知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20210317	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本	报告编号	保密级别
chainswap 智能合约审计报告	V1.0		项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2021 年 3 月 15 日开始到 2021 年 3 月 17 日结束，在此期间针对 **chainswap 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

本次智能合约安全审计的范围，不包含外部合约调用，不包含未来可能出现的新型攻击方式，不包含合约升级或篡改后的代码(随着项目方的发展，智能合约可能会增加新的 pool、新的功能模块，新的外部合约调用等)，不包含前端安全与服务器安全。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**通过**。

本次智能合约安全审计结果： 通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次审计的报告信息：

报告编号：

报告查询地址链接:

本次审计的目标信息：

条目	描述	
代币(项目)简称	chainswap	
合约地址	MappingToken Factory	
代码类型	代币代码、swap 跨链代码、以太坊智能合约代码	
代码语言	solidity	

合约文件及哈希：

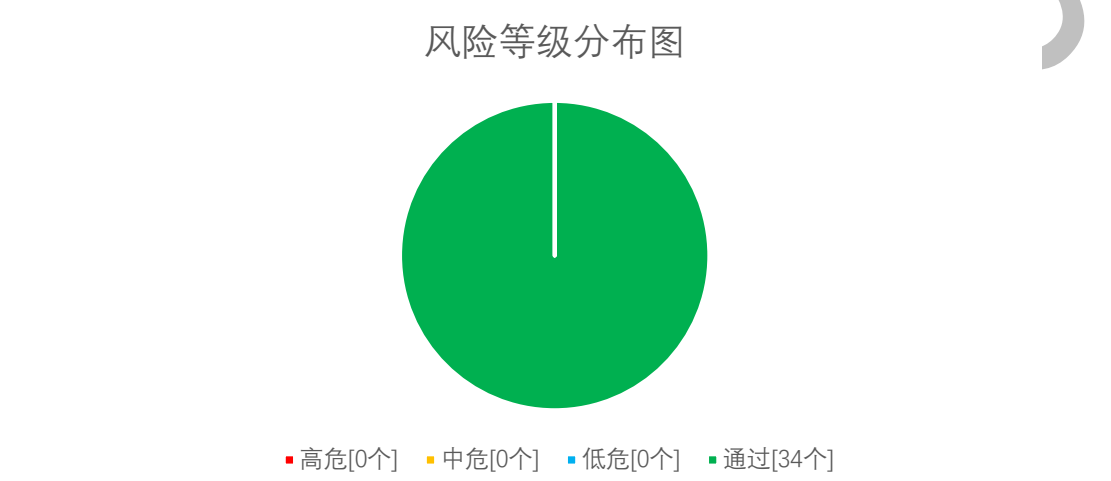
合约文件	MD5
MappingTokenFactory.sol	D1181CB6C8F6D3D88B57BF12B28E5E33

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	34



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	MappingBase 合约变量及修饰器	通过	经检测，不存在安全问题。
	MappingBase 合约增加/减少制定地址额度功能	通过	经检测，不存在安全问题。
	MappingBase 合约 send 函数	通过	经检测，不存在安全问题。
	MappingBase 合约 receive 函数	通过	经检测，不存在安全问题。
	MappingTokenFactory 合约增加和减少指定地址 signatory 可验证次数功能	通过	经检测，不存在安全问题。
	MappingTokenFactory 合约注册 token 映射函数	通过	经检测，不存在安全问题。
	MappingTokenFactory 合约创建合约函数	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。

	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. MappingBase 合约变量及修饰器【通过】

审计分析： MappingBase 合约变量定义及修神器函数设计合理。

```
abstract contract MappingBase is ContextUpgradeSafe, Constants {
    using SafeMath for uint; //knownsec// 使用 SafeMath 库，用于安全数学运算
    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// 声明变量 factory 用于记录工厂地址
    uint256 public mainChainId;//knownsec// 声明变量 mainChainId 用于记录链 Id
    address public token;//knownsec// 声明变量 token 用于记录代币地址
    address public creator;//knownsec// 声明变量 creator 用于记录创建者地址
    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// 修饰器，要求调用者必须为 factory 地址
        require(msg.sender == factory, 'Only called by Factory');
        _;
    }
```

