KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: Knownsec blockchain security team

Version Summary

Content	Date	Version
Editing Document	20210317	V1.0

Report Information

Title	Version	Document Number	Type
chainswap Smart Contract Audit Report	V1.0		Open to project team

Copyright Notice

Knownsec only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. Knownsec is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. Knownsec's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, Knownsec shall not be liable for any losses and adverse effects caused thereby.

Table of Contents

1. Introduction

The effective test time of this report is from From March 15, 2021 to March 17, 2021 . During this period, the security and standardization of **the smart contract code of the chainswap** will be audited and used as the statistical basis for the report.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the chainswap** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit: SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Report information of this audit:

Report Number:

Report query address link:

Target information of the chainswap audit:

Target information	
Token name	Chainswap
Code type	Token code, swap cross-chain code, Ethereum smart contract code
Code language	solidity

Contract documents and hash:

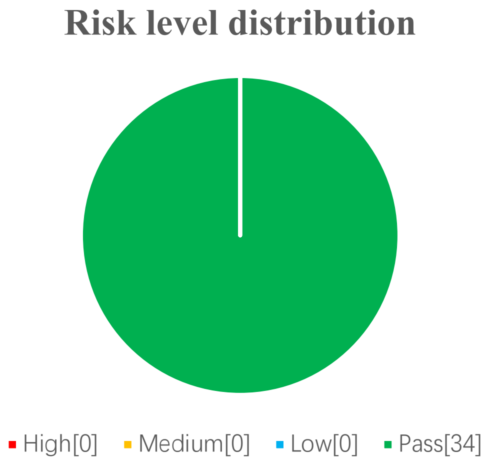
Contract documents	MD5
MappingTokenFactory.sol	D1181CB6C8F6D3D88B57BF12B28E5E33

2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	34



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	MappingBase contract variables and modifiers	Pass	After testing, there is no such safety vulnerability.
	MappingBase contract increase/decrease the function of setting address quota	Pass	After testing, there is no such safety vulnerability.
	MappingBase contract send function	Pass	After testing, there is no such safety vulnerability.
	MappingBase contract receive function	Pass	After testing, there is no such safety vulnerability.
	The MappingTokenFactory contract increases and reduces the number of times the specified address signatory can be verified	Pass	After testing, there is no such safety vulnerability.
	MappingTokenFactory contract registration token mapping function	Pass	After testing, there is no such safety vulnerability.
	MappingTokenFactory contract creation contract function	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	Compiler version security	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.

	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.oriigin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.

	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.
	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. MappingBase contract variables and modifiers【PASS】

Audit analysis: The definition of MappingBase contract variables and the design of the repair artifact function are reasonable.

```
    abstract contract MappingBase is ContextUpgradeSafe, Constants {
        using SafeMath for uint; //knownsec// Use SafeMath library for safe mathematical operations
        bytes32 public constant RECEIVE_TYPEHASH = keccak256("Receive(uint256 fromChainId,address to,uint256 nonce,uint256 volume,address signatory)");
        bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256 chainId,address verifyingContract)");
        bytes32 internal _DOMAIN_SEPARATOR;
        function DOMAIN_SEPARATOR() virtual public view returns (bytes32) {  return _DOMAIN_SEPARATOR;  }

        address public factory;//knownsec// Declare the variable factory to record the factory location
        uint256 public mainChainId;//knownsec// Declare the variable mainChainId to record the chain Id d
        address public token;//knownsec// Declare the variable token to record the token address
        address public creator;//knownsec// Declare the variable creator to record the creator address
        mapping (address => uint) public authQuotaOf; // signatory => quota
        mapping (uint => mapping (address => uint)) public sentCount; // toChainId => to => sentCount
        mapping (uint => mapping (address => mapping (uint => uint))) public sent; // toChainId => to => nonce => volume
        mapping (uint => mapping (address => mapping (uint => uint))) public received; // fromChainId => to => nonce => volume
        modifier onlyFactory {//knownsec// Decorator, the caller must be the factory address
```

```
            require(msg.sender == factory, 'Only called by Factory');
            _;
        }
```

Recommendation: nothing.

3.2. MappingBase contract increase/decrease the function of setting address quota【PASS】

Audit analysis: The increaseAuthQuota and decreaseAuthQuota functions of the MappingBase contract are called by the factory contract to increase and decrease the quota of the specified address.

```
  //knownsec// Increase the quota of specified addresses in batches
function increaseAuthQuotas(address[] memory signatorys, uint[] memory increments) virtual external returns (uint[] memory quotas) {
        require(signatorys.length == increments.length, 'two array lenth not equal');
        quotas = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            quotas[i] = increaseAuthQuota(signatorys[i], increments[i])//knownsec// Call the increaseAuthQuota function to increase the quota of the specified address

    }
    //knownsec// Increase the quota for the specified address
    function increaseAuthQuota(address signatory, uint increment) virtual public onlyFactory returns (uint quota) {
        quota = authQuotaOf[signatory].add(increment);
        authQuotaOf[signatory] = quota; //knownsec// Update the quota of the specified address
        emit IncreaseAuthQuota(signatory, increment, quota);
    }
    event IncreaseAuthQuota(address indexed signatory, uint increment, uint quota);
    //knownsec// Reduce the quota corresponding to the specified address, if not enough, it will be 0
function decreaseAuthQuotas(address[] memory signatorys, uint[] memory decrements) virtual
```

```
external returns (uint[] memory quotas) {
        require(signatorys.length == decrements.length, 'two array lenth not equal');
        quotas = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            quotas[i] = decreaseAuthQuota(signatorys[i], decrements[i]);
    }

    function decreaseAuthQuota(address signatory, uint decrement) virtual public onlyFactory
returns (uint quota) {
        quota = authQuotaOf[signatory];
        if(quota < decrement)//knownsec// If the quota of the specified address is less than the
reduced value, the reduced value is set to the current value
            decrement = quota;
        return _decreaseAuthQuota(signatory, decrement);
    }

    function _decreaseAuthQuota(address signatory, uint decrement) virtual internal returns (uint
quota) {
        quota = authQuotaOf[signatory].sub(decrement);
        authQuotaOf[signatory] = quota;//knownsec// Update quota for specified address
        emit DecreaseAuthQuota(signatory, decrement, quota);
    }
    event DecreaseAuthQuota(address indexed signatory, uint decrement, uint quota);
```

Recommendation： nothing.

3.3. MappingBase contract send function 【PASS】

Audit analysis: The send function of the MappingBase contract is used by the user to exchange the token of Ethereum with the token of the specified chain, and pass in other chain ids, to addresses and exchange amounts.

```
    function send(uint toChainId, address to, uint volume) virtual external payable returns (uint nonce) {
        return sendFrom(_msgSender(), toChainId, to, volume);
    }

    function sendFrom(address from, uint toChainId, address to, uint volume) virtual public payable returns (uint nonce) {
        _chargeFee();//knownsec// Call internal functions to charge transaction fees
        _sendFrom(from, volume);//knownsec// Call the internal function _sendFrom for token transfer
        //knownsec// Record user sent data
        nonce = sentCount[toChainId][to]++;
        sent[toChainId][to][nonce] = volume;
        emit Send(from, toChainId, to, nonce, volume);
    }
```

Recommendation: nothing.

3.4. MappingBase contract receive function【PASS】

Audit analysis: The receive in the MappingBase contract is used to accept tokens exchanged from other chains and requires verification information from more than 3 nodes.

```
    function receive(uint256 fromChainId, address to, uint256 nonce, uint256 volume, Signature[] memory signatures) virtual external payable {
        _chargeFee();//knownsec// Call internal functions to charge transaction fees
        require(received[fromChainId][to][nonce] == 0, 'withdrawn already');
        uint N = signatures.length;
        require(N >= MappingTokenFactory(factory).getConfig(_minSignatures_), 'too few signatures');//knownsec// The number of signatures required to be greater than or equal to the configured minimum
        for(uint i=0; i<N; i++) {
            //knownsec// Verify signature information
            bytes32 structHash = keccak256(abi.encode(RECEIVE_TYPEHASH, fromChainId, to, nonce, volume, signatures[i].signatory));
            bytes32 digest = keccak256(abi.encodePacked("\x19\x01", _DOMAIN_SEPARATOR, structHash));
            address signatory = ecrecover(digest, signatures[i].v, signatures[i].r, signatures[i].s);
            require(signatory != address(0), "invalid signature");
            require(signatory == signatures[i].signatory, "unauthorized");
            _decreaseAuthQuota(signatures[i].signatory, volume);//knownsec// Call the internal function _decreaseAuthQuota to specify the quota corresponding to the address
            emit Authorize(fromChainId, to, nonce, volume, signatory);
        }
        received[fromChainId][to][nonce] = volume; //knownsec// Update the value corresponding to the nonce of the specified chain to address
        _receive(to, volume);//knownsec// Call the internal function _receive for token transfer
        emit Receive(fromChainId, to, nonce, volume);
    }
    event Receive(uint256 indexed fromChainId, address indexed to, uint256 indexed nonce, uint256 volume);
```

Recommendation: nothing.

