



智能合约安全审计报告



审计编号: 2019082291148

源代码 hash (SHA256): 72437977f78c0e81c401a66b6694929b810c4a698dbd50ec80066087654bbabf

审计合约名称:

序号	合约文件名
1	contracts\base\Owned.sol
2	contracts\base\SafeMath.sol
3	contracts\AccountCreator.sol
4	contracts\AccountLogic.sol
5	contracts\AccountStorage.sol
6	contracts\BaseAccount.sol
7	contracts\BaseAccountProxy.sol
8	contracts\BaseLogic.sol
9	contracts\DappLogic.sol
10	contracts\DualsignsLogic.sol
11	contracts\LogicManager.sol
12	contracts\TransferLogic.sol

合约审计开始日期: 2019.08.15

合约审计完成日期: 2019.08.29

审计结果: 通过 (优)

审计团队: 成都链安科技有限公司

审计类型及结果:

序号	审计类型	审计子项	审计结果
1	代码规范审计	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过
		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过
		拒绝服务攻击审计	通过
		函数调用权限审计	通过

		call/delegatecall 安全审计	通过
		返回值安全审计	通过
		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	通过

备注：审计意见及建议请见代码注释。

免责声明：本次审计仅针对审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链安科技仅根据本报告出具前已经存在或发生的攻击或漏洞出具本报告，并就此承担相应责任。对于出具以后存在或发生的新的攻击或漏洞，成都链安科技无法判断其对智能合约安全状况可能的影响，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，仅基于合约提供者截至本报告出具时向成都链安科技提供的文件和资料，文件和资料不应存在缺失、被篡改、删减或隐瞒的情形；如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符等情况，成都链安科技对由此而导致的损失和不利影响不承担任何责任。

审计结果说明

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对MYKEY项目合约的代码规范性、安全性以及业务逻辑三个方面进行多维度的安全审计。**MYKEY项目的合约通过所有检测项，合约审计结果为通过(优)，合约可正常使用。**

代码规范审计

1. 编译器版本安全审计

使用老版本的编译器编译合约可能会导致各种已知安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

在TransferLogic合约中，onERC721Received函数存在编译器警告如下图1所示：

```
browser/TransferLogic.sol:121:31: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes calldata _data)
external pure returns (bytes4) {
^-----^

browser/TransferLogic.sol:121:50: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes calldata _data)
external pure returns (bytes4) {
^-----^

browser/TransferLogic.sol:121:65: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes calldata _data)
external pure returns (bytes4) {
^-----^

browser/TransferLogic.sol:121:83: Warning: Unused function parameter. Remove or comment out the variable name to silence this warning.
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes calldata _data)
external pure returns (bytes4) {
^-----^
```

图 1 编译器警告截图

- **安全建议：**建议固定编译器版本并完善onERC721Received函数以消除编译器警告。
- **审计结果：**通过

2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw、years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- **安全建议：**无
- **审计结果：**通过

3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- **安全建议：**无
- **审计结果：**通过

4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用（及其所有子调用）中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- **安全建议：**无
- **审计结果：**通过

5. gas 消耗审计

以太坊虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因gas不足导致函数执行一直失败。

- **安全建议：**无
- **审计结果：**通过

通用漏洞审计

1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。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uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- **安全建议：**无

- 审计结果：通过

2. 重入攻击审计

重入漏洞是最典型的以太坊智能合约漏洞，曾导致了The DAO被攻击。该漏洞原因是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的逻辑顺序存在错误时，就会存在重入攻击的风险。

- 安全建议：无
- 审计结果：通过

3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- 安全建议：无
- 审计结果：通过

4. 交易顺序依赖审计

在以太坊的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- 安全建议：无
- 审计结果：通过

5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在以太坊智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约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有很多种，包括在作为交易接收方时的恶意revert、代码设计缺陷导致gas耗尽等等。

- 安全建议：无
- 审计结果：通过

6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币、自毁、change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- 安全建议：无
- 审计结果：通过

7. call/delegatecall 安全审计

Solidity 中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- 安全建议：无
- 审计结果：通过

8. 返回值安全审计

在Solidity中存在transfer()、send()、call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑；另外在ERC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- 安全建议：无
- 审计结果：通过

9. tx.origin使用安全审计

在以太坊智能合约的复杂调用中，tx.origin表示交易的初始发起者地址，如果使用tx.origin进行权限判断，可能会出现错误；另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- 安全建议：无
- 审计结果：通过

10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- 安全建议：无
- 审计结果：通过

11. 变量覆盖审计

以太坊存在着复杂的变量类型，例如结构体、动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- 安全建议：无
- 审计结果：通过

业务审计

1、LogicManager合约

- 相关函数：
submitUpdate、triggerUpdateLogic。
- 功能描述：

本合约作为项目业务逻辑模块管理者，实现对业务逻辑模块增删的管理功能。管理员用户通过调用 submitUpdate 函数向合约提交添加/移除逻辑合约的请求，在延迟期过后任何人都可调用 triggerUpdateLogic 函数执行该请求，从而实现项目业务逻辑模块的添加/移除。

- 安全建议：无
- 审计结果：通过

2、BaseLogic 合约

- 相关函数：

executeProposal、checkApproval、checkRelation、getSignHash、verifySig、checkAndUpdateNonce 等。

- 功能描述：

本合约作为项目逻辑功能的基石，所有逻辑合约都继承该合约。该合约提供了必要的身份验证、提案结果验证相关函数，包括用于签名验证的 getSignHash、verifySig 函数；nonce 检查更新的 checkAndUpdateNonce 函数；紧急联系人验证的 checkRelation 函数；检查提案是否通过的 checkApproval 函数等等。还为提案提供了执行入口函数 executeProposal，可执行的提案包括 AccountLogic 合约中定义的账户紧急联系人修改 admin 公钥提案以及 DualsignsLogic 合约中定义三个的无延迟提案。

- 安全建议：无
- 审计结果：通过

3、AccountStorage 合约

- 相关函数：

initAccount 、 getProposalDataApproval 、 setProposalData 、 clearProposalData 、 getDelayDataDueTime、setDelayData、clearDelayData、getBackupAddress、setBackup、getKeyData、setKeyData 等。

- 功能描述：

本合约作为项目数据存储合约，提供了创建账户时初始化账户数据（账户操作的公钥、账户的紧急联系人）函数、账户的操作公钥和冻结状态 get/set 方法、账户紧急联系人和生效过期时间的 get/set 方法、逻辑动作延迟数据的 get/set 方法、提案请求及对提案授权的紧急联系人的 get/set 方法。实现了对账户数据、账户动作数据、提案数据的存储与管理。

- 安全建议：无
- 审计结果：通过

4、AccountCreator 合约

- 相关函数：

createAccount。

- 功能描述：

项目管理员调用合约的createAccount函数为每个用户创建账户。创建账户时用户需提供的数据包括操作作用的公钥数组和紧急联系人数组。函数限制了创建账户的调用者只能是项目管理员，但是用户可以通过baseAccount合约的init函数绕过管理员自行创建账户。

➤ **安全建议：**

- 1、建议增加公钥和联系人的数量限制，并添加紧急联系人数组元素重复性检查，避免由于不规范创建账户而造成对AccountStorage合约内存空间的浪费以及额外的gas开销。
- 2、建议在baseAccount合约的init函数添加权限控制逻辑。

➤ **修复结果：**经项目方确认参数规范由链下进行限制，合约不做修改。

➤ **审计结果：**通过

5、BaseAccount合约

➤ **相关函数：**

init、invoke、enableStaticCall、fallback函数等。

➤ **功能描述：**

合约提供了用于部分逻辑合约(DappLogic、TransferLogic)功能调用的入口函数invoke，并且当项目有新的逻辑功能也可以通过该函数去执行。

➤ **安全建议：**无

➤ **审计结果：**通过

6、BaseAccountProxy合约

➤ **相关函数：**

fallback函数。

➤ **功能描述：**

每个账户对应一个BaseAccountProxy合约，在创建账户时生成，作为BaseAccount合约的代理，所有逻辑都delegate call到BaseAccount合约以减少账户合约部署的成本。

➤ **安全建议：**无

➤ **审计结果：**通过

7、AccountLogic合约

➤ **相关函数：**

enter 、 changeAdminKey 、 triggerChangeAdminKey 、 changeAdminKeyByBackup 、 triggerChangeAdminKeyByBackup 、 addOperationKey 、 changeAllOperationKeys 、 triggerChangeAllOperationKeys 、 freeze 、 unfreeze 、 triggerUnfreeze 、 removeBackup 、 cancelAddBackup、proposeAsBackup、approveProposal等。

➤ **功能描述：**

本合约作为面向账户操作的逻辑合约，提供了账户对其公钥的增删、修改、冻结；紧急联系人的删除、到期后进行延期、紧急联系人发起修改admin公钥的提案、紧急联系人对指定提案投票等功能。这些操作分为普通操作和一些紧急操作，普通操作通过本合约的enter入口调用而紧急操作则通过executeProposal入口进行调用。

- 安全建议：无
- 审计结果：通过

8、TransferLogic合约

- 相关函数：

enter、transferEth、transferErc20、transferApprovedErc20、transferNft等。

- 功能描述：

本合约作为用户进行转账相关操作的逻辑合约，提供了ETH转账函数、ERC20标准代币转账函数、ERC20标准代币代理转账函数、Nft代币转账函数。

- 安全建议：无
- 审计结果：通过

9、DualsignLogic合约

- 相关函数：

enter 、 changeAdminKeyWithoutDelay 、 changeAllOperationKeysWithoutDelay 、 unfreezeWithoutDelay、addBackup、proposeByBoth等。