安全建议：无。

3.2. MappingBase 合约增加/减少制定地址额度函数【通过】

审计分析：MappingBase 合约的 increaseAuthQuota 和 decreaseAuthQuota 函数是由工厂合约调用去增加和减少指定地址的额度。

```
    //knownsec// 批量增加指定地址的配额
    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// 调用 increaseAuthQuota 函数增加指定地址的配额

    }
    //knownsec// 增加指定地址的配额
    function increaseAuthQuota(address signatory, uint increment) virtual public onlyFactory returns (uint quota) {
        quota = authQuotaOf[signatory].add(increment);
        authQuotaOf[signatory] = quota; //knownsec// 更新指定地址的配额
        emit IncreaseAuthQuota(signatory, increment, quota);
    }
    event IncreaseAuthQuota(address indexed signatory, uint increment, uint quota);
    //knownsec// 减少指定地址对应的配额，如果不足，则为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// 如果指定地址的配额小于减少值，则减少的值设定为当前值
            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// 更新指定地址的配额
        emit DecreaseAuthQuota(signatory, decrement, quota);
    }
    event DecreaseAuthQuota(address indexed signatory, uint decrement, uint quota);
```

安全建议：无。

3.3. MappingBase 合约 send 函数【通过】

审计分析： MappingBase 合约的 send 函数用于用户将以太坊的 token 与指定的链的 token 做兑换，传入其它链 id，to 地址以及兑换金额。

```
    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// 调用内部函数收取交易手续费
        _sendFrom(from, volume);//knownsec// 调用内部函数_sendFrom进行代币转账
        //knownsec// 记录用户sent的数据
        nonce = sentCount[toChainId][to]++;
        sent[toChainId][to][nonce] = volume;
        emit Send(from, toChainId, to, nonce, volume);
    }
```

安全建议：无。

3.4. MappingBase 合约 receive 函数【通过】

审计分析： MappingBase 合约中的 receive 用于接受从其它链兑换的 token 代币，需要超过 3 个节点的验证信息。

```
    function receive(uint256 fromChainId, address to, uint256 nonce, uint256 volume, Signature[] memory signatures) virtual external payable {
        _chargeFee();//knownsec// 调用内部函数收取交易手续费
        require(received[fromChainId][to][nonce] == 0, 'withdrawn already');
        uint N = signatures.length;
        require(N >= MappingTokenFactory(factory).getConfig(_minSignatures_), 'too few signatures');//knownsec// 要求签名的数量大于等于配置的最小值
        for(uint i=0; i<N; i++) {
            //knownsec// 验证签名信息
            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// 调用内部函
```

```
数_decreaseAuthQuota 指定地址对应的配额
                emit Authorize(fromChainId, to, nonce, volume, signatory);
            }
            received[fromChainId][to][nonce] = volume; //knownsec// 更新指定链 to 地址 nonce
对应的值
            _receive(to, volume);//knownsec// 调用内部函数_receive 进行代币转账
            emit Receive(fromChainId, to, nonce, volume);
        }
        event Receive(uint256 indexed fromChainId, address indexed to, uint256 indexed nonce,
uint256 volume);
```