3.5. The MappingTokenFactory contract increases and reduces the number of times the specified address signatory can be verified【PASS】

Audit analysis: The authorized caller in the MappingTokenFactory contract can call the following functions to increase or decrease the number of verifications for the specified address.

```
    //knownsec// Increase the number of verifications for specified addresses in batches
    function increaseAuthCount(address[] memory signatorys, uint[] memory increments) virtual external returns (uint[] memory counts) {
        require(signatorys.length == increments.length, 'two array lenth not equal');
        counts = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            counts[i] = increaseAuthCount(signatorys[i], increments[i]);
    }
    //knownsec// Increase the number of verifications for the specified address
function increaseAuthCount(address signatory, uint increment) virtual public onlyAuthorty returns (uint count) {
//knownsec// Reduce the number of signatory verifications for the specified address
        count = authCountOf[signatory].add(increment);
        authCountOf[signatory] = count;
        emit IncreaseAuthQuota(_msgSender(), signatory, increment, count);
    }
    event IncreaseAuthQuota(address indexed authorty, address indexed signatory, uint increment, uint quota);
    //knownsec// Reduce the number of verifications for specified addresses in batches
    function decreaseAuthCounts(address[] memory signatorys, uint[] memory decrements) virtual external returns (uint[] memory counts) {
        require(signatorys.length == decrements.length, 'two array lenth not equal');
        counts = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            counts[i] = decreaseAuthCount(signatorys[i], decrements[i]);
    }
    //knownsec// Reduce the number of verifications for the specified address
    function decreaseAuthCount(address signatory, uint decrement) virtual public onlyAuthorty returns (uint count) {
        count = authCountOf[signatory];
        if(count < decrement) //knownsec// If the current number of verifications of the specified signatory is less than the reduced value, then modify the number of signatory verifications of the specified address to reduce the number of times
            decrement = count;
        return _decreaseAuthCount(signatory, decrement); //knownsec// Call the internal function _decreaseAuthCount to update the number of signatory verifications at the specified
```

```
address
    }
function _decreaseAuthCount(address signatory, uint decrement) virtual internal returns (uint count) {
    //knownsec// Reduce the number of signatory verifications for the specified address
        count = authCountOf[signatory].sub(decrement);
        authCountOf[signatory] = count;
        emit DecreaseAuthCount(_msgSender(), signatory, decrement, count);
    }
```

Recommendation： nothing.

3.6. MappingTokenFactory contract registration token mapping function【PASS】

Audit analysis: Register the token mapping by calling the registerMapping function by the governor of the MappingTokenFactory contract or through node verification.

```
    function _registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory mappingTokenMappeds_) virtual internal {
        require(chainId() == 1 || chainId() == 3, 'called only on ethereum mainnet');
        require(chainIds.length == mappingTokenMappeds_.length, 'two array lenth not equal');
        require(isSupportChainId(mainChainId), 'Not support mainChainId');
        for(uint i=0; i<chainIds.length; i++) {
            require(isSupportChainId(chainIds[i]), 'Not support chainId');
            require(_mainChainIdTokens[mappingTokenMappeds_[i]] == 0 || _mainChainIdTokens[mappingTokenMappeds_[i]] == (mainChainId << 160) | uint(token), 'mainChainIdTokens exist already');
            require(mappingTokenMappeds[token][chainIds[i]] == address(0), 'mappingTokenMappeds exist already');//knownsec// Require token => chainId => mappingToken or tokenMapped does not exist
            if(_mainChainIdTokens[mappingTokenMappeds_[i]] == 0)//knownsec// If the chain id+token value corresponding to the token address specified by _mainChainIdTokens is 0
```

```
                _mainChainIdTokens[mappingTokenMappeds_[i]] = (mainChainId << 160) | uint(token);//knownsec// Record the chain id+token corresponding to the specified token address
            mappingTokenMappeds[token][chainIds[i]] = mappingTokenMappeds_[i];//knownsec// Record token => chainId => mappingToken or tokenMapped data
            emit RegisterMapping(mainChainId, token, chainIds[i], mappingTokenMappeds_[i]);
        }
    }
    event RegisterMapping(uint mainChainId, address token, uint chainId, address mappingTokenMapped);
//knownsec// governor registered token mapping
    function registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory mappingTokenMappeds_) virtual external governance {
        _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
    }
//knownsec// Register token mapping by signing
    function registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory mappingTokenMappeds_, Signature[] memory signatures) virtual external payable {
        _chargeFee();
        uint N = signatures.length;
        require(N >= getConfig(_minSignatures_), 'too few signatures');
        for(uint i=0; i<N; i++) {
        //knownsec// Signature verification
            bytes32 structHash = keccak256(abi.encode(REGISTER_TYPEHASH, mainChainId, token, chainIds, mappingTokenMappeds_, signatures[i].signatory));
            bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, structHash));
            address signatory = ecrecover(digest, signatures[i].v, signatures[i].r, signatures[i].s);
            require(signatory != address(0), "invalid signature");
            require(signatory == signatures[i].signatory, "unauthorized");
            _decreaseAuthCount(signatures[i].signatory, 1);//knownsec// Update the
```

```
verification times of the specified address
                emit AuthorizeRegister(mainChainId, token, signatory);
            }
            _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
        }
        event AuthorizeRegister(uint indexed mainChainId, address indexed token, address indexed
signatory);
```

Recommendation: nothing.

3.7. MappingTokenFactory contract creation contract function【PASS】

Audit analysis: Take createTokenMapped in the MappingTokenFactory contract as an example, the caller can call this function to create a tokenMapped contract and initialize it through the same contract.

```
    function createTokenMapped(address token) external payable returns (address tokenMapped) {
            _chargeFee(); //knownsec// Call the internal function _chargeFee to charge a fee
            IERC20(token).totalSupply();//knownsec// Get the total amount of tokens from the token
address                                                                     // just for check
            require(tokenMappeds[token]      ==      address(0),      'TokenMapped      created
already');//knownsec// requires the token address to be created for the first time

            bytes32 salt = keccak256(abi.encodePacked(chainId(), token));

            bytes memory bytecode = type(InitializableProductProxy).creationCode;
            assembly {
                tokenMapped := create2(0, add(bytecode, 32), mload(bytecode), salt)
            }

InitializableProductProxy(payable(tokenMapped)).__InitializableProductProxy_init(address(    th
is), _TokenMapped_, abi.encodeWithSignature('__TokenMapped_init(address)', token));
```

```
        tokenMappeds[token] = tokenMapped;//knownsec// Record the tokenMapped address corresponding to the token
        emit CreateTokenMapped(_msgSender(), token, tokenMapped);
    }
    event CreateTokenMapped(address indexed creator, address indexed token, address indexed tokenMapped);
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the smart contract code has formulated the compiler version 0.6.0 within the major version, and there is no such security problem.

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.6. Fallback function safety【PASS】

Check whether the fallback function is used correctly in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.7. tx.origin authentication【PASS】

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.9. Gas consumption detection【PASS】

Check whether the consumption of gas exceeds the maximum block limit.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Audit result: After testing, the smart contract code has the function of issuing additional tokens, but due to the setting of the upper limit of tokens, it passed.

Recommendation: nothing.

4.13. Access control defect detection 【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.14. Numerical overflow detection 【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation: nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the smart contract code does not use structure, there is no such problem.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send ETH to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to ETH sending failure.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Audit result： After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send ETH. When the **call.value()** function to send ETH occurs before the actual reduction of the sender’s account balance, There is a risk of reentry attacks.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

5. Appendix A：Contract code

Source code：

MappingTokenFactory .sol

```
/**
  *Submitted for verification at Etherscan.io on 2021-03-12
*/

// SPDX-License-Identifier: MIT

pragma solidity ^0.6.0;
pragma experimental ABIEncoderV2;

/**
  * @title Proxy
  * @dev Implements delegation of calls to other contracts, with proper
  * forwarding of return values and bubbling of failures.
  * It defines a fallback function that delegates all calls to the address
  * returned by the abstract _implementation() internal function.
  */
abstract contract Proxy {
    /**
      * @dev Fallback function.
      * Implemented entirely in `_fallback`.
      */
    fallback () payable external {
        _fallback();
    }

    receive () payable external {
        _fallback();
    }

    /**
      * @return The Address of the implementation.
      */
    function _implementation() virtual internal view returns (address);

    /**
      * @dev Delegates execution to an implementation contract.
      * This is a low level function that doesn't return to its internal call site.
      * It will return to the external caller whatever the implementation returns.
      * @param implementation Address to delegate.
      */
    function _delegate(address implementation) internal {
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /**
      * @dev Function that is run as the first thing in the fallback function.
      * Can be redefined in derived contracts to add functionality.
      * Redefinitions must call super._willFallback().
      */
    function _willFallback() virtual internal {

