CERTIK

# Audit Report

Produced by CertiK

for

Oct 15, 2019

# CertiK Audit Report For MyKey

Request Date: 2019-08-28
Revision Date: 2019-10-14
Platform Name: Ethereum

CERTIK

# Contents

## Disclaimer

## About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK, in partnership with grants from IBM and the Ethereum Foundation, has developed a proprietary Formal Verification technology to apply rigorous and complete mathematical reasoning against code. This process ensures algorithms, protocols, and business functionalities are secured and working as intended across all platforms.

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis, and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 6.2B in assets.

For more information: https://certik.org/

## Executive Summary

This report has been prepared for MyKey to discover issues and vulnerabilities in the source code of their smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practice and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line by line manual review of the entire codebase by industry experts.

## Vulnerability Classification

CertiK categorizes issues into 3 buckets based on overall risk levels:

**Critical**

The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.

**Medium**

The code implementation does not match the specification under certain conditions, or it could affect the security standard by lost of access control.

**Low**

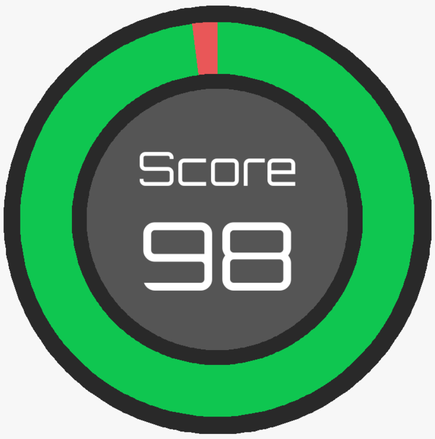
The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilies further down the line.

## Testing Summary

**PASS**

CERTIK *believes this smart contract passes security qualifications to be listed on digital asset exchanges.*

*Oct 14, 2019*



### Type of Issues

CertiK smart label engine applied 100% formal verification coverage on the source code. Our team of engineers ao scanned the source code using our proprietary static analysis tools and code-review methodologies. The following technical issues were found:

| Title | Description | Issues | SWC ID |
|---|---|---|---|
| Integer Overflow and Underflow | An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type. | 0 | SWC-101 |
| Function incorrectness | Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities. | 0 | |
| Buffer Overflow | An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens | 0 | SWC-124 |
| Reentrancy | A malicious contract can call back into the calling contract before the first invocation of the function is finished. | 0 | SWC-107 |
| Transaction Order Dependence | A race condition vulnerability occurs when code depends on the order of the transactions submitted to it. | 0 | SWC-114 |
| Timestamp Dependence | Timestamp can be influenced by minors to some degree. | 1 | SWC-116 |
| Insecure Compiler Version | Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used. | 1 | SWC-102 SWC-103 |
| Insecure Randomness | Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree. | 0 | SWC-120 |

| | | | |
|---|---|---|---|
| "tx.origin" for authorization | tx.origin should not be used for authorization. Use msg.sender instead. | 0 | SWC-115 |
| Delegatecall to Untrusted Callee | Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized. | 0 | SWC-112 |
| State Variable Default Visibility | Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable. | 0 | SWC-108 |
| Function Default Visibility | Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility. | 0 | SWC-100 |
| Uninitialized variables | Uninitialized local storage variables can point to other unexpected storage variables in the contract. | 0 | SWC-109 |
| Assertion Failure | The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement. | 0 | SWC-110 |
| Deprecated Solidity Features | Several functions and operators in Solidity are deprecated and should not be used as best practice. | 0 | SWC-111 |
| Unused variables | Unused variables reduce code quality | 0 | |

## Vulnerability Details

### Critical

No issue found.

### Medium

No issue found.

### Low

No issue found.

## Manual Review Notes

## Review Details

MyKey, a Self-sovereign Identity System built on various public blockchains. It mission is building a one-stop digital life platform for users through digital currency storage, trading, wealth management, games and community, and builds a variety of businesses for developers. The model's blockchain application development and operation ecosystem. In MyKey, users can control their assets autonomously, and when they lose their account, they can easily freeze and recover their accounts. In addition, MyKey is also part of the Web of Trust. In the Web 3.0, MyKey returns the data sovereignty to the user, which fundamentally protects the user's privacy rights.
MyKey Smart Contract Wallet provides following features such as:

- Creating wallet
- Signing a transaction
- Multi-signing
- Managing crypto assets
- Submitting proposals
- Restoring key

### Scope of Audit

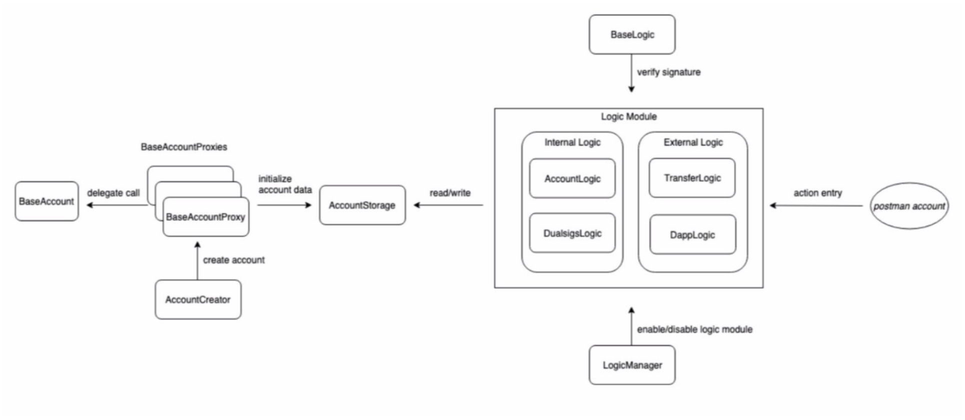
CertiK was chosen by MyKey to audit the design and implementation of its soon to be released smart contract. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

#### Source Code SHA-256 Checksum

- **Account.sol**
  `d91ec9f494b653d3bc32421a1d520605c05bc0a69f8be423bec2bff711980aed`
- **AccountCreator.sol**
  `17193c08483c9a4b4d69d953f2d4de267a12e4c6d0c65e6bc5af9ebb9b94f606`
- **AccountProxy.sol**
  `f334c7926ba32f68f52c64f01ac1d03b7ccdb7f5e88e664a449724b7e81c0dbf`
- **AccountStorage.sol**
  `f8e378640f804e688113395bb1c2baef73c6b6560bbf3667c6940b0cb16892bb`
- **LogicManager.sol**
  `cdfc6120153db8e95f362cd6a73ae05a714c7e1fcce8f7d1d815694735db795f`

- **AccountLogic.sol**
  `38e3f140ac80177442886dfa9f7c6e2808236e98582ea69ac1b2cfa4b0ea3468`
- **DappLogic.sol**
  `28a3581bb9fe59f5a8636b3a7a1500bdeaabead18dc734b14157a431c83c1fb1`
- **DualsigsLogic.sol**
  `65b3a1b70eae76a5df29a20e9842308e5d529c17d0b2cf56abefda7ab2b6e6fd`
- **TransferLogic.sol**
  `6515eb85a68af6e14f740a4fdd858e4cb670e39d097bccb2d3edd3aaff4de62d`
- **AccountBaseLogic.sol**
  `b25c1a8833ead00b1c75feb4115f9e191065eb99ee92b1dd578c4d086d6319cd`
- **BaseLogic.sol**
  `333a0800a7746d72731a4cea07d1e1bf37552611d3bc3c57bdeb8cb9e6197a40`
- **MyNft.sol**
  `b41eb4f8d4f96722562e31d68c15e5e224c771342680379954f51ce4fbbb8b4d`
- **MyToken.sol**
  `ad67e648646af505fc51152dd2d1cf81e4f5bf139a5b55cd1104e3cbfa5042a2`
- **MultiOwned.sol**
  `51d174dc864e45d2fefb3551aab784320b34f3dedb2c75be789274df8d827df1`
- **Owned.sol**
  `9c3fe9adaedbbe27940e0f25c27c3d8e5811a3d3ad658e4d058a1840afcef09e`
- **SafeMath.sol**
  `8f5ffacb100244d0da64f334543c3298be1c48a7ce9aadae06516c5e01f47714`

## MyKey Architect & Workflow Overview



System Overview:

1. For each MyKey account will provide an corresponding `Account Proxy` contract address (Not an externally owned account)

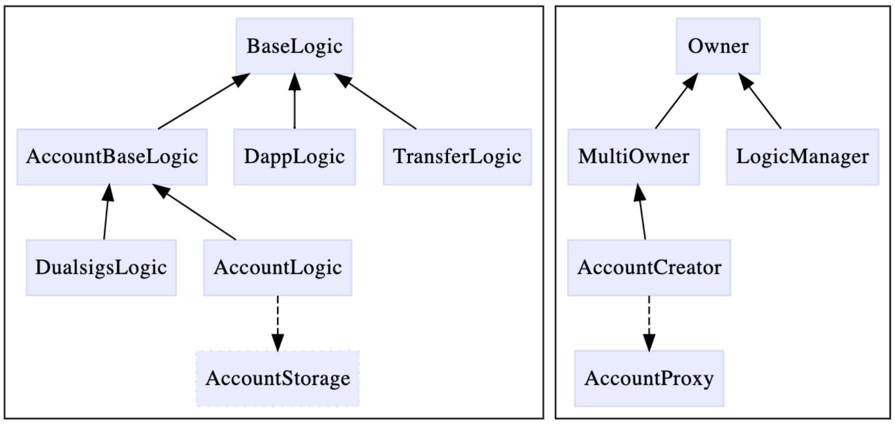
2. While creating a new MyKey account, `MyKey Lab` will set as one of the backup keys as default setting, users can add more backup keys later.