- 功能描述：

本合约作为账户紧急操作的逻辑合约，提供了无延迟的修改admin公钥函数、修改账户所有操作公钥函数、解除公钥冻结函数以及添加紧急联系人、共同发起提案函数。

- 安全建议：无
- 审计结果：通过

10、DappLogic合约

- 相关函数：

enter、callContract等。

- 功能描述：

本合约作为账户操作其它Dapp的逻辑合约，提供了实现其它合约功能调用的函数入口callContract。

- 安全建议：无
- 审计结果：通过

合约源代码审计注释:

```
// 成都链安 // ### File: contracts\base\SafeMath.sol ###
pragma solidity ^0.5.4;

/* The MIT License (MIT)

Copyright (c) 2016 Smart Contract Solutions, Inc.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE. */

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

    /**
     * @dev Multiplies two numbers, reverts on 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-solidity/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b);
    }
}
```



```
        return c;
    }

    /**
     * @dev Integer division of two numbers truncating the quotient, reverts on division by
     zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0); // Solidity only automatically asserts when dividing by 0
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater than
     minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Adds two numbers, reverts on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
     * reverts when dividing by zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }

    /**
     * @dev Returns ceil(a / b).
     */
```



```
function ceil(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a / b;
    if(a % b == 0) {
        return c;
    }
    else {
        return c + 1;
    }
}
}

// 成都链安 // ### File: contracts\base\Owned.sol ###
pragma solidity ^0.5.4;

/**
 * @title Owned
 * @dev Basic contract to define an owner.
 * @author Julien Niset - <julien@argent.im>
 */
contract Owned {

    // The owner
    address public owner; // 成都链安 // 声明 owner 变量，存储合约管理员

    event OwnerChanged(address indexed _newOwner); // 成都链安 // 声明 OwnernerChanged 事件

    /**
     * @dev Throws if the sender is not the owner.
     */
    modifier onlyOwner {
        require(msg.sender == owner, "Must be owner");
        _;
    }

    // 成都链安 // 构造函数，初始化合约管理员
    constructor() public {
        owner = msg.sender;
    }

    /**
     * @dev Lets the owner transfer ownership of the contract to a new owner.
     * @param _newOwner The new owner.
     */
    function changeOwner(address _newOwner) external onlyOwner {
        require(_newOwner != address(0), "Address must not be null"); // 成都链安 //
        _newOwner 非零地址检查，避免管理员权限丢失
        owner = _newOwner; // 成都链安 // 将_newOwner 设置为新的合约管理员
        emit OwnerChanged(_newOwner); // 成都链安 // 触发 OwnerChanged 事件
    }
}
```

```
// 成都链安 // ### File:contracts\BaseAccount.sol ###
pragma solidity ^0.5.4;

import "./LogicManager.sol";
import "./BaseLogic.sol";

contract BaseAccount {

    // The implementation of the proxy
    address public implementation;

    // Logic manager
    address public manager;

    // The enabled static calls
    mapping (bytes4 => address) public enabled;

    event EnabledStaticCall(address indexed module, bytes4 indexed method); // 成都链安 //
    声明 EnabledStaticCall 事件
    event Invoked(address indexed module, address indexed target, uint indexed value,
bytes data); // 成都链安 // 声明 Invoked 事件
    event Received(uint indexed value, address indexed sender, bytes data); // 成都链安 //
    声明 Received 事件

    event AccountInit(address indexed account); // 成都链安 // 声明 AccountInit 事件
    // 成都链安 // 函数修饰器，被该修饰器修饰的函数只能被经授权的逻辑合约调用
    modifier onlyManagerAuthorised {
        require(LogicManager(manager).isAuthorised(msg.sender), "not an authorized
logic");
        _;
    }
    // 成都链安 // 初始化函数，用于设置逻辑管理合约地址并在基本逻辑合约中初始化本合约账户
    function init(address _manager, address _accountStorage, address[] calldata _logics,
address[] calldata _keys, address[] calldata _backups)
        external
    {
        require(manager == address(0), "BaseAccount: account already initialized"); // 成
都链安 // 当前逻辑管理合约地址必须是零地址，即只能初始化一次
        require(_manager != address(0) && _accountStorage != address(0), "BaseAccount:
address is null"); // 成都链安 // _accountStorage 非零地址检查
        manager = _manager; // 成都链安 // 设置逻辑管理合约地址为_manager
        // 成都链安 // 在各逻辑合约中初始化合约账户
        for (uint i = 0; i < _logics.length; i++) {
            address logic = _logics[i];
            require(LogicManager(manager).isAuthorised(logic), "must be authorised
logic");
        }
    }
}
```



```
BaseLogic(logic).initAccount(this);
}
// 成都链安 // 在 AccountStorage 合约中初始化合约账户，并存储其公钥和紧急联系人
AccountStorage(_accountStorage).initAccount(this, _keys, _backups);
// 成都链安 // 触发 AccountInit 事件
emit AccountInit(address(this));
}

function invoke(address _target, uint _value, bytes calldata _data)
    external
    onlyManagerAuthorised
{
    // solium-disable-next-line security/no-call-value
    (bool success,) = _target.call.value(_value)(_data); // 成都链安 // 通过目标合约地址进行 call 调用
    require(success, "call to target failed"); // 成都链安 // call 调用返回值检查，避免调用失败后仍触发事件 Invoked 且交易回执显示的交易状态仍为成功
    emit Invoked(msg.sender, _target, _value, _data); // 成都链安 // 触发 Invoked 事件
}

/**
 * @dev Enables a static method by specifying the target module to which the call must be delegated.
 * @param _module The target module.
 * @param _method The static method signature.
 */
function enableStaticCall(address _module, bytes4 _method) external
onlyManagerAuthorised {
    enabled[_method] = _module; // 成都链安 // 记录_method 方法对应的合约地址_module
    emit EnabledStaticCall(_module, _method); // 成都链安 // 触发 EnabledStaticCall 事件
}

/**
 * @dev This method makes it possible for the wallet to comply to interfaces expecting the wallet to
 * implement specific static methods. It delegates the static call to a target contract if the data corresponds
 * to an enabled method, or logs the call otherwise.
 */
function() external payable {
    if(msg.data.length > 0) {
        address logic = enabled[msg.sig]; // 成都链安 // 根据函数标识符获取其所在的逻辑合约地址
        if(logic == address(0)) { // 成都链安 // 若对应的地址为零地址，即该函数标识符未记录，那么直接触发 Received 事件
            emit Received(msg.value, msg.sender, msg.data);
        }
    }
}
```

```
        else {
            require(LogicManager(manager).isAuthorised(logic), "must be an authorized
logic for static call"); // 成都链安 // 否则，判断对应的逻辑合约是否经过授权
            // solium-disable-next-line security/no-inline-assembly
            assembly { // 成都链安 // 使用 staticcall 调用该函数
                calldatacopy(0, 0, calldatasize())
                let result := staticcall(gas, logic, 0, calldatasize(), 0, 0)
                returndatacopy(0, 0, returndatasize())
                switch result
                case 0 {revert(0, returndatasize())}
                default {return (0, returndatasize())}
            }
        }
    }
}

// 成都链安 // #### File:contracts\BaseAccountProxy.sol ####
pragma solidity ^0.5.4;

contract BaseAccountProxy {

    address implementation; // 成都链安 // 声明 implementation 变量，记录 BaseAccount 地址

    event Received(uint indexed value, address indexed sender, bytes data); // 成都链安 //
    声明 Received 事件
    // 成都链安 // 构造函数，初始化相关变量
    constructor(address _implementation) public {
        implementation = _implementation;
    }
    // 成都链安 // fallback 函数，用户实现功能调用的主要入口
    function() external payable {

        if(msg.data.length == 0 && msg.value > 0) {
            emit Received(msg.value, msg.sender, msg.data);
        }
        else {
            // solium-disable-next-line security/no-inline-assembly
            assembly {
                let target := sload(0)
                calldatacopy(0, 0, calldatasize())
                let result := delegatecall(gas, target, 0, calldatasize(), 0, 0)
                returndatacopy(0, 0, returndatasize())
                switch result
                case 0 {revert(0, returndatasize())}
                default {return (0, returndatasize())}
            }
        }
    }
}
```

```
}
// 成都链安 // #### File:contracts\AccountStorage.sol ####
pragma solidity ^0.5.4;

import "./BaseAccount.sol";
import "./LogicManager.sol";

contract AccountStorage {
    // 成都链安 // 函数修饰器，要求调用被该修饰器修饰的函数时传入的_account 地址和函数调用者 msg.sender 必须相同
    modifier onlyAccount(BaseAccount _account) {
        require(msg.sender == address(_account), "caller must be account");
        _;
    }
    // 成都链安 // 函数修饰器，要求被该修饰器修饰的函数的调用者必须是经授权的逻辑合约
    modifier onlyManagerAuthorised(address payable _account) {
        require(LogicManager(BaseAccount(_account).manager()).isAuthorised(msg.sender), "not an authorized logic");
        _;
    }
    // 成都链安 // 公钥结构体，包含公钥和对应的状态(是否被冻结)
    struct KeyItem {
        address pubKey;
        uint256 status;
    }
    // 成都链安 // 紧急联系人结构体，包含紧急联系人地址和其生效/失效时间
    struct BackupAccount {
        address backup;
        uint256 effectiveDate;//means not effective until this timestamp
        uint256 expiryDate;//means effective until this timestamp
    }
    // 成都链安 // 延迟结构体，包含动作的哈希和延迟的到期时间
    struct DelayItem {
        bytes32 hash;
        uint256 dueTime;
    }
    // 成都链安 // 多签提案结构体，包含提案的哈希和同意提案的紧急联系人数组