    }

    /**
      * @dev fallback implementation.
      * Extracted to enable manual triggering.
      */
    function _fallback() internal {
        if(OpenZeppelinUpgradesAddress.isContract(msg.sender) && msg.data.length == 0 && gasleft() <= 2300)
// for receive ETH only from other contract
            return;
        _willFallback();
        _delegate(_implementation());
    }
}


/**
  * @title BaseUpgradeabilityProxy
  * @dev This contract implements a proxy that allows to change the
  * implementation address to which it will delegate.
  * Such a change is called an implementation upgrade.
  */
abstract contract BaseUpgradeabilityProxy is Proxy {
    /**
     * @dev Emitted when the implementation is upgraded.
     * @param implementation Address of the new implementation.
     */
    event Upgraded(address indexed implementation);

    /**
     * @dev Storage slot with the address of the current implementation.
     * This is the keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32        internal        constant        IMPLEMENTATION_SLOT        =
0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc;

    /**
      * @dev Returns the current implementation.
      * @return impl Address of the current implementation
      */
    function _implementation() override internal view returns (address impl) {
        bytes32 slot = IMPLEMENTATION_SLOT;
        assembly {
            impl := sload(slot)
        }
    }

    /**
      * @dev Upgrades the proxy to a new implementation.
      * @param newImplementation Address of the new implementation.
      */
    function _upgradeTo(address newImplementation) internal {
        _setImplementation(newImplementation);
        emit Upgraded(newImplementation);
    }

    /**
      * @dev Sets the implementation address of the proxy.
      * @param newImplementation Address of the new implementation.
      */
    function _setImplementation(address newImplementation) internal {
        require(OpenZeppelinUpgradesAddress.isContract(newImplementation), "Cannot set a proxy implementation
to a non-contract address");

        bytes32 slot = IMPLEMENTATION_SLOT;

        assembly {
            sstore(slot, newImplementation)
        }
    }
}


/**
  * @title BaseAdminUpgradeabilityProxy
  * @dev This contract combines an upgradeability proxy with an authorization
  * mechanism for administrative tasks.
  * All external functions in this contract must be guarded by the
  * `ifAdmin` modifier. See ethereum/solidity#3864 for a Solidity
  * feature proposal that would enable this to be done automatically.
  */
contract BaseAdminUpgradeabilityProxy is BaseUpgradeabilityProxy {
    /**
      * @dev Emitted when the administration has been transferred.
      * @param previousAdmin Address of the previous admin.
      * @param newAdmin Address of the new admin.
      */
    event AdminChanged(address previousAdmin, address newAdmin);

    /**
      * @dev Storage slot with the admin of the contract.
      * This is the keccak-256 hash of "eip1967.proxy.admin" subtracted by 1, and is
      * validated in the constructor.
      */
```

```
  bytes32 internal constant ADMIN_SLOT =
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

  /**
   * @dev Modifier to check whether the `msg.sender` is the admin.
   * If it is, it will run the function. Otherwise, it will delegate the call
   * to the implementation.
   */
  modifier ifAdmin() {
    if (msg.sender == _admin()) {
      _;
    } else {
      _fallback();
    }
  }

  /**
   * @return The address of the proxy admin.
   */
  function admin() external ifAdmin returns (address) {
    return _admin();
  }

  /**
   * @return The address of the implementation.
   */
  function implementation() external ifAdmin returns (address) {
    return _implementation();
  }

  /**
   * @dev Changes the admin of the proxy.
   * Only the current admin can call this function.
   * @param newAdmin Address to transfer proxy administration to.
   */
  function changeAdmin(address newAdmin) external ifAdmin {
    require(newAdmin != address(0), "Cannot change the admin of a proxy to the zero address");
    emit AdminChanged(_admin(), newAdmin);
    _setAdmin(newAdmin);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy.
   * Only the admin can call this function.
   * @param newImplementation Address of the new implementation.
   */
  function upgradeTo(address newImplementation) external ifAdmin {
    _upgradeTo(newImplementation);
  }

  /**
   * @dev Upgrade the backing implementation of the proxy and call a function
   * on the new implementation.
   * This is useful to initialize the proxied contract.
   * @param newImplementation Address of the new implementation.
   * @param data Data to send as msg.data in the low level call.
   * It should include the signature and the parameters of the function to be called, as described in
   * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
   */
  function upgradeToAndCall(address newImplementation, bytes calldata data) payable external ifAdmin {
    _upgradeTo(newImplementation);
    (bool success,) = newImplementation.delegatecall(data);
    require(success);
  }

  /**
   * @return adm The admin slot.
   */
  function _admin() internal view returns (address adm) {
    bytes32 slot = ADMIN_SLOT;
    assembly {
      adm := sload(slot)
    }
  }

  /**
   * @dev Sets the address of the proxy admin.
   * @param newAdmin Address of the new proxy admin.
   */
  function _setAdmin(address newAdmin) internal {
    bytes32 slot = ADMIN_SLOT;

    assembly {
      sstore(slot, newAdmin)
    }
  }
```

```
    /**
     * @dev Only fall back when the sender is not the admin.
     */
    function _willFallback() virtual override internal {
        require(msg.sender != _admin(), "Cannot call fallback function from the proxy admin");
        //super._willFallback();
    }
}

interface IAdminUpgradeabilityProxyView {
    function admin() external view returns (address);
    function implementation() external view returns (address);
}


/**
 * @title UpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with a constructor for initializing
 * implementation and init data.
 */
abstract contract UpgradeabilityProxy is BaseUpgradeabilityProxy {
    /**
     * @dev Contract constructor.
     * @param _logic Address of the initial implementation.
     * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
     * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
     */
    constructor(address _logic, bytes memory _data) public payable {
        assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
        _setImplementation(_logic);
        if(_data.length > 0) {
            (bool success,) = _logic.delegatecall(_data);
            require(success);
        }
    }

    //function _willFallback() virtual override internal {
        //super._willFallback();
    //}
}


/**
 * @title AdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with a constructor for
 * initializing the implementation, admin, and init data.
 */
contract AdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy, UpgradeabilityProxy {
    /**
     * Contract constructor.
     * @param _logic address of the initial implementation.
     * @param _admin Address of the proxy administrator.
     * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
     * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
     */
    constructor(address _admin, address _logic, bytes memory _data) UpgradeabilityProxy(_logic, _data) public
payable {
        assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
        _setAdmin(_admin);
    }

    function _willFallback() override(Proxy, BaseAdminUpgradeabilityProxy) internal {
        super._willFallback();
    }
}


/**
 * @title InitializableUpgradeabilityProxy
 * @dev Extends BaseUpgradeabilityProxy with an initializer for initializing
 * implementation and init data.
 */
abstract contract InitializableUpgradeabilityProxy is BaseUpgradeabilityProxy {
    /**
     * @dev Contract initializer.
     * @param _logic Address of the initial implementation.
     * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
     * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
     */
    function initialize(address _logic, bytes memory _data) public payable {
        require(_implementation() == address(0));
        assert(IMPLEMENTATION_SLOT == bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1));
        _setImplementation(_logic);
        if(_data.length > 0) {
            (bool success,) = _logic.delegatecall(_data);
            require(success);
        }
    }
}


/**
 * @title InitializableAdminUpgradeabilityProxy
 * @dev Extends from BaseAdminUpgradeabilityProxy with an initializer for
 * initializing the implementation, admin, and init data.
 */
contract InitializableAdminUpgradeabilityProxy is BaseAdminUpgradeabilityProxy,
InitializableUpgradeabilityProxy {
    /**
     * Contract initializer.
     * @param _logic address of the initial implementation.
     * @param _admin Address of the proxy administrator.
     * @param _data Data to send as msg.data to the implementation to initialize the proxied contract.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
     * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
     */
    function initialize(address _admin, address _logic, bytes memory _data) public payable {
        require(_implementation() == address(0));
        InitializableUpgradeabilityProxy.initialize(_logic, _data);
        assert(ADMIN_SLOT == bytes32(uint256(keccak256('eip1967.proxy.admin')) - 1));
        _setAdmin(_admin);
    }

    function _willFallback() override(Proxy, BaseAdminUpgradeabilityProxy) internal {
        super._willFallback();
    }

}


interface IProxyFactory {
        function productImplementation() external view returns (address);
        function productImplementations(bytes32 name) external view returns (address);
}


/**
 * @title ProductProxy
 * @dev This contract implements a proxy that
 * it is deploied by ProxyFactory,
 * and it's implementation is stored in factory.
 */
contract ProductProxy is Proxy {