3. All MyKey user related data will storage in contract `AccountStorage`, for instance account admin key, 6(max) backup operation keys, delayItem and multi-sign Proposal Items

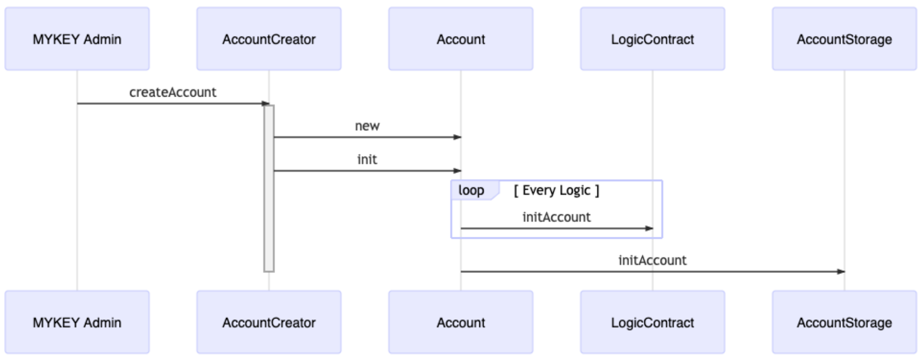
4. Logic Modules, including all the contract logic such as transfer, multi-signing proposal, dapp, and account related logic

5. LogicManager, as named handling all the logic contracts upgradeability, allow contracts to be upgraded due to its business expansion, and vulnerability fixes etc...

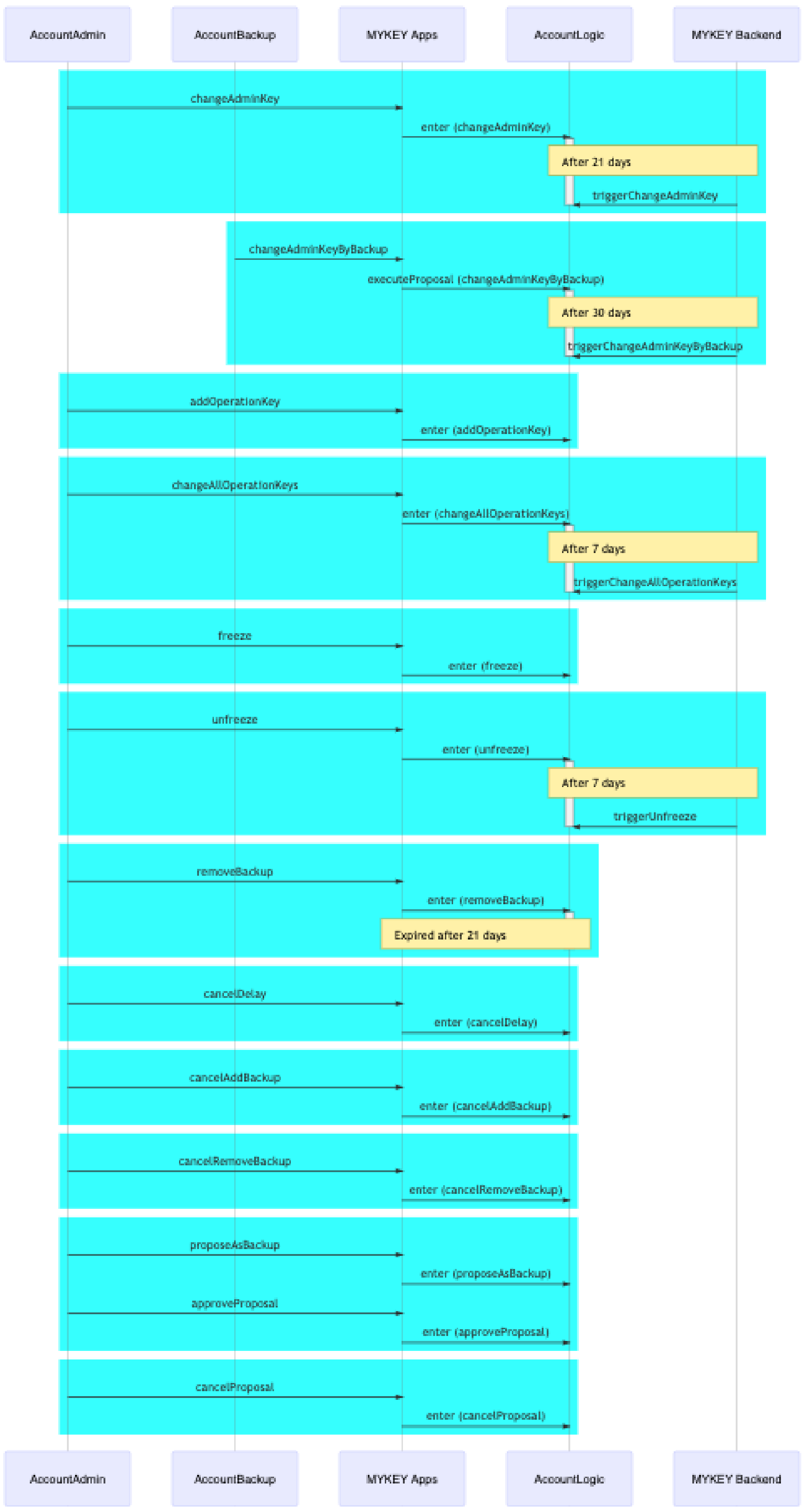
MyKey team provide the smart contract wallet design architecture diagram, each module workflow process can be illustrate as following:



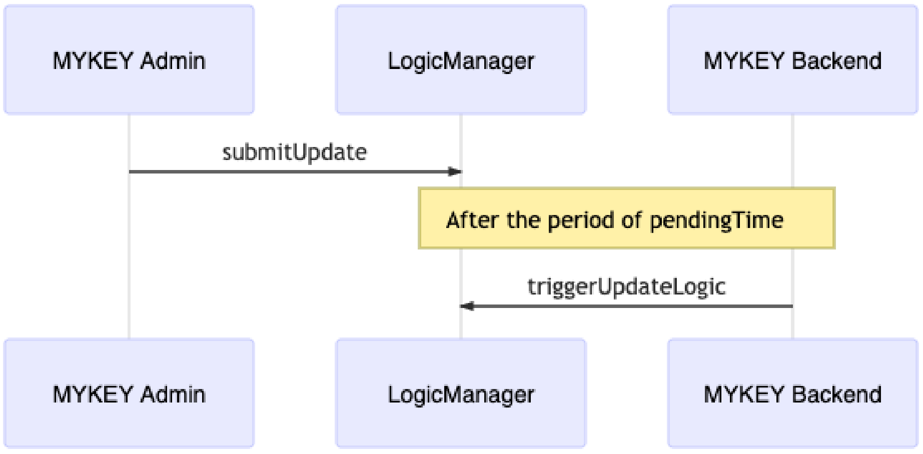
Account Creation Workflow



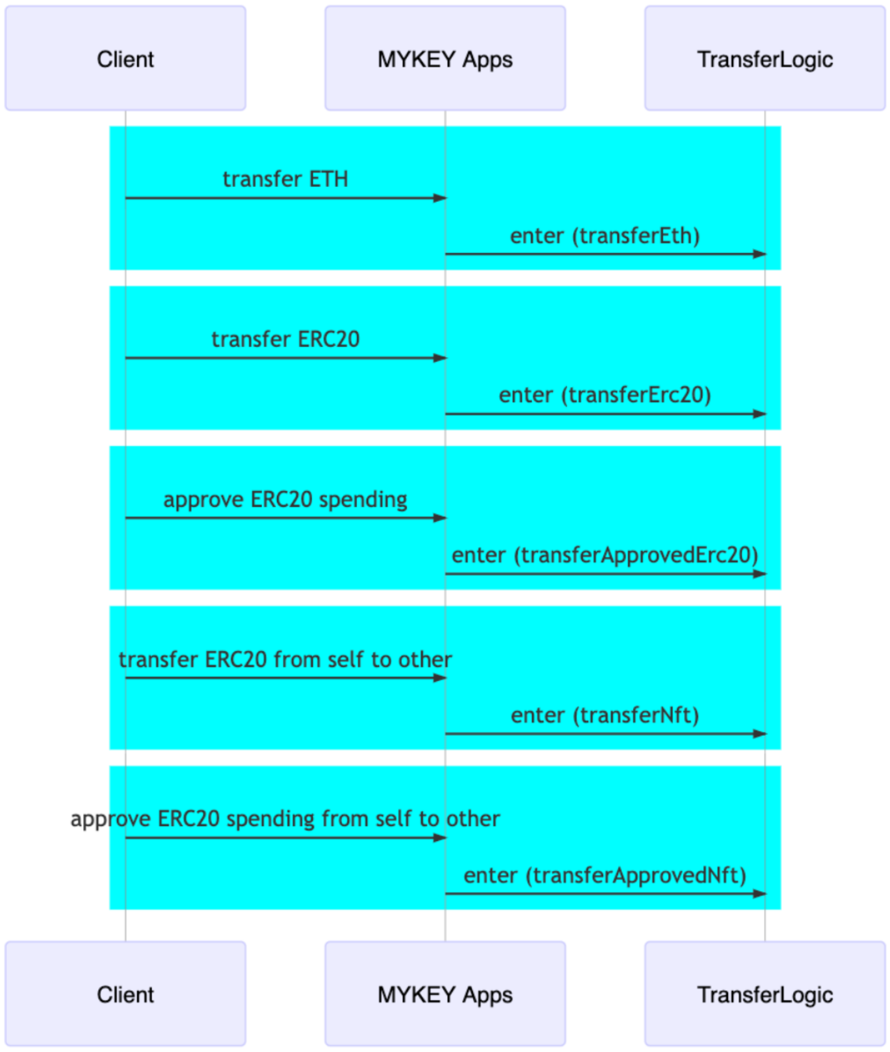
Account Logic Workflow



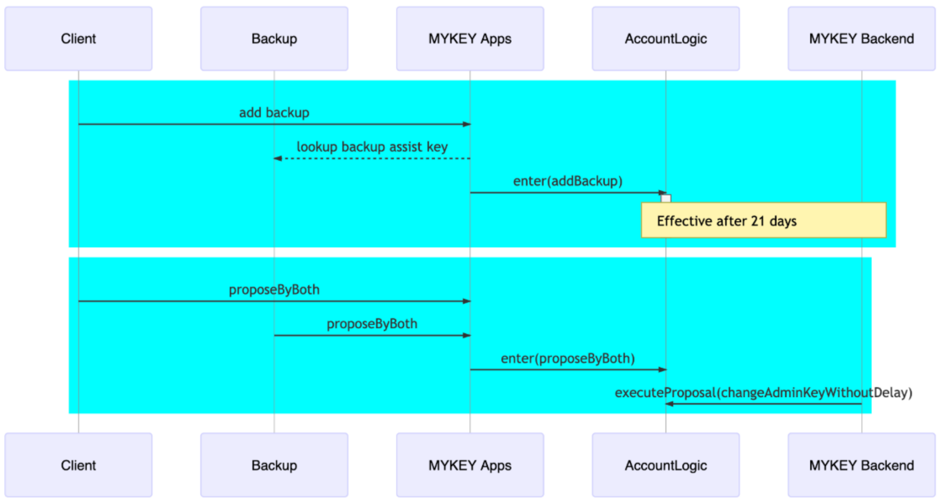
Account Logic Update Workflow



Account Logic Transfer Workflow



Account Logic Dualsig Workflow



## Review Comments

### BasicLogic.sol

- INFO Consider using `enum` for `ENVIROMENT` type for better readability.
  - ✓ MyKey The `ENVIROMENT` type is removed on mainnet release. Its original purpose was for development simulation benefits and testings.
- MINOR `getSignHash()` Recommend declaring the `prefix` variable as a constant for gas optimization.
  - ✓ MyKey The code is updated and reflected in the latest commit
- MINOR `verifySig()` Recommend checking the `_signature` length is 65 `require( _signature.length == 65, ‘’invalid _signature length’’)`
  - ✓ MyKey The code is updated and reflected in the latest commit
- MINOR `verifySig()` The `signatureSplit()` mentioned the `bytes is not working due to the Solidity parser` would you mind to share any references or case failure examples?
  - ✓ MyKey The `signatureSplit()` is removed and updated to `recover()` and reflected in the latest commit.
- MINOR `checkAndUpdateNonce()` Consider using SafeMath library for adding `now` + 86400 to prevent the issue cause by integer underflow or overflow

### AccountCreator.sol

- INFO `constructor()` Recommend to check the variables `_mgr`, `_storage`, `_accountImpl` are not an zero address for minimizing the human errors.

- $\boxed{\text{MINOR}}$ Given `close()` will invoke `selfdestruct`, a very low-level opcode call, highly recommend to emit an event for future reference as a best practice.
  - ✓ $\boxed{\text{MyKey}}$ The code is updated and reflected in the latest commit.

### AccountLogic.sol

- $\boxed{\text{INFO}}$ Recommend to remove the declaration of `actionId` variable, instead use the constant variable directly.
  1. `changeAllOperationKeys`
  2. `triggerChangeAdminKeyByBackup`
  3. `changeAllOperationKeys`
  4. `triggerChangeAllOperationKeys`
  5. ✓ $\boxed{\text{MyKey}}$ The code is updated and reflected in the latest commit.

- $\boxed{\text{MINOR}}$ Recommend declaring the local memory variable outside the for loop for gas optimization.
  1. `changeAllOperationKeys`
  2. `triggerChangeAdminKeyByBackup`
  3. `changeAllOperationKeys`
  4. `triggerChangeAllOperationKeys`
  5. ✓ $\boxed{\text{MyKey}}$ The code is updated and reflected in the latest commit.

```
address r
for (uint i = 0; i < keys.length; i++){
    r = keys[i] // reuse the variable r instead of creating a new reference every-time
    ....
}
```

- $\boxed{\text{MINOR}}$ Recommend emitting event logs for states changing functions. First, it is a good practice using logging for the purpose of history tracing and user behaviors analysis. Second, as the functions declare as `external`, that refer as any users can triggered directly from outside the contract, not necessary go thru by `enter()`.
  - `addOperationKey`
  - `changeAllOperationKeys`
  - `freeze`
  - `unfreeze`
  - `removeBackup`
  - `cancelDelay`
  - `cancelAddBackup`
  - `cancelRemoveBackup`

  - approveProposal
  - ✓ MyKey The code is updated and reflected in the latest commit.

- INFO `findBackup` Recommend checking the given `_account` is not an zero address.
  - ✓ MyKey The code is updated and reflected in the latest commit.

## AccountStorage.sol

- INFO `setKeyStatus()`: Recommend adding `require()` to ensure `_status` is `0` or `1`.
- INFO `setBackup()`: Recommend adding `require()` to ensure following
  - `_backup` is a non zero address
  - `_effective` should be greater than `now`
  - `_expiry` is later than `now`
  - `_effective` is not later than `_expiry`
- INFO `setBackupExpiryDate()`: Recommend adding `require()` to ensure `_expiry` is later than `now`
- INFO `setDelayData()`: Recommend adding `require()` to ensure
  - `_hash` is a non zero address
  - `_dueTime` is later than `now`

## AccountProxy.sol

- INFO Recommend defining the visibility level for variable `implemetation` implicitly regarding to the best practice guide

## DualsigsLogic.sol

- INFO Recommend changing `isActionWithDualSigs()` from a function to a modifier.
  - ✓ MyKey The `isActionWithDualSigs` is renamed to `allowDualSigsActionOnly` with modifier decorator
- INFO Recommend changing `isFastAction()` from a function to a modifier.
- MINOR `addBackup()` Consider using SafeMath library for adding `now` + getDelayTime to prevent the issue cause by integer underflow or overflow
  - ✓ MyKey The `getDelayTime()` is removed, only (7, 14, 21) days are valid delayed time on mainnet.

### Owned.sol

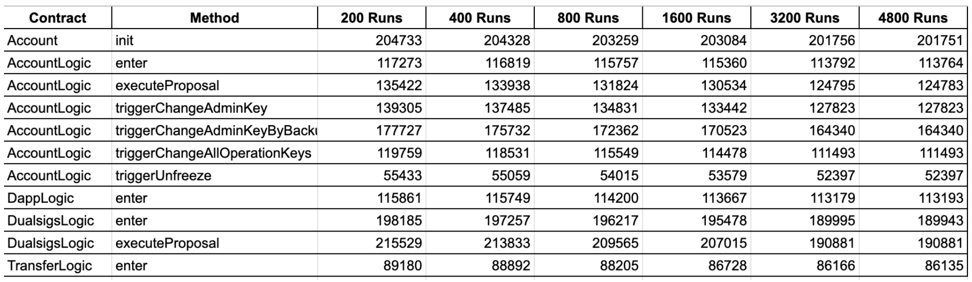
- $\boxed{\text{INFO}}$ Given `constructor()` not taking any input parameter, consider keeping the function as `internal`.
- $\boxed{\text{INFO}}$ Recommend to record the previous owner address in the event `OwnerChanged` for better tracing context. - i.e: `event OwnerChanged(address indexed previousOwner, address indexed _newOwner);`
  - ✓ $\boxed{\text{MyKey}}$ The code is updated and reflected in the latest commit.
- $\boxed{\text{INFO}}$ Highly recommend using pull-over-push pattern for ownership transfer, openzepplin's `Ownable` contract, which is a good reference for consideration.

### LogicManager.sol

- $\boxed{\text{INFO}}$ Recommend changing `if (authorized[_logic] != _value)` in `updateLogic()` to be `require(authorized[_logic] != p.value)` in `triggerUpdateLogic()` before calling `updateLogic()`.
- $\boxed{\text{INFO}}$ Recommend `submitUpdate` using SafeMath for `now` + pendingTime for preventing the arithmetic vulnerability

### Gas Consumption

The gas consumption is based on localhost environment with optimizer mode and runs with 200, 400, 800, 1600, 3200, and 4000 times



| Contract | Method | 200 Runs | 400 Runs | 800 Runs | 1600 Runs | 3200 Runs | 4800 Runs |
|---|---|---|---|---|---|---|---|
| Account | init | 204733 | 204328 | 203259 | 203084 | 201756 | 201751 |
| AccountLogic | enter | 117273 | 116819 | 115757 | 115360 | 113792 | 113764 |
| AccountLogic | executeProposal | 135422 | 133938 | 131824 | 130534 | 124795 | 124783 |
| AccountLogic | triggerChangeAdminKey | 139305 | 137485 | 134831 | 133442 | 127823 | 127823 |
| AccountLogic | triggerChangeAdminKeyByBacku | 177727 | 175732 | 172362 | 170523 | 164340 | 164340 |
| AccountLogic | triggerChangeAllOperationKeys | 119759 | 118531 | 115549 | 114478 | 111493 | 111493 |
| AccountLogic | triggerUnfreeze | 55433 | 55059 | 54015 | 53579 | 52397 | 52397 |
| DappLogic | enter | 115861 | 115749 | 114200 | 113667 | 113179 | 113193 |
| DualsigsLogic | enter | 198185 | 197257 | 196217 | 195478 | 189995 | 189943 |
| DualsigsLogic | executeProposal | 215529 | 213833 | 209565 | 207015 | 190881 | 190881 |
| TransferLogic | enter | 89180 | 88892 | 88205 | 86728 | 86166 | 86135 |

## Best practice

Smart contract development requires a particular engineering mindset. A failure in the initial construction can be catastrophic, and changing the project after the fact can be exceedingly difficult.
To ensure success and to avoid the challenges above smart contracts should here to best practices at their conception. Below, we summarized a checklist of key points & vulnerability vectors that help to indicate a high overall quality of the current MyKey project. (✓ indicates satisfaction; × indicates unsatisfaction; – indicates inapplicable)

### General

Overall, smart contract coding practice baseline such as environment setting, compiler version, testing, logging, and code layout.
Compiling

- ✓ Correct environment settings, e.g. compiler version, test framework
- ✓ No compiler warnings

Logging

- ✓ Provide error message along with `assert` & `require`
- ✓ Use events to monitor contract activities

Code Layout

- ✓ According to Solidity Tutorial, Layout contract elements should following below order:
    1. Pragma statements
    2. Import statements
    3. Interfaces
    4. Libraries
    5. Contracts
- × Each contract, library or interface should following below order:
    1. Type declarations
    2. State variables
    3. Events
    4. Functions
- × According to Solidity Tutorial, functions should be grouped according to their visibility and ordered:
    1. constructor
    2. fallback function (if exists)
    3. external
    4. public
    5. internal
    6. private

### Arithmetic Vulnerability

EVM specifies fixed-size data types for integers, in which means that has only a certain range of numbers it can store or represent.
Two's Complement / Integer underflow / overflow

- ✓ Use Math library as SafeMath for all arithmetic operations to handle integer overflow and underflow

Floating Points and Precision

- – Correct handling the right precision when dealing ratios and rates

### Access & Privilege Control Vulnerability

Authorization of end-user and administrator and his/her assessment rights
Circuit Breaker

- ✓ Provide pause functionality for control and emergency handling

Restriction

- ✓ Provide proper access control for functions
- ✓ Establish rate limiter for certain operations
- ✓ Restrict access to sensitive functions
- ✓ Restrict permission to contract destruction
- ✓ Establish speed bumps slow down some sensitive actions, any malicious actions occur, there is time to recover.

### DoS Vulnerability

A type of attacks that make the contract inoperable with certain period of time or permanently.
Unexpected Revert

- ✓ Use favor pull over push pattern for handling unexpected revert

Block Gas Limit

- – Use favor pull over push pattern for handling gas spent exceeds its limit on Contract via unbounded operations
- ✓ Use favor pull over push pattern for handling gas spent exceeds its limit on the network via block stuffing

### Miner Manipulation Vulnerability

BlockNumber Dependence

– Understand the security risk level and trade-off of using `block.number` as one of core factors in the contract. Be aware that `block.number` can not be manipulated by the miner, but can lead to large than expected time differences. With assumptions of an Ethereum block confirmation takes 13 seconds. However, the average block time is between 13  15 seconds. During the difficulty bomb stage or hard/soft fork upgrade of the network, `block.number` to a time is dangerous and inaccurate as expected.

Timestamp Dependence

✓ Understand the security risk level and trade-off of using `block.timestamp` or alias `now` as one of core factors in the contract.

✓ Correct use of 15-second rule to minimize the impact caused by timestamp variance

Transaction Ordering Or Front-Running

– Understand the security risk level and the `gasPrice` rule in this vulnerability

– Correct placing an upper bound on the `gasPrice` for preventing the users taking the benefit of transaction ordering

### External Referencing Vulnerability

External calls may execute malicious code in that contract or any other contract that it depends upon. As such, every external call should be treated as a potential security risk

✓ Correct using the pull over push favor for external calls to reduce reduces the chance of problems with the gas limit.

Avoid state changes after external calls

✓ Correct using checks-effects-interactions pattern to minimize the state changes after external contract or call referencing.

Handle errors in external calls

✓ Correct handling errors in any external contract or call referencing by checking its return value

### Race Conditions Vulnerability

A type of vulnerability caused by calling external contracts that attacker can take over the control flow, and make changes to the data that the calling function wasn't expecting.

- Type of race conditions:
  - Reentrancy
    A state variable is changed after a contract uses `call.value()()`.
  - Cross-function Race Conditions
    An attacker may also be able to do a similar attack using two different functions that share the same state

✓ Avoid using `call.value()()`, instead use `send()`, `transfer()` that consumes 2300 gas. This will prevent any external code from being executed continuously

✓ Finish all internal work before calling the external function for unavoidable external call.

### Low-level Call Vulnerability

The low-level function or opcodes are very useful and danger as for allowing the Libraries implementation and modularized code. However it opens up the doors to vulnerabilities as essentially your contract is allowing anyone to do whatever they want with their state Code Injection by delegatecall

✓ Ensure the libraries implementation is stateless and non-self-destructable

### Visibility Vulnerability

Solidity functions have 4 difference visibility dictate how functions are allowed to be called. The visibility determines whether a function can be called externally by users, by other derived contracts, only internally or only externally.

✓ Specify the visibility of all functions in a contract, even if they are intentionally public

### Incorrect Interface Vulnerability

A contract interface defines functions with a different type signature than the implementation, causing two different method id's to be created. As a result, when the interface is called, the fallback method will be executed.

✓ Ensure the defined function signatures are match with the contract interface and implementation

### Bad Randomness Vulnerability

Pseudo random number generation is not supported by Solidity as default, which it is an unsafe operation.

✓ Avoid using randomness for block variables, there may be a chance manipulated by the miners

**Documentation**

✓ Provide project README and execution guidance

✓ Provide inline comment for complex functions intention

✓ Provide instruction to initialize and execute the test files

**Testing**

✓ Provide migration scripts for continuously contracts deployment to the Ethereum network

✓ Provide test scripts and coverage for potential scenarios

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

## Static Analysis Results

### INSECURE_COMPILER_VERSION

Line 1 in File DualsigsLogic.sol

```
pragma solidity ^0.5.4;
```

Only these compiler versions are safe to compile your code: 0.5.10

### TIMESTAMP_DEPENDENCY

Line 131 in File DualsigsLogic.sol

```
    accountStorage.setBackup(_account, index, _backup, now + DELAY_CHANGE_BACKUP),
        uint256(-1));
```

"now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 145 in File DualsigsLogic.sol

```
      if ((backup == _backup) && (expiryDate > now)) {
```

"now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 150 in File DualsigsLogic.sol

```
        if ((backup == address(0)) || (expiryDate <= now)) {
```

"now" can be influenced by minors to some degree

### INSECURE_COMPILER_VERSION

Line 1 in File AccountLogic.sol

```
pragma solidity ^0.5.4;
```

Only these compiler versions are safe to compile your code: 0.5.10

### TIMESTAMP_DEPENDENCY

Line 75 in File AccountLogic.sol

```
    accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY, hash, now +
        DELAY_CHANGE_ADMIN_KEY);
```

"now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 85 in File AccountLogic.sol

```
    require(due <= now, "too early to trigger changeAdminKey");
```

"now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 103 in File AccountLogic.sol

```
103    accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP, hash, now +
           DELAY_CHANGE_ADMIN_KEY_BY_BACKUP);
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 113 in File AccountLogic.sol

```
113    require(due <= now, "too early to trigger changeAdminKeyByBackup");
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 150 in File AccountLogic.sol

```
150    accountStorage.setDelayData(_account, CHANGE_ALL_OPERATION_KEYS, hash, now +
           DELAY_CHANGE_OPERATION_KEY);
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 160 in File AccountLogic.sol

```
160    require(due <= now, "too early to trigger changeAllOperationKeys");
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 186 in File AccountLogic.sol

```
186    accountStorage.setDelayData(_account, UNFREEZE, hash, now + DELAY_UNFREEZE_KEY);
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 196 in File AccountLogic.sol

```
196    require(due <= now, "too early to trigger unfreeze");
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 214 in File AccountLogic.sol

```
214    accountStorage.setBackupExpiryDate(_account, index, now + DELAY_CHANGE_BACKUP);
```

❗ ”now” can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 247 in File AccountLogic.sol

```
    require(effectiveDate > now, "already effective");
```

! "now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 256 in File AccountLogic.sol

```
    require(expiryDate > now, "already expired");
```

! "now" can be influenced by minors to some degree

### INSECURE_COMPILER_VERSION

Line 1 in File DappLogic.sol

```
pragma solidity ^0.5.4;
```

i Only these compiler versions are safe to compile your code: 0.5.10

### INSECURE_COMPILER_VERSION

Line 1 in File AccountBaseLogic.sol

```
pragma solidity ^0.5.4;
```

i Only these compiler versions are safe to compile your code: 0.5.10

### TIMESTAMP_DEPENDENCY

Line 146 in File AccountBaseLogic.sol

```
        return (_effectiveDate <= now) && (_expiryDate > now);
```

! "now" can be influenced by minors to some degree

### TIMESTAMP_DEPENDENCY

Line 146 in File AccountBaseLogic.sol

```
        return (_effectiveDate <= now) && (_expiryDate > now);
```

! "now" can be influenced by minors to some degree

### INSECURE_COMPILER_VERSION

Line 1 in File BaseLogic.sol

```
pragma solidity ^0.5.4;
```

i Only these compiler versions are safe to compile your code: 0.5.10

### TIMESTAMP_DEPENDENCY

Line 156 in File BaseLogic.sol

```
        require(SafeMath.div(_nonce, 1000000) <= now + 86400, "nonce too big"); //
            86400=24*3600 seconds
```