    struct Proposal {
        bytes32 hash;
        address[] approval;
    }

    // account => quantity of keys
    mapping (address => uint256) keyCount;

    // account => index => KeyItem
```

```
mapping (address => mapping(uint256 => KeyItem)) keyData; // 成都链安 // 声明 keyData
变量，用于存储账户各序列对应的公钥数据

// account => index => backup account
mapping (address => mapping(uint256 => BackupAccount)) backupData; // 成都链安 // 声明
keyData 变量，用于存储账户各序列对应的紧急联系人数据

/* account => actionId => DelayItem

delayData applies to these 4 actions:
changeAdminKey, changeAllOperationKeys, unfreeze, changeAdminKeyByBackup
*/
mapping (address => mapping(bytes4 => DelayItem)) delayData; // 成都链安 // 声明
delayData 变量，用于存储账户动作对应的延迟数据

// client account => proposer account => proposed actionId => Proposal
mapping (address => mapping(address => mapping(bytes4 => Proposal))) proposalData; //
成都链安 // 声明 proposalData 变量，用于存储当事人账户和某一紧急联系人发起的提案对应的多签
数据

// ***** keyCount ***** //

function getKeyCount(address _account) external view returns(uint256) {
    return keyCount[_account];
}

function incrementKeyCount(address payable _account) external
onlyManagerAuthorised(_account) {
    keyCount[_account] = keyCount[_account] + 1;
}

// ***** keyData ***** //
// 成都链安 // 获取公钥函数，根据账户和序号获取对应的公钥
function getKeyData(address _account, uint256 _index) public view returns(address) {
    KeyItem memory item = keyData[_account][_index];
    return item.pubKey;
}
// 成都链安 // 设置公钥函数，设置账户和序号对应的公钥
function setKeyData(address payable _account, uint256 _index, address _key) external
onlyManagerAuthorised(_account) {
    require(_key != address(0), "invalid _key value");
    KeyItem storage item = keyData[_account][_index];
    item.pubKey = _key;
}

// ***** keyStatus ***** //
// 成都链安 // 获取冻结状态函数，根据账户和序号获取冻结状态
```

```
function getKeyStatus(address _account, uint256 _index) external view returns(uint256)
{
    KeyItem memory item = keyData[_account][_index];
    return item.status;
}

// 成都链安 // 设置冻结状态函数，设置账户和序号对应公钥的冻结状态
function setKeyStatus(address payable _account, uint256 _index, uint256 _status)
external onlyManagerAuthorised(_account) {
    KeyItem storage item = keyData[_account][_index];
    item.status = _status;
}

// ***** backupData ***** //
// 成都链安 // 以下函数实现了紧急联系人数据的 get/set 以及清除方法
function getBackupAddress(address _account, uint256 _index) external view
returns(address) {
    BackupAccount memory b = backupData[_account][_index];
    return b.backup;
}

function getBackupEffectiveDate(address _account, uint256 _index) external view
returns(uint256) {
    BackupAccount memory b = backupData[_account][_index];
    return b.effectiveDate;
}

function getBackupExpiryDate(address _account, uint256 _index) external view
returns(uint256) {
    BackupAccount memory b = backupData[_account][_index];
    return b.expiryDate;
}

function setBackup(address payable _account, uint256 _index, address _backup, uint256
_effective, uint256 _expiry)
external
onlyManagerAuthorised(_account)
{
    BackupAccount storage b = backupData[_account][_index];
    b.backup = _backup;
    b.effectiveDate = _effective;
    b.expiryDate = _expiry;
}

function setBackupExpiryDate(address payable _account, uint256 _index, uint256
_expiry)
external
onlyManagerAuthorised(_account)
{
```



```
BackupAccount storage b = backupData[_account][_index];
b.expiryDate = _expiry;
}

function clearBackupData(address payable _account, uint256 _index) external
onlyManagerAuthorised(_account) {
    delete backupData[_account][_index];
}

// ***** delayData ***** //
// 成都链安 // 以下函数实现了账户操作数据的 get/set 以及清除操作数据的方法
function getDelayDataHash(address payable _account, bytes4 _actionId) external view
returns(bytes32) {
    DelayItem memory item = delayData[_account][_actionId];
    return item.hash;
}

function getDelayDataDueTime(address payable _account, bytes4 _actionId) external view
returns(uint256) {
    DelayItem memory item = delayData[_account][_actionId];
    return item.dueTime;
}

function setDelayData(address payable _account, bytes4 _actionId, bytes32 _hash,
uint256 _dueTime) external onlyManagerAuthorised(_account) {
    DelayItem storage item = delayData[_account][_actionId];
    item.hash = _hash;
    item.dueTime = _dueTime;
}

function clearDelayData(address payable _account, bytes4 _actionId) external
onlyManagerAuthorised(_account) {
    delete delayData[_account][_actionId];
}

// ***** proposalData ***** //
// 成都链安 // 以下函数实现了多签提案数据的 get/set 以及清除提案数据的方法
function getProposalDataHash(address _client, address _proposer, bytes4 _actionId)
external view returns(bytes32) {
    Proposal memory p = proposalData[_client][_proposer][_actionId];
    return p.hash;
}

function getProposalDataApproval(address _client, address _proposer, bytes4 _actionId)
external view returns(address[] memory) {
    Proposal memory p = proposalData[_client][_proposer][_actionId];
    return p.approval;
}
```

```
function setProposalData(address payable _client, address _proposer, bytes4 _actionId,
bytes32 _hash, address _approvedBackup)
    external
    onlyManagerAuthorised(_client)
{
    Proposal storage p = proposalData[_client][_proposer][_actionId];
    if (p.hash > 0) {
        require(p.hash == _hash, "proposal hash unmatched");
        for (uint256 i = 0; i < p.approval.length; i++) {
            require(p.approval[i] != _approvedBackup, "backup already exists");
        }
        p.approval.push(_approvedBackup);
    } else {
        p.hash = _hash;
        p.approval.push(_approvedBackup);
    }
}

function clearProposalData(address payable _client, address _proposer, bytes4
_actionId) external onlyManagerAuthorised(_client) {
    delete proposalData[_client][_proposer][_actionId];
}

// ***** init ***** //
// 成都链安 // 初始化账户信息函数，用于创建账户时在本合约设置该账户初始的公钥和紧急联
系人信息
function initAccount(BaseAccount _account, address[] calldata _keys, address[]
calldata _backups)
    external
    onlyAccount(_account)
{
    require(getKeyData(address(_account), 0) == address(0), "AccountStorage: account
already initialized!"); // 成都链安 // _account 账户检查，该账户不能是已存在的账户
    require(_keys.length > 0, "empty keys array"); // 成都链安 // _keys 公钥检查，账户
至少需要有一个公钥

    keyCount[address(_account)] = _keys.length; // 成都链安 // 记录账户添加公钥的数量

    for (uint256 index = 0; index < _keys.length; index++) { // 成都链安 // 初始化账户
对应的公钥组
        address _key = _keys[index];
        require(_key != address(0), "_key cannot be 0x0");
        KeyItem storage item = keyData[address(_account)][index];
        item.pubKey = _key;
        item.status = 0;
    }
}
```

```
// avoid backup duplication if _backups.length > 1
// normally won't check duplication, in most cases only one initial backup when
initialization
if (_backups.length > 1) {
    address[] memory bkps = _backups;
    for (uint256 i = 0; i < _backups.length; i++) {
        for (uint256 j = 0; j < i; j++) {
            require(bkps[j] != _backups[i], "duplicate backup");
        }
    }
}

for (uint256 index = 0; index < _backups.length; index++) { // 成都链安 // 初始化
账户对应的紧急联系人
    address _backup = _backups[index];
    require(_backup != address(0), "backup cannot be 0x0");
    require(_backup != address(_account), "cannot be backup of oneself");

    backupData[address(_account)][index] = BackupAccount(_backup, now, uint256(-
1)); // 成都链安 // 紧急联系人的生效时间为当前，失效时间为 2**256 - 1
}
}

// 成都链安 // #### File:contracts\AccountCreator.sol ####
pragma solidity ^0.5.4;

import "./base/Owned.sol";
import "./BaseAccount.sol";
import "./BaseAccountProxy.sol";

contract AccountCreator is Owned {

    address public logicManager; // 成都链安 // 声明 logicManager 变量，用于存储逻辑模块管
理合约地址
    address public accountStorage; // 成都链安 // 声明 accountStorage 变量，用于存储数据合
约地址
    address public accountImpl; // 成都链安 // 声明 accountImpl 变量，用于存储 BaseAccount
合约地址
    address[] public logics; // 成都链安 // 声明 logics 变量，用于存储逻辑合约地址数组

    // ***** Events ***** //
    event AccountCreated(address indexed _wallet, address[] _keys, address[] _backups); //
成都链安 // 声明 AccountCreated 事件

    // ***** Constructor ***** //
    // 成都链安 // 构造函数，初始化合约变量
```

```
    constructor(address _mgr, address _storage, address _accountImpl, address[] memory
_logsics) public {
    logicManager = _mgr;
    accountStorage = _storage;
    accountImpl = _accountImpl;
    logics = _logics;
}

// ***** External Functions ***** //
// 成都链安 // 账户创建函数，用户调用该函数可在 AccountStorage 合约中创建账户，并初始化
账户基本信息
    function createAccount(address[] calldata _keys, address[] calldata _backups) external
onlyOwner {
    BaseAccountProxy accountProxy = new BaseAccountProxy(accountImpl); // 成都链安 //
创建代理合约
    BaseAccount(address(accountProxy)).init(logicManager, accountStorage, logics,
_keys, _backups); // 成都链安 // 使用代理合约地址实例化的 BaseAccount 合约进行账户的初始化

    emit AccountCreated(address(accountProxy), _keys, _backups); // 成都链安 // 触发
AccountCreated 事件
}

// ***** Suicide ***** //
// 成都链安 // 自毁函数，管理员可调用该函数销毁本合约