    /**
     * @dev Storage slot with the address of the ProxyFactory.
     * This is the keccak-256 hash of "eip1967.proxy.factory" subtracted by 1, and is
     * validated in the constructor.
     */
    bytes32 internal constant FACTORY_SLOT =
0x7a45a402e4cb6e08ebc196f20f66d5d30e67285a2a8aa80503fa409e727a4af1;
    bytes32 internal constant NAME_SLOT =
0x4cd9b827ca535ceb0880425d70eff88561ecdf04dc32fcf7ff3b15c587f8a870; //
bytes32(uint256(keccak256('eip1967.proxy.name')) - 1)

    function _name() virtual internal view returns (bytes32 name_) {
        bytes32 slot = NAME_SLOT;
        assembly {  name_ := sload(slot)  }
    }

    function _setName(bytes32 name_) internal {
        bytes32 slot = NAME_SLOT;
        assembly {  sstore(slot, name_)  }
    }

    /**
     * @dev Sets the factory address of the ProductProxy.
     * @param newFactory Address of the new factory.
     */
    function _setFactory(address newFactory) internal {
        require(OpenZeppelinUpgradesAddress.isContract(newFactory), "Cannot set a factory to a non-contract
address");

        bytes32 slot = FACTORY_SLOT;

        assembly {
            sstore(slot, newFactory)
        }
    }

    /**
     * @dev Returns the factory.
     * @return factory_ Address of the factory.
     */
    function _factory() internal view returns (address factory_) {
        bytes32 slot = FACTORY_SLOT;
        assembly {
            factory_ := sload(slot)
        }
    }

    /**
     * @dev Returns the current implementation.
     * @return Address of the current implementation
     */
    function _implementation() virtual override internal view returns (address) {
        address factory_ = _factory();
        if(OpenZeppelinUpgradesAddress.isContract(factory_))
            return IProxyFactory(factory_).productImplementations(_name());
        else
            return address(0);
    }

}


/**
 * @title InitializableProductProxy
 * @dev Extends ProductProxy with an initializer for initializing
 * factory and init data.
 */
contract InitializableProductProxy is ProductProxy {
    /**
     * @dev Contract initializer.
     * @param factory_ Address of the initial factory.
     * @param data_ Data to send as msg.data to the implementation to initialize the proxied contract.
     * It should include the signature and the parameters of the function to be called, as described in
     * https://solidity.readthedocs.io/en/v0.4.24/abi-spec.html#function-selector-and-argument-encoding.
     * This parameter is optional, if no data is given the initialization call to proxied contract will be skipped.
     */
    function __InitializableProductProxy_init(address factory_, bytes32 name_, bytes memory data_) public payable
{
        require(_factory() == address(0));
        assert(FACTORY_SLOT == bytes32(uint256(keccak256('eip1967.proxy.factory')) - 1));
        assert(NAME_SLOT        == bytes32(uint256(keccak256('eip1967.proxy.name')) - 1));
        _setFactory(factory_);
        _setName(name_);
        if(data_.length > 0) {
            (bool success,) = _implementation().delegatecall(data_);
            require(success);
        }
    }
}


/**
 * @title Initializable
 *
 * @dev Helper contract to support initializer functions. To use it, replace
 * the constructor with a function that has the `initializer` modifier.
 * WARNING: Unlike constructors, initializer functions must be manually
 * invoked. This applies both to deploying an Initializable contract, as well
 * as extending an Initializable contract via inheritance.
 * WARNING: When used with inheritance, manual care must be taken to not invoke
 * a parent initializer twice, or ensure that all initializers are idempotent,
 * because this is not dealt with automatically as with constructors.
 */
contract Initializable {

    /**
     * @dev Indicates that the contract has been initialized.
     */
    bool private initialized;

    /**
     * @dev Indicates that the contract is in the process of being initialized.
     */
    bool private initializing;

    /**
     * @dev Modifier to use in the initializer function of a contract.
     */
    modifier initializer() {
        require(initializing || isConstructor() || !initialized, "Contract instance has already been initialized");

        bool isTopLevelCall = !initializing;
        if (isTopLevelCall) {
            initializing = true;
            initialized = true;
        }

        _;

        if (isTopLevelCall) {
            initializing = false;
        }
    }

    /// @dev Returns true if and only if the function is running in the constructor
    function isConstructor() private view returns (bool) {
        // extcodesize checks the size of the code stored in an address, and
        // address returns the current address. Since the code is still not
        // deployed when running a constructor, any checks on its code size will
        // yield zero, making it an effective way to detect if a contract is
        // under construction or not.
        address self = address(this);
        uint256 cs;
        assembly { cs := extcodesize(self) }
        return cs == 0;
    }

    // Reserved storage space to allow for layout changes in the future.
    uint256[50] private ______gap;
}


/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract ContextUpgradeSafe is Initializable {
        // Empty internal constructor, to prevent people from mistakenly deploying
        // an instance of this contract, which should be used via inheritance.

        function __Context_init() internal initializer {
                __Context_init_unchained();
        }

        function __Context_init_unchained() internal initializer {


        }


        function _msgSender() internal view virtual returns (address payable) {
                return msg.sender;
        }

        function _msgData() internal view virtual returns (bytes memory) {
                this; // silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
                return msg.data;
        }

        uint256[50] private __gap;
}

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
        /**
         * @dev Returns the largest of two numbers.
         */
        function max(uint256 a, uint256 b) internal pure returns (uint256) {
                return a >= b ? a : b;
        }

        /**
         * @dev Returns the smallest of two numbers.
         */
        function min(uint256 a, uint256 b) internal pure returns (uint256) {
                return a < b ? a : b;
        }
    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function sub0(uint256 a, uint256 b) internal pure returns (uint256) {
        return a > b ? a - b : 0;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

/**
 * Utility library of inline functions on addresses
 *
 * Source https://raw.githubusercontent.com/OpenZeppelin/openzeppelin-
solidity/v2.1.3/contracts/utils/Address.sol
 * This contract is copied here and renamed from the original to avoid clashes in the compiled artifacts
 * when the user imports a zos-lib contract (that transitively causes this contract to be compiled and added to the
 * build/artifacts folder) as well as the vanilla Address implementation from an openzeppelin version.
 */
library OpenZeppelinUpgradesAddress {
    /**
     * Returns whether the target address is a contract
     * @dev This function will return false if invoked during the constructor of a contract,
     * as the code is not actually created until after the constructor finishes.
     * @param account address of the account to check
     * @return whether the target address is a contract
     */
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        // XXX Currently there is no better way to check if there is a contract in an address
        // than to check the size of the code at that address.
        // See https://ethereum.stackexchange.com/a/14016/36603
        // for more details about how this works.
        // TODO Check this again before the Serenity release, because all addresses will be
        // contracts then.
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
}

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20MinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20UpgradeSafe is Initializable, ContextUpgradeSafe, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowances;

    uint256 private _totalSupply;

    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
     * a default value of 18.
     *
     * To select a different value for {decimals}, use {_setupDecimals}.
     *
     * All three of these values are immutable: they can only be set once during
     * construction.
     */

    function __ERC20_init(string memory name, string memory symbol) internal initializer {
        __Context_init_unchained();
        __ERC20_init_unchained(name, symbol);
    }

    function __ERC20_init_unchained(string memory name, string memory symbol) internal initializer {


        _name = name;
        _symbol = symbol;
        _decimals = 18;

    }


    /**
     * @dev Returns the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
     * called.
     *
     * NOTE: This information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * {IERC20-balanceOf} and {IERC20-transfer}.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }

    /**
     * @dev See {IERC20-totalSupply}.
     */
    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See {IERC20-balanceOf}.
     */
    function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
    }
    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    /**
     * @dev See {IERC20-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool)
{
        _transfer(sender, recipient, amount);
        if(sender != _msgSender() && _allowances[sender][_msgSender()] != uint(-1))
            _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20:
transfer amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
```

```
decreased allowance below zero"));
            return true;
      }

      /**
        * @dev Moves tokens `amount` from `sender` to `recipient`.
        *
        * This is internal function is equivalent to {transfer}, and can be used to
        * e.g. implement automatic token fees, slashing mechanisms, etc.
        *
        * Emits a {Transfer} event.
        *
        * Requirements:
        *
        * - `sender` cannot be the zero address.
        * - `recipient` cannot be the zero address.
        * - `sender` must have a balance of at least `amount`.
        */
      function _transfer(address sender, address recipient, uint256 amount) internal virtual {
            require(sender != address(0), "ERC20: transfer from the zero address");
            require(recipient != address(0), "ERC20: transfer to the zero address");

            _beforeTokenTransfer(sender, recipient, amount);

            _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
            _balances[recipient] = _balances[recipient].add(amount);
            emit Transfer(sender, recipient, amount);
      }

      /** @dev Creates `amount` tokens and assigns them to `account`, increasing
        * the total supply.
        *
        * Emits a {Transfer} event with `from` set to the zero address.
        *
        * Requirements
        *
        * - `to` cannot be the zero address.
        */
      function _mint(address account, uint256 amount) internal virtual {
            require(account != address(0), "ERC20: mint to the zero address");

            _beforeTokenTransfer(address(0), account, amount);

            _totalSupply = _totalSupply.add(amount);
            _balances[account] = _balances[account].add(amount);
            emit Transfer(address(0), account, amount);
      }

