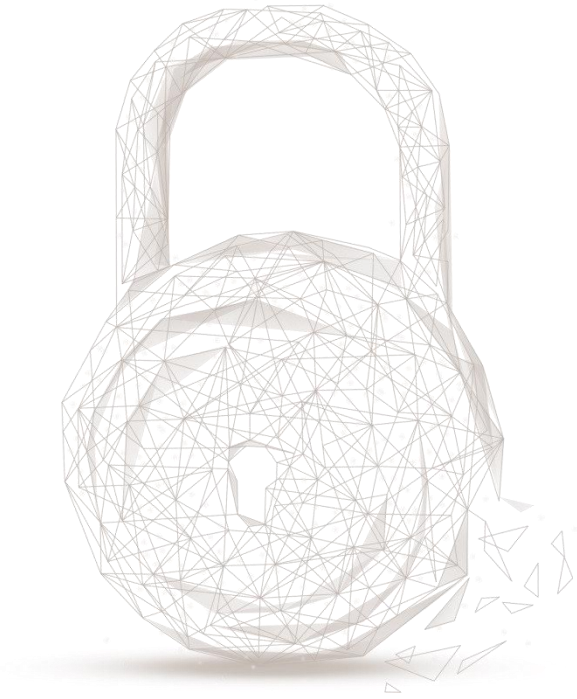
成都链安
BEOSIN

Smart contract security audit report



Audit Number：202008281850

Smart Contract Name：

GXC & relay

Smart Contract Address Link：

https://github.com/gxchain/gxc-relay-contract

Commit Hash：

fd829ed3d56c2a7ccaf3485925af8b918d416fce

Start Date：2020.08.26

Completion Date：2020.08.28

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

1. ETH Business

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass

		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

2. GXC Business

No.	Audit Items	Results
1	Access Control of Owner	Pass
2	System API Usage	Pass
3	get_trx_origin Use Security	Pass
4	Overflow/Underflow	Pass
5	Pseudo-random Number Generator (PRNG)	Pass
6	Handling Fee Consumption	Pass
7	Redundant Code	Pass
8	Exception Check	Pass
9	Asset Check	Pass
10	Replay Attack	Pass
11	Business Logics	Pass

Note: Audit results and suggestions in code comments

the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contracts GXC & relay, including Coding Standards, Security, and Business Logic. **The GXC & relay contract passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

ERC20 Token Audit:

1. Basic Token Information

Token name	None
Token symbol	None
decimals	5
totalSupply	Initial total supply is 0 (Mintable without cap; Destroyable)
Token type	ERC20

Table 1 Basic Token Information

2. Token Vesting Information

No Vesting.

ETH Business Audit:

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.

- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract.
- Result: Pass

2.10 Replay Attack

- Description: Check the weather the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

Check whether the business is secure.

3.1 Token deliver function

- Description:

The token contract implements the *deliver* function to transfer a certain amount of tokens to the specified address for delivery the cross-chain assets to the target address. This function restricts it can only be called by users with the specified DELIVER_ROLE permission. Contract owner pre-sends tokens to the authority address, and then distributes GXC assets (ETH chain) through this address.

```
34      function deliver(
35          address to,
36          uint256 amount,
37          string memory from,
38          string memory txid
39      ) public {
40          require(
41              amount >= _minDeliver,
42              "The minimum value must be greater than minDeliver"
43          );
44          require(hasRole(DELIVER_ROLE, _msgSender()), "Must have deliver role to deliver");
45          for (uint256 i = 0; i < arrayLength; i++) {
46              require(
47                  keccak256(abi.encodePacked(txidArray[i])) !=
48                      keccak256(abi.encodePacked(txid)),
49                  "The txid has existed"
50              );
51          }
52          uint256 id_number = id % arrayLength;
53          txidArray[id_number] = txid;
54          id++;
55          transfer(to, amount);
56          emit Deliver(to, amount, from, txid);
57      }
```

Figure 1 source code of function deliver

- Related functions: *deliver*, *transfer*
- Result: Pass

3.2 Burn for cross-chain withdraw

- Description:

  The token contract implements the *burn(uint256 amount, string memory to)* function. The user destroys GXC assets (ETH chain) through this function, and at the same time specifies the GXC account to. After the specified destruction operation is monitored by the relay service, the corresponding assets on the GXC chain will be withdrawn.

```
59      function burn(uint256 amount, string memory to) public {
60          require(
61              amount >= _minBurn,
62              "The minimum value must be greater than minBurn"
63          );
64          super.burn(amount);
65          emit Burn(msg.sender, amount, to);
66      }
```

Figure 2 source code of function burn

- Related functions: *burn(uint256 amount, string memory to)*, *burn(uint256 amount)*
- Result: Pass

3.3 Adjust related parameters

- Description:

The token contract implements the *adjustParams* function to adjust the specified *deliver* and *burn(uint256 amount, string memory to)* function related restriction parameters. This function restricts it can only be called by users with the specified ADJUST_ROLE permission.

```
68      function adjustParams(uint256 minDeliver , uint256 minBurn)
69          public
70      {
71          require(hasRole(ADJUST_ROLE, _msgSender()), "Adjust role required");
72          _minDeliver = minDeliver;
73          _minBurn = minBurn;
74      }
```

Figure 3 source code of function adjustParams

- Related functions: *adjustParams*
- Result: Pass

GXC Business Audit:

1. Access Control of Administrator

- Description: Due to the business requirements of the project, smart contracts generally have functional functions with high authority requirements. For example, the *withdraw*, *confirmd*, and *confirmw* functions of this contract require the caller to be the administrator of the contract. Therefore, it is necessary to control the calling permissions of contract functions to avoid security problems caused by permission leakage.

```
44      //@abi action
45      void withdraw(std::string to_account, contract_asset amount, std::string from_target, std::string txid, std::string from_account)
46      {
47          int64_t account_id = get_account_id(to_account.c_str(), to_account.size());
48          uint64_t sender = get_trx_sender();
49          auto coin_kind = find(TARGETS.begin(), TARGETS.end(), from_target);
50          graphene_assert(amount.asset_id == 1, "Only support GXC");
51          graphene_assert(amount.amount >= MIN_WITHDRAW, "Must greater than min number");
52          graphene_assert(coin_kind != TARGETS.end(), "Invalid target");
53          graphene_assert(sender == adminAccount, "No authority");
54          graphene_assert(account_id >= 0, "Invalid account_name to_account");
55          graphene_assert(amount.amount > 0, "Invalid amount");
56          if (from_target == "ETH")
57          {
58              for(auto id_begin = eth_withdraw_table.begin(); id_begin != eth_withdraw_table.end(); id_begin++){
59                  graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
60              }
```

Figure 4 partial source code of function withdraw

```
90      //@abi action
91      void confirmd(uint64_t order_id, std::string target, std::string addr, contract_asset amount, std::string txid)
92      {
93          uint64_t sender = get_trx_sender();
94          graphene_assert(sender == adminAccount, "You have no authority");
95          auto idx = fund_in_table.find(order_id);
96          graphene_assert(idx != fund_in_table.end(), "There is no that order_id");
97          graphene_assert((*idx).target == target, "Unmatched chain name");
98          graphene_assert((*idx).asset_id == amount.asset_id, "Unmatched assert id");
99          graphene_assert((*idx).amount == amount.amount, "Unmatched assert amount");
100         if (target == "ETH")
101         {
102             for(auto id_begin = eth_confirm_table.begin(); id_begin != eth_confirm_table.end(); id_begin++){
103                 graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
104             }
```

Figure 5 partial source code of function confirmd

- Result: Pass

2. **System API Usage**

   - Description: GXChain itself provides a large number of built-in APIs for smart contracts. In the process of contract development, contract developers generally use these built-in APIs to complete complex business logic. Therefore, if the built-in APIs are used improperly, it will cause unpredictable safe question.
   - Result: Pass

3. **get_trx_origin Use Security**

   - Description: In the smart contract of GXChain, the *get_trx_origin* function obtains the original caller account ID of this transaction; and the *get_trx_sender* function obtains the direct caller ID of this transaction. If the two are used confusingly, may cause logic errors.
   - Result: Pass

4. **Overflow/Underflow**

   - Description: Overflow is a security problem in many languages, and they are especially dangerous in smart contracts. For example, uint64_t is expressed as an integer from 0 to (2**64-1), and its maximum value is 2**64-1, but if the maximum value is increased by 1, it will overflow and get 0. Similarly, 0 minus 1 will underflow to get the maximum value. Overflow conditions can lead to incorrect results, especially if the possible results are not expected, which may affect the reliability and safety of the program.
   - Result: Pass

5. **Pseudo-random Number Generator (PRNG)**

   - Description: Random numbers may be used in smart contracts. At present, the most common method is to use block information as a random factor to generate, but such use is insecure, because block information can be controlled by miners or obtained by attackers during transactions, This kind of random number is predictable or collidable to a certain extent.
   - Result: Pass

6. **Handling Fee Consumption**

   - Description: In GXChain's smart contract, when the storage space is consumed, the corresponding handling fee needs to be paid. Therefore, when designing the contract, it is necessary to consider the handling fee consumption to avoid excessive handling fee consumption.
   - Result: Pass

7. **Redundant Code**

   - Description: Redundant codes in smart contracts will reduce code readability and may consume more GXC when deploying contracts. It is recommended to eliminate code redundancy.
   - Result: Pass

8. Exception Check

- Description: GXChain uses state recovery exceptions to handle errors. This mechanism will undo all changes made to the state in the current call (and all its sub-calls). The built-in function *graphene_assert* is used to check the condition and throw an exception when the condition is not met, and return the corresponding error message.
- Result: Pass

9. Asset Check

- Description: The assets used by GXChain consist of two parts: the number of assets and the asset ID. The asset ID is the unique identifier of the asset on the GXChain. Therefore, when the contract checks the asset transaction, it must check the asset number as well as the asset ID.
- Result: Pass

10. Replay Attack

- Description: A replay attack means that if two contracts are implemented using the same code, and the identity authentication is in the parameter transmission, when the user executes a transaction to one contract, the transaction information can be copied and retransmitted to the other contract to execute the transaction.
- Result: Pass

11. Business Logics

11.1 Adjust related parameters

- Description:

  By calling this function and specifying the target chain and target address, and attaching GXC assets, the deposit operation is performed. This function performs related checks and restricts the deposit asset type and the deposit amount. After the project relay service monitors the user's deposit operation, it calls the relevant *deliver* function on the external relay contract of the target chain to deliver (issuance) the target chain assets.

```
20      // @abi action
21      // @abi payable
22      void deposit(std::string target, std::string addr)
23      {
24          int64_t asset_amount = get_action_asset_amount();
25          uint64_t asset_id = get_action_asset_id();
26          graphene_assert(asset_id == 1, "Only support GXC ");
27          graphene_assert(asset_amount >= MIN_DEPOSIT, "Must greater than minnumber ");
28          contract_asset amount{asset_amount, asset_id};
29          uint64_t id_number = fund_in_table.available_primary_key();
30          auto coin_kind = find(TARGETS.begin(), TARGETS.end(), target);
31          graphene_assert(coin_kind != TARGETS.end(), "Invalid chain name");
32          uint64_t sender = get_trx_sender();
33          fund_in_table.emplace(sender, [&](auto &o) {
34              o.id = id_number;
35              o.from = sender;
36              o.asset_id = asset_id;
37              o.amount = asset_amount;
38              o.target = target;
39              o.to = addr;
40              o.state = 0;
41          });
42      }
```

Figure 6 source code of function deposit

- Related functions: *deposit*
- Result: Pass

11.2 Confirm deposit function

- Description:

  This function is used to confirm the user's deposit and cross-chain delivery (issuance) transfer transaction. When the user deposits successfully and the corresponding deliver call is executed in the external relay contract, the corresponding txid obtained by the relay service and the deposit record order are used as parameters to verify the deposit order. This function restricts its caller (sender) to only the specified contract administrator (adminAccount). In addition, the function also checks the cross-chain target, the type of assets stored in the order, and the number of assets.

```
90      //@abi action
91      void confirmd(uint64_t order_id, std::string target, std::string addr, contract_asset amount, std::string txid)
92      {
93          uint64_t sender = get_trx_sender();
94          graphene_assert(sender == adminAccount, "You have no authority");
95          auto idx = fund_in_table.find(order_id);
96          graphene_assert(idx != fund_in_table.end(), "There is no that order_id");
97          graphene_assert((*idx).target == target, "Unmatched chain name");
98          graphene_assert((*idx).asset_id == amount.asset_id, "Unmatched assert id");
99          graphene_assert((*idx).amount == amount.amount, "Unmatched assert amount");
100         if (target == "ETH")
101         {
102             for(auto id_begin = eth_confirm_table.begin(); id_begin != eth_confirm_table.end(); id_begin++){
103                 graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
104             }
105             auto id_number = eth_confirm_table.available_primary_key();
106             eth_confirm_table.emplace(sender, [&](auto &o) {
107                 o.id = id_number;
108                 o.txid = txid;
109             });
110             auto begin_iterator = eth_confirm_table.begin();
111             if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
112             {
113                 eth_confirm_table.erase(begin_iterator);
114             }
115             fund_in_table.modify(idx, sender, [&](auto &o) {
116                 o.state = 1;
117             });
118             fund_in_table.erase(idx);
119         }
120     }
```

Figure 7 source code of function confirmd

- Related functions: *confirmd*
- Result: Pass

11.3 User withdrawal request function

- Description:

  This function is used to initiate a user withdrawal request operation. When the relay service monitors the user's destruction operation on the corresponding target chain, it calls this function to initiate the user withdrawal request operation. This function restricts its caller (sender) to only the specified contract administrator (adminAccount). In addition, it also checks the type of withdrawal assets and the amount of withdrawals.

```
44      //@abi action
45      void withdraw(std::string to_account, contract_asset amount, std::string from_target, std::string txid, std::string from_account)
46      {
47          int64_t account_id = get_account_id(to_account.c_str(), to_account.size());
48          uint64_t sender = get_trx_sender();
49          auto coin_kind = find(TARGETS.begin(), TARGETS.end(), from_target);
50          graphene_assert(amount.asset_id == 1, "Only support GXC");
51          graphene_assert(amount.amount >= MIN_WITHDRAW, "Must greater than min number");
52          graphene_assert(coin_kind != TARGETS.end(), "Invalid target");
53          graphene_assert(sender == adminAccount, "No authority");
54          graphene_assert(account_id >= 0, "Invalid account_name to_account");
55          graphene_assert(amount.amount > 0, "Invalid amount");
56          if (from_target == "ETH")
57          {
58              for(auto id_begin = eth_withdraw_table.begin(); id_begin != eth_withdraw_table.end(); id_begin++){
59                   graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
60              }
61              auto id_number = eth_withdraw_table.available_primary_key();
62              eth_withdraw_table.emplace(sender, [&](auto &o) {
63                  o.id = id_number;
64                  o.txid = txid;
65              });
66              auto begin_iterator = eth_withdraw_table.begin();
67              if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
68              {
69                  eth_withdraw_table.erase(begin_iterator);
70              }
71              auto contract_id = current_receiver();
72              auto contract_balance = get_balance(contract_id, amount.asset_id);
73              graphene_assert(contract_balance > amount.amount, "Balance not enough");
74              //withdraw_asset(_self, account_id, amount.asset_id, amount.amount);
75              auto id_number2 = fund_out_table.available_primary_key();
76              int64_t block_time = get_head_block_time();
77              fund_out_table.emplace(sender, [&](auto &o){
78                  o.id = id_number2;
79                  o.to_account = account_id;
80                  o.asset_id = amount.asset_id;
81                  o.amount = amount.amount;
82                  o.from_target = from_target;
83                  o.txid = txid;
84                  o.from_account = from_account;
85                  o.block_time = block_time;
86              });
87          }
88      }
```

Figure 8 source code of function withdraw

- Related functions: *withdraw*
- Result: Pass

11.4 Confirm withdraw function

- Description:

  This function is used to confirm the user's withdrawal request and send the corresponding withdrawal asset to the corresponding user. The relay service will monitor the contents of the withdrawal table, regularly agree to the 24-hour confirmation request in the withdrawal table, and complete the withdrawal operation. This function restricts its caller (sender) to only the specified contract administrator (adminAccount).

```
122     //@abi action
123     void confirmw()
124     {
125        uint64_t sender = get_trx_sender();
126        graphene_assert(sender == adminAccount, "You have no authority");
127        int64_t block_time_now = get_head_block_time();
128        auto idx = fund_out_table.begin();
129        auto number_index = 0;
130        graphene_assert(idx != fund_out_table.end(), "There id nothing to withdraw");
131        while((idx != fund_out_table.end()) && number_index < NUMBER_LIMIT){
132            if(((*idx).block_time + TIME_GAP) > block_time_now){
133                break;
134            }
135            withdraw_asset(_self, (*idx).to_account, (*idx).asset_id, (*idx).amount);
136            idx = fund_out_table.erase(idx);
137            number_index++;
138     }
```

Figure 9 source code of function confirmw

- Related functions: *confirmw*
- Result: Pass

Audited Source Code (ETH) with Comments:

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.4.21 <0.8.0; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
pragma experimental ABIEncoderV2; // Beosin (Chengdu LianAn) // Enable the Encoder of encoding and decoding nested arrays and structures.

import "@openzeppelin/contracts/presets/ERC20PresetMinterPauser.sol";

contract GXC is ERC20PresetMinterPauser {
    // Beosin (Chengdu LianAn) // Declare the bytes32 constant for storing the specified role index.
    bytes32 public constant ADJUST_ROLE = keccak256("ADJUST_ROLE");
    bytes32 public constant DELIVER_ROLE = keccak256("DELIVER_ROLE");

    string[10] private txidArray; // Beosin (Chengdu LianAn) // Declare the string array that the length is 10.
    uint256 arrayLength = 10;
    uint256 private id;

    uint256 private _minDeliver = 50000;
    uint256 private _minBurn = 50000;

    uint8 private decimals_ = 5;
    // Beosin (Chengdu LianAn) // Declare the event 'Deliver'.
    event Deliver(address indexed to, uint256 amount, string from, string txid);
    // Beosin (Chengdu LianAn) // Declare the event 'Burn'.
```

```
    event Burn(address indexed from, uint256 amount, string to);
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic token information and set the initial permission of 'ADJUST_ROLE'.
    constructor(string memory name, string memory symbol)
        public
        ERC20PresetMinterPauser(name, symbol)
    {
        super._setupDecimals(decimals_); // Beosin (Chengdu LianAn) // Call the function '_setupDecimals' to set the token decimals.
        _setupRole(ADJUST_ROLE, _msgSender()); // Beosin (Chengdu LianAn) // Set the initial permission of 'ADJUST_ROLE' to the deployer.
    }

    // Beosin (Chengdu LianAn) // The function 'deliver' is defined to transfer a certain amount of tokens to specified address.
    function deliver(
        address to,
        uint256 amount,
        string memory from,
        string memory txid
    ) public {
        require(
            amount >= _minDeliver,
            "The minimum value must be greater than minDeliver"
        ); // Beosin (Chengdu LianAn) // Require that the amount should not be less than the minimum deliver amount.
        require(hasRole(DELIVER_ROLE, _msgSender()), "Must have deliver role to deliver"); // Beosin (Chengdu LianAn) // Check whether the caller has the permission of 'DELIVER_ROLE'.
        // Beosin (Chengdu LianAn) // Traversal the txidArray array to check whether the specifide 'txid' exists in it.
        for (uint256 i = 0; i < arrayLength; i++) {
            require(
                keccak256(abi.encodePacked(txidArray[i])) !=
                    keccak256(abi.encodePacked(txid)),
                "The txid has existed"
            ); // Beosin (Chengdu LianAn) // Require that the specified 'txid' does not exist in the 'txidArray'.
        }
        uint256 id_number = id % arrayLength; // Beosin (Chengdu LianAn) // Get the index of the specified 'txid' to be stored.
        txidArray[id_number] = txid; // Beosin (Chengdu LianAn) // Store/Update the transaction id in specified index.
        id++;
        transfer(to, amount); // Beosin (Chengdu LianAn) // Call the 'transfer' function to transfer tokens.
        emit Deliver(to, amount, from, txid); // Beosin (Chengdu LianAn) // Trigger the event 'Deliver'.
    }
    // Beosin (Chengdu LianAn) // The function 'burn' is defined to destroy a certain amount of tokens
```

```
of caller.
    function burn(uint256 amount, string memory to) public {
        require(
            amount >= _minBurn,
            "The minimum value must be greater than minBurn"
        ); // Beosin (Chengdu LianAn) // Require that the destruction amount should not be less than the minimum burn amount.
        super.burn(amount); // Beosin (Chengdu LianAn) // Call the 'burn' function of the parent contract to destroy tokens.
        emit Burn(msg.sender, amount, to); // Beosin (Chengdu LianAn) // Trigger the event 'Burn'.
    }
    // Beosin (Chengdu LianAn) // The function 'adjustParams' is defined to adjust the parameters about functions 'deliver' and 'burn'.
    function adjustParams(uint256 minDeliver , uint256 minBurn)
        public
    {
        require(hasRole(ADJUST_ROLE, _msgSender()), "Adjust role required"); // Beosin (Chengdu LianAn) // Check whether the caller has the permission of 'ADJUST_ROLE'.
        _minDeliver = minDeliver;
        _minBurn = minBurn;
    }
    // Beosin (Chengdu LianAn) // The function 'getParams' is defined to query the parameters about functions 'deliver' and 'burn'.
    function getParams() public returns (uint256 ,uint256){
        return (_minDeliver, _minBurn);
    }
    // Beosin (Chengdu LianAn) // The function 'getTxids' is defined to query the stored 10 txids at the current time.
    function getTxids() public returns (string[10] memory) {
        return txidArray;
    }
}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com