address. It signs a message containing the new signer's address using the old signer's private key, and updates the environment with the new private key.
- **Restoring Old Signer Endpoint (/restoreOldSigner):** This endpoint restores the old signer by setting the private key back to the old private key. It updates the environment with the old private key.

These helper functions and endpoints, along with the chaincode functions they call, provide the core functionalities of HypeRoot, enabling it to interact with both the RSK network and the Fabric network, and perform operations such as enrolling users, locking and unlocking tokens, and transferring tokens.

2.3 Rootstock Network Smart Contract

The Rootstock network smart contract is a crucial part of the HypeRoot system. Written in Solidity, this contract oversees the locking of RBTC tokens. Users interact with this contract to lock their RBTC tokens, which triggers the issuance of an equivalent number of tokens on the Hyperledger Fabric network. The contract utilizes OpenZeppelin libraries, which are known for their robustness, security, and community trust.

Security Considerations

The smart contract employs several security measures to prevent attacks:

- **ReentrancyGuard:** This is a pattern used to prevent re-entrancy attacks, where an attacker can drain a contract by repeatedly calling a function before the first function call is finished. By using the nonReentrant modifier on all external functions that change the contract's state, we ensure that nested (reentrant) calls are not allowed.
- **Signature Verification:** The contract verifies the signatures of transactions to ensure they are initiated by the rightful owner of the tokens.
- **Address and ECDSA libraries:** These libraries from OpenZeppelin provide utility functions for address checks and elliptic curve cryptography operations, further enhancing the security of the contract.
- **Locking of Funds:** The contract locks the RBTC tokens when the user decides to lock them. This operation prevents double-spending, as the same tokens cannot be spent on both the RSK and Fabric networks simultaneously.
- **Prevention of Malleability Attacks:** In Ethereum/RSK, transaction hashes are dependent on the entire transaction content and signature. This makes the network inherently resistant to malleability attacks, as a third party cannot change the transaction content without also invalidating the signature.
- **Dynamic Signer Key:** The signer key used for unlocking tokens is dynamically updated every time a lock transaction occurs. This design choice enhances the security of the contract by ensuring that a specific key-signature pair cannot be reused for fraudulent transactions. This also reinforces the prevention of replay attacks, as the changing of the signer key with each lock transaction invalidates previous signatures, disallowing their reuse.

Smart Contract Functions

- **constructor(address _signerAddress, address _walletAddress):** The constructor function sets the initial signer and wallet addresses.

- **lockTokens(address _newSignerAddress, bytes memory _signature):** This function allows users to lock their RBTC tokens. It verifies the signature to ensure it matches the expected signer, then locks the specified amount of tokens, deducts a fee, and updates the signer key with a new key provided in the arguments. This is a critical security measure, as it ensures that the signer key is constantly changing, making it more difficult for an attacker to gain unauthorized access.
- **unlockTokens(uint256 _amount, bytes memory _signature):** This function allows users to unlock their previously locked RBTC tokens. It verifies the signature to ensure it matches the expected signer, then unlocks the specified amount of tokens.
- **recoverTokens(int256 _amount, bytes memory _signature):** This function allows users to recover their tokens if there's a discrepancy between the token balance on the RSK and Fabric networks. It verifies the signature to ensure it matches the expected signer, then adjusts the token balance and sends the difference to the user.

Events

- **Locked(address indexed user, uint256 amount):** This event is emitted when a user locks tokens.
- **Unlocked(address indexed user, uint256 amount):** This event is emitted when a user unlocks tokens.
- **Recovered(address indexed user, uint256 amount):** This event is emitted when a user recovers tokens.

2.4 Hyperledger Fabric Chaincode

The system includes a chaincode deployed on the Hyperledger Fabric network. This chaincode, written in JavaScript, mirrors the transactions happening on the RSK network. When RBTC tokens are locked on the RSK network, the chaincode issues equivalent tokens on the Fabric network. These tokens can then be unlocked or transferred within the Fabric network. The chaincode also contains functions to check transaction data and verify if a user exists within the system.

The chaincode script consists of several critical functions:

Helper Functions

Helper functions are used to assist the chaincode functions in various tasks such as transaction verification and checking the balance. These functions are:

- **verifyTransaction(txObject):** This function verifies the transaction object by checking if the 'from' and 'to' addresses match the ones in the transaction object, and ensures that the transaction data matches the one in the transaction object.
- **verifyAmount(data, amount):** This function verifies the amount in the transaction data. It does so by comparing the amount in Wei from the transaction data with the provided amount.
- **verifyTransfer(to, amount, messageReceived):** This function verifies the transfer by comparing the received message with the created message.

Chaincode Functions

Chaincode functions are the ones that directly interact with the ledger state on the Hyperledger Fabric network. These functions are:

- **InitIdentity(ctx, rootstockAddress, timestamp, message, signature):** This function initializes a new identity for a user with a given Rootstock address. It verifies the message signature to ensure it matches the expected signer, then creates a new user identity with a balance of 0 Fabric tokens.
- **IssueTokens(ctx, rootstockAddress, fabricTokens, timestamp, tx):** This function issues tokens to a user's account. It verifies the transaction and updates the user's balance with the new amount of Fabric tokens.
- **DissolveTokens(ctx, rootstockAddress, fabricTokens, signature, timestamp):** This function dissolves tokens from a user's account. It verifies the signature to ensure it matches the expected signer, then deducts the specified amount of tokens from the user's balance.
- **TransferTokens(ctx, fromRootstockAddress, toRootstockAddress, fabricTokens, timestamp, message, signature):** This function enables token transfer between two users. It verifies the transfer and updates the balances of the sender and receiver accordingly.
- **CheckTransactionData(ctx, username):** This function checks the transaction data for a user. It returns the transaction history for a given user.
- **IdentityExists(ctx, username):** This function checks if a user identity exists in the system. It returns a boolean value indicating whether the user exists.