      /**
        * @dev Destroys `amount` tokens from `account`, reducing the
        * total supply.
        *
        * Emits a {Transfer} event with `to` set to the zero address.
        *
        * Requirements
        *
        * - `account` cannot be the zero address.
        * - `account` must have at least `amount` tokens.
        */
      function _burn(address account, uint256 amount) internal virtual {
            require(account != address(0), "ERC20: burn from the zero address");

            _beforeTokenTransfer(account, address(0), amount);

            _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
            _totalSupply = _totalSupply.sub(amount);
            emit Transfer(account, address(0), amount);
      }

      /**
        * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
        *
        * This is internal function is equivalent to `approve`, and can be used to
        * e.g. set automatic allowances for certain subsystems, etc.
        *
        * Emits an {Approval} event.
        *
        * Requirements:
        *
        * - `owner` cannot be the zero address.
        * - `spender` cannot be the zero address.
        */
      function _approve(address owner, address spender, uint256 amount) internal virtual {
            require(owner != address(0), "ERC20: approve from the zero address");
            require(spender != address(0), "ERC20: approve to the zero address");

            _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual { }

    uint256[44] private __gap;
}


/**
 * @dev Extension of {ERC20} that adds a cap to the supply of tokens.
 */
abstract contract ERC20CappedUpgradeSafe is Initializable, ERC20UpgradeSafe {
    uint256 private _cap;

    /**
     * @dev Sets the value of the `cap`. This value is immutable, it can only be
     * set once during construction.
     */

    function __ERC20Capped_init(uint256 cap) internal initializer {
        __Context_init_unchained();
        __ERC20Capped_init_unchained(cap);
    }

    function __ERC20Capped_init_unchained(uint256 cap) internal initializer {


        require(cap > 0, "ERC20Capped: cap is 0");
        _cap = cap;

    }


    /**
     * @dev Returns the cap on the token's total supply.
     */
    function cap() public view returns (uint256) {
        return _cap;
    }

    /**
     * @dev See {ERC20-_beforeTokenTransfer}.
     *
     * Requirements:
     *
     * - minted tokens must not cause the total supply to go over the cap.
     */
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override {
        super._beforeTokenTransfer(from, to, amount);

        if (from == address(0)) { // When minting tokens
            require(totalSupply().add(amount) <= _cap, "ERC20Capped: cap exceeded");
        }
    }

    uint256[49] private __gap;
}


/**
 * @title SafeERC20
```

```
  * @dev Wrappers around ERC20 operations that throw on failure (when the token
  * contract returns false). Tokens that return no value (and instead revert or
  * throw on failure) are also supported, non-reverting calls are assumed to be
  * successful.
  * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
  * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
  */
library SafeERC20 {
        using SafeMath for uint256;
        using Address for address;

        function safeTransfer(IERC20 token, address to, uint256 value) internal {
                _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
        }

        function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
                _callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
        }

        function safeApprove(IERC20 token, address spender, uint256 value) internal {
                // safeApprove should only be called when setting an initial allowance,
                // or when resetting it to zero. To increase and decrease it, use
                // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
                // solhint-disable-next-line max-line-length
                require((value == 0) || (token.allowance(address(this), spender) == 0),
                        "SafeERC20: approve from non-zero to non-zero allowance"
                );
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
        }

        function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256 newAllowance = token.allowance(address(this), spender).add(value);
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
        }

        function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256  newAllowance  =  token.allowance(address(this),  spender).sub(value,  "SafeERC20:  decreased
allowance below zero");
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
        }

        /**
          * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the requirement
          * on the return value: the return value is optional (but if data is returned, it must not be false).
          * @param token The token targeted by the call.
          * @param data The call data (encoded using abi.encode or one of its variants).
          */
        function _callOptionalReturn(IERC20 token, bytes memory data) private {
                // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism, since
                // we're implementing it ourselves.

                // A Solidity high level call has three parts:
                //   1. The target address is checked to verify it contains contract code
                //   2. The call itself is made, and success asserted
                //   3. The return value is decoded, which in turn checks the size of the returned data.
                // solhint-disable-next-line max-line-length
                require(address(token).isContract(), "SafeERC20: call to non-contract");

                // solhint-disable-next-line avoid-low-level-calls
                (bool success, bytes memory returndata) = address(token).call(data);
                require(success, "SafeERC20: low-level call failed");

                if (returndata.length > 0) { // Return data is optional
                        // solhint-disable-next-line max-line-length
                        require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
                }
        }
}


contract Governable is Initializable {
        address public governor;

        event GovernorshipTransferred(address indexed previousGovernor, address indexed newGovernor);

        /**
          * @dev Contract initializer.
          * called once by the factory at time of deployment
          */
        function __Governable_init_unchained(address governor_) virtual public initializer {
                governor = governor_;
                emit GovernorshipTransferred(address(0), governor);
        }

        modifier governance() {
                require(msg.sender == governor);
                _;
```

```
    }

    /**
     * @dev Allows the current governor to relinquish control of the contract.
     * @notice Renouncing to governorship will leave the contract without an governor.
     * It will not be possible to call the functions with the `governance`
     * modifier anymore.
     */
    function renounceGovernorship() public governance {
        emit GovernorshipTransferred(governor, address(0));
        governor = address(0);
    }

    /**
     * @dev Allows the current governor to transfer control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function transferGovernorship(address newGovernor) public governance {
        _transferGovernorship(newGovernor);
    }

    /**
     * @dev Transfers control of the contract to a newGovernor.
     * @param newGovernor The address to transfer governorship to.
     */
    function _transferGovernorship(address newGovernor) internal {
        require(newGovernor != address(0));
        emit GovernorshipTransferred(governor, newGovernor);
        governor = newGovernor;
    }
}


contract Configurable is Governable {

    mapping (bytes32 => uint) internal config;

    function getConfig(bytes32 key) public view returns (uint) {
        return config[key];
    }
    function getConfig(bytes32 key, uint index) public view returns (uint) {
        return config[bytes32(uint(key) ^ index)];
    }
    function getConfig(bytes32 key, address addr) public view returns (uint) {
        return config[bytes32(uint(key) ^ uint(addr))];
    }

    function _setConfig(bytes32 key, uint value) internal {
        if(config[key] != value)
            config[key] = value;
    }
    function _setConfig(bytes32 key, uint index, uint value) internal {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function _setConfig(bytes32 key, address addr, uint value) internal {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }

    function setConfig(bytes32 key, uint value) external governance {
        _setConfig(key, value);
    }
    function setConfig(bytes32 key, uint index, uint value) external governance {
        _setConfig(bytes32(uint(key) ^ index), value);
    }
    function setConfig(bytes32 key, address addr, uint value) public governance {
        _setConfig(bytes32(uint(key) ^ uint(addr)), value);
    }
}


contract Constants {
    bytes32 internal constant _TokenMapped_        = 'TokenMapped';
    bytes32 internal constant _MappableToken_      = 'MappableToken';
    bytes32 internal constant _MappingToken_       = 'MappingToken';
    bytes32 internal constant _fee_                = 'fee';
    bytes32 internal constant _feeCreate_          = 'feeCreate';
    bytes32 internal constant _feeTo_              = 'feeTo';
    bytes32 internal constant _minSignatures_      = 'minSignatures';
    bytes32 internal constant _uniswapRounter_     = 'uniswapRounter';

    function chainId() public pure returns (uint id) {
        assembly { id := chainid() }
    }
}

struct Signature {
    address signatory;
```

```
        uint8     v;
        bytes32 r;
        bytes32 s;
}

abstract contract MappingBase is ContextUpgradeSafe, Constants {
         using SafeMath for uint;

        bytes32 public constant RECEIVE_TYPEHASH = keccak256("Receive(uint256 fromChainId,address
to,uint256 nonce,uint256 volume,address signatory)");
        bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256
chainId,address verifyingContract)");
        bytes32 internal _DOMAIN_SEPARATOR;
        function DOMAIN_SEPARATOR() virtual public view returns (bytes32) { return
_DOMAIN_SEPARATOR;  }

        address public factory;
        uint256 public mainChainId;
        address public token;
        address public creator;

        mapping (address => uint) public authQuotaOf;                                    // signatory
=> quota
        mapping (uint => mapping (address => uint)) public sentCount;                    // toChainId =>
to => sentCount
        mapping (uint => mapping (address => mapping (uint => uint))) public sent;       // toChainId => to
=> nonce => volume
        mapping (uint => mapping (address => mapping (uint => uint))) public received;   // fromChainId =>
to => nonce => volume

        modifier onlyFactory {
                require(msg.sender == factory, 'Only called by Factory');
                _;
        }

        function increaseAuthQuotas(address[] memory signatorys, uint[] memory increments) virtual external returns
(uint[] memory quotas) {
                require(signatorys.length == increments.length, 'two array lenth not equal');
                quotas = new uint[](signatorys.length);
                for(uint i=0; i<signatorys.length; i++)
                        quotas[i] = increaseAuthQuota(signatorys[i], increments[i]);
        }