安全建议：无。

3.5. MappingTokenFactory 合约增加/减少指定地址验证次数功能【通过】

审计分析：MappingTokenFactory 合约中有权限的调用者可调用以下函数增加或减少指定地址的验证次数

```
//knownsec// 批量增加指定地址的验证次数
    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// 增加指定地址的验证次数
     function increaseAuthCount(address signatory, uint increment) virtual public onlyAuthorty
returns (uint count) {
        //knownsec// 减少指定地址 signatory 的验证次数
        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// 批量减少指定地址的验证次数
    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// 减少指定地址的验证次数
    function decreaseAuthCount(address signatory, uint decrement) virtual public onlyAuthorty returns (uint count) {
        count = authCountOf[signatory];
        if(count < decrement) //knownsec// 如果指定 signatory 的当前验证次数少于减少的减少的值，则修改指定地址 signatory 验证次数为本次减少次数
            decrement = count;
        return _decreaseAuthCount(signatory, decrement); //knownsec// 调用内部函数 _decreaseAuthCount 更新指定地址 signatory 验证次数
    }
    function _decreaseAuthCount(address signatory, uint decrement) virtual internal returns (uint count) {
        //knownsec// 减少指定地址 signatory 的验证次数
        count = authCountOf[signatory].sub(decrement);
        authCountOf[signatory] = count;
        emit DecreaseAuthCount(_msgSender(), signatory, decrement, count);
    }
```

安全建议：无。

3.6. MappingTokenFactory 合约注册 token 映射函数【通过】

审计分析：MappingTokenFactory 合约 governor 或通过节点验证的方式通过调用 registerMapping 函数注册 token 映射。

```
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// 要求 token => chainId => mappingToken or tokenMapped 不存在
            if(_mainChainIdTokens[mappingTokenMappeds_[i]] == 0)//knownsec// 如果 _mainChainIdTokens 指定的 token 地址对应的链 id+token 值为 0
                _mainChainIdTokens[mappingTokenMappeds_[i]] = (mainChainId << 160) | uint(token);//knownsec// 则记录指定 token 地址对应的链 id+token
            mappingTokenMappeds[token][chainIds[i]] = mappingTokenMappeds_[i];//knownsec// 记录 token => chainId => mappingToken or tokenMapped 数据
            emit RegisterMapping(mainChainId, token, chainIds[i], mappingTokenMappeds_[i]);
        }
    }
    event RegisterMapping(uint mainChainId, address token, uint chainId, address mappingTokenMapped);
```

```
    //knownsec// governor 注册代币映射
    function registerMapping(uint mainChainId, address token, uint[] memory chainIds, address[] memory mappingTokenMappeds_) virtual external governance {
        _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
    }
    //knownsec// 通过签名注册 token 映射
    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// 签名验证
            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// 更新指定地址的验证次数
            emit AuthorizeRegister(mainChainId, token, signatory);
        }
        _registerMapping(mainChainId, token, chainIds, mappingTokenMappeds_);
    }
    event AuthorizeRegister(uint indexed mainChainId, address indexed token, address indexed signatory);
```

安全建议：无。

3.7. MappingTokenFactory 合约创建合约函数【通过】

审计分析：以 MappingTokenFactory 合约中 createTokenMapped 为例，调用者可调用该函数创建 tokenMapped 合约并通过同理合约初始化。

```
    function createTokenMapped(address token) external payable returns (address tokenMapped) {
        _chargeFee(); //knownsec// 调用内部函数_chargeFee 收取手续费
        IERC20(token).totalSupply();//knownsec// 获取 token 地址的代币总量
// just for check
        require(tokenMappeds[token] == address(0), 'TokenMapped created already');//knownsec// 要求 token 地址第一次创建

        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;//knownsec// 记录 token 对应的 tokenMapped 地址
        emit CreateTokenMapped(_msgSender(), token, tokenMapped);
    }
    event CreateTokenMapped(address indexed creator, address indexed token, address indexed tokenMapped);
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.6.0 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在增发代币的功能，但由于设定代币上限，故通过。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；

传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果：经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

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. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

7.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

7.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

7.3 securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

7.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

7.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合

约的字节码，并尝试建立一系列交易以找出并确认错误。

7.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

7.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

7.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

7.9 知道创宇区块链安全审计人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509