These functions together provide a robust system for handling tokens on the Fabric network. They ensure that tokens are issued, transferred, and dissolved accurately while maintaining precise transaction histories for each user. Through the verification of transactions and signatures, the chaincode contributes to maintaining the integrity of the system and preventing fraudulent activity.

2.5 Enroll Admin and Register Private Key Scripts

Before starting the server, two scripts are run to set up the necessary entities in the system: an admin user identity for the Hyperledger Fabric network and an Ethereum wallet with a registered private key. This private key is crucial for our system as it is used to sign transactions when unlocking tokens.

2.5.1 enrollAdmin.js

This script enrolls an admin user for the organization Org1 in the Hyperledger Fabric network.

The script makes use of various utilities provided by Fabric SDK such as Wallets and FabricCAServices. It also uses custom utility functions from CAUtil.js and AppUtil.js for building the certificate authority client and wallet.

The enrollment process involves connecting to the certificate authority, enrolling the admin, and storing the admin's identity in the wallet.

Here is a breakdown of the script:

- **buildCCPOrg1():** Builds and returns the connection profile which is needed to connect to the Fabric network.
- **buildCAClient():** Returns a new instance of the FabricCAServices class which is used to interact with the certificate authority.
- **buildWallet():** Returns an instance of the Wallet class. The wallet is used to manage identities.
- **enrollAdmin():** Enrolls the admin user and stores the admin's identity in the wallet.

2.5.2 registerPrivateKey.js

This script creates a new Ethereum wallet, extracts the private key, and stores it in a .env file. This private key can be used to sign transactions and messages.

The script makes use of the ethers library to create a random Ethereum wallet and to extract the private key. It uses the fs module to write the private key to the .env file.

Here's a breakdown of the script:

- **ethers.Wallet.createRandom():** Creates a new random Ethereum wallet.
- **ethWallet.privateKey:** Extracts the private key from the Ethereum wallet.
- **fs.writeFileSync():** Writes the private key to a .env file. This private key can be accessed later for signing transactions or messages.

3. User Interface Workflow

The user interface (UI) serves as the medium for users to interact with the system. The website presents several actions a user can perform, and the flow of these actions is as follows:

- **Connect Metamask:** On clicking the 'Connect Metamask' button, the connect() function from the frontend JavaScript application is triggered. This function checks if the Metamask extension is installed in the user's browser. If installed, it requests Metamask to connect with the current Ethereum account.
- **Join Fabric Network:** After successfully connecting the Metamask account, the user can join the Fabric network by clicking on the 'Join Fabric Network' button. This action triggers the joinFabric() function in the frontend JavaScript application. The function creates a message and signs it using the user's Ethereum account private key. This message is then sent to the Express.js server's /join endpoint, which will interact with the Fabric network to add the user to the network.
- **Get Transaction Data:** The 'Get Transaction Data' button triggers the getTransactionData() function. This function fetches the transaction history of the user from the Fabric network by calling the Express.js server's /transactionData endpoint.
- **Lock Tokens:** The user can lock their RBTC tokens by entering an amount and clicking on the 'Lock Tokens' button. This triggers the lockTokens() function,

which sends a transaction to the smart contract on the RSK network to lock the specified amount of RBTC tokens. The function also communicates with the Express.js server to update the Fabric network about the token locking.

- **Unlock Tokens:** Similarly, the 'Unlock Tokens' button allows the user to unlock their locked RBTC tokens. The unlockTokens() function is called upon clicking the button. The function signs a message with the user's Ethereum private key and sends a request to the Express.js server's /unlock endpoint. Then, a transaction is sent to the RSK network's smart contract to unlock the tokens.
- **Transfer Tokens:** The 'Transfer Tokens' button allows the user to transfer their Fabric tokens to another user. On entering the recipient's account address and the amount to transfer, and clicking the button, the transferTokens() function is triggered. This function signs a message containing the transaction details and sends a request to the Express.js server's /transfer endpoint.
- **Recover Tokens:** The 'Recover Tokens' button triggers the recoverTokens() function. This function is particularly useful in a scenario where a user, while unlocking tokens, approves the first popup (for signing the message) but rejects the second popup (for paying gas fee for contract calling). In this case, the Fabric tokens would be dissolved, but the equivalent RBTC wouldn't be unlocked due to the user's rejection of the transaction. The recoverTokens() function allows the user to recover the RBTC tokens that were not unlocked due to this interruption in the process. This function does so by reconciling the user's RBTC balance in the RSK network smart contract and the Fabric token balance.

4. Conclusion

In conclusion, this documentation provides a comprehensive overview of the hypeRoot platform, a blockchain-based system integrating both Hyperledger Fabric and Rootstock (RSK) networks. The detailed technical explanation of each component, their functionality, and the interactions between them, helps in understanding the overall functioning of the system.

hypeRoot, through its innovative design, enables the fluid movement of tokens between the public blockchain (RSK) and the private blockchain (Fabric). Moreover, it maintains the necessary security measures to ensure the integrity and safety of users' transactions.

The journey of a transaction, from the user's interface to the backend network, and finally to the blockchain, is also well illustrated in this document. This deep-dive into the system's workflow provides a clearer understanding of the process, aiding developers, users, and stakeholders in effectively utilizing and improving the system.

For further enhancements or modifications, it is advised to refer to this document to understand the existing architecture and functionality. This will help in aligning new updates with the existing system seamlessly.

Remember, the power of blockchain lies in its community and open-source nature. Let's continue to build and innovate together.