        function increaseAuthQuota(address signatory, uint increment) virtual public onlyFactory returns (uint quota)
{
                quota = authQuotaOf[signatory].add(increment);
                authQuotaOf[signatory] = quota;
                emit IncreaseAuthQuota(signatory, increment, quota);
        }
        event IncreaseAuthQuota(address indexed signatory, uint increment, uint quota);

        function decreaseAuthQuotas(address[] memory signatorys, uint[] memory decrements) virtual external
returns (uint[] memory quotas) {
                require(signatorys.length == decrements.length, 'two array lenth not equal');
                quotas = new uint[](signatorys.length);
                for(uint i=0; i<signatorys.length; i++)
                        quotas[i] = decreaseAuthQuota(signatorys[i], decrements[i]);
        }

        function decreaseAuthQuota(address signatory, uint decrement) virtual public onlyFactory returns (uint quota)
{
                quota = authQuotaOf[signatory];
                if(quota < decrement)
                        decrement = quota;
                return _decreaseAuthQuota(signatory, decrement);
        }

        function _decreaseAuthQuota(address signatory, uint decrement) virtual internal returns (uint quota) {
                quota = authQuotaOf[signatory].sub(decrement);
                authQuotaOf[signatory] = quota;
                emit DecreaseAuthQuota(signatory, decrement, quota);
        }
        event DecreaseAuthQuota(address indexed signatory, uint decrement, uint quota);


        function send(uint toChainId, address to, uint volume) virtual external payable returns (uint nonce) {
                return sendFrom(_msgSender(), toChainId, to, volume);
        }

        function sendFrom(address from, uint toChainId, address to, uint volume) virtual public payable returns (uint
nonce) {
                _chargeFee();
                _sendFrom(from, volume);
                nonce = sentCount[toChainId][to]++;
                sent[toChainId][to][nonce] = volume;
                emit Send(from, toChainId, to, nonce, volume);
        }
```

```
    event Send(address indexed from, uint indexed toChainId, address indexed to, uint nonce, uint volume);

    function _sendFrom(address from, uint volume) virtual internal;

    function receive(uint256 fromChainId, address to, uint256 nonce, uint256 volume, Signature[] memory
signatures) virtual external payable {
        _chargeFee();
        require(received[fromChainId][to][nonce] == 0, 'withdrawn already');
        uint N = signatures.length;
        require(N >= MappingTokenFactory(factory).getConfig(_minSignatures_), 'too few signatures');
        for(uint i=0; i<N; i++) {
            bytes32 structHash = keccak256(abi.encode(RECEIVE_TYPEHASH, fromChainId, to, nonce,
volume, signatures[i].signatory));
            bytes32 digest = keccak256(abi.encodePacked("\x19\x01", _DOMAIN_SEPARATOR, structHash));
            address signatory = ecrecover(digest, signatures[i].v, signatures[i].r, signatures[i].s);
            require(signatory != address(0), "invalid signature");
            require(signatory == signatures[i].signatory, "unauthorized");
            _decreaseAuthQuota(signatures[i].signatory, volume);
            emit Authorize(fromChainId, to, nonce, volume, signatory);
        }
        received[fromChainId][to][nonce] = volume;
        _receive(to, volume);
        emit Receive(fromChainId, to, nonce, volume);
    }
    event Receive(uint256 indexed fromChainId, address indexed to, uint256 indexed nonce, uint256 volume);
    event Authorize(uint256 fromChainId, address indexed to, uint256 indexed nonce, uint256 volume, address
indexed signatory);

    function _receive(address to, uint256 volume) virtual internal;

    function _chargeFee() virtual internal {
        require(msg.value >= MappingTokenFactory(factory).getConfig(_fee_), 'fee is too low');
        address payable feeTo = address(MappingTokenFactory(factory).getConfig(_feeTo_));
        if(feeTo == address(0))
            feeTo = address(uint160(factory));
        feeTo.transfer(msg.value);
        emit ChargeFee(_msgSender(), feeTo, msg.value);
    }
    event ChargeFee(address indexed from, address indexed to, uint value);

    uint256[50] private __gap;
}


contract TokenMapped is MappingBase {
    using SafeERC20 for IERC20;

     function __TokenMapped_init(address token_) external initializer {
        __Context_init_unchained();
          __TokenMapped_init_unchained(token_);
     }

     function __TokenMapped_init_unchained(address token_) public initializer {
        factory = _msgSender();
        mainChainId = _chainId();
        token = token_;
        creator = address(0);
         _DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH,
keccak256(bytes(ERC20UpgradeSafe(token).name())), _chainId(), address(this)));
     }

    function totalMapped() virtual public view returns (uint) {
        return IERC20(token).balanceOf(address(this));
    }

    function _sendFrom(address from, uint volume) virtual override internal {
        IERC20(token).safeTransferFrom(from, address(this), volume);
    }

    function _receive(address to, uint256 volume) virtual override internal {
        IERC20(token).safeTransfer(to, volume);
    }

    uint256[50] private __gap;
}


abstract contract Permit {
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH =
0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    function DOMAIN_SEPARATOR() virtual public view returns (bytes32);

    mapping (address => uint) public nonces;

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external
{
        require(deadline >= block.timestamp, 'permit EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR(),
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++,
deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'permit
INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }

    function _approve(address owner, address spender, uint256 amount) internal virtual;

    uint256[50] private __gap;
}

contract MappableToken is Permit, ERC20UpgradeSafe, MappingBase {
     function __MappableToken_init(address creator_, string memory name_, string memory symbol_, uint8
decimals_, uint256 totalSupply_) external initializer {
        __Context_init_unchained();
          __ERC20_init_unchained(name_, symbol_);
          _setupDecimals(decimals_);
          _mint(creator_, totalSupply_);
          __MappableToken_init_unchained(creator_);
     }

     function __MappableToken_init_unchained(address creator_) public initializer {
        factory = _msgSender();
        mainChainId = chainId();
        token = address(0);
        creator = creator_;
        _DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name())),
chainId(), address(this)));
     }

    function DOMAIN_SEPARATOR() virtual override(Permit, MappingBase) public view returns (bytes32) {
        return MappingBase.DOMAIN_SEPARATOR();
    }

    function _approve(address owner, address spender, uint256 amount) virtual override(Permit,
ERC20UpgradeSafe) internal {
        return ERC20UpgradeSafe._approve(owner, spender, amount);
    }

    function totalMapped() virtual public view returns (uint) {
        return balanceOf(address(this));
    }

    function _sendFrom(address from, uint volume) virtual override internal {
        transferFrom(from, address(this), volume);
    }

    function _receive(address to, uint256 volume) virtual override internal {
        _transfer(address(this), to, volume);
    }

    uint256[50] private __gap;
}


contract MappingToken is Permit, ERC20CappedUpgradeSafe, MappingBase {
     function __MappingToken_init(uint mainChainId_, address token_, address creator_, string memory name_,
string memory symbol_, uint8 decimals_, uint cap_) external initializer {
        __Context_init_unchained();
          __ERC20_init_unchained(name_, symbol_);
          _setupDecimals(decimals_);
          __ERC20Capped_init_unchained(cap_);
          __MappingToken_init_unchained(mainChainId_, token_, creator_);
     }

     function __MappingToken_init_unchained(uint mainChainId_, address token_, address creator_) public
initializer {
        factory = _msgSender();
        mainChainId = mainChainId_;
        token = token_;
        creator = (token_ == address(0)) ? creator_ : address(0);
        _DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name())),
chainId(), address(this)));
     }

    function DOMAIN_SEPARATOR() virtual override(Permit, MappingBase) public view returns (bytes32) {
        return MappingBase.DOMAIN_SEPARATOR();
    }
    function _approve(address owner, address spender, uint256 amount) virtual override(Permit,
ERC20UpgradeSafe) internal {
        return ERC20UpgradeSafe._approve(owner, spender, amount);
    }

    function _sendFrom(address from, uint volume) virtual override internal {
        _burn(from, volume);
        if(from != _msgSender() && allowance(from, _msgSender()) != uint(-1))
            _approve(from, _msgSender(), allowance(from, _msgSender()).sub(volume, "ERC20: transfer
volume exceeds allowance"));
    }

    function _receive(address to, uint256 volume) virtual override internal {
        _mint(to, volume);
    }

    uint256[50] private __gap;
}


contract MappingTokenFactory is ContextUpgradeSafe, Configurable, Constants {
    using SafeERC20 for IERC20;
    using SafeMath for uint;

    bytes32 public constant REGISTER_TYPEHASH = keccak256("RegisterMapping(uint
mainChainId,address token,uint[] chainIds,address[] mappingTokenMappeds_)");
    bytes32 public constant CREATE_TYPEHASH = keccak256("CreateMappingToken(address creator,uint
mainChainId,address token,string name,string symbol,uint8 decimals,uint cap)");
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256
chainId,address verifyingContract)");
    bytes32 public DOMAIN_SEPARATOR;

    mapping (bytes32 => address) public productImplementations;
    mapping (address => address) public tokenMappeds;                // token => tokenMapped
    mapping (address => address) public mappableTokens;              // creator => mappableTokens
    mapping (uint256 => mapping (address => address)) public mappingTokens;     // mainChainId => token
or creator => mappableTokens
    mapping (address => bool) public authorties;