    function close() external onlyOwner {
        selfdestruct(msg.sender);
    }
}

// 成都链安 // ### File:contracts\BaseLogic.sol ###
pragma solidity ^0.5.4;

import "./BaseAccount.sol";
import "./AccountStorage.sol";
import "./base/SafeMath.sol";

contract BaseLogic {

    /*
    mainnet: 0;
    local: 1;
    ropsten: 2;
    */
    uint256 constant internal ENVIRONMENT = 1;
    // 成都链安 // 定义各类动作对应的操作码，用于设置动作延迟
    uint256 constant internal TYPE_CHANGE_ADMIN_KEY = 0;
```

```
uint256 constant internal TYPE_CHANGE_OPERATION_KEY = 1;
uint256 constant internal TYPE_UNFREEZE_KEY = 2;
uint256 constant internal TYPE_CHANGE_BACKUP = 3;
uint256 constant internal TYPE_CHANGE_ADMIN_KEY_BY_BACKUP = 4;
// 成都链安 // 定义账户对应的紧急联系人数量
uint256 constant internal MAX_DEFINED_BACKUP_INDEX = 5;

mapping (address => uint256) keyNonce; // 成都链安 // 声明 keyNonce 变量，用于存储公钥
对应的 nonce 值
AccountStorage public accountStorage; // 成都链安 // 声明 accountStorage 变量，用于存储
AccountStorage 合约地址

// 成都链安 // 函数修饰器，被该函数修饰器修饰的函数的调用者只能是合约本身
modifier onlySelf() {
    require (msg.sender == address(this), "only internal call is allowed");
    _;
}

// 成都链安 // 函数修饰器，被该函数修饰器修饰的函数在被调用时传入的_account 必须和调用
者相同
modifier onlyAccount(BaseAccount _account) {
    require(msg.sender == address(_account), "caller must be account");
    _;
}

event LogicInitialised(address wallet); // 成都链安 // 声明 LogicInitialised 事件
event ProposalExecuted(address indexed client, address indexed proposer, bytes
functionData); // 成都链安 // 声明 ProposalExecuted 事件

// ***** Constructor ***** //
// 成都链安 // 构造函数，初始化 AccountStorage 合约地址
constructor(AccountStorage _accountStorage) public {
    accountStorage = _accountStorage;
}

// ***** Initialization ***** //

function initAccount(BaseAccount _account) external onlyAccount(_account) {
    emit LogicInitialised(address(_account));
}

// ***** Getter ***** //
// 成都链安 // 获取延迟时间函数，在不同环境根据动作的操作码获取对应的延迟时间
function getDelayTime(uint256 _actionType) internal pure returns(uint256) {
    if (ENVIRONMENT == 0) { //mainnet
        if (_actionType == TYPE_CHANGE_ADMIN_KEY) {
            return 21 days;
        } else if (_actionType == TYPE_CHANGE_OPERATION_KEY) {
            return 7 days;
        }
    }
}
```



```
    } else if (_actionType == TYPE_UNFREEZE_KEY) {
        return 7 days;
    } else if (_actionType == TYPE_CHANGE_BACKUP) {
        return 21 days;
    } else if (_actionType == TYPE_CHANGE_ADMIN_KEY_BY_BACKUP) {
        return 30 days;
    }
} else if (ENVIRONMENT == 1) { //local
    return 1 seconds;
} else if (ENVIRONMENT == 2) { //ropsten
    if (_actionType == TYPE_CHANGE_ADMIN_KEY) {
        return 21*10 seconds;
    } else if (_actionType == TYPE_CHANGE_OPERATION_KEY) {
        return 7*10 seconds;
    } else if (_actionType == TYPE_UNFREEZE_KEY) {
        return 7*10 seconds;
    } else if (_actionType == TYPE_CHANGE_BACKUP) {
        return 21*10 seconds;
    } else if (_actionType == TYPE_CHANGE_ADMIN_KEY_BY_BACKUP) {
        return 30*10 seconds;
    }
}
revert("invalid type or environment");
}

// 成都链安 // 获取动作 ID 函数，根据函数名和参数类型的 bytes 数据获取函数标识符作为动作 ID 并返回
function getActionId(bytes memory _name) internal pure returns(bytes4) {
    return bytes4(keccak256(_name));
}

// 成都链安 // 获取 nonce 值，根据公钥地址获取对应的 nonce 值
function getKeyNonce(address _key) external view returns(uint256) {
    return keyNonce[_key];
}

// ***** Proposal ***** //

/* â€œexecuteProposalâ€œ™ is shared by AccountLogic and DualsignsLogic,
proposed actions called from 'executeProposal':
    AccountLogic: changeAdminKeyByBackup
    DualsignsLogic: changeAdminKeyWithoutDelay, changeAllOperationKeysWithoutDelay,
unfreezeWithoutDelay
*/

// 成都链安 // 执行提案函数，根据提案发起者和动作哈希，验证并执行提案
function executeProposal(address payable _client, address _proposer, bytes calldata
_functionData) external {
    bytes4 proposedActionId = getMethodId(_functionData); // 成都链安 // 获取提案的前
四个字节作为提案 ID
```

```
bytes32 functionHash = keccak256(_functionData); // 成都链安 // 根据提案数据获取动作哈希
```

```
checkApproval(_client, _proposer, proposedActionId, functionHash); // 成都链安 // 调用 checkApproval 函数，检查是否授权通过
```

```
// call functions with/without delay  
// solium-disable-next-line security/no-low-level-calls  
(bool success,) = address(this).call(_functionData); // 成都链安 // 使用 call 调用执行提案对应的动作
```

```
require(success, "executeProposal failed"); // 成都链安 // 验证 call 调用是否成功
```

```
accountStorage.clearProposalData(_client, _proposer, proposedActionId); // 成都链安 // 从 AccountStorage 合约中清除提案数据
```

```
emit ProposalExecuted(_client, _proposer, _functionData); // 成都链安 // 触发 ProposalExecuted 事件
```

```
}  
// 成都链安 // 提案授权情况检查函数，检查指定提案是否通过授权
```

```
function checkApproval(address _client, address _proposer, bytes4 _proposedActionId, bytes32 _functionHash) internal view {
```

```
    bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer, _proposedActionId); // 成都链安 // 根据传入参数获取提案对应的动作哈希
```

```
    require(hash == _functionHash, "proposal hash unmatched"); // 成都链安 // 检查是否存在相应的提案
```

```
    // 成都链安 // 临时变量 backupCount 和 approvedCount，分别用于记录紧急联系人数量和已对该提案授权的紧急联系人数量
```

```
    uint256 backupCount;
```

```
    uint256 approvedCount;
```

```
    address[] memory approved = accountStorage.getProposalDataApproval(_client, _proposer, _proposedActionId); // 成都链安 // 获取已对该提案授权的紧急联系人数组
```

```
    require(approved.length > 0, "no approval"); // 成都链安 // 已通过该提案的紧急联系人数组必须大于 0
```

```
    // iterate backup list
```

```
    for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) { // 成都链安 // 根据序号遍历获取当事人的紧急联系人对该提案的授权情况
```

```
        address backup = accountStorage.getBackupAddress(_client, i);
```

```
        uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_client, i);
```

```
        uint256 expiryDate = accountStorage.getBackupExpiryDate(_client, i);
```

```
        if (backup != address(0) && isEffectiveBackup(effectiveDate, expiryDate)) {
```

```
            // count how many backups in backup list
```

```
            backupCount += 1;
```

```
            // iterate approved array
```

```
            for (uint256 k = 0; k < approved.length; k++) {
```

```
                if (backup == approved[k]) {
```

```
                    // count how many approved backups still exist in backup list
```

```
                    approvedCount += 1;
```

```
                }
```

```
    }
    }
    }
    require(backupCount > 0, "no backup in list"); // 成都链安 // 当事人的紧急联系人数量必须大于 0
    uint256 threshold = SafeMath.ceil(backupCount*6, 10); // 成都链安 // 获取已对该提案授权的紧急联系人所占比例
    require(approvedCount >= threshold, "must have 60% approval at least"); // 成都链安 // 要求所占比例必须大于等于 60%
}
// 成都链安 // 关系检查函数, 检查_backup 是否是_client 当前生效的紧急联系人
function checkRelation(address _client, address _backup) internal view {
    // 成都链安 // 非零地址检查
    require(_backup != address(0), "backup cannot be 0x0");
    require(_client != address(0), "client cannot be 0x0");
    bool isBackup;
    for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) { // 成都链安 // 遍历_client 的紧急联系人
        address backup = accountStorage.getBackupAddress(_client, i);
        uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_client, i);
        uint256 expiryDate = accountStorage.getBackupExpiryDate(_client, i);
        // backup match and effective and not expired
        if (_backup == backup && isEffectiveBackup(effectiveDate, expiryDate)) {
            isBackup = true;
            break;
        }
    }
    require(isBackup, "backup does not exist in list");
}

function isEffectiveBackup(uint256 _effectiveDate, uint256 _expiryDate) internal view returns(bool) {
    return (_effectiveDate <= now) && (_expiryDate > now);
}

// ***** Signature ***** //
// 成都链安 // 计算签名哈希
function getSignHash(bytes memory _data, uint256 _nonce) internal view returns(bytes32) {
    // use EIP 191
    // 0x1900 + this logic address + data + nonce of signing key
    bytes32 msgHash = keccak256(abi.encodePacked(byte(0x19), byte(0), address(this), _data, _nonce));
    bytes memory prefix = "\x19Ethereum Signed Message:\n32";
    bytes32 prefixedHash = keccak256(abi.encodePacked(prefix, msgHash));
    return prefixedHash;
}
// 成都链安 // 验证签名
```

```
function verifySig(address _signingKey, bytes memory _signature, bytes32 _signHash)
internal pure {
    uint8 v;
    bytes32 r;
    bytes32 s;

    (v,r,s) = signatureSplit(_signature);
    address recoveredAddr = ecrecover(_signHash, v, r, s);
    require(recoveredAddr == _signingKey, "signature verification failed");
}

/**
 * @dev divides bytes signature into `uint8 v, bytes32 r, bytes32 s`
 * @param _signature concatenated vrs signatures
 */
function signatureSplit(bytes memory _signature)
    internal
    pure
    returns (uint8 v, bytes32 r, bytes32 s)
{
    // The signature format is a compact form of:
    // {bytes32 r}{bytes32 s}{uint8 v}
    // Compact means, uint8 is not padded to 32 bytes.
    // solium-disable-next-line security/no-inline-assembly
    assembly {
        r := mload(add(_signature, 32))
        s := mload(add(_signature, 64))
        // Here we are loading the last 32 bytes, including 31 bytes
        // of 's'. There is no 'mload8' to do this.
        //
        // 'byte' is not working due to the Solidity parser, so lets
        // use the second best option, 'and'
        v := and(mload(add(_signature, 65)), 0xff)
    }
    if (v < 27) {
        v += 27;
    }
    require(v == 27 || v == 28, "signature v invalid");
}

/* get signer account address from data
 * @dev Gets an address encoded as the first argument in transaction data
 * @param b The byte array that should have an address as first argument
 * @returns a The address retrieved from the array
 */
function getSignerAddress(bytes memory _b) internal pure returns (address _a) {
    require(_b.length >= 36, "invalid bytes");
    // solium-disable-next-line security/no-inline-assembly
```

```
assembly {
    let mask := 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    _a := and(mask, mload(add(_b, 36)))
    // b = {length:32}{method sig:4}{address:32}{...}
    // 36 is the offset of the first parameter of the data, if encoded properly.
    // 32 bytes for the length of the bytes array, and the first 4 bytes for the
function signature.
    // 32 bytes is the length of the bytes array!!!!
}

// get method id, first 4 bytes of data
function getMethodId(bytes memory _b) internal pure returns (bytes4 _a) {
    require(_b.length >= 4, "invalid data");
    // solium-disable-next-line security/no-inline-assembly
    assembly {
        // 32 bytes is the length of the bytes array
        _a := mload(add(_b, 32))
    }
}

// _nonce is timestamp in microsecond(1/1000000 second)
// 成都链安 // 检查并更新账户指定公钥对应的 nonce 值
function checkAndUpdateNonce(address _account, uint256 _nonce, uint256 _index)
internal {
    // check operation key status
    if (_index > 0) {
        require(accountStorage.getKeyStatus(_account, _index) != 1, "frozen key");
    }
    address key = accountStorage.getKeyData(_account, _index); // 成都链安 // 根据账户
和序号获取对应的公钥
    // 成都链安 // nonce 值检查, 需要_nonce 大于该公钥上一次操作的 nonce 并且不超过一天
    require(_nonce > keyNonce[key], "nonce too small");
    require(SafeMath.div(_nonce, 1000000) <= now + 86400, "nonce too big"); //
86400=24*3600 seconds

    keyNonce[key] = _nonce; // 成都链安 // 更新公钥对应的 nonce 值
}

// 成都链安 // ### File:contracts\AccountLogic.sol ###
pragma solidity ^0.5.4;

import "./BaseLogic.sol";

/**
 * @title AccountLogic
 */
contract AccountLogic is BaseLogic {
```



```
// 成都链安 // 账户操作相关的事件声明
event AccountLogicEntered(address indexed account, bytes data, uint256 indexed nonce);
event AccountLogicInitialised(address indexed account);
event ChangeAdminKeyTriggered(address indexed account, address pkNew);
event ChangeAdminKeyByBackupTriggered(address indexed account, address pkNew);
event changeAllOperationKeysTriggered(address indexed account, address[] pks);
event UnfreezeTriggered(address indexed account);

// ***** Constructor ***** //

constructor(AccountStorage _accountStorage)
    BaseLogic(_accountStorage)
    public
{
}

// ***** Initialization ***** //
// 成都链安 // 初始化账户，在本合约中触发 AccountLogicInitialised 事件
function initAccount(BaseAccount _account) external onlyAccount(_account) {
    emit AccountLogicInitialised(address(_account));
}

// ***** action entry ***** //

/* AccountLogic has 12 actions called from 'enter':
    changeAdminKey, addOperationKey, changeAllOperationKeys, freeze, unfreeze,
    removeBackup, cancelDelay, cancelAddBackup, cancelRemoveBackup,
    proposeAsBackup, approveProposal, cancelProposal
*/
function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
external {
    require(getMethodId(_data) !=
bytes4(keccak256("changeAdminKeyByBackup(address,address)")), "invalid data"); // 成都链安
// 不允许动作为 changeAdminKeyByBackup
    address account = getSignerAddress(_data); // 成都链安 // 获取原始账户地址
    uint256 keyIndex = getKeyIndex(_data); // 成都链安 // 根据动作数据获取公钥对应的序号
    checkAndUpdateNonce(account, _nonce, keyIndex); // 成都链安 // 检查并更新公钥对应的 nonce 值
    address signingKey = accountStorage.getKeyData(account, keyIndex); // 成都链安 // 获取签名公钥
    bytes32 signHash = getSignHash(_data, _nonce); // 成都链安 // 计算签名哈希
    verifySig(signingKey, _signature, signHash); // 成都链安 // 验证签名

    // solium-disable-next-line security/no-low-level-calls
    (bool success,) = address(this).call(_data); // 成都链安 // 使用 call 调用执行账户动作
    require(success, "calling self failed"); // 成都链安 // 验证 call 调用是否成功
```

```
emit AccountLogicEntered(account, _data, _nonce); // 成都链安 // 触发
AccountLogicEntered 事件
}

// ***** change admin key ***** //

// called from 'enter'
// 成都链安 // 修改账户对应的 admin 公钥，该函数只能通过本合约的 enter 函数调用
function changeAdminKey(address payable _account, address _pkNew) external onlySelf {
    require(_pkNew != address(0), "0x0 is invalid"); // 成都链安 // 新的 admin 公钥不能
    为零地址
    address pk = accountStorage.getKeyData(_account, 0); // 成都链安 // 获取账户原来的
    admin 公钥
    require(pk != _pkNew, "identical admin key exists"); // 成都链安 // 新的 admin 公钥
    不能和账户原来的 admin 公钥相同
    bytes4 actionId = getActionId("changeAdminKey(address,address)"); // 成都链安 //
    获取动作 ID
    require(accountStorage.getDelayDataHash(_account, actionId) == 0, "delay data
    already exists"); // 成都链安 // 该动作不能已存在，即需要将上次修改 admin 公钥的动作执行后
    才能再次修改 admin 公钥
    bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew)); //
    成都链安 // 生成动作哈希
    accountStorage.setDelayData(_account, actionId, hash, now +
    getDelayTime(TYPE_CHANGE_ADMIN_KEY)); // 成都链安 // 设置动作延迟
}

// called from external
// 成都链安 // 触发(执行)修改账户对应的 admin 公钥的动作
function triggerChangeAdminKey(address payable _account, address _pkNew) external {
    bytes4 actionId = getActionId("changeAdminKey(address,address)"); // 成都链安 //
    获取动作 ID
    bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew)); //
    成都链安 // 生成动作哈希
    require(hash == accountStorage.getDelayDataHash(_account, actionId), "delay hash
    unmatched"); // 成都链安 // 对比生成的动作哈希与等待处理的动作哈希是否一致
    // 成都链安 // 验证当前触发时间是否合法
    uint256 due = accountStorage.getDelayDataDueTime(_account, actionId);
    require(due > 0, "delay data not found");
    require(due <= now, "too early to trigger changeAdminKey");
    // 成都链安 // 修改 admin 公钥，清除对应的动作延迟数据
    accountStorage.setKeyData(_account, 0, _pkNew);
    accountStorage.clearDelayData(_account, actionId);
    emit ChangeAdminKeyTriggered(_account, _pkNew); // 成都链安 // 触发
    ChangeAdminKeyTriggered 事件
}

// ***** change admin key by backup proposal ***** //
```

```
// called from 'executeProposal'
// 成都链安 // 通过紧急联系人修改 admin 公钥，该函数只能通过 executeProposal 函数调用
function changeAdminKeyByBackup(address payable _account, address _pkNew) external
onlySelf {
    require(_pkNew != address(0), "0x0 is invalid"); // 成都链安 // 新的 admin 公钥不能
    为零地址
    address pk = accountStorage.getKeyData(_account, 0); // 成都链安 // 获取账户原来的
    admin 公钥
    require(pk != _pkNew, "identical admin key exists"); // 成都链安 // 新的 admin 公钥
    不能和账户原来的 admin 公钥相同
    bytes4 actionId = getActionId("changeAdminKeyByBackup(address,address)"); // 成都
    链安 // 获取动作 ID
    require(accountStorage.getDelayDataHash(_account, actionId) == 0, "delay data
    already exists"); // 成都链安 // 该动作不能已存在，即需要将上次修改 admin 公钥的动作执行后
    才能再次修改 admin 公钥
    bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account,
    _pkNew)); // 成都链安 // 生成动作哈希
    accountStorage.setDelayData(_account, actionId, hash, now +
    getDelayTime(TYPE_CHANGE_ADMIN_KEY_BY_BACKUP)); // 成都链安 // 设置动作延迟数据
}

// called from external
// 成都链安 // 触发(执行)账户的紧急联系人修改 admin 公钥的动作
function triggerChangeAdminKeyByBackup(address payable _account, address _pkNew)
external {
    bytes4 actionId = getActionId("changeAdminKeyByBackup(address,address)"); // 成都
    链安 // 获取动作 ID
    bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account,
    _pkNew)); // 成都链安 // 生成动作哈希
    require(hash == accountStorage.getDelayDataHash(_account, actionId), "delay hash
    unmatched"); // 成都链安 // 对比生成的动作哈希与等待处理的动作哈希是否一致
    // 成都链安 // 验证当前触发时间是否合法
    uint256 due = accountStorage.getDelayDataDueTime(_account, actionId);
    require(due > 0, "delay data not found");
    require(due <= now, "too early to trigger changeAdminKeyByBackup");
    // 成都链安 // 修改 admin 公钥，清除对应的动作延迟数据
    accountStorage.setKeyData(_account, 0, _pkNew);
    accountStorage.clearDelayData(_account, actionId);
    emit ChangeAdminKeyByBackupTriggered(_account, _pkNew); // 成都链安 // 触发
    ChangeAdminKeyTriggered 事件
}

// ***** add operation key ***** //

// called from 'enter'
// 成都链安 // 添加公钥函数，该函数只能通过本合约的 enter 函数调用
function addOperationKey(address payable _account, address _pkNew, uint256 _index)
external onlySelf {
```

```
require(_index > 0, "invalid operation key index"); // 成都链安 // _index 必须大于
零
require(_pkNew != address(0), "0x0 is invalid"); // 成都链安 // _pkNew 不能为零地
址
address pk = accountStorage.getKeyData(_account, _index); // 成都链安 // 获取账户
指定序号对应的公钥
require(pk == address(0), "operation key already exists"); // 成都链安 // 该序号对
应的公钥需要未被添加
accountStorage.setKeyData(_account, _index, _pkNew); // 成都链安 // 添加账户公钥
accountStorage.incrementKeyCount(_account); // 成都链安 // 更新账户的公钥数量
}

// ***** change all operation keys ***** //

// called from 'enter'
function changeAllOperationKeys(address payable _account, address[] calldata _pks)
external onlySelf {
    // 成都链安 // 检查_pks 数组长度
    uint256 keyCount = accountStorage.getKeyCount(_account);
    require(_pks.length == keyCount, "invalid number of keys");
    bytes4 actionId = getActionId("changeAllOperationKeys(address,address[])"); // 成
都链安 // 获取动作 ID
    require(accountStorage.getDelayDataHash(_account, actionId) == 0, "delay data
already exists"); // 成都链安 // 该动作不能已存在，即上次提交的修改该账户所有的 operation
公钥的动作请求必须执行完成后才能再次修改该账户的所有公钥
    for (uint256 i = 0; i < keyCount; i++) { // 成都链安 // 传入的_pks 公钥数组元素不
能有零地址
        address pk = _pks[i];
        require(pk != address(0), "0x0 is invalid");
    }
    bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account,
_pks)); // 成都链安 // 生成动作哈希
    accountStorage.setDelayData(_account, actionId, hash, now +
getDelayTime(TYPE_CHANGE_OPERATION_KEY)); // 成都链安 // 设置动作延迟数据
}

// called from external
function triggerChangeAllOperationKeys(address payable _account, address[] calldata
_pks) external {
    bytes4 actionId = getActionId("changeAllOperationKeys(address,address[])"); // 成
都链安 // 获取动作 ID
    bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account,
_pks)); // 成都链安 // 生成动作哈希
    require(hash == accountStorage.getDelayDataHash(_account, actionId), "delay hash
unmatch"); // 成都链安 // 对比生成的动作哈希与等待处理的动作哈希是否一致
    // 成都链安 // 验证当前触发时间是否合法
    uint256 due = accountStorage.getDelayDataDueTime(_account, actionId);
    require(due > 0, "delay data not found");
```

```
require(due <= now, "too early to trigger changeAllOperationKeys");
for (uint256 i = 0; i < accountStorage.getKeyCount(_account); i++) { // 成都链安
// 修改账户对应的所有 operation 公钥
    address pk = _pks[i];
    accountStorage.setKeyData(_account, i+1, pk);
    accountStorage.setKeyStatus(_account, i+1, 0);
}
accountStorage.clearDelayData(_account, actionId); // 成都链安 // 清除该动作的延迟
数据
    emit changeAllOperationKeysTriggered(_account, _pks); // 成都链安 // 触发
changeAllOperationKeysTriggered 事件
}

// ***** freeze/unfreeze all operation keys ***** //

// called from 'enter'
function freeze(address payable _account) external onlySelf {
    for (uint256 i = 1; i <= accountStorage.getKeyCount(_account); i++) { // 成都链安
// 对账户的所有 operation 公钥进行冻结
        if (accountStorage.getKeyStatus(_account, i) == 0) {
            accountStorage.setKeyStatus(_account, i, 1);
        }
    }
}

// called from 'enter'
// 成都链安 // 解除对账户的 operation 公钥的冻结, 该函数只能通过本合约的 enter 函数调用
function unfreeze(address payable _account) external onlySelf {
    bytes4 actionId = getActionId("unfreeze(address)");
    require(accountStorage.getDelayDataHash(_account, actionId) == 0, "delay data
already exists");
    bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account));
    accountStorage.setDelayData(_account, actionId, hash, now +
getDelayTime(TYPE_UNFREEZE_KEY));
}

// called from external
// 成都链安 // 触发(执行)对账户 operation 公钥冻结状态的解除
function triggerUnfreeze(address payable _account) external {
    bytes4 actionId = getActionId("unfreeze(address)"); // 成都链安 // 获取动作 ID
    bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account)); // 成都链安 //
生成动作哈希
    require(hash == accountStorage.getDelayDataHash(_account, actionId), "delay hash
unmatch"); // 成都链安 // 对比生成的动作哈希与等待处理的动作哈希是否一致
    // 成都链安 // 验证当前触发时间是否合法
    uint256 due = accountStorage.getDelayDataDueTime(_account, actionId);
    require(due > 0, "delay data not found");
    require(due <= now, "too early to trigger unfreeze");
}
```



```
// 成都链安 // 解除对账户所有 operation 公钥的冻结
for (uint256 i = 1; i <= accountStorage.getKeyCount(_account); i++) {
    if (accountStorage.getKeyStatus(_account, i) == 1) {
        accountStorage.setKeyStatus(_account, i, 0);
    }
}
accountStorage.clearDelayData(_account, actionId); // 成都链安 // 清除该动作的延迟
数据
emit UnfreezeTriggered(_account); // 成都链安 // 触发 UnfreezeTriggered 事件
}

// ***** remove backup ***** //

// called from 'enter'
// 成都链安 // 移除紧急联系人，该函数只能通过本合约的 enter 函数调用
function removeBackup(address payable _account, address _backup) external onlySelf {
    uint256 index = findBackup(_account, _backup); // 成都链安 // 调用 findBackup 函数
    获取该紧急联系人对应序号
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");

    accountStorage.setBackupExpiryDate(_account, index, now +
    getDelayTime(TYPE_CHANGE_BACKUP)); // 成都链安 // 设置动作延迟数据
}

// return backupData index(0~5), 6 means not found
// do make sure _backup is not 0x0
function findBackup(address _account, address _backup) public view returns(uint) {
    uint index = MAX_DEFINED_BACKUP_INDEX + 1;
    if (_backup == address(0)) {
        return index;
    }
    for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
        address b = accountStorage.getBackupAddress(_account, i);
        if (b == _backup) {
            index = i;
            break;
        }
    }
    return index;
}

// ***** cancel delay action ***** //

// called from 'enter'
// 成都链安 // 清除指定的动作请求，该函数只能通过本合约的 enter 函数调用
function cancelDelay(address payable _account, bytes4 _actionId) external onlySelf {
    uint256 due = accountStorage.getDelayDataDueTime(_account, _actionId);
    require(due > now, "delay item already due");
}
```



```
accountStorage.clearDelayData(_account, _actionId); // 成都链安 // 清除指定动作请
求的延迟数据
}

// called from 'enter'
// 成都链安 // 取消添加紧急联系人，该函数只能通过本合约的 enter 函数调用
function cancelAddBackup(address payable _account, address _backup) external onlySelf
{
    uint256 index = findBackup(_account, _backup); // 成都链安 // 调用 findBackup 函数
    获取指定紧急联系人对应的序号
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
    // 成都链安 // 指定的紧急联系人必须未生效
    uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_account, index);
    require(effectiveDate > now, "already effective");
    accountStorage.clearBackupData(_account, index); // 成都链安 // 删除账户对应的紧急
    联系人添加请求
}

// called from 'enter'
// 成都链安 // 延长紧急联系人失效时间，该函数只能通过本合约的 enter 函数调用
function cancelRemoveBackup(address payable _account, address _backup) external
onlySelf {
    uint256 index = findBackup(_account, _backup); // 成都链安 // 调用 findBackup 函数
    获取指定紧急联系人对应的序号
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
    // 成都链安 // 指定的紧急联系人必须未失效
    uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, index);
    require(expiryDate > now, "already expired");
    accountStorage.setBackupExpiryDate(_account, index, uint256(-1)); // 成都链安 //
    设置该紧急联系人的失效时间为 2**256-1
}

// ***** propose, approve and cancel proposal ***** //

// called from 'enter'
// proposer is backup in the case of 'proposeAsBackup'
// 成都链安 // 账户紧急联系人发起修改账户 admin 公钥提案，该函数只能通过本合约的 enter
函数调用
function proposeAsBackup(address _backup, address payable _client, bytes calldata
_functionData) external onlySelf {
    // 成都链安 // 只能是 changeAdminKeyByBackup 动作
    bytes4 proposedActionId = getMethodId(_functionData);
    require(proposedActionId ==
bytes4(keccak256("changeAdminKeyByBackup(address,address)")), "invalid proposal by
backup");
    checkRelation(_client, _backup); // 成都链安 // 发起者与账户关系检查，发起者必须是
    账户的紧急联系人
    bytes32 functionHash = keccak256(_functionData);
```

```
accountStorage.setProposalData(_client, _backup, proposedActionId, functionHash,
_backup); // 成都链安 // 提交提案
}

// called from 'enter'
// 成都链安 // 紧急联系人对指定提案投票，该函数只能通过本合约的 enter 函数调用
function approveProposal(address _backup, address payable _client, address _proposer,
bytes4 _proposedActionId, bytes32 _functionHash) external onlySelf {
    require(_functionHash != 0, "invalid hash");
    checkRelation(_client, _backup);
    bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer,
_proposedActionId);
    require(hash == _functionHash, "proposal unmatched");
    accountStorage.setProposalData(_client, _proposer, _proposedActionId,
_functionHash, _backup);
}

// called from 'enter'
// 成都链安 // 取消指定提案，该函数只能通过本合约的 enter 函数调用
function cancelProposal(address payable _client, address _proposer, bytes4
_proposedActionId) external onlySelf {
    accountStorage.clearProposalData(_client, _proposer, _proposedActionId);
}

// ***** internal functions ***** //

/*
index 0: admin key
    1: asset(transfer)
    2: adding
    3: reserved(dapp)
    4: assist
*/
function getKeyIndex(bytes memory _data) internal pure returns (uint256) {
    uint256 index;
    bytes4 methodId = getMethodId(_data);
    if (methodId == bytes4(keccak256("addOperationKey(address,address,uint256)"))) {
        index = 2; //adding key
    } else if (methodId ==
bytes4(keccak256("proposeAsBackup(address,address,bytes)"))) {
        index = 4; //assist key
    } else if (methodId ==
bytes4(keccak256("approveProposal(address,address,address,bytes4,bytes32)"))) {
        index = 4; //assist key
    } else {
        index = 0; //admin key
    }
    return index;
}
```

```
}  
  
}  
  
// 成都链安 // #### File:contracts\DappLogic.sol ####  
pragma solidity ^0.5.4;  
  
import "./BaseLogic.sol";  
  
contract DappLogic is BaseLogic {  
  
    /*  
    index 0: admin key  
        1: asset(transfer)  
        2: adding  
        3: reserved(dapp)  
        4: assist  
  
    */  
    uint constant internal DappKeyIndex = 3;  
  
    // ***** Events ***** //  
  
    event DappLogicInitialised(address indexed account); // 成都链安 // 声明  
DappLogicInitialised 事件  
    event DappLogicEntered(address indexed account, bytes data, uint256 indexed nonce); //  
成都链安 // 声明 DappLogicEntered 事件  
  
    // ***** Constructor ***** //  
    constructor(AccountStorage _accountStorage)  
        BaseLogic(_accountStorage)  
    public  
    {  
    }  
  
    // ***** Initialization ***** //  
  
    function initAccount(BaseAccount _account) external onlyAccount(_account) {  
        emit DappLogicInitialised(address(_account));  
    }  
  
    // ***** action entry ***** //  
    // 成都链安 // DappLogic 合约的操作入口函数  
    function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)  
external {  
        address account = getSignerAddress(_data); // 成都链安 // 验证该动作的原始账户与传  
入的_account 账户是否一致  
        checkAndUpdateNonce(account, _nonce, DappKeyIndex); // 成都链安 // 检查并更新公钥  
对应的 nonce 值
```

```
        address dappKey = accountStorage.getKeyData(account, DappKeyIndex); // 成都链安 //
获取签名公钥
        bytes32 signHash = getSignHash(_data, _nonce); // 成都链安 // 计算签名哈希
        verifySig(dappKey, _signature, signHash); // 成都链安 // 签名验证

        // solium-disable-next-line security/no-low-level-calls
        (bool success,) = address(this).call(_data); // 成都链安 // 使用 call 调用执行账户
动作
        require(success, "calling self failed"); // 成都链安 // 验证 call 调用是否成功
        emit DappLogicEntered(account, _data, _nonce); // 成都链安 // 触发
DappLogicEntered 事件
    }

    // ***** call Dapp ***** //

    // called from 'enter'
    // call other contract from base account
    function callContract(address payable _account, address payable _target, uint256
_value, bytes calldata _methodData) external onlySelf {
        BaseAccount(_account).invoke(_target, _value, _methodData);
    }
}

// 成都链安 // #### File:contracts\DualsigsLogic.sol ####
pragma solidity ^0.5.4;

import "./BaseLogic.sol";

/**
 * @title DualsigsLogic
 */
contract DualsigsLogic is BaseLogic {

    event DualsigsLogicInitialised(address indexed account); // 成都链安 // 声明
DualsigsLogicInitialised 事件
    event DualsigsLogicEntered(address indexed client, address indexed backup, address ind
exed client, address indexed backup, bytes data, uint256 indexed clientNonce, uint256 back
upNonce); // 成都链安 // 声明 DualsigsLogicEntered 事件

    // ***** Constructor ***** //

    constructor(AccountStorage _accountStorage)
        BaseLogic(_accountStorage)
    public
    {
```

```
}

// ***** Initialization ***** //

function initAccount(BaseAccount _account) external onlyAccount(_account) {
    emit DualsigsLogicInitialised(address(_account));
}

// ***** action entry ***** //

/* DualsigsLogic has 2 actions called from 'enter':
    addBackup, proposeByBoth
*/
// 成都链安 // DualsigsLogic 合约的操作入口函数
function enter(
    bytes calldata _data, bytes calldata _clientSig, bytes calldata _backupSig,
    uint256 _clientNonce, uint256 _backupNonce
)
    external
{
    require(isActionWithDualSigs(_data), "wrong entry"); // 成都链安 // 验证提案方法
    verifyClient(_data, _clientSig, _clientNonce); // 成都链安 // 验证当事人签名
    verifyBackup(_data, _backupSig, _backupNonce); // 成都链安 // 验证共同发起人签名

    // solium-disable-next-line security/no-low-level-calls
    (bool success,) = address(this).call(_data); // 成都链安 // 使用 call 调用执行账户
    动作
    require(success, "enterWithDualSigs failed");
    emit DualsigsLogicEntered(_data, _clientNonce, _backupNonce); // 成都链安 // 触发
    DualsigsLogicEntered 事件
}
// 成都链安 // 当事人签名验证
function verifyClient(bytes memory _data, bytes memory _clientSig, uint256
_clientNonce) internal {
    address client = getSignerAddress(_data);
    //client sign with admin key
    uint256 clientKeyIndex = 0;
    checkAndUpdateNonce(client, _clientNonce, clientKeyIndex);
    verifySig(accountStorage.getKeyData(client, clientKeyIndex), _clientSig,
    getSignHash(_data, _clientNonce));
}
// 成都链安 // 紧急联系人签名验证
function verifyBackup(bytes memory _data, bytes memory _backupSig, uint256
_backupNonce) internal {
    address backup = getSecondSignerAddress(_data);
    //backup sign with assist key
    uint256 backupKeyIndex = 4;
    checkAndUpdateNonce(backup, _backupNonce, backupKeyIndex);
```

```
        verifySig(accountStorage.getKeyData(backup, backupKeyIndex), _backupSig,
getSignHash(_data, _backupNonce));
    }

    // ***** change admin key ***** //

    // called from 'executeProposal'
    // 成都链安 // 无延迟修改账户的 admin 公钥，该函数只能通过 executeProposal 函数调用
    function changeAdminKeyWithoutDelay(address payable _account, address _pkNew) external
onlySelf {
        // 成都链安 // 新的 admin 公钥不能和原来的相同
        address pk = accountStorage.getKeyData(_account, 0);
        require(pk != _pkNew, "identical admin key already exists");
        require(_pkNew != address(0), "0x0 is invalid"); // 成都链安 // 新的 admin 公钥不能
为零地址
        accountStorage.setKeyData(_account, 0, _pkNew); // 成都链安 // 修改 admin 公钥
    }

    // ***** change all operation keys ***** //

    // called from 'executeProposal'
    // 成都链安 // 无延迟修改所有 operation 公钥，该函数只能通过 executeProposal 函数调用
    function changeAllOperationKeysWithoutDelay(address payable _account, address[]
calldata _pks) external onlySelf {
        uint256 keyCount = accountStorage.getKeyCount(_account); // 成都链安 // 获取
operation 公钥数量
        require(_pks.length == keyCount, "invalid number of keys"); // 成都链安 // 传入的
新的 operation 公钥数量需要和原始的 operation 公钥数量相同
        for (uint256 i = 0; i < keyCount; i++) { // 成都链安 // 替换所有原始的 operation 公
钥

            address pk = _pks[i];
            require(pk != address(0), "0x0 is invalid");
            accountStorage.setKeyData(_account, i+1, pk);
            accountStorage.setKeyStatus(_account, i+1, 0);

        }
    }

    // ***** freeze/unfreeze all operation keys ***** //

    // called from 'executeProposal'
    // 成都链安 // 无延迟解除对 operation 公钥的冻结，该函数只能通过 executeProposal 函数调
用
    function unfreezeWithoutDelay(address payable _account) external onlySelf {
        for (uint256 i = 0; i < accountStorage.getKeyCount(_account); i++) { // 成都链安
// 解除对账户所有的 operation 公钥的冻结
            if (accountStorage.getKeyStatus(_account, i+1) == 1) {
                accountStorage.setKeyStatus(_account, i+1, 0);
            }
        }
    }
}
```



```
}

// ***** add backup ***** //

// called from 'enter'
// 成都链安 // 添加紧急联系人, 该函数只能通过本合约的 enter 函数调用
function addBackup(address payable _account, address _backup) external onlySelf {
    require(_account != _backup, "cannot be backup of oneself"); // 成都链安 // 账户本
身不能成为紧急联系人
    uint256 index = findAvailableSlot(_account, _backup); // 成都链安 // 获取空的或过
期的紧急联系人最低序号
    require(index <= MAX_DEFINED_BACKUP_INDEX, "invalid or duplicate or no vacancy");
// 成都链安 // 紧急联系人数量限制
    accountStorage.setBackup(_account, index, _backup, now +
getDelayTime(TYPE_CHANGE_BACKUP), uint256(-1)); // 成都链安 // 添加紧急联系人并设置其失效
时间为 2**256-1
}

// return backupData index(0~5), 6 means not found
// 'available' means empty or expired
function findAvailableSlot(address _account, address _backup) public view
returns(uint) {
    uint index = MAX_DEFINED_BACKUP_INDEX + 1;
    if (_backup == address(0)) {
        return index;
    }
    for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
        address backup = accountStorage.getBackupAddress(_account, i);
        uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, i);
        // _backup already exists and not expired
        if ((backup == _backup) && (expiryDate > now)) {
            return MAX_DEFINED_BACKUP_INDEX + 1;
        }
        if (index > MAX_DEFINED_BACKUP_INDEX) {
            // zero address or backup expired
            if ((backup == address(0)) || (expiryDate <= now)) {
                index = i;
            }
        }
    }
    return index;
}

// ***** propose, approve, execute and cancel proposal
***** //

// called from 'enter'
// proposer is client in the case of 'proposeByBoth'
```

```
// 成都链安 // 账户与一个紧急联系人共同发起提案
function proposeByBoth(address payable _client, address _backup, bytes calldata
_functionData) external onlySelf {
    bytes4 proposedActionId = getMethodId(_functionData); // 成都链安 // 获取提案的动
    作 ID
    require(isFastAction(proposedActionId), "invalid proposal"); // 成都链安 // 验证动
    作 ID, 只允许是设定好的动作
    checkRelation(_client, _backup); // 成都链安 // 验证_backup 地址是否为账户的紧急联
    系人
    bytes32 functionHash = keccak256(_functionData); // 成都链安 // 生成动作哈希
    accountStorage.setProposalData(_client, _client, proposedActionId, functionHash,
    _backup); // 成都链安 // 提交提案
}

// 成都链安 // 内部函数, 规定可使用共同提案的动作
function isFastAction(bytes4 _actionId) internal pure returns(bool) {
    if ((_actionId ==
bytes4(keccak256("changeAdminKeyWithoutDelay(address,address)"))) ||
        (_actionId ==
bytes4(keccak256("changeAllOperationKeysWithoutDelay(address,address[])"))) ||
        (_actionId == bytes4(keccak256("unfreezeWithoutDelay(address)"))))
    {
        return true;
    }
    return false;
}

// ***** internal functions ***** //
// 成都链安 // 内部函数, 规定可使用 enter 入口调用的动作
function isActionWithDualSigs(bytes memory _data) internal pure returns(bool) {
    bytes4 methodId = getMethodId(_data);
    if ((methodId == bytes4(keccak256("addBackup(address,address)"))) ||
        (methodId == bytes4(keccak256("proposeByBoth(address,address,bytes)")))) {
        return true;
    }
    return false;
}

function getSecondSignerAddress(bytes memory _b) internal pure returns (address _a) {
    require(_b.length >= 68, "data length too short");
    // solium-disable-next-line security/no-inline-assembly
    assembly {
        //68 = 32 + 4 + 32
        let mask := 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
        _a := and(mask, mload(add(_b, 68)))
    }
}
```

```
// 成都链安 // #### File:contracts\TransferLogic.sol ####
pragma solidity ^0.5.4;

import "./BaseLogic.sol";

contract TransferLogic is BaseLogic {

    /*
    index 0: admin key
        1: asset(transfer)
        2: adding
        3: reserved(dapp)
        4: assist
    */
    uint constant internal TranferKeyIndex = 1; // 成都链安 // 用于操作 TransferLogic 合约
    的是账户序号为 1 的公钥

    // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)"))`
    bytes4 private constant ERC721_RECEIVED = 0x150b7a02;

    modifier onlySelf() {
        require (msg.sender == address(this), "only internal call is allowed");
        _;
    }

    // ***** Events ***** //

    event TransferLogicInitialised(address indexed account); // 成都链安 // 声明
    TransferLogicInitialised 事件
    event TransferLogicEntered(address indexed account, bytes data, uint256 indexed
    nonce); // 成都链安 // 声明 TransferLogicEntered 事件
    // event Transfer(address indexed from, address indexed to, address indexed token,
    uint256 amount);
    // event NftTransfer(address indexed from, address to, address indexed nftContract,
    uint256 indexed tokenId, bytes data);

    // ***** Constructor ***** //

    constructor(AccountStorage _accountStorage)
        BaseLogic(_accountStorage)
        public
    {
    }

    // ***** Initialization ***** //

    // enable staic call 'onERC721Received' from base account
```

```
function initAccount(BaseAccount _account) external onlyAccount(_account) {
    _account.enableStaticCall(address(this), ERC721_RECEIVED);
    emit TransferLogicInitialised(address(_account));
}

// ***** action entry ***** //
// 成都链安 // TransferLogic 合约的操作入口函数
function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
external {
    address account = getSignerAddress(_data);
    checkAndUpdateNonce(account, _nonce, TransferKeyIndex); // 成都链安 // 检查并更新
Transfer 公钥对应的 nonce 值

    address assetKey = accountStorage.getKeyData(account, TransferKeyIndex); // 成都链
安 // 获取对应公钥
    bytes32 signHash = getSignHash(_data, _nonce); // 成都链安 // 生成签名哈希
    verifySig(assetKey, _signature, signHash); // 成都链安 // 验证签名

    // solium-disable-next-line security/no-low-level-calls
    (bool success,) = address(this).call(_data); // 成都链安 // 使用 call 调用执行账户
动作
    require(success, "calling self failed"); // 成都链安 // 检查 call 调用是否成功
    emit TransferLogicEntered(account, _data, _nonce); // 成都链安 // 触发
TransferLogicEntered 事件
}

// ***** transfer assets ***** //

// called from 'enter'
// signer is '_from'
function transferEth(address payable _from, address _to, uint256 _amount) external
onlySelf {
    BaseAccount(_from).invoke(_to, _amount, "");
}

// called from 'enter'
// signer is '_from'
function transferErc20(address payable _from, address _to, address _token, uint256
_amount) external onlySelf {
    bytes memory methodData = abi.encodeWithSignature("transfer(address,uint256)",
_to, _amount);
    BaseAccount(_from).invoke(_token, 0, methodData);
}

// called from 'enter'
// signer is '_approvedSpender'
// make sure '_from' has approved allowance to '_approvedSpender'
```

```
function transferApprovedErc20(address payable _approvedSpender, address _from,
address _to, address _token, uint256 _amount) external onlySelf {
    bytes memory methodData =
abi.encodeWithSignature("transferFrom(address,address,uint256)", _from, _to, _amount);
    BaseAccount(_approvedSpender).invoke(_token, 0, methodData);
}

// called from 'enter'
// signer is '_from'
function transferNft(
    address payable _from, address _to, address _nftContract, uint256 _tokenId, bytes
calldata _data, bool _safe)
    external
    onlySelf
{
    bytes memory methodData;
    if(_safe) {
        methodData =
abi.encodeWithSignature("safeTransferFrom(address,address,uint256,bytes)", _from, _to,
_tokenId, _data);
    } else {
        methodData = abi.encodeWithSignature("transferFrom(address,address,uint256)",
_from, _to, _tokenId);
    }
    BaseAccount(_from).invoke(_nftContract, 0, methodData);
}

// called from 'enter'
// signer is '_approvedSpender'
// make sure '_from' has approved nftToken to '_approvedSpender'
function transferApprovedNft(
    address payable _approvedSpender, address _from, address _to, address
_nftContract, uint256 _tokenId, bytes calldata _data, bool _safe)
    external
    onlySelf
{
    bytes memory methodData;
    if(_safe) {
        methodData =
abi.encodeWithSignature("safeTransferFrom(address,address,uint256,bytes)", _from, _to,
_tokenId, _data);
    } else {
        methodData = abi.encodeWithSignature("transferFrom(address,address,uint256)",
_from, _to, _tokenId);
    }
    BaseAccount(_approvedSpender).invoke(_nftContract, 0, methodData);
}
```

```
// ***** callback of safeTransferFrom ***** //
```

```
function onERC721Received(address _operator, address _from, uint256 _tokenId, bytes
calldata _data) external pure returns (bytes4) {
    return ERC721_RECEIVED;
}
}
```

```
// 成都链安 // #### File:contracts\LogicManager.sol ####
pragma solidity ^0.5.4;

import "./BaseLogic.sol";
import "./base/Owned.sol";
import "./BaseAccount.sol";
import "./AccountStorage.sol";

contract LogicManager is Owned {

    event UpdateLogicSubmitted(address indexed logic, bool value); // 成都链安 // 声明
UpdateLogicSubmitted 事件
    event UpdateLogicDone(address indexed logic, bool value); // 成都链安 // 声明
UpdateLogicDone 事件
    // 成都链安 // 定义添加/移除逻辑合约延迟时间的结构体 pending
    struct pending {
        bool value;
        uint dueTime;
    }

    // The authorised logic modules
    mapping (address => bool) public authorised;

    // updated logics and their due time of becoming effective
    mapping (address => pending) pendingLogics;

    // pending time before updated logics take effect
    uint public pendingTime;

    // how many authorised logics
    uint public logicCount;
    // 成都链安 // 构造函数，初始化逻辑合约
    constructor(address[] memory _initialLogics, uint256 _pendingTime) public
    {
        for (uint i = 0; i < _initialLogics.length; i++) { // 成都链安 // 初始化逻辑合约
            address logic = _initialLogics[i];
            authorised[logic] = true;
            logicCount += 1;
        }
    }
}
```



```
// pendingTime: 4 days for mainnet, 4 minutes for ropsten testnet
pendingTime = _pendingTime; // 成都链安 // 初始化延迟时间
}

// 成都链安 // 判断指定合约是否授权生效
function isAuthorised(address _logic) external view returns (bool) {
    return authorised[_logic];
}

// 成都链安 // 提交添加/移除逻辑合约请求
function submitUpdate(address _logic, bool _value) external onlyOwner {
    pending storage p = pendingLogics[_logic];
    p.value = _value;
    p.dueTime = now + pendingTime;
    emit UpdateLogicSubmitted(_logic, _value);
}

// 成都链安 // 内部函数，根据请求更新逻辑合约
function updateLogic(address _logic, bool _value) internal {
    if (authorised[_logic] != _value) {
        if(_value == true) { // 成都链安 // 添加逻辑合约
            logicCount += 1;
            authorised[_logic] = true;
        }
        else { // 成都链安 // 移除逻辑合约
            logicCount -= 1;
            require(logicCount > 0, "must have at least one logic module");
            delete authorised[_logic];
        }
        emit UpdateLogicDone(_logic, _value);
    }
}

// 成都链安 // 触发(执行)添加/移除逻辑合约的请求
function triggerUpdateLogic(address _logic) external {
    // 成都链安 // 生效时间检查
    pending memory p = pendingLogics[_logic];
    require(p.dueTime > 0, "pending logic not found");
    require(p.dueTime <= now, "too early to trigger updateLogic");
    updateLogic(_logic, p.value); // 成都链安 // 调用内部函数 updateLogic 执行请求
    delete pendingLogics[_logic]; // 成都链安 // 删除请求记录
}
}
```



成都链安
B E O S I N

官方网址

<https://lianantech.com>

电子邮箱

vaas@lianantech.com

微信公众号