! "now" can be influenced by minors to some degree

## INSECURE_COMPILER_VERSION

Line 1 in File MyToken.sol

```
pragma solidity ^0.5.0;
```

Only these compiler versions are safe to compile your code: 0.5.10

## Source Code

File logics/AccountLogic.sol

```
pragma solidity ^0.5.4;

import "./base/AccountBaseLogic.sol";

/**
* @title AccountLogic
*/
contract AccountLogic is AccountBaseLogic {

  // Equals to bytes4(keccak256("changeAllOperationKeys(address,address[])"))
  bytes4 private constant CHANGE_ALL_OPERATION_KEYS = 0xd3b9d4d6;
  // Equals to bytes4(keccak256("unfreeze(address)"))
  bytes4 private constant UNFREEZE = 0x45c8b1a6;
  // Equals to bytes4(keccak256("addOperationKey(address,address)"))
  bytes4 private constant ADD_OPERATION_KEY = 0x9a7f6101;
  // Equals to bytes4(keccak256("proposeAsBackup(address,address,bytes)"))
  bytes4 private constant PROPOSE_AS_BACKUP = 0xd470470f;
  // Equals to bytes4(keccak256("approveProposal(address,address,address,bytes)"))
  bytes4 private constant APPROVE_PROPOSAL = 0x3713f742;

    event AccountLogicEntered(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)
    AccountBaseLogic(_accountStorage)
    public
  {
  }

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

  function initAccount(Account _account) external allowAccountCallsOnly(_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) != CHANGE_ADMIN_KEY_BY_BACKUP, "invalid data");
    address account = getSignerAddress(_data);
    uint256 keyIndex = getKeyIndex(_data);
    checkAndUpdateNonce(account, _nonce, keyIndex);
```

```
      address signingKey = accountStorage.getKeyData(account, keyIndex);
      bytes32 signHash = getSignHash(_data, _nonce);
      verifySig(signingKey, _signature, signHash);

      // solium-disable-next-line security/no-low-level-calls
      (bool success,) = address(this).call(_data);
      require(success, "calling self failed");
      emit AccountLogicEntered(_data, _nonce);
    }

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

      // called from 'enter'
    function changeAdminKey(address payable _account, address _pkNew) external allowSelfCallsOnly {
      require(_pkNew != address(0), "0x0 is invalid");
      address pk = accountStorage.getKeyData(_account, 0);
      require(pk != _pkNew, "identical admin key exists");
      require(accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY) == 0, "delay data already exists");
      bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew));
      accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY, hash, now + DELAY_CHANGE_ADMIN_KEY);
    }

      // called from external
    function triggerChangeAdminKey(address payable _account, address _pkNew) external {
      bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew));
      require(hash == accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY), "delay hash unmatch");

      uint256 due = accountStorage.getDelayDataDueTime(_account, CHANGE_ADMIN_KEY);
      require(due > 0, "delay data not found");
      require(due <= now, "too early to trigger changeAdminKey");
      accountStorage.setKeyData(_account, 0, _pkNew);
      //clear any existing related delay data and proposal
      accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
      accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
      clearRelatedProposalAfterAdminKeyChanged(_account);
      emit ChangeAdminKeyTriggered(_account, _pkNew);
    }

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

      // called from 'executeProposal'
    function changeAdminKeyByBackup(address payable _account, address _pkNew) external allowSelfCallsOnly {
      require(_pkNew != address(0), "0x0 is invalid");
      address pk = accountStorage.getKeyData(_account, 0);
      require(pk != _pkNew, "identical admin key exists");
      require(accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY_BY_BACKUP) == 0, "delay data already exists");
      bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account, _pkNew));
      accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP, hash, now + DELAY_CHANGE_ADMIN_KEY_BY_BACKUP);
    }

    // called from external
  function triggerChangeAdminKeyByBackup(address payable _account, address _pkNew)
      external {
    bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account,
        _pkNew));
    require(hash == accountStorage.getDelayDataHash(_account,
        CHANGE_ADMIN_KEY_BY_BACKUP), "delay hash unmatch");

    uint256 due = accountStorage.getDelayDataDueTime(_account,
        CHANGE_ADMIN_KEY_BY_BACKUP);
    require(due > 0, "delay data not found");
    require(due <= now, "too early to trigger changeAdminKeyByBackup");
    accountStorage.setKeyData(_account, 0, _pkNew);
    //clear any existing related delay data and proposal
    accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
    accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
    clearRelatedProposalAfterAdminKeyChanged(_account);
    emit ChangeAdminKeyByBackupTriggered(_account, _pkNew);
  }

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

    // called from 'enter'
  function addOperationKey(address payable _account, address _pkNew) external
      allowSelfCallsOnly {
    uint256 index = accountStorage.getOperationKeyCount(_account) + 1;
    require(index > 0, "invalid operation key index");
    // set a limit to prevent unnecessary trouble
    require(index < 20, "index exceeds limit");
    require(_pkNew != address(0), "0x0 is invalid");
    address pk = accountStorage.getKeyData(_account, index);
    require(pk == address(0), "operation key already exists");
    accountStorage.setKeyData(_account, index, _pkNew);
    accountStorage.increaseKeyCount(_account);
  }

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

    // called from 'enter'
  function changeAllOperationKeys(address payable _account, address[] calldata _pks)
      external allowSelfCallsOnly {
    uint256 keyCount = accountStorage.getOperationKeyCount(_account);
    require(_pks.length == keyCount, "invalid number of keys");
    require(accountStorage.getDelayDataHash(_account, CHANGE_ALL_OPERATION_KEYS) ==
        0, "delay data already exists");
    address pk;
    for (uint256 i = 0; i < keyCount; i++) {
      pk = _pks[i];
      require(pk != address(0), "0x0 is invalid");
    }
    bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account,
        _pks));
    accountStorage.setDelayData(_account, CHANGE_ALL_OPERATION_KEYS, hash, now +
        DELAY_CHANGE_OPERATION_KEY);
  }

    // called from external
    function triggerChangeAllOperationKeys(address payable _account, address[]
        calldata _pks) external {
      bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account,
          _pks));
      require(hash == accountStorage.getDelayDataHash(_account,
          CHANGE_ALL_OPERATION_KEYS), "delay hash unmatch");

      uint256 due = accountStorage.getDelayDataDueTime(_account,
          CHANGE_ALL_OPERATION_KEYS);
      require(due > 0, "delay data not found");
      require(due <= now, "too early to trigger changeAllOperationKeys");
      address pk;
      for (uint256 i = 0; i < accountStorage.getOperationKeyCount(_account); i++) {
        pk = _pks[i];
        accountStorage.setKeyData(_account, i+1, pk);
        accountStorage.setKeyStatus(_account, i+1, 0);
      }
      accountStorage.clearDelayData(_account, CHANGE_ALL_OPERATION_KEYS);
      emit ChangeAllOperationKeysTriggered(_account, _pks);
    }

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

      // called from 'enter'
    function freeze(address payable _account) external allowSelfCallsOnly {
      for (uint256 i = 1; i <= accountStorage.getOperationKeyCount(_account); i++) {
        if (accountStorage.getKeyStatus(_account, i) == 0) {
          accountStorage.setKeyStatus(_account, i, 1);
        }
      }
    }

      // called from 'enter'
    function unfreeze(address payable _account) external allowSelfCallsOnly {
      require(accountStorage.getDelayDataHash(_account, UNFREEZE) == 0, "delay data
          already exists");
      bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account));
      accountStorage.setDelayData(_account, UNFREEZE, hash, now + DELAY_UNFREEZE_KEY);
    }

      // called from external
    function triggerUnfreeze(address payable _account) external {
      bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account));
      require(hash == accountStorage.getDelayDataHash(_account, UNFREEZE), "delay hash
           unmatch");

      uint256 due = accountStorage.getDelayDataDueTime(_account, UNFREEZE);
      require(due > 0, "delay data not found");
      require(due <= now, "too early to trigger unfreeze");

      for (uint256 i = 1; i <= accountStorage.getOperationKeyCount(_account); i++) {
        if (accountStorage.getKeyStatus(_account, i) == 1) {
          accountStorage.setKeyStatus(_account, i, 0);
        }
      }
      accountStorage.clearDelayData(_account, UNFREEZE);
      emit UnfreezeTriggered(_account);
    }

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

    // called from 'enter'
  function removeBackup(address payable _account, address _backup) external
      allowSelfCallsOnly {
    uint256 index = findBackup(_account, _backup);
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");

    accountStorage.setBackupExpiryDate(_account, index, now + DELAY_CHANGE_BACKUP);
  }

    // 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;
    }
    address b;
    for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
      b = accountStorage.getBackupAddress(_account, i);
      if (b == _backup) {
        index = i;
        break;
      }
    }
    return index;
  }

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

    // called from 'enter'
  function cancelDelay(address payable _account, bytes4 _actionId) external
      allowSelfCallsOnly {
    accountStorage.clearDelayData(_account, _actionId);
  }

    // called from 'enter'
  function cancelAddBackup(address payable _account, address _backup) external
      allowSelfCallsOnly {
    uint256 index = findBackup(_account, _backup);
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
    uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_account, index);
    require(effectiveDate > now, "already effective");
    accountStorage.clearBackupData(_account, index);
  }

    // called from 'enter'
  function cancelRemoveBackup(address payable _account, address _backup) external
      allowSelfCallsOnly {
    uint256 index = findBackup(_account, _backup);
    require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
    uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, index);
    require(expiryDate > now, "already expired");
    accountStorage.setBackupExpiryDate(_account, index, uint256(-1));
  }

```

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

    // called from 'enter'
  // proposer is backup in the case of 'proposeAsBackup'
  function proposeAsBackup(address _backup, address payable _client, bytes calldata
      _functionData) external allowSelfCallsOnly {
    bytes4 proposedActionId = getMethodId(_functionData);
    require(proposedActionId == CHANGE_ADMIN_KEY_BY_BACKUP, "invalid proposal by
        backup");
    checkRelation(_client, _backup);
    bytes32 functionHash = keccak256(_functionData);
    accountStorage.setProposalData(_client, _backup, proposedActionId, functionHash,
         _backup);
  }

    // called from 'enter'
  function approveProposal(address _backup, address payable _client, address
      _proposer, bytes calldata _functionData) external allowSelfCallsOnly {
    bytes32 functionHash = keccak256(_functionData);
    require(functionHash != 0, "invalid hash");
    checkRelation(_client, _backup);
    bytes4 proposedActionId = getMethodId(_functionData);
    bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer,
        proposedActionId);
    require(hash == functionHash, "proposal unmatch");
    accountStorage.setProposalData(_client, _proposer, proposedActionId,
        functionHash, _backup);
  }

    // called from 'enter'
  function cancelProposal(address payable _client, address _proposer, bytes4
      _proposedActionId) external allowSelfCallsOnly {
    require(_client != _proposer, "cannot cancel dual signed proposal");
    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; //index default value is 0, admin key
    bytes4 methodId = getMethodId(_data);
    if (methodId == ADD_OPERATION_KEY) {
        index = 2; //adding key
    } else if (methodId == PROPOSE_AS_BACKUP || methodId == APPROVE_PROPOSAL) {
        index = 4; //assist key
    }
    return index;
  }

}
```

File logics/DappLogic.sol

```
pragma solidity ^0.5.4;

import "./base/BaseLogic.sol";

contract DappLogic is BaseLogic {

    /*
    index 0: admin key
          1: asset(transfer)
          2: adding
          3: reserved(dapp)
          4: assist
     */
    uint constant internal DAPP_KEY_INDEX = 3;

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

    event DappLogicInitialised(address indexed account);
    event DappLogicEntered(bytes data, uint256 indexed nonce);

    // *************** Constructor ********************** //
    constructor(AccountStorage _accountStorage)
        BaseLogic(_accountStorage)
        public
    {
    }

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

    function initAccount(Account _account) external allowAccountCallsOnly(_account){
        emit DappLogicInitialised(address(_account));
    }

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

    function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
         external {
        address account = getSignerAddress(_data);
        checkAndUpdateNonce(account, _nonce, DAPP_KEY_INDEX);

        address dappKey = accountStorage.getKeyData(account, DAPP_KEY_INDEX);
        bytes32 signHash = getSignHash(_data, _nonce);
        verifySig(dappKey, _signature, signHash);

        // solium-disable-next-line security/no-low-level-calls
        (bool success,) = address(this).call(_data);
        require(success, "calling self failed");
        emit DappLogicEntered(_data, _nonce);
    }

    // *************** 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 allowSelfCallsOnly {
        Account(_account).invoke(_target, _value, _methodData);
    }

}
```

File logics/DualsigsLogic.sol

```
pragma solidity ^0.5.4;

import "./base/AccountBaseLogic.sol";

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

  // Equals to bytes4(keccak256("changeAllOperationKeysWithoutDelay(address,address
      []")"))
  bytes4 private constant CHANGE_ALL_OPERATION_KEYS_WITHOUT_DELAY = 0x02064abc;
  // Equals to bytes4(keccak256("unfreezeWithoutDelay(address)"))
  bytes4 private constant UNFREEZE_WITHOUT_DELAY = 0x69521650;
  // Equals to bytes4(keccak256("addBackup(address,address)"))
  bytes4 private constant ADD_BACKUP = 0x426b7407;
  // Equals to bytes4(keccak256("proposeByBoth(address,address,bytes)"))
  bytes4 private constant PROPOSE_BY_BOTH = 0x7548cb94;

    event DualsigsLogicInitialised(address indexed account);
    event DualsigsLogicEntered(bytes data, uint256 indexed clientNonce, uint256
        backupNonce);

  modifier allowDualSigsActionOnly(bytes memory _data) {
    bytes4 methodId = getMethodId(_data);
    require ((methodId == ADD_BACKUP) ||
          (methodId == PROPOSE_BY_BOTH), "wrong entry");
    _;
  }

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

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

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

    function initAccount(Account _account) external allowAccountCallsOnly(_account){
        emit DualsigsLogicInitialised(address(_account));
    }

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

    /* DualsigsLogic has 2 actions called from 'enter':
        addBackup, proposeByBoth
  */
  function enter(
    bytes calldata _data, bytes calldata _clientSig, bytes calldata _backupSig,
        uint256 _clientNonce, uint256 _backupNonce
  )
```

```
    external allowDualSigsActionOnly(_data)
  {
        verifyClient(_data, _clientSig, _clientNonce);
        verifyBackup(_data, _backupSig, _backupNonce);

    // solium-disable-next-line security/no-low-level-calls
    (bool success,) = address(this).call(_data);
    require(success, "enterWithDualSigs failed");
    emit DualsigsLogicEntered(_data, _clientNonce, _backupNonce);
  }

  function verifyClient(bytes memory _data, bytes memory _clientSig, uint256
      _clientNonce) internal {
    address client = getSignerAddress(_data);
    //client sign with admin key
    uint256 clientKeyIndex = 0;
    if ((getMethodId(_data) == PROPOSE_BY_BOTH) &&
        (getProposedMethodId(_data) == CHANGE_ADMIN_KEY_WITHOUT_DELAY)) {
      // if proposed action is 'changeAdminKeyWithoutDelay', do not check
          _clientNonce
      verifySig(accountStorage.getKeyData(client, clientKeyIndex), _clientSig,
          getSignHashWithoutNonce(_data));
    } else {
      checkAndUpdateNonce(client, _clientNonce, clientKeyIndex);
      verifySig(accountStorage.getKeyData(client, clientKeyIndex), _clientSig,
          getSignHash(_data, _clientNonce));
    }
  }

    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'
  function changeAdminKeyWithoutDelay(address payable _account, address _pkNew)
      external allowSelfCallsOnly {
    address pk = accountStorage.getKeyData(_account, 0);
    require(pk != _pkNew, "identical admin key already exists");
    require(_pkNew != address(0), "0x0 is invalid");
    accountStorage.setKeyData(_account, 0, _pkNew);
    //clear any existing related delay data and proposal
    accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
    accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
    clearRelatedProposalAfterAdminKeyChanged(_account);
  }

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

    // called from 'executeProposal'
  function changeAllOperationKeysWithoutDelay(address payable _account, address[]
        calldata _pks) external allowSelfCallsOnly {
      uint256 keyCount = accountStorage.getOperationKeyCount(_account);
      require(_pks.length == keyCount, "invalid number of keys");
      for (uint256 i = 0; i < keyCount; i++) {
        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'
    function unfreezeWithoutDelay(address payable _account) external
        allowSelfCallsOnly {
      for (uint256 i = 0; i < accountStorage.getOperationKeyCount(_account); i++) {
        if (accountStorage.getKeyStatus(_account, i+1) == 1) {
          accountStorage.setKeyStatus(_account, i+1, 0);
        }
      }
    }

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

      // called from 'enter'
    function addBackup(address payable _account, address _backup) external
        allowSelfCallsOnly {
      require(_account != _backup, "cannot be backup of oneself");
      uint256 index = findAvailableSlot(_account, _backup);
      require(index <= MAX_DEFINED_BACKUP_INDEX, "invalid or duplicate or no vacancy")
          ;
      accountStorage.setBackup(_account, index, _backup, now + DELAY_CHANGE_BACKUP,
          uint256(-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'
  function proposeByBoth(address payable _client, address _backup, bytes calldata _functionData) external allowSelfCallsOnly {
    bytes4 proposedActionId = getMethodId(_functionData);
    require(isFastAction(proposedActionId), "invalid proposal");
    checkRelation(_client, _backup);
    bytes32 functionHash = keccak256(_functionData);
    accountStorage.setProposalData(_client, _client, proposedActionId, functionHash, _backup);
  }

  function isFastAction(bytes4 _actionId) internal pure returns(bool) {
    if ((_actionId == CHANGE_ADMIN_KEY_WITHOUT_DELAY) ||
      (_actionId == CHANGE_ALL_OPERATION_KEYS_WITHOUT_DELAY) ||
      (_actionId == UNFREEZE_WITHOUT_DELAY))
    {
      return true;
    }
    return false;
  }

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

  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)))
    }
  }

    function getProposedMethodId(bytes memory _b) internal pure returns (bytes4 _a) {
    require(_b.length >= 164, "data length too short");
        // solium-disable-next-line security/no-inline-assembly
        assembly {
      /* 'proposeByBoth' data example:
      0x
      7548cb94                                                     // method id
      000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5 // param 0
      00000000000000000000000011390e32ccdfb3f85e92b949c72fe482d77838f3 // param 1
      0000000000000000000000000000000000000000000000000000000000000060 // data length including padding
      0000000000000000000000000000000000000000000000000000000000000044 // true data length
      441d2e50                                                     // method id(proposed method: changeAdminKeyWithoutDelay)
      000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5 // param 0
```

```
        000000000000000000000000013667a2711960c95fae074f90e0f739bc324d1ed // param 1
        0000000000000000000000000000000000000000000000000000000000000000     // padding
        */
            // the first 32 bytes is the length of the bytes array _b
        // 32 + 4 + 32 + 32 + 32 + 32 = 164
            _a := mload(add(_b, 164))
        }
    }

    function getSignHashWithoutNonce(bytes memory _data) internal view returns(
        bytes32) {
        // use EIP 191
        // 0x1900 + this logic address + data
        bytes32 msgHash = keccak256(abi.encodePacked(byte(0x19), byte(0), address(
            this), _data));
        bytes32 prefixedHash = keccak256(abi.encodePacked(SIGN_HASH_PREFIX, msgHash))
            ;
        return prefixedHash;
    }

}
```

File logics/TransferLogic.sol

```
pragma solidity ^0.5.4;

import "./base/BaseLogic.sol";

contract TransferLogic is BaseLogic {

    /*
    index 0: admin key
          1: asset(transfer)
          2: adding
          3: reserved(dapp)
          4: assist
     */
    uint constant internal TRANSFER_KEY_INDEX = 1;

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

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

    event TransferLogicInitialised(address indexed account);
    event TransferLogicEntered(bytes data, uint256 indexed nonce);

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

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

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

    // enable staic call 'onERC721Received' from base account
  function initAccount(Account _account) external allowAccountCallsOnly(_account){
      _account.enableStaticCall(address(this), ERC721_RECEIVED);
      emit TransferLogicInitialised(address(_account));
  }

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

  function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
      external {
      address account = getSignerAddress(_data);
      checkAndUpdateNonce(account, _nonce, TRANSFER_KEY_INDEX);

      address assetKey = accountStorage.getKeyData(account, TRANSFER_KEY_INDEX);
      bytes32 signHash = getSignHash(_data, _nonce);
      verifySig(assetKey, _signature, signHash);

      // solium-disable-next-line security/no-low-level-calls
      (bool success,) = address(this).call(_data);
      require(success, "calling self failed");
      emit TransferLogicEntered(_data, _nonce);
  }

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

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

  // called from 'enter'
  // signer is '_from'
  function transferErc20(address payable _from, address _to, address _token,
      uint256 _amount) external allowSelfCallsOnly {
      bytes memory methodData = abi.encodeWithSignature("transfer(address,uint256)"
          , _to, _amount);
      Account(_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 allowSelfCallsOnly {
      bytes memory methodData = abi.encodeWithSignature("transferFrom(address,
          address,uint256)", _from, _to, _amount);
      Account(_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
      allowSelfCallsOnly
  {
      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);
      }
      Account(_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
      allowSelfCallsOnly
  {
      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);
      }
      Account(_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;
  }
}
```

File logics/base/AccountBaseLogic.sol

```
pragma solidity ^0.5.4;

import "./BaseLogic.sol";

contract AccountBaseLogic is BaseLogic {

    uint256 constant internal DELAY_CHANGE_ADMIN_KEY = 21 days;
    uint256 constant internal DELAY_CHANGE_OPERATION_KEY = 7 days;
    uint256 constant internal DELAY_UNFREEZE_KEY = 7 days;
    uint256 constant internal DELAY_CHANGE_BACKUP = 21 days;
    uint256 constant internal DELAY_CHANGE_ADMIN_KEY_BY_BACKUP = 30 days;

    uint256 constant internal MAX_DEFINED_BACKUP_INDEX = 5;

  // Equals to bytes4(keccak256("changeAdminKey(address,address)"))
  bytes4 internal constant CHANGE_ADMIN_KEY = 0xd595d935;
  // Equals to bytes4(keccak256("changeAdminKeyByBackup(address,address)"))
  bytes4 internal constant CHANGE_ADMIN_KEY_BY_BACKUP = 0xfdd54ba1;
```

```
// Equals to bytes4(keccak256("changeAdminKeyWithoutDelay(address,address)"))
bytes4 internal constant CHANGE_ADMIN_KEY_WITHOUT_DELAY = 0x441d2e50;


  event ProposalExecuted(address indexed client, address indexed proposer, bytes
      functionData);

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

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

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

  /* 'executeProposal' is shared by AccountLogic and DualsigsLogic,
     proposed actions called from 'executeProposal':
       AccountLogic: changeAdminKeyByBackup
       DualsigsLogic: changeAdminKeyWithoutDelay, changeAllOperationKeysWithoutDelay,
            unfreezeWithoutDelay
  */
  function executeProposal(address payable _client, address _proposer, bytes
      calldata _functionData) external {
      bytes4 proposedActionId = getMethodId(_functionData);
      bytes32 functionHash = keccak256(_functionData);

      checkApproval(_client, _proposer, proposedActionId, functionHash);

      // call functions with/without delay
      // solium-disable-next-line security/no-low-level-calls
      (bool success,) = address(this).call(_functionData);
      require(success, "executeProposal failed");

      accountStorage.clearProposalData(_client, _proposer, proposedActionId);
      emit ProposalExecuted(_client, _proposer, _functionData);
  }

  function checkApproval(address _client, address _proposer, bytes4
      _proposedActionId, bytes32 _functionHash) internal view {
      bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer,
          _proposedActionId);
      require(hash == _functionHash, "proposal hash unmatch");

      uint256 backupCount;
      uint256 approvedCount;
      address[] memory approved = accountStorage.getProposalDataApproval(_client,
          _proposer, _proposedActionId);
      require(approved.length > 0, "no approval");

      // iterate backup list
      for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
          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");
        uint256 threshold = SafeMath.ceil(backupCount*6, 10);
        require(approvedCount >= threshold, "must have 60% approval at least");
    }

    function checkRelation(address _client, address _backup) internal view {
        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++) {
            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);
    }

    function clearRelatedProposalAfterAdminKeyChanged(address payable _client)
        internal {
        //clear any existing proposal proposed by both, proposer is _client
        accountStorage.clearProposalData(_client, _client,
            CHANGE_ADMIN_KEY_WITHOUT_DELAY);

        //clear any existing proposal proposed by backup, proposer is one of the
            backups
        for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
            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)) {
                accountStorage.clearProposalData(_client, backup,
                    CHANGE_ADMIN_KEY_BY_BACKUP);
            }
        }
    }

}
```

File logics/base/BaseLogic.sol

```
pragma solidity ^0.5.4;

import "../../Account.sol";
import "../../AccountStorage.sol";
import "../../utils/SafeMath.sol";

contract BaseLogic {

    bytes constant internal SIGN_HASH_PREFIX = "\x19Ethereum Signed Message:\n32";

    mapping (address => uint256) keyNonce;
    AccountStorage public accountStorage;

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

    modifier allowAccountCallsOnly(Account _account) {
        require(msg.sender == address(_account), "caller must be account");
        _;
    }

    event LogicInitialised(address wallet);

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

    constructor(AccountStorage _accountStorage) public {
        accountStorage = _accountStorage;
    }

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

    function initAccount(Account _account) external allowAccountCallsOnly(_account){
        emit LogicInitialised(address(_account));
    }

    // *************** Getter ********************** //

    function getKeyNonce(address _key) external view returns(uint256) {
        return keyNonce[_key];
    }

    // *************** Signature ********************** //

    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));
        bytes32 prefixedHash = keccak256(abi.encodePacked(SIGN_HASH_PREFIX, msgHash))
            ;
        return prefixedHash;
    }

    function verifySig(address _signingKey, bytes memory _signature, bytes32
            _signHash) internal pure {
        address recoveredAddr = recover(_signHash, _signature);
        require(recoveredAddr == _signingKey, "signature verification failed");
    }

    /**
     * @dev Returns the address that signed a hashed message (‘hash‘) with
     * ‘signature‘. This address can then be used for verification purposes.
     *
     * The ‘ecrecover‘ EVM opcode allows for malleable (non-unique) signatures:
     * this function rejects them by requiring the ‘s‘ value to be in the lower
     * half order, and the ‘v‘ value to be either 27 or 28.
     *
     * NOTE: This call _does not revert_ if the signature is invalid, or
     * if the signer is otherwise unable to be retrieved. In those scenarios,
     * the zero address is returned.
     *
     * IMPORTANT: ‘hash‘ _must_ be the result of a hash operation for the
     * verification to be secure: it is possible to craft signatures that
     * recover to arbitrary addresses for non-hashed data. A safe way to ensure
     * this is by receiving a hash of the original message (which may otherwise)
     * be too long), and then calling {toEthSignedMessageHash} on it.
     */
    function recover(bytes32 hash, bytes memory signature) internal pure returns (
        address) {
        // Check the signature length
        if (signature.length != 65) {
            return (address(0));
        }

        // Divide the signature in r, s and v variables
        bytes32 r;
        bytes32 s;
        uint8 v;

        // ecrecover takes the signature parameters, and the only way to get them
        // currently is to use assembly.
        // solhint-disable-next-line no-inline-assembly
        assembly {
            r := mload(add(signature, 0x20))
            s := mload(add(signature, 0x40))
            v := byte(0, mload(add(signature, 0x60)))
        }

        // EIP-2 still allows signature malleability for ecrecover(). Remove this
            possibility and make the signature
        // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.
            io/yellowpaper/paper.pdf), defines
        // the valid range for s in (281): 0 < s < secp256k1n // 2 + 1, and for v in
            (282): v \in {27, 28}. Most
        // signatures from current libraries generate a unique signature with an s-
            value in the lower half order.
        //
        // If your library generates malleable signatures, such as s-values in the
            upper range, calculate a new s-value
        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 -
            s1 and flip v from 27 to 28 or
        // vice versa. If your library also generates signatures with 0/1 for v
```

```
            instead 27/28, add 27 to v to accept
        // these malleable signatures as well.
        if (uint256(s) > 0
            x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
            return address(0);
        }

        if (v != 27 && v != 28) {
            return address(0);
        }

        // If the signature is valid (and not malleable), return the signer address
        return ecrecover(hash, v, r, s);
    }

    /* get signer 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 := 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
            _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)
    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);
        require(_nonce > keyNonce[key], "nonce too small");
        require(SafeMath.div(_nonce, 1000000) <= now + 86400, "nonce too big"); //
            86400=24*3600 seconds

        keyNonce[key] = _nonce;
    }
}
```

File testUtils/MyToken.sol

```
pragma solidity ^0.5.0;

// import "openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Mintable.sol";

contract MyToken is ERC20Mintable {
  string private _name;
    string private _symbol;
    uint8 private _decimals;
  uint256 public val;

  constructor(string memory name, string memory symbol, uint8 decimals/*, address
      account, uint256 amount*/) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
        // mint(account, amount);
  }

    /**
     * @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.
     *
     * > Note that 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;
    }

}
```

File utils/MultiOwned.sol

```
pragma solidity ^0.5.4;

import "./Owned.sol";

contract MultiOwned is Owned {
    mapping (address => bool) public multiOwners;

    modifier onlyMultiOwners {
        require(multiOwners[msg.sender] == true, "must be one of owners");
        _;
    }

    event OwnerAdded(address indexed _owner);
    event OwnerRemoved(address indexed _owner);

    function addOwner(address _owner) external onlyOwner {
        require(_owner != address(0), "owner must not be 0x0");
        if(multiOwners[_owner] == false) {
            multiOwners[_owner] = true;
            emit OwnerAdded(_owner);
        }
    }

    function removeOwner(address _owner) external onlyOwner {
        require(multiOwners[_owner] == true, "owner not exist");
        delete multiOwners[_owner];
        emit OwnerRemoved(_owner);
    }
}
```

File utils/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;

    event OwnerChanged(address indexed _newOwner);

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

    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");
        owner = _newOwner;
        emit OwnerChanged(_newOwner);
    }
}
```

File utils/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;
        }
    }
}
```

File Account.sol

```
pragma solidity ^0.5.4;

import "./LogicManager.sol";
import "./logics/base/BaseLogic.sol";
import "./AccountStorage.sol";

contract Account {

    // 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);
    event Invoked(address indexed module, address indexed target, uint indexed value
        , bytes data);
    event Received(uint indexed value, address indexed sender, bytes data);

    event AccountInit(address indexed account);

    modifier allowAuthorizedLogicContractsCallsOnly {
        require(LogicManager(manager).isAuthorized(msg.sender), "not an authorized
            logic");
        _;
    }

    function init(address _manager, address _accountStorage, address[] calldata
        _logics, address[] calldata _keys, address[] calldata _backups)
        external
    {
        require(manager == address(0), "Account: account already initialized");
        require(_manager != address(0) && _accountStorage != address(0), "Account:
            address is null");
        manager = _manager;

        for (uint i = 0; i < _logics.length; i++) {
            address logic = _logics[i];
            require(LogicManager(manager).isAuthorized(logic), "must be authorized
                logic");

            BaseLogic(logic).initAccount(this);
        }

        AccountStorage(_accountStorage).initAccount(this, _keys, _backups);

        emit AccountInit(address(this));
    }

    function invoke(address _target, uint _value, bytes calldata _data)
        external
        allowAuthorizedLogicContractsCallsOnly
    {
        // solium-disable-next-line security/no-call-value
        (bool success,) = _target.call.value(_value)(_data);
        require(success, "call to target failed");
        emit Invoked(msg.sender, _target, _value, _data);
    }

    /**
    * @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
        allowAuthorizedLogicContractsCallsOnly {
        enabled[_method] = _module;
        emit EnabledStaticCall(_module, _method);
    }

     /**
     * @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];
            if(logic == address(0)) {
                emit Received(msg.value, msg.sender, msg.data);
            }
            else {
                require(LogicManager(manager).isAuthorized(logic), "must be an
                    authorized logic for static call");
                // solium-disable-next-line security/no-inline-assembly
                assembly {
                    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())}
                }
            }
        }
    }
}
```

File AccountCreator.sol

```
pragma solidity ^0.5.4;

import "./utils/MultiOwned.sol";
import "./Account.sol";
import "./AccountProxy.sol";

contract AccountCreator is MultiOwned {

    address public logicManager;
    address public accountStorage;
    address public accountImpl;
    address[] public logics;

    // *************** Events *************************** //
    event AccountCreated(address indexed wallet, address[] keys, address[] backups);
    event Closed(address indexed sender);

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

    // *************** External Functions ********************* //

    function createAccount(address[] calldata _keys, address[] calldata _backups)
        external onlyMultiOwners {
        AccountProxy accountProxy = new AccountProxy(accountImpl);
        Account(address(accountProxy)).init(logicManager, accountStorage, logics,
            _keys, _backups);

        emit AccountCreated(address(accountProxy), _keys, _backups);
    }

    // *************** Suicide ******************** //

    function close() external onlyMultiOwners {
        selfdestruct(msg.sender);
        emit Closed(msg.sender);
    }
}
```

File AccountProxy.sol

```
pragma solidity ^0.5.4;

contract AccountProxy {

    address implementation;

    event Received(uint indexed value, address indexed sender, bytes data);

    constructor(address _implementation) public {
        implementation = _implementation;
    }

    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())}
            }
        }
    }
}
```

File AccountStorage.sol

```
pragma solidity ^0.5.4;

import "./Account.sol";
import "./LogicManager.sol";


contract AccountStorage {

    modifier allowAccountCallsOnly(Account _account) {
        require(msg.sender == address(_account), "caller must be account");
        _;
    }

    modifier allowAuthorizedLogicContractsCallsOnly(address payable _account) {
        require(LogicManager(Account(_account).manager()).isAuthorized(msg.sender), "
            not an authorized logic");
        _;
    }

    struct KeyItem {
        address pubKey;
        uint256 status;
    }

    struct BackupAccount {
        address backup;
        uint256 effectiveDate;//means not effective until this timestamp
        uint256 expiryDate;//means effective until this timestamp
    }

    struct DelayItem {
        bytes32 hash;
        uint256 dueTime;
    }

    struct Proposal {
        bytes32 hash;
        address[] approval;
    }

    // account => quantity of operation keys (index >= 1)
    mapping (address => uint256) operationKeyCount;

    // account => index => KeyItem
    mapping (address => mapping(uint256 => KeyItem)) keyData;

    // account => index => backup account
    mapping (address => mapping(uint256 => BackupAccount)) backupData;

    /* account => actionId => DelayItem

       delayData applies to these 4 actions:
       changeAdminKey, changeAllOperationKeys, unfreeze, changeAdminKeyByBackup
    */
    mapping (address => mapping(bytes4 => DelayItem)) delayData;

    // client account => proposer account => proposed actionId => Proposal
    mapping (address => mapping(address => mapping(bytes4 => Proposal)))
        proposalData;

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

    function getOperationKeyCount(address _account) external view returns(uint256) {
        return operationKeyCount[_account];
    }

    function increaseKeyCount(address payable _account) external
        allowAuthorizedLogicContractsCallsOnly(_account) {
        operationKeyCount[_account] = operationKeyCount[_account] + 1;
    }

    // *************** keyData ********************** //

    function getKeyData(address _account, uint256 _index) public view returns(
        address) {
        KeyItem memory item = keyData[_account][_index];
        return item.pubKey;
    }

    function setKeyData(address payable _account, uint256 _index, address _key)
        external allowAuthorizedLogicContractsCallsOnly(_account) {
        require(_key != address(0), "invalid _key value");
        KeyItem storage item = keyData[_account][_index];
        item.pubKey = _key;
    }

    // *************** keyStatus ********************** //

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

    function setKeyStatus(address payable _account, uint256 _index, uint256 _status)
         external allowAuthorizedLogicContractsCallsOnly(_account) {
        KeyItem storage item = keyData[_account][_index];
        item.status = _status;
    }

    // *************** backupData ********************** //

    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
      allowAuthorizedLogicContractsCallsOnly(_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
      allowAuthorizedLogicContractsCallsOnly(_account)
  {
      BackupAccount storage b = backupData[_account][_index];
      b.expiryDate = _expiry;
  }

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

  // *************** delayData ********************** //

  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 allowAuthorizedLogicContractsCallsOnly(_account)
      {
      DelayItem storage item = delayData[_account][_actionId];
```

```
            item.hash = _hash;
            item.dueTime = _dueTime;
        }

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

        // *************** proposalData ********************** //

        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
            allowAuthorizedLogicContractsCallsOnly(_client)
        {
            Proposal storage p = proposalData[_client][_proposer][_actionId];
            if (p.hash > 0) {
                if (p.hash == _hash) {
                    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.length = 0;
                }
            } else {
                p.hash = _hash;
                p.approval.push(_approvedBackup);
            }
        }

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


        // *************** init ********************** //
        function initAccount(Account _account, address[] calldata _keys, address[]
            calldata _backups)
            external
            allowAccountCallsOnly(_account)
        {
```

```
        require(getKeyData(address(_account), 0) == address(0), "AccountStorage:
            account already initialized!");
        require(_keys.length > 0, "empty keys array");

        operationKeyCount[address(_account)] = _keys.length - 1;

        for (uint256 index = 0; index < _keys.length; index++) {
            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++) {
            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));
        }
    }
}
```

File LogicManager.sol

```
pragma solidity ^0.5.4;

import "./utils/Owned.sol";

contract LogicManager is Owned {

    event UpdateLogicSubmitted(address indexed logic, bool value);
    event UpdateLogicDone(address indexed logic, bool value);

    struct pending {
        bool value;
        uint dueTime;
    }

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

    // 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 authorized logics
    uint public logicCount;

    constructor(address[] memory _initialLogics, uint256 _pendingTime) public
    {
        for (uint i = 0; i < _initialLogics.length; i++) {
            address logic = _initialLogics[i];
            authorized[logic] = true;
            logicCount += 1;
        }

        // pendingTime: 4 days for mainnet, 4 minutes for ropsten testnet
        pendingTime = _pendingTime;
    }

    function isAuthorized(address _logic) external view returns (bool) {
        return authorized[_logic];
    }

    function submitUpdate(address _logic, bool _value) external onlyOwner {
        pending storage p = pendingLogics[_logic];
        p.value = _value;
        p.dueTime = now + pendingTime;
        emit UpdateLogicSubmitted(_logic, _value);
    }

    function updateLogic(address _logic, bool _value) internal {
        if (authorized[_logic] != _value) {
            if(_value) {
                logicCount += 1;
                authorized[_logic] = true;
            }
            else {
                logicCount -= 1;
                require(logicCount > 0, "must have at least one logic module");
                delete authorized[_logic];
            }
            emit UpdateLogicDone(_logic, _value);
        }
    }

    function triggerUpdateLogic(address _logic) external {
        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);
        delete pendingLogics[_logic];
    }
}
```

CERTIK

Building Fully Trustworthy Smart Contracts and Blockchain Ecosystems