    // only on ethereum mainnet
    mapping (address => uint) public authCountOf;                    // signatory => count
    mapping (address => uint256) internal _mainChainIdTokens;        // mappingToken =>
mainChainId+token
    mapping (address => mapping (uint => address)) public mappingTokenMappeds;   // token => chainId =>
mappingToken or tokenMapped
    uint[] public supportChainIds;
    mapping (string => uint256) internal _certifiedTokens;           // symbol => mainChainId+token
    string[] public certifiedSymbols;

    function __MappingTokenFactory_init(address _governor, address _implTokenMapped, address
_implMappableToken, address _implMappingToken, address _feeTo) external initializer {
        __Governable_init_unchained(_governor);
        __MappingTokenFactory_init_unchained(_implTokenMapped, _implMappableToken,
_implMappingToken, _feeTo);
    }

    function __MappingTokenFactory_init_unchained(address _implTokenMapped, address
_implMappableToken, address _implMappingToken, address _feeTo) public governance {
        config[_fee_]                            = 0.005 ether;
        //config[_feeCreate_]                    = 0.200 ether;
        config[_feeTo_]                          = uint(_feeTo);
        config[_minSignatures_]                  = 3;
        config[_uniswapRounter_]                                                    =
uint(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);

        DOMAIN_SEPARATOR = keccak256(abi.encode(DOMAIN_TYPEHASH,
keccak256(bytes('MappingTokenFactory')), chainId(), address(this)));
        upgradeProductImplementationsTo(_implTokenMapped, _implMappableToken, _implMappingToken);
    }

    function upgradeProductImplementationsTo(address _implTokenMapped, address _implMappableToken,
address _implMappingToken) public governance {
        productImplementations[_TokenMapped_]    = _implTokenMapped;
        productImplementations[_MappableToken_] = _implMappableToken;
        productImplementations[_MappingToken_]  = _implMappingToken;
    }

    function setAuthorty(address authorty, bool enable) virtual external governance {
        authorties[authorty] = enable;
        emit SetAuthorty(authorty, enable);
    }
    event SetAuthorty(address indexed authorty, bool indexed enable);

    modifier onlyAuthorty {
        require(authorties[_msgSender()], 'only authorty');
        _;
    }

    function increaseAuthQuotas(address mappingTokenMapped, address[] memory signatorys, uint[] memory
increments) virtual external onlyAuthorty returns (uint[] memory quotas) {
        quotas = MappingBase(mappingTokenMapped).increaseAuthQuotas(signatorys, increments);
        for(uint i=0; i<signatorys.length; i++)
            emit IncreaseAuthQuota(_msgSender(), mappingTokenMapped, signatorys[i], increments[i],
quotas[i]);
    }

    function increaseAuthQuota(address mappingTokenMapped, address signatory, uint increment) virtual
external onlyAuthorty returns (uint quota) {
        quota = MappingBase(mappingTokenMapped).increaseAuthQuota(signatory, increment);
        emit IncreaseAuthQuota(_msgSender(), mappingTokenMapped, signatory, increment, quota);
    }
    event IncreaseAuthQuota(address indexed authorty, address indexed mappingTokenMapped, address indexed
signatory, uint increment, uint quota);

    function decreaseAuthQuotas(address mappingTokenMapped, address[] memory signatorys, uint[] memory
decrements) virtual external onlyAuthorty returns (uint[] memory quotas) {
        quotas = MappingBase(mappingTokenMapped).decreaseAuthQuotas(signatorys, decrements);
        for(uint i=0; i<signatorys.length; i++)
            emit DecreaseAuthQuota(_msgSender(), mappingTokenMapped, signatorys[i], decrements[i],
quotas[i]);
    }

    function decreaseAuthQuota(address mappingTokenMapped, address signatory, uint decrement) virtual
external onlyAuthorty returns (uint quota) {
        quota = MappingBase(mappingTokenMapped).decreaseAuthQuota(signatory, decrement);
        emit DecreaseAuthQuota(_msgSender(), mappingTokenMapped, signatory, decrement, quota);
    }
    event DecreaseAuthQuota(address indexed authorty, address indexed mappingTokenMapped, address indexed
signatory, uint decrement, uint quota);

    function increaseAuthCount(address[] memory signatorys, uint[] memory increments) virtual external returns
(uint[] memory counts) {
        require(signatorys.length == increments.length, 'two array lenth not equal');
        counts = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            counts[i] = increaseAuthCount(signatorys[i], increments[i]);
    }

    function increaseAuthCount(address signatory, uint increment) virtual public onlyAuthorty returns (uint count)
{
        count = authCountOf[signatory].add(increment);
        authCountOf[signatory] = count;
        emit IncreaseAuthQuota(_msgSender(), signatory, increment, count);
    }
    event IncreaseAuthQuota(address indexed authorty, address indexed signatory, uint increment, uint quota);

    function decreaseAuthCounts(address[] memory signatorys, uint[] memory decrements) virtual external
returns (uint[] memory counts) {
        require(signatorys.length == decrements.length, 'two array lenth not equal');
        counts = new uint[](signatorys.length);
        for(uint i=0; i<signatorys.length; i++)
            counts[i] = decreaseAuthCount(signatorys[i], decrements[i]);
    }

    function decreaseAuthCount(address signatory, uint decrement) virtual public onlyAuthorty returns (uint count)
{
        count = authCountOf[signatory];
        if(count < decrement)
            decrement = count;
        return _decreaseAuthCount(signatory, decrement);
    }

    function _decreaseAuthCount(address signatory, uint decrement) virtual internal returns (uint count) {
        count = authCountOf[signatory].sub(decrement);
        authCountOf[signatory] = count;
        emit DecreaseAuthCount(_msgSender(), signatory, decrement, count);
    }
    event DecreaseAuthCount(address indexed authorty, address indexed signatory, uint decrement, uint count);

    function supportChainCount() public view returns (uint) {
        return supportChainIds.length;
    }

    function mainChainIdTokens(address mappingToken) virtual public view returns(uint mainChainId, address
token) {
        uint256 chainIdToken = _mainChainIdTokens[mappingToken];
        mainChainId = chainIdToken >> 160;
        token = address(chainIdToken);
    }

    function chainIdMappingTokenMappeds(address tokenOrMappingToken) virtual external view returns (uint[]
memory chainIds, address[] memory mappingTokenMappeds_) {
        (, address token) = mainChainIdTokens(tokenOrMappingToken);
```

```
            if(token == address(0))
                token = tokenOrMappingToken;
            uint N = 0;
            for(uint i=0; i<supportChainCount(); i++)
                if(mappingTokenMappeds[token][supportChainIds[i]] != address(0))
                    N++;
            chainIds = new uint[](N);
            mappingTokenMappeds_ = new address[](N);
            uint j = 0;
            for(uint i=0; i<supportChainCount(); i++) {
                uint chainId = supportChainIds[i];
                address mappingTokenMapped = mappingTokenMappeds[token][chainId];
                if(mappingTokenMapped != address(0)) {
                    chainIds[j] = chainId;
                    mappingTokenMappeds_[j] = mappingTokenMapped;
                    j++;
                }
            }
        }

        function isSupportChainId(uint chainId) virtual public view returns (bool) {
            for(uint i=0; i<supportChainCount(); i++)
                if(supportChainIds[i] == chainId)
                    return true;
            return false;
        }

        function registerSupportChainId(uint chainId_) virtual external governance {
            require(chainId() == 1 || chainId() == 3, 'called only on ethereum mainnet');
            require(!isSupportChainId(chainId_), 'support chainId already');
            supportChainIds.push(chainId_);
        }

        function _registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory
mappingTokenMappeds_) virtual internal {
            require(chainId() == 1 || chainId() == 3, 'called only on ethereum mainnet');
            require(chainIds.length == mappingTokenMappeds_.length, 'two array lenth not equal');
            require(isSupportChainId(mainChainId), 'Not support mainChainId');
            for(uint i=0; i<chainIds.length; i++) {
                require(isSupportChainId(chainIds[i]), 'Not support chainId');
                require(_mainChainIdTokens[mappingTokenMappeds_[i]]          ==          0          ||
_mainChainIdTokens[mappingTokenMappeds_[i]] == (mainChainId << 160) | uint(token), 'mainChainIdTokens
exist already');
                require(mappingTokenMappeds[token][chainIds[i]] == address(0), 'mappingTokenMappeds exist
already');
                if(_mainChainIdTokens[mappingTokenMappeds_[i]] == 0)
                    _mainChainIdTokens[mappingTokenMappeds_[i]] = (mainChainId << 160) | uint(token);
                mappingTokenMappeds[token][chainIds[i]] = mappingTokenMappeds_[i];
                emit RegisterMapping(mainChainId, token, chainIds[i], mappingTokenMappeds_[i]);
            }
        }
        event RegisterMapping(uint mainChainId, address token, uint chainId, address mappingTokenMapped);

        function registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory
mappingTokenMappeds_) virtual external governance {
            _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
        }

        function registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory
mappingTokenMappeds_, Signature[] memory signatures) virtual external payable {
            _chargeFee();
            uint N = signatures.length;
            require(N >= getConfig(_minSignatures_), 'too few signatures');
            for(uint i=0; i<N; i++) {
                bytes32   structHash   =   keccak256(abi.encode(REGISTER_TYPEHASH,   mainChainId,   token,
chainIds, mappingTokenMappeds_, signatures[i].signatory));
                bytes32 digest = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, structHash));
                address signatory = ecrecover(digest, signatures[i].v, signatures[i].r, signatures[i].s);
                require(signatory != address(0), "invalid signature");
                require(signatory == signatures[i].signatory, "unauthorized");
                _decreaseAuthCount(signatures[i].signatory, 1);
                emit AuthorizeRegister(mainChainId, token, signatory);
            }
            _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
        }
        event AuthorizeRegister(uint indexed mainChainId, address indexed token, address indexed signatory);

        function certifiedCount() external view returns (uint) {
            return certifiedSymbols.length;
        }

        function certifiedTokens(string memory symbol) public view returns (uint mainChainId, address token) {
            uint256 chainIdToken = _certifiedTokens[symbol];
            mainChainId = chainIdToken >> 160;
            token = address(chainIdToken);
        }
```

```
    function allCertifiedTokens() external view returns (string[] memory symbols, uint[] memory chainIds,
address[] memory tokens) {
        symbols = certifiedSymbols;
        uint N = certifiedSymbols.length;
        chainIds = new uint[](N);
        tokens = new address[](N);
        for(uint i=0; i<N; i++)
            (chainIds[i], tokens[i]) = certifiedTokens(certifiedSymbols[i]);
    }

    function registerCertified(string memory symbol, uint mainChainId, address token) external governance {
        require(chainId() == 1 || chainId() == 3, 'called only on ethereum mainnet');
        require(isSupportChainId(mainChainId), 'Not support mainChainId');
        require(_certifiedTokens[symbol] == 0, 'Certified added already');
        if(mainChainId == chainId())
            require(keccak256(bytes(symbol))    ==    keccak256(bytes(ERC20UpgradeSafe(token).symbol())),
'symbol different');
        _certifiedTokens[symbol] = (mainChainId << 160) | uint(token);
        certifiedSymbols.push(symbol);
        emit RegisterCertified(symbol, mainChainId, token);
    }
    event RegisterCertified(string indexed symbol, uint indexed mainChainId, address indexed token);

    function createTokenMapped(address token) external payable returns (address tokenMapped) {
        _chargeFee();
        IERC20(token).totalSupply();
// just for check
        require(tokenMappeds[token] == address(0), 'TokenMapped created already');

        bytes32 salt = keccak256(abi.encodePacked(chainId(), token));

        bytes memory bytecode = type(InitializableProductProxy).creationCode;
        assembly {
            tokenMapped := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(payable(tokenMapped)).__InitializableProductProxy_init(address(this),
_TokenMapped_, abi.encodeWithSignature('__TokenMapped_init(address)', token));

        tokenMappeds[token] = tokenMapped;
        emit CreateTokenMapped(_msgSender(), token, tokenMapped);
    }
    event CreateTokenMapped(address indexed creator, address indexed token, address indexed tokenMapped);

    function createMappableToken(string memory name, string memory symbol, uint8 decimals, uint totalSupply)
external payable returns (address mappableToken) {
        _chargeFee();
        require(mappableTokens[_msgSender()] == address(0), 'MappableToken created already');

        bytes32 salt = keccak256(abi.encodePacked(chainId(), _msgSender()));

        bytes memory bytecode = type(InitializableProductProxy).creationCode;
        assembly {
            mappableToken := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(payable(mappableToken)).__InitializableProductProxy_init(address(this),
_MappableToken_,          abi.encodeWithSignature('__MappableToken_init(address,string,string,uint8,uint256)',
_msgSender(), name, symbol, decimals, totalSupply));

        mappableTokens[_msgSender()] = mappableToken;
        emit CreateMappableToken(_msgSender(), name, symbol, decimals, totalSupply, mappableToken);
    }
    event  CreateMappableToken(address  indexed  creator,  string  name,  string  symbol,  uint8  decimals,  uint
totalSupply, address indexed mappableToken);

    function _createMappingToken(uint mainChainId, address token, address creator, string memory name, string
memory symbol, uint8 decimals, uint cap) internal returns (address mappingToken) {
        _chargeFee();
        address tokenOrCreator = (token == address(0)) ? creator : token;
        require(mappingTokens[mainChainId][tokenOrCreator]    ==    address(0),    'MappingToken    created
already');

        bytes32 salt = keccak256(abi.encodePacked(mainChainId, tokenOrCreator));

        bytes memory bytecode = type(InitializableProductProxy).creationCode;
        assembly {
            mappingToken := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(payable(mappingToken)).__InitializableProductProxy_init(address(this),
_MappingToken_,
abi.encodeWithSignature('__MappingToken_init(uint256,address,address,string,string,uint8,uint256)',
mainChainId, token, creator, name, symbol, decimals, cap));

        mappingTokens[mainChainId][tokenOrCreator] = mappingToken;
        emit CreateMappingToken(mainChainId, token, creator, name, symbol, decimals, cap, mappingToken);
    }
    event CreateMappingToken(uint mainChainId, address indexed token, address indexed creator, string name,
string symbol, uint8 decimals, uint cap, address indexed mappingToken);
```

```
    function createMappingToken(uint mainChainId, address token, address creator, string memory name, string memory symbol, uint8 decimals, uint cap) public payable governance returns (address mappingToken) {
        return _createMappingToken(mainChainId, token, creator, name, symbol, decimals, cap);
    }

    function createMappingToken(uint mainChainId, address token, string memory name, string memory symbol, uint8 decimals, uint cap, Signature[] memory signatures) public payable returns (address mappingToken) {
        uint N = signatures.length;
        require(N >= getConfig(_minSignatures_), 'too few signatures');
        for(uint i=0; i<N; i++) {
            bytes32 hash = keccak256(abi.encode(CREATE_TYPEHASH, _msgSender(), mainChainId, token, name, symbol, decimals, cap, signatures[i].signatory));
            hash = keccak256(abi.encodePacked("\x19\x01", DOMAIN_SEPARATOR, hash));
            address signatory = ecrecover(hash, signatures[i].v, signatures[i].r, signatures[i].s);
            require(signatory != address(0), "invalid signature");
            require(signatory == signatures[i].signatory, "unauthorized");
            _decreaseAuthCount(signatures[i].signatory, 1);
            emit AuthorizeCreate(mainChainId, token, _msgSender(), name, symbol, decimals, cap, signatory);
        }
        return _createMappingToken(mainChainId, token, _msgSender(), name, symbol, decimals, cap);
    }
    event AuthorizeCreate(uint mainChainId, address indexed token, address indexed creator, string name, string symbol, uint8 decimals, uint cap, address indexed signatory);

    function _chargeFee() virtual internal {
        require(msg.value >= config[_feeCreate_], 'fee for Create is too low');
        address payable feeTo = address(config[_feeTo_]);
        if(feeTo == address(0))
            feeTo = address(uint160(address(this)));
        feeTo.transfer(msg.value);
        emit ChargeFee(_msgSender(), feeTo, msg.value);
    }
    event ChargeFee(address indexed from, address indexed to, uint value);

    uint256[50] private __gap;
}
```

6. Appendix B：Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	Level Description
High	Vulnerabilities that can directly cause the loss of token contracts or user funds，such as: value overflow loopholes that can cause the value of tokens to zero，fake recharge loopholes that can cause exchanges to lose tokens，and can cause contract accounts to lose ETH or tokens. Access loopholes，etc.； Vulnerabilities that can cause loss of ownership of token contracts，such as: access control defects of key functions，call injection leading to bypassing of access control of key functions，etc.； Vulnerabilities that can cause the token contract to not work properly，such as: denial of service vulnerability caused by sending ETH to malicious addresses，and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger，such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions，and logical design defects that cannot cause direct capital losses，etc.
Low	Vulnerabilities that are difficult to be triggered，vulnerabilities with limited damage after triggering，such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger，vulnerabilities where attackers cannot

	directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

7. Appendix C：Introduction to auditing tools

7.1 Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2 Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3 securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4 Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

7.5 MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6 ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.8 Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

7.9 Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing