CERTIK

Audit Report

Produced by CertiK

for 

Sep 30, 2019

CertiK Audit Report For MyKey

Request Date: 2019-08-28
Revision Date: 2019-09-30
Platform Name: Ethereum

CERTIK

Contents

Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and MyKey(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

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 as the product of the Smart Contract Audit request by MyKey. This audit was conducted to discover issues and vulnerabilities in the source code of MyKey's Smart Contracts. Utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review, a comprehensive examination has been performed. The auditing process pays special attention to the following considerations.

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessment of the codebase for 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

For every issue found, CertiK categorizes them into 3 buckets based on its risk level:

Critical

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

Medium

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

Low

The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilies, but no concern found yet.

Testing Summary

PASS

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

Sep 30, 2019

Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code using our proprietary static analysis and formal verification engine to detect the follow type of issues.

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

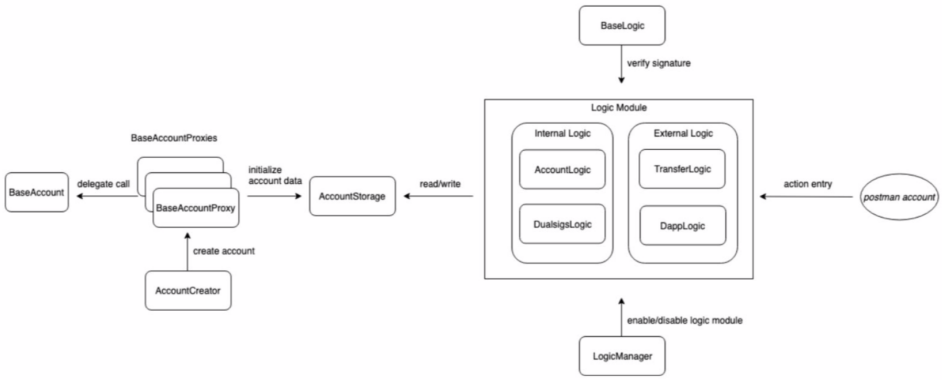
CertiK was chosen by MyKey to audit the design and implementation of its soon to be released smart contract. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That 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**
  `7e18ac57c4fbf2c375ea7833f54576d392a6415fa579ea26734d1d57e4974c07`
- **AccountProxy.sol**
  `f334c7926ba32f68f52c64f01ac1d03b7ccdb7f5e88e664a449724b7e81c0dbf`
- **AccountStorage.sol**
  `8df921ecd0616212f2398049c80528266737e8bbf5b82d16ed6d16878dd2699b`
- **LogicManager.sol**
  `cdfc6120153db8e95f362cd6a73ae05a714c7e1fcce8f7d1d815694735db795f`

- **AccountLogic.sol**
  `ab2c1e82d044d102578c9af07195ad168d49ee7ccfccb0c5012a1efa297dcc96`
- **DappLogic.sol**
  `f9180dbcfbdd840efb66d51df35b8d54de37354a5cc362fdbf4569dc5d6daa3a`
- **DualsigsLogic.sol**
  `65b3a1b70eae76a5df29a20e9842308e5d529c17d0b2cf56abefda7ab2b6e6fd`
- **TransferLogic.sol**
  `6515eb85a68af6e14f740a4fdd858e4cb670e39d097bccb2d3edd3aaff4de62d`
- **AccountBaseLogic.sol**
  `5bb152cdb100990bad89b9c00eae246e04b48585d92b7db3b03672c131b625b4`
- **BaseLogic.sol**
  `63086d4adc804621465390f2c9688aa6b7c7ea006d7bb01d6dafeda89597b86e`
- **MyNft.sol**
  `b41eb4f8d4f96722562e31d68c15e5e224c771342680379954f51ce4fbbb8b4d`
- **MyToken.sol**
  `ad67e648646af505fc51152dd2d1cf81e4f5bf139a5b55cd1104e3cbfa5042a2`
- **Owned.sol**
  `9c3fe9adaedbbe27940e0f25c27c3d8e5811a3d3ad658e4d058a1840afcef09e`
- **SafeMath.sol**
  `8f5ffacb100244d0da64f334543c3298be1c48a7ce9aadae06516c5e01f47714`

MyKey Architect & Workflow Overview



System Overview:

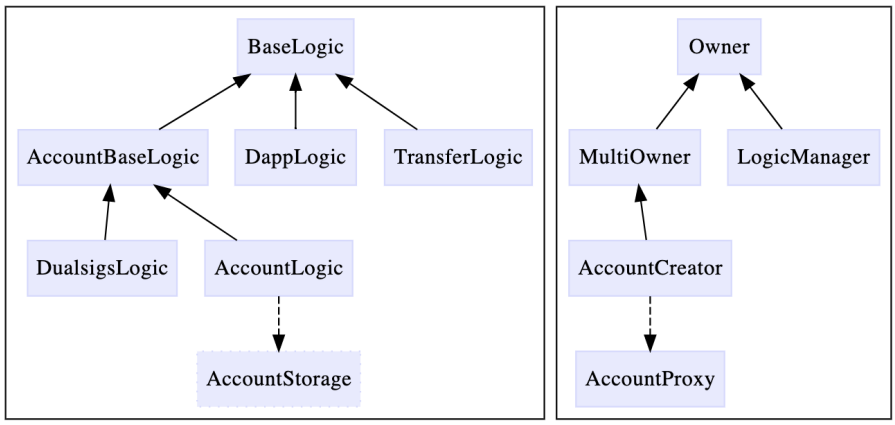
1. For each MyKey account will provide an corresponding `Account Proxy` contract address (Externally owned account)
2. While creating a new MyKey account, `MyKey Lab` will set as one of the backup keys as default setting, users can add more backup keys later.

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

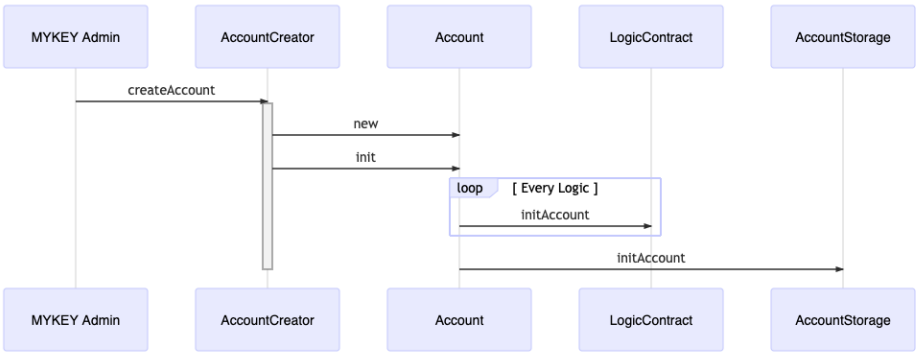
4. Logic Modules, including all the contract logic such as transfer, multi-signing proposal, dapp, and account related logic

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

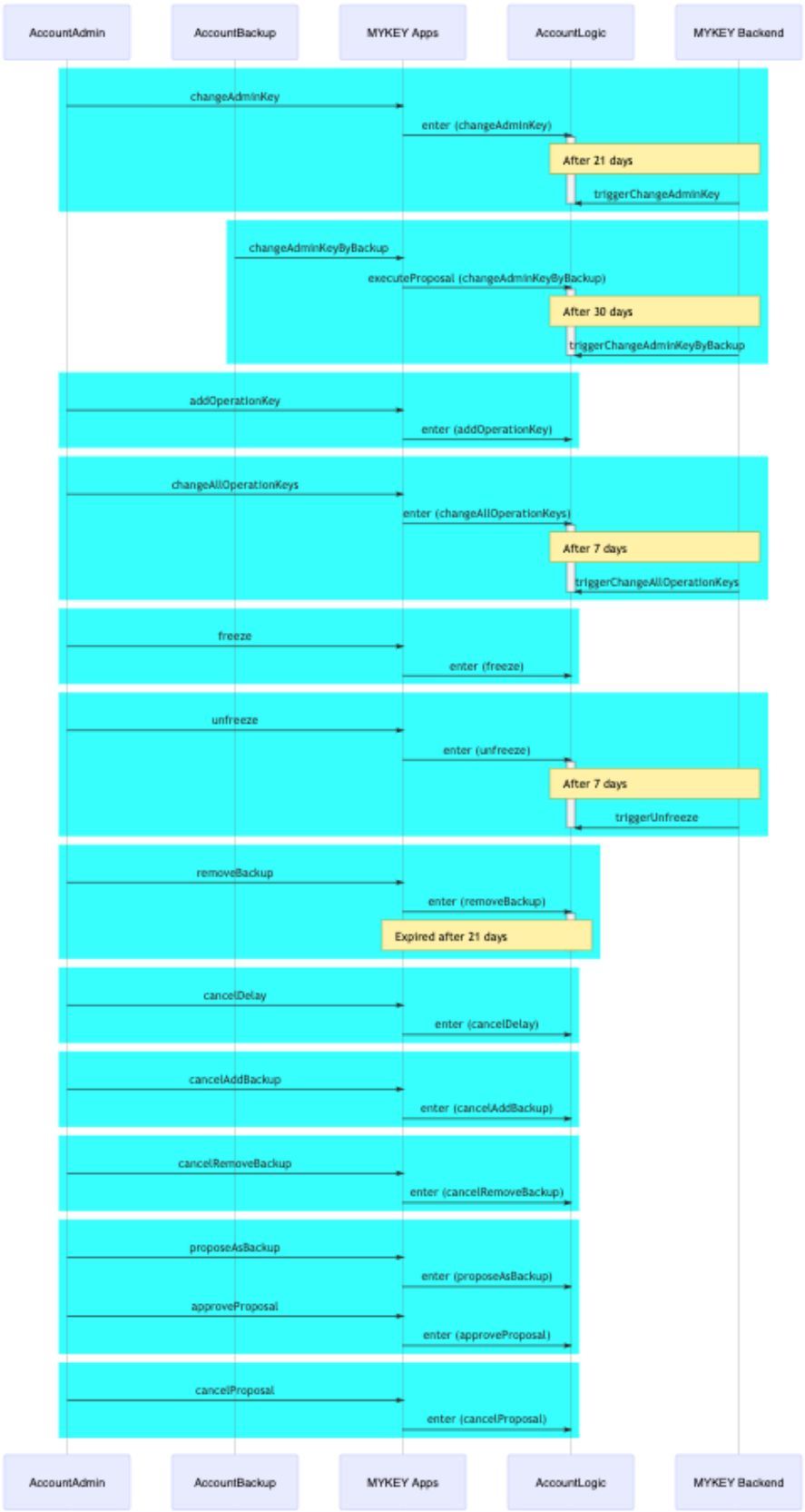
MyKey team provide the smart contract wallet design architecture diagram, each module workflow process can be illustrated as following:



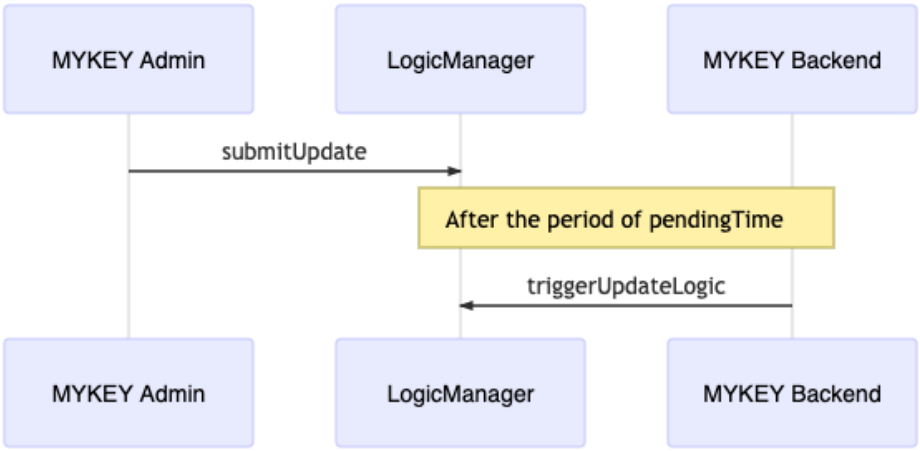
Account Creation Workflow



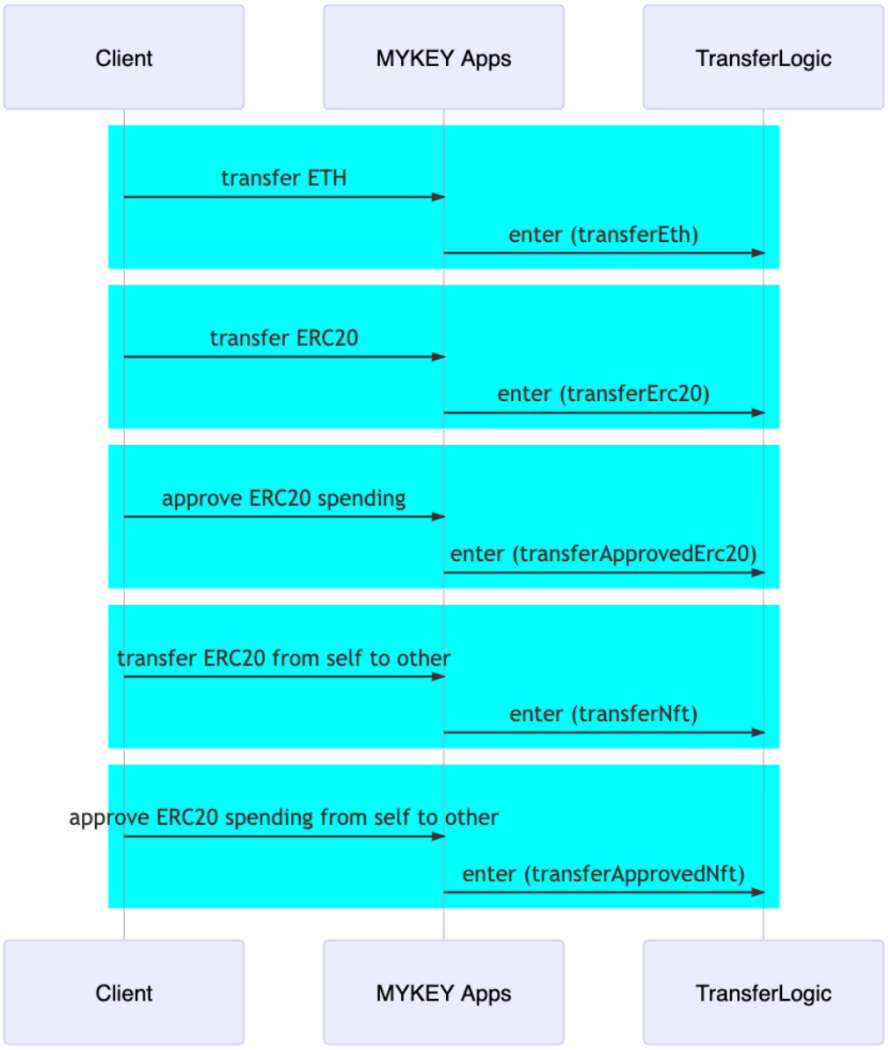
Account Logic Workflow



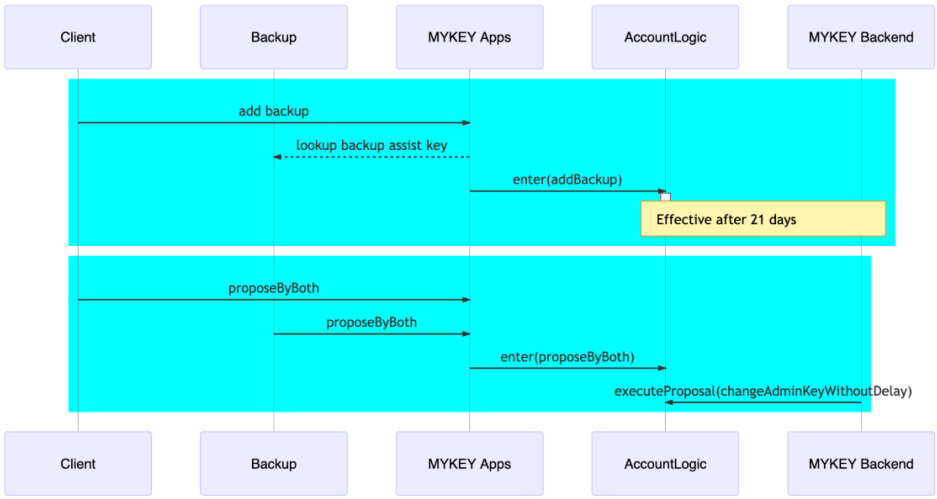
Account Logic Update Workflow



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 will be removed when the contract go live. The current implementation is benefit for testing and developing purpose.
- 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.
  - ✓ MyKey The code is updated and reflected in the latest commit.

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

AccountLogic.sol

- INFO Recommend to remove the declaration of `actionId` variable, instead use the constant variable directly.
  1. `changeAllOperationKeys`
  2. `triggerChangeAdminKeyByBackup`
  3. `changeAllOperationKeys`
  4. `triggerChangeAllOperationKeys`
  5. ✓ MyKey The code is updated and reflected in the latest commit.
- MINOR Recommend declaring the local memory variable outside the for loop for gas optimization.
  1. `changeAllOperationKeys`
  2. `triggerChangeAdminKeyByBackup`
  3. `changeAllOperationKeys`
  4. `triggerChangeAllOperationKeys`
  5. ✓ 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
    ....
}
```

- 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`
  - ✓ MyKey The code is updated and reflected in the latest commit.
- INFO `setBackupExpiryDate()`: Recommend adding `require()` to ensure `_expiry` is later than `now`
  - ✓ MyKey The code is updated and reflected in the latest commit.
- INFO `setDelayData()`: Recommend adding `require()` to ensure
  - `_hash` is a non zero address
  - `_dueTime` is later than `now`
  - ✓ MyKey The code is updated and reflected in the latest commit.

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

Owned.sol

- INFO Given `constructor()` not taking any input parameter, consider keeping the function as `internal`.
- 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);`
  - ✓ MyKey The code is updated and reflected in the latest commit.
- INFO Highly recommend using pull-over-push pattern for ownership transfer, openzepplin's `Ownable` contract, which is a good reference for consideration.

LogicManager.sol

- INFO Recommend changing `if (authorized[_logic] != _value)` in `updateLogic()` to be `require(authorized[_logic] != p.value)` in `triggerUpdateLogic()` before calling `updateLogic()`.
- 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

✓ Aware the risk of 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

```
1 pragma solidity ^0.5.4;
```

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

TIMESTAMP_DEPENDENCY

Line 131 in File DualsigsLogic.sol

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

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

TIMESTAMP_DEPENDENCY

Line 145 in File DualsigsLogic.sol

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

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

TIMESTAMP_DEPENDENCY

Line 150 in File DualsigsLogic.sol

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

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

INSECURE_COMPILER_VERSION

Line 1 in File AccountLogic.sol

```
1 pragma solidity ^0.5.4;
```

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

TIMESTAMP_DEPENDENCY

Line 75 in File AccountLogic.sol

```
75     accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY, hash, now + getDelayTime(
           TYPE_CHANGE_ADMIN_KEY));
```

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

TIMESTAMP_DEPENDENCY

Line 85 in File AccountLogic.sol

```
85     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 +
            getDelayTime(TYPE_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 +
            getDelayTime(TYPE_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 + getDelayTime(
            TYPE_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 + getDelayTime(
            TYPE_CHANGE_BACKUP));
```

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 247 in File AccountLogic.sol

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

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 256 in File AccountLogic.sol

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

"now" can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File DappLogic.sol

```
1 pragma solidity ^0.5.4;
```

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

INSECURE_COMPILER_VERSION

Line 1 in File AccountBaseLogic.sol

```
1 pragma solidity ^0.5.4;
```

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

TIMESTAMP_DEPENDENCY

Line 146 in File AccountBaseLogic.sol

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

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 146 in File AccountBaseLogic.sol

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

"now" can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File BaseLogic.sol

```
1 pragma solidity ^0.5.4;
```

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

TIMESTAMP_DEPENDENCY

Line 156 in File BaseLogic.sol

```
156         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

```
1 pragma solidity ^0.5.0;
```

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

Formal Verification Results

How to read

Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms

CERTIK *label location*: Line 30-34 in File howtoread.sol

CERTIK *label*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

Raw code location: Line 35-41 in File howtoread.sol

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample: This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

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);
```

```
54      checkAndUpdateNonce(account, _nonce, keyIndex);
55      address signingKey = accountStorage.getKeyData(account, keyIndex);
56      bytes32 signHash = getSignHash(_data, _nonce);
57      verifySig(signingKey, _signature, signHash);
58
59      // solium-disable-next-line security/no-low-level-calls
60      (bool success,) = address(this).call(_data);
61      require(success, "calling self failed");
62      emit AccountLogicEntered(_data, _nonce);
63    }
64
65    // *************** change admin key ********************** //
66
67      // called from 'enter'
68    function changeAdminKey(address payable _account, address _pkNew) external
          allowSelfCallsOnly {
69      require(_pkNew != address(0), "0x0 is invalid");
70      address pk = accountStorage.getKeyData(_account, 0);
71      require(pk != _pkNew, "identical admin key exists");
72      require(accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY) == 0, "delay
            data already exists");
73      bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew));
74      accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY, hash, now + getDelayTime(
            TYPE_CHANGE_ADMIN_KEY));
75    }
76
77      // called from external
78    function triggerChangeAdminKey(address payable _account, address _pkNew) external {
79      bytes32 hash = keccak256(abi.encodePacked('changeAdminKey', _account, _pkNew));
80      require(hash == accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY), "
            delay hash unmatch");
81
82      uint256 due = accountStorage.getDelayDataDueTime(_account, CHANGE_ADMIN_KEY);
83      require(due > 0, "delay data not found");
84      require(due <= now, "too early to trigger changeAdminKey");
85      accountStorage.setKeyData(_account, 0, _pkNew);
86      //clear any existing related delay data and proposal
87      accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
88      accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
89      clearRelatedProposalAfterAdminKeyChanged(_account);
90      emit ChangeAdminKeyTriggered(_account, _pkNew);
91    }
92
93    // *************** change admin key by backup proposal ********************** //
94
95      // called from 'executeProposal'
96    function changeAdminKeyByBackup(address payable _account, address _pkNew) external
          allowSelfCallsOnly {
97      require(_pkNew != address(0), "0x0 is invalid");
98      address pk = accountStorage.getKeyData(_account, 0);
99      require(pk != _pkNew, "identical admin key exists");
100     require(accountStorage.getDelayDataHash(_account, CHANGE_ADMIN_KEY_BY_BACKUP) ==
            0, "delay data already exists");
101     bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account,
            _pkNew));
102     accountStorage.setDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP, hash, now +
            getDelayTime(TYPE_CHANGE_ADMIN_KEY_BY_BACKUP));
103   }
```

```
104
105     // called from external
106   function triggerChangeAdminKeyByBackup(address payable _account, address _pkNew)
          external {
107     bytes32 hash = keccak256(abi.encodePacked('changeAdminKeyByBackup', _account,
          _pkNew));
108     require(hash == accountStorage.getDelayDataHash(_account,
          CHANGE_ADMIN_KEY_BY_BACKUP), "delay hash unmatch");
109
110     uint256 due = accountStorage.getDelayDataDueTime(_account,
          CHANGE_ADMIN_KEY_BY_BACKUP);
111     require(due > 0, "delay data not found");
112     require(due <= now, "too early to trigger changeAdminKeyByBackup");
113     accountStorage.setKeyData(_account, 0, _pkNew);
114     //clear any existing related delay data and proposal
115     accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
116     accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
117     clearRelatedProposalAfterAdminKeyChanged(_account);
118     emit ChangeAdminKeyByBackupTriggered(_account, _pkNew);
119   }
120
121   // *************** add operation key ********************** //
122
123     // called from 'enter'
124   function addOperationKey(address payable _account, address _pkNew) external
          allowSelfCallsOnly {
125     uint256 index = accountStorage.getOperationKeyCount(_account) + 1;
126     require(index > 0, "invalid operation key index");
127     // set a limit to prevent unnecessary trouble
128     require(index < 20, "index exceeds limit");
129     require(_pkNew != address(0), "0x0 is invalid");
130     address pk = accountStorage.getKeyData(_account, index);
131     require(pk == address(0), "operation key already exists");
132     accountStorage.setKeyData(_account, index, _pkNew);
133     accountStorage.increaseKeyCount(_account);
134   }
135
136   // *************** change all operation keys ********************** //
137
138     // called from 'enter'
139   function changeAllOperationKeys(address payable _account, address[] calldata _pks)
          external allowSelfCallsOnly {
140     uint256 keyCount = accountStorage.getOperationKeyCount(_account);
141     require(_pks.length == keyCount, "invalid number of keys");
142     require(accountStorage.getDelayDataHash(_account, CHANGE_ALL_OPERATION_KEYS) == 0,
            "delay data already exists");
143     address pk;
144     for (uint256 i = 0; i < keyCount; i++) {
145       pk = _pks[i];
146       require(pk != address(0), "0x0 is invalid");
147     }
148     bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account, _pks
          ));
149     accountStorage.setDelayData(_account, CHANGE_ALL_OPERATION_KEYS, hash, now +
          getDelayTime(TYPE_CHANGE_OPERATION_KEY));
150   }
151
152     // called from external
```

```
153   function triggerChangeAllOperationKeys(address payable _account, address[] calldata
          _pks) external {
154     bytes32 hash = keccak256(abi.encodePacked('changeAllOperationKeys', _account, _pks
            ));
155     require(hash == accountStorage.getDelayDataHash(_account,
            CHANGE_ALL_OPERATION_KEYS), "delay hash unmatch");
156
157     uint256 due = accountStorage.getDelayDataDueTime(_account,
            CHANGE_ALL_OPERATION_KEYS);
158     require(due > 0, "delay data not found");
159     require(due <= now, "too early to trigger changeAllOperationKeys");
160     address pk;
161     for (uint256 i = 0; i < accountStorage.getOperationKeyCount(_account); i++) {
162       pk = _pks[i];
163       accountStorage.setKeyData(_account, i+1, pk);
164       accountStorage.setKeyStatus(_account, i+1, 0);
165     }
166     accountStorage.clearDelayData(_account, CHANGE_ALL_OPERATION_KEYS);
167     emit ChangeAllOperationKeysTriggered(_account, _pks);
168   }
169
170   // *************** freeze/unfreeze all operation keys ********************** //
171
172     // called from 'enter'
173   function freeze(address payable _account) external allowSelfCallsOnly {
174     for (uint256 i = 1; i <= accountStorage.getOperationKeyCount(_account); i++) {
175       if (accountStorage.getKeyStatus(_account, i) == 0) {
176         accountStorage.setKeyStatus(_account, i, 1);
177       }
178     }
179   }
180
181     // called from 'enter'
182   function unfreeze(address payable _account) external allowSelfCallsOnly {
183     require(accountStorage.getDelayDataHash(_account, UNFREEZE) == 0, "delay data
            already exists");
184     bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account));
185     accountStorage.setDelayData(_account, UNFREEZE, hash, now + getDelayTime(
            TYPE_UNFREEZE_KEY));
186   }
187
188     // called from external
189   function triggerUnfreeze(address payable _account) external {
190     bytes32 hash = keccak256(abi.encodePacked('unfreeze', _account));
191     require(hash == accountStorage.getDelayDataHash(_account, UNFREEZE), "delay hash
            unmatch");
192
193     uint256 due = accountStorage.getDelayDataDueTime(_account, UNFREEZE);
194     require(due > 0, "delay data not found");
195     require(due <= now, "too early to trigger unfreeze");
196
197     for (uint256 i = 1; i <= accountStorage.getOperationKeyCount(_account); i++) {
198       if (accountStorage.getKeyStatus(_account, i) == 1) {
199         accountStorage.setKeyStatus(_account, i, 0);
200       }
201     }
202     accountStorage.clearDelayData(_account, UNFREEZE);
203     emit UnfreezeTriggered(_account);
```

```
204   }
205
206   // *************** remove backup ********************** //
207
208     // called from 'enter'
209   function removeBackup(address payable _account, address _backup) external
          allowSelfCallsOnly {
210     uint256 index = findBackup(_account, _backup);
211     require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
212
213     accountStorage.setBackupExpiryDate(_account, index, now + getDelayTime(
            TYPE_CHANGE_BACKUP));
214   }
215
216     // return backupData index(0~5), 6 means not found
217     // do make sure _backup is not 0x0
218   function findBackup(address _account, address _backup) public view returns(uint) {
219     uint index = MAX_DEFINED_BACKUP_INDEX + 1;
220     if (_backup == address(0)) {
221       return index;
222     }
223     address b;
224     for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
225       b = accountStorage.getBackupAddress(_account, i);
226       if (b == _backup) {
227         index = i;
228         break;
229       }
230     }
231     return index;
232   }
233
234   // *************** cancel delay action ********************** //
235
236     // called from 'enter'
237   function cancelDelay(address payable _account, bytes4 _actionId) external
          allowSelfCallsOnly {
238     accountStorage.clearDelayData(_account, _actionId);
239   }
240
241     // called from 'enter'
242   function cancelAddBackup(address payable _account, address _backup) external
          allowSelfCallsOnly {
243     uint256 index = findBackup(_account, _backup);
244     require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
245     uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_account, index);
246     require(effectiveDate > now, "already effective");
247     accountStorage.clearBackupData(_account, index);
248   }
249
250     // called from 'enter'
251   function cancelRemoveBackup(address payable _account, address _backup) external
          allowSelfCallsOnly {
252     uint256 index = findBackup(_account, _backup);
253     require(index <= MAX_DEFINED_BACKUP_INDEX, "backup invalid or not exist");
254     uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, index);
255     require(expiryDate > now, "already expired");
256     accountStorage.setBackupExpiryDate(_account, index, uint256(-1));
```

```
257   }
258
259   // *************** propose, approve and cancel proposal ********************** //
260
261     // called from 'enter'
262   // proposer is backup in the case of 'proposeAsBackup'
263   function proposeAsBackup(address _backup, address payable _client, bytes calldata
          _functionData) external allowSelfCallsOnly {
264     bytes4 proposedActionId = getMethodId(_functionData);
265     require(proposedActionId == CHANGE_ADMIN_KEY_BY_BACKUP, "invalid proposal by
            backup");
266     checkRelation(_client, _backup);
267     bytes32 functionHash = keccak256(_functionData);
268     accountStorage.setProposalData(_client, _backup, proposedActionId, functionHash,
            _backup);
269   }
270
271     // called from 'enter'
272   function approveProposal(address _backup, address payable _client, address _proposer
          , bytes calldata _functionData) external allowSelfCallsOnly {
273     bytes32 functionHash = keccak256(_functionData);
274     require(functionHash != 0, "invalid hash");
275     checkRelation(_client, _backup);
276     bytes4 proposedActionId = getMethodId(_functionData);
277     bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer,
            proposedActionId);
278     require(hash == functionHash, "proposal unmatch");
279     accountStorage.setProposalData(_client, _proposer, proposedActionId, functionHash,
             _backup);
280   }
281
282     // called from 'enter'
283   function cancelProposal(address payable _client, address _proposer, bytes4
          _proposedActionId) external allowSelfCallsOnly {
284     require(_client != _proposer, "cannot cancel dual signed proposal");
285     accountStorage.clearProposalData(_client, _proposer, _proposedActionId);
286   }
287
288   // *************** internal functions ********************** //
289
290     /*
291     index 0: admin key
292           1: asset(transfer)
293           2: adding
294           3: reserved(dapp)
295           4: assist
296      */
297   function getKeyIndex(bytes memory _data) internal pure returns (uint256) {
298     uint256 index; //index default value is 0, admin key
299     bytes4 methodId = getMethodId(_data);
300     if (methodId == ADD_OPERATION_KEY) {
301         index = 2; //adding key
302     } else if (methodId == PROPOSE_AS_BACKUP || methodId == APPROVE_PROPOSAL) {
303         index = 4; //assist key
304     }
305     return index;
306   }
307
```

```
308 }
```

File logics/DappLogic.sol

```
1  pragma solidity ^0.5.4;
2
3  import "./base/BaseLogic.sol";
4
5  contract DappLogic is BaseLogic {
6
7      /*
8      index 0: admin key
9            1: asset(transfer)
10           2: adding
11           3: reserved(dapp)
12           4: assist
13      */
14     uint constant internal DAPP_KEY_INDEX = 3;
15
16     // *************** Events *************************** //
17
18     event DappLogicInitialised(address indexed account);
19     event DappLogicEntered(bytes data, uint256 indexed nonce);
20
21     // *************** Constructor ********************** //
22     constructor(AccountStorage _accountStorage)
23         BaseLogic(_accountStorage)
24         public
25     {
26     }
27
28     // *************** Initialization ******************** //
29
30     function initAccount(Account _account) external allowAccountCallsOnly(_account){
31         emit DappLogicInitialised(address(_account));
32     }
33
34     // *************** action entry ********************** //
35
36     function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
           external {
37         address account = getSignerAddress(_data);
38         checkAndUpdateNonce(account, _nonce, DAPP_KEY_INDEX);
39
40         address dappKey = accountStorage.getKeyData(account, DAPP_KEY_INDEX);
41         bytes32 signHash = getSignHash(_data, _nonce);
42         verifySig(dappKey, _signature, signHash);
43
44         // solium-disable-next-line security/no-low-level-calls
45         (bool success,) = address(this).call(_data);
46         require(success, "calling self failed");
47         emit DappLogicEntered(_data, _nonce);
48     }
49
50     // *************** call Dapp ********************** //
51
52     // called from 'enter'
53     // call other contract from base account
54     function callContract(address payable _account, address payable _target, uint256
```

```
        _value, bytes calldata _methodData) external allowSelfCallsOnly {
55      Account(_account).invoke(_target, _value, _methodData);
56    }
57
58  }
```

File logics/DualsigsLogic.sol

```
1   pragma solidity ^0.5.4;
2
3   import "./base/AccountBaseLogic.sol";
4
5   /**
6   * @title DualsigsLogic
7   */
8   contract DualsigsLogic is AccountBaseLogic {
9
10    // Equals to bytes4(keccak256("changeAllOperationKeysWithoutDelay(address,address
          [])"))
11    bytes4 private constant CHANGE_ALL_OPERATION_KEYS_WITHOUT_DELAY = 0x02064abc;
12    // Equals to bytes4(keccak256("unfreezeWithoutDelay(address)"))
13    bytes4 private constant UNFREEZE_WITHOUT_DELAY = 0x69521650;
14    // Equals to bytes4(keccak256("addBackup(address,address)"))
15    bytes4 private constant ADD_BACKUP = 0x426b7407;
16    // Equals to bytes4(keccak256("proposeByBoth(address,address,bytes)"))
17    bytes4 private constant PROPOSE_BY_BOTH = 0x7548cb94;
18
19      event DualsigsLogicInitialised(address indexed account);
20      event DualsigsLogicEntered(bytes data, uint256 indexed clientNonce, uint256
            backupNonce);
21
22    modifier allowDualSigsActionOnly(bytes memory _data) {
23      bytes4 methodId = getMethodId(_data);
24      require ((methodId == ADD_BACKUP) ||
25             (methodId == PROPOSE_BY_BOTH), "wrong entry");
26      _;
27    }
28
29    // *************** Constructor ********************** //
30
31    constructor(AccountStorage _accountStorage)
32      AccountBaseLogic(_accountStorage)
33      public
34    {
35    }
36
37      // *************** Initialization ******************** //
38      function initAccount(Account _account) external allowAccountCallsOnly(_account){
39          emit DualsigsLogicInitialised(address(_account));
40      }
41
42    // *************** action entry ********************** //
43
44      /* DualsigsLogic has 2 actions called from 'enter':
45          addBackup, proposeByBoth
46    */
47    function enter(
48      bytes calldata _data, bytes calldata _clientSig, bytes calldata _backupSig,
          uint256 _clientNonce, uint256 _backupNonce
```

```
49      )
50        external allowDualSigsActionOnly(_data)
51      {
52            verifyClient(_data, _clientSig, _clientNonce);
53            verifyBackup(_data, _backupSig, _backupNonce);
54
55        // solium-disable-next-line security/no-low-level-calls
56        // (bool success,) = address(this).call(_data);
57        // require(success, "enterWithDualSigs failed");
58        emit DualsigsLogicEntered(_data, _clientNonce, _backupNonce);
59      }
60
61      function verifyClient(bytes memory _data, bytes memory _clientSig, uint256
            _clientNonce) internal {
62        address client = getSignerAddress(_data);
63        //client sign with admin key
64        uint256 clientKeyIndex = 0;
65        if ((getMethodId(_data) == PROPOSE_BY_BOTH) &&
66            (getProposedMethodId(_data) == CHANGE_ADMIN_KEY_WITHOUT_DELAY)) {
67          // if proposed action is 'changeAdminKeyWithoutDelay', do not check
                _clientNonce
68          verifySig(accountStorage.getKeyData(client, clientKeyIndex), _clientSig,
                getSignHashWithoutNonce(_data));
69        } else {
70          checkAndUpdateNonce(client, _clientNonce, clientKeyIndex);
71          verifySig(accountStorage.getKeyData(client, clientKeyIndex), _clientSig,
                getSignHash(_data, _clientNonce));
72        }
73      }
74
75        function verifyBackup(bytes memory _data, bytes memory _backupSig, uint256
              _backupNonce) internal {
76        address backup = getSecondSignerAddress(_data);
77        //backup sign with assist key
78        uint256 backupKeyIndex = 4;
79        checkAndUpdateNonce(backup, _backupNonce, backupKeyIndex);
80        verifySig(accountStorage.getKeyData(backup, backupKeyIndex), _backupSig,
              getSignHash(_data, _backupNonce));
81      }
82
83      // *************** change admin key ********************** //
84
85        // called from 'executeProposal'
86      function changeAdminKeyWithoutDelay(address payable _account, address _pkNew)
            external allowSelfCallsOnly {
87        address pk = accountStorage.getKeyData(_account, 0);
88        require(pk != _pkNew, "identical admin key already exists");
89        require(_pkNew != address(0), "0x0 is invalid");
90        accountStorage.setKeyData(_account, 0, _pkNew);
91        //clear any existing related delay data and proposal
92        accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY);
93        accountStorage.clearDelayData(_account, CHANGE_ADMIN_KEY_BY_BACKUP);
94        clearRelatedProposalAfterAdminKeyChanged(_account);
95      }
96
97      // *************** change all operation keys ********************** //
98
99        // called from 'executeProposal'
```

```
100     function changeAllOperationKeysWithoutDelay(address payable _account, address[]
            calldata _pks) external allowSelfCallsOnly {
101       uint256 keyCount = accountStorage.getOperationKeyCount(_account);
102       require(_pks.length == keyCount, "invalid number of keys");
103       for (uint256 i = 0; i < keyCount; i++) {
104         address pk = _pks[i];
105         require(pk != address(0), "0x0 is invalid");
106         accountStorage.setKeyData(_account, i+1, pk);
107         accountStorage.setKeyStatus(_account, i+1, 0);
108       }
109     }
110
111     // *************** freeze/unfreeze all operation keys ********************** //
112
113       // called from 'executeProposal'
114     function unfreezeWithoutDelay(address payable _account) external
            allowSelfCallsOnly {
115       for (uint256 i = 0; i < accountStorage.getOperationKeyCount(_account); i++) {
116         if (accountStorage.getKeyStatus(_account, i+1) == 1) {
117           accountStorage.setKeyStatus(_account, i+1, 0);
118         }
119       }
120     }
121
122     // *************** add backup ********************** //
123
124       // called from 'enter'
125     function addBackup(address payable _account, address _backup) external
            allowSelfCallsOnly {
126       require(_account != _backup, "cannot be backup of oneself");
127       uint256 index = findAvailableSlot(_account, _backup);
128       require(index <= MAX_DEFINED_BACKUP_INDEX, "invalid or duplicate or no vacancy")
              ;
129       accountStorage.setBackup(_account, index, _backup, now + getDelayTime(
            TYPE_CHANGE_BACKUP), uint256(-1));
130     }
131
132       // return backupData index(0~5), 6 means not found
133       // 'available' means empty or expired
134     function findAvailableSlot(address _account, address _backup) public view returns(
            uint) {
135       uint index = MAX_DEFINED_BACKUP_INDEX + 1;
136       if (_backup == address(0)) {
137         return index;
138       }
139       for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
140             address backup = accountStorage.getBackupAddress(_account, i);
141             uint256 expiryDate = accountStorage.getBackupExpiryDate(_account, i);
142         // _backup already exists and not expired
143         if ((backup == _backup) && (expiryDate > now)) {
144           return MAX_DEFINED_BACKUP_INDEX + 1;
145         }
146         if (index > MAX_DEFINED_BACKUP_INDEX) {
147           // zero address or backup expired
148           if ((backup == address(0)) || (expiryDate <= now)) {
149                   index = i;
150           }
151         }
```

```
152       }
153       return index;
154     }
155
156     // *************** propose, approve, execute and cancel proposal
            ********************** //
157
158       // called from 'enter'
159     // proposer is client in the case of 'proposeByBoth'
160     function proposeByBoth(address payable _client, address _backup, bytes calldata
            _functionData) external allowSelfCallsOnly {
161       bytes4 proposedActionId = getMethodId(_functionData);
162       require(isFastAction(proposedActionId), "invalid proposal");
163       checkRelation(_client, _backup);
164       bytes32 functionHash = keccak256(_functionData);
165       accountStorage.setProposalData(_client, _client, proposedActionId, functionHash,
              _backup);
166     }
167
168     function isFastAction(bytes4 _actionId) internal pure returns(bool) {
169       if ((_actionId == CHANGE_ADMIN_KEY_WITHOUT_DELAY) ||
170         (_actionId == CHANGE_ALL_OPERATION_KEYS_WITHOUT_DELAY) ||
171         (_actionId == UNFREEZE_WITHOUT_DELAY))
172       {
173         return true;
174       }
175       return false;
176     }
177
178     // *************** internal functions ********************** //
179
180     function getSecondSignerAddress(bytes memory _b) internal pure returns (address _a
            ) {
181       require(_b.length >= 68, "data length too short");
182       // solium-disable-next-line security/no-inline-assembly
183       assembly {
184         //68 = 32 + 4 + 32
185         let mask := 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
186         _a := and(mask, mload(add(_b, 68)))
187       }
188     }
189
190       function getProposedMethodId(bytes memory _b) internal pure returns (bytes4 _a)
              {
191       require(_b.length >= 164, "data length too short");
192           // solium-disable-next-line security/no-inline-assembly
193           assembly {
194         /* 'proposeByBoth' data example:
195         0x
196         7548cb94                                                    // method id
197         000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5 // param 0
198         00000000000000000000000011390e32ccdfb3f85e92b949c72fe482d77838f3 // param 1
199         0000000000000000000000000000000000000000000000000000000000000060 // data length
                including padding
200         0000000000000000000000000000000000000000000000000000000000000044 // true data
              length
201         441d2e50                                                    // method id(
              proposed method: changeAdminKeyWithoutDelay)
```

```
202         000000000000000000000000b7055946345ad40f8cca3feb075dfadd9e2641b5 // param 0
203         00000000000000000000000013667a2711960c95fae074f90e0f739bc324d1ed // param 1
204         0000000000000000000000000000000000000000000000000000000000000000     // padding
205         */
206               // the first 32 bytes is the length of the bytes array _b
207         // 32 + 4 + 32 + 32 + 32 + 32 = 164
208               _a := mload(add(_b, 164))
209           }
210       }
211
212       function getSignHashWithoutNonce(bytes memory _data) internal view returns(
              bytes32) {
213           // use EIP 191
214           // 0x1900 + this logic address + data
215           bytes32 msgHash = keccak256(abi.encodePacked(byte(0x19), byte(0), address(
                  this), _data));
216           bytes32 prefixedHash = keccak256(abi.encodePacked(SIGN_HASH_PREFIX, msgHash))
                  ;
217           return prefixedHash;
218       }
219
220   }
```

File logics/TransferLogic.sol

```
1   pragma solidity ^0.5.4;
2
3   import "./base/BaseLogic.sol";
4
5   contract TransferLogic is BaseLogic {
6
7       /*
8       index 0: admin key
9             1: asset(transfer)
10            2: adding
11            3: reserved(dapp)
12            4: assist
13       */
14      uint constant internal TRANSFER_KEY_INDEX = 1;
15
16      // Equals to `bytes4(keccak256("onERC721Received(address,address,uint256,bytes)
            "))`
17      bytes4 private constant ERC721_RECEIVED = 0x150b7a02;
18
19      // *************** Events *************************** //
20
21      event TransferLogicInitialised(address indexed account);
22      event TransferLogicEntered(bytes data, uint256 indexed nonce);
23
24      // *************** Constructor ********************** //
25
26      constructor(AccountStorage _accountStorage)
27      BaseLogic(_accountStorage)
28      public
29    {
30    }
31
32      // *************** Initialization ******************** //
33
```

```
34      // enable staic call 'onERC721Received' from base account
35      function initAccount(Account _account) external allowAccountCallsOnly(_account){
36          _account.enableStaticCall(address(this), ERC721_RECEIVED);
37          emit TransferLogicInitialised(address(_account));
38      }
39
40      // *************** action entry ******************** //
41
42      function enter(bytes calldata _data, bytes calldata _signature, uint256 _nonce)
            external {
43          address account = getSignerAddress(_data);
44          checkAndUpdateNonce(account, _nonce, TRANSFER_KEY_INDEX);
45
46          address assetKey = accountStorage.getKeyData(account, TRANSFER_KEY_INDEX);
47          bytes32 signHash = getSignHash(_data, _nonce);
48          verifySig(assetKey, _signature, signHash);
49
50          // solium-disable-next-line security/no-low-level-calls
51          // (bool success,) = address(this).call(_data);
52          // require(success, "calling self failed");
53          emit TransferLogicEntered(_data, _nonce);
54      }
55
56      // *************** transfer assets ******************** //
57
58      // called from 'enter'
59      // signer is '_from'
60      function transferEth(address payable _from, address _to, uint256 _amount)
            external allowSelfCallsOnly {
61          Account(_from).invoke(_to, _amount, "");
62      }
63
64      // called from 'enter'
65      // signer is '_from'
66      function transferErc20(address payable _from, address _to, address _token,
            uint256 _amount) external allowSelfCallsOnly {
67          bytes memory methodData = abi.encodeWithSignature("transfer(address,uint256)"
                , _to, _amount);
68          Account(_from).invoke(_token, 0, methodData);
69      }
70
71      // called from 'enter'
72      // signer is '_approvedSpender'
73      // make sure '_from' has approved allowance to '_approvedSpender'
74      function transferApprovedErc20(address payable _approvedSpender, address _from,
            address _to, address _token, uint256 _amount) external allowSelfCallsOnly {
75          bytes memory methodData = abi.encodeWithSignature("transferFrom(address,
                address,uint256)", _from, _to, _amount);
76          Account(_approvedSpender).invoke(_token, 0, methodData);
77      }
78
79      // called from 'enter'
80      // signer is '_from'
81      function transferNft(
82          address payable _from, address _to, address _nftContract, uint256 _tokenId,
                bytes calldata _data, bool _safe)
83          external
84          allowSelfCallsOnly
```

```
85      {
86          bytes memory methodData;
87          if(_safe) {
88              methodData = abi.encodeWithSignature("safeTransferFrom(address,address,
                    uint256,bytes)", _from, _to, _tokenId, _data);
89          } else {
90              methodData = abi.encodeWithSignature("transferFrom(address,address,
                    uint256)", _from, _to, _tokenId);
91          }
92          Account(_from).invoke(_nftContract, 0, methodData);
93      }
94
95      // called from 'enter'
96      // signer is '_approvedSpender'
97      // make sure '_from' has approved nftToken to '_approvedSpender'
98      function transferApprovedNft(
99          address payable _approvedSpender, address _from, address _to, address
                _nftContract, uint256 _tokenId, bytes calldata _data, bool _safe)
100         external
101         allowSelfCallsOnly
102     {
103         bytes memory methodData;
104         if(_safe) {
105             methodData = abi.encodeWithSignature("safeTransferFrom(address,address,
                    uint256,bytes)", _from, _to, _tokenId, _data);
106         } else {
107             methodData = abi.encodeWithSignature("transferFrom(address,address,
                    uint256)", _from, _to, _tokenId);
108         }
109         Account(_approvedSpender).invoke(_nftContract, 0, methodData);
110     }
111
112     // *************** callback of safeTransferFrom ******************** //
113
114     function onERC721Received(address _operator, address _from, uint256 _tokenId,
            bytes calldata _data) external pure returns (bytes4) {
115         return ERC721_RECEIVED;
116     }
117 }
```

File logics/base/AccountBaseLogic.sol

```
1  pragma solidity ^0.5.4;
2
3  import "./BaseLogic.sol";
4
5  contract AccountBaseLogic is BaseLogic {
6
7      /*
8        mainnet: 0;
9        local: 1;
10       ropsten: 2;
11     */
12     uint256 constant internal ENVIRONMENT = 1;
13
14     uint256 constant internal TYPE_CHANGE_ADMIN_KEY = 0;
15     uint256 constant internal TYPE_CHANGE_OPERATION_KEY = 1;
16     uint256 constant internal TYPE_UNFREEZE_KEY = 2;
17     uint256 constant internal TYPE_CHANGE_BACKUP = 3;
```

```
18    uint256 constant internal TYPE_CHANGE_ADMIN_KEY_BY_BACKUP = 4;
19
20    uint256 constant internal MAX_DEFINED_BACKUP_INDEX = 5;
21
22  // Equals to bytes4(keccak256("changeAdminKey(address,address)"))
23  bytes4 internal constant CHANGE_ADMIN_KEY = 0xd595d935;
24  // Equals to bytes4(keccak256("changeAdminKeyByBackup(address,address)"))
25  bytes4 internal constant CHANGE_ADMIN_KEY_BY_BACKUP = 0xfdd54ba1;
26  // Equals to bytes4(keccak256("changeAdminKeyWithoutDelay(address,address)"))
27  bytes4 internal constant CHANGE_ADMIN_KEY_WITHOUT_DELAY = 0x441d2e50;
28
29
30    event ProposalExecuted(address indexed client, address indexed proposer, bytes
          functionData);
31
32    // *************** Constructor ********************** //
33
34  constructor(AccountStorage _accountStorage)
35    BaseLogic(_accountStorage)
36    public
37  {
38  }
39
40    // *************** Getter ********************** //
41
42    function getDelayTime(uint256 _actionType) internal pure returns(uint256) {
43        if (ENVIRONMENT == 0) { //mainnet
44            if (_actionType == TYPE_CHANGE_ADMIN_KEY) {
45                return 21 days;
46            } else if (_actionType == TYPE_CHANGE_OPERATION_KEY) {
47                return 7 days;
48            } else if (_actionType == TYPE_UNFREEZE_KEY) {
49                return 7 days;
50            } else if (_actionType == TYPE_CHANGE_BACKUP) {
51                return 21 days;
52            } else if (_actionType == TYPE_CHANGE_ADMIN_KEY_BY_BACKUP) {
53                return 30 days;
54            }
55        } else if (ENVIRONMENT == 1) { //local
56            return 2 seconds;
57        } else if (ENVIRONMENT == 2) { //ropsten
58            if (_actionType == TYPE_CHANGE_ADMIN_KEY) {
59                return 21*100 seconds;
60            } else if (_actionType == TYPE_CHANGE_OPERATION_KEY) {
61                return 7*100 seconds;
62            } else if (_actionType == TYPE_UNFREEZE_KEY) {
63                return 7*100 seconds;
64            } else if (_actionType == TYPE_CHANGE_BACKUP) {
65                return 21*100 seconds;
66            } else if (_actionType == TYPE_CHANGE_ADMIN_KEY_BY_BACKUP) {
67                return 30*100 seconds;
68            }
69        }
70        revert("invalid type or environment");
71    }
72
73    // *************** Proposal ********************** //
74
```

```
75      /* 'executeProposal' is shared by AccountLogic and DualsigsLogic,
76         proposed actions called from 'executeProposal':
77           AccountLogic: changeAdminKeyByBackup
78           DualsigsLogic: changeAdminKeyWithoutDelay, changeAllOperationKeysWithoutDelay,
                 unfreezeWithoutDelay
79      */
80      function executeProposal(address payable _client, address _proposer, bytes
            calldata _functionData) external {
81          bytes4 proposedActionId = getMethodId(_functionData);
82          bytes32 functionHash = keccak256(_functionData);
83
84          checkApproval(_client, _proposer, proposedActionId, functionHash);
85
86          // call functions with/without delay
87          // solium-disable-next-line security/no-low-level-calls
88          (bool success,) = address(this).call(_functionData);
89          require(success, "executeProposal failed");
90
91          accountStorage.clearProposalData(_client, _proposer, proposedActionId);
92          emit ProposalExecuted(_client, _proposer, _functionData);
93      }
94
95      function checkApproval(address _client, address _proposer, bytes4
            _proposedActionId, bytes32 _functionHash) internal view {
96          bytes32 hash = accountStorage.getProposalDataHash(_client, _proposer,
                _proposedActionId);
97          require(hash == _functionHash, "proposal hash unmatch");
98
99          uint256 backupCount;
100         uint256 approvedCount;
101         address[] memory approved = accountStorage.getProposalDataApproval(_client,
                _proposer, _proposedActionId);
102         require(approved.length > 0, "no approval");
103
104         // iterate backup list
105         for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
106             address backup = accountStorage.getBackupAddress(_client, i);
107             uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_client, i);
108             uint256 expiryDate = accountStorage.getBackupExpiryDate(_client, i);
109             if (backup != address(0) && isEffectiveBackup(effectiveDate, expiryDate)) {
110                 // count how many backups in backup list
111                 backupCount += 1;
112                 // iterate approved array
113                 for (uint256 k = 0; k < approved.length; k++) {
114                     if (backup == approved[k]) {
115                         // count how many approved backups still exist in backup list
116                         approvedCount += 1;
117                     }
118                 }
119             }
120         }
121         require(backupCount > 0, "no backup in list");
122         uint256 threshold = SafeMath.ceil(backupCount*6, 10);
123         require(approvedCount >= threshold, "must have 60% approval at least");
124     }
125
126     function checkRelation(address _client, address _backup) internal view {
127         require(_backup != address(0), "backup cannot be 0x0");
```

```
128         require(_client != address(0), "client cannot be 0x0");
129         bool isBackup;
130         for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
131             address backup = accountStorage.getBackupAddress(_client, i);
132             uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_client, i);
133             uint256 expiryDate = accountStorage.getBackupExpiryDate(_client, i);
134             // backup match and effective and not expired
135             if (_backup == backup && isEffectiveBackup(effectiveDate, expiryDate)) {
136                 isBackup = true;
137                 break;
138             }
139         }
140         require(isBackup, "backup does not exist in list");
141     }
142 
143     function isEffectiveBackup(uint256 _effectiveDate, uint256 _expiryDate) internal
        view returns(bool) {
144         return (_effectiveDate <= now) && (_expiryDate > now);
145     }
146 
147     function clearRelatedProposalAfterAdminKeyChanged(address payable _client)
        internal {
148         //clear any existing proposal proposed by both, proposer is _client
149         accountStorage.clearProposalData(_client, _client,
            CHANGE_ADMIN_KEY_WITHOUT_DELAY);
150 
151         //clear any existing proposal proposed by backup, proposer is one of the
            backups
152         for (uint256 i = 0; i <= MAX_DEFINED_BACKUP_INDEX; i++) {
153             address backup = accountStorage.getBackupAddress(_client, i);
154             uint256 effectiveDate = accountStorage.getBackupEffectiveDate(_client, i);
155             uint256 expiryDate = accountStorage.getBackupExpiryDate(_client, i);
156             if (backup != address(0) && isEffectiveBackup(effectiveDate, expiryDate)) {
157                 accountStorage.clearProposalData(_client, backup,
                        CHANGE_ADMIN_KEY_BY_BACKUP);
158             }
159         }
160     }
161 
162 }
```

File logics/base/BaseLogic.sol

```
1  pragma solidity ^0.5.4;
2 
3  import "../../Account.sol";
4  import "../../AccountStorage.sol";
5  import "../../utils/SafeMath.sol";
6 
7  contract BaseLogic {
8 
9      bytes constant internal SIGN_HASH_PREFIX = "\x19Ethereum Signed Message:\n32";
10 
11     mapping (address => uint256) keyNonce;
12     AccountStorage public accountStorage;
13 
14     modifier allowSelfCallsOnly() {
15         require (msg.sender == address(this), "only internal call is allowed");
16         _;
```

```
17      }
18
19      modifier allowAccountCallsOnly(Account _account) {
20          require(msg.sender == address(_account), "caller must be account");
21          _;
22      }
23
24      event LogicInitialised(address wallet);
25
26      // *************** Constructor ********************** //
27
28      constructor(AccountStorage _accountStorage) public {
29          accountStorage = _accountStorage;
30      }
31
32      // *************** Initialization ******************** //
33
34      function initAccount(Account _account) external allowAccountCallsOnly(_account){
35          emit LogicInitialised(address(_account));
36      }
37
38      // *************** Getter ********************** //
39
40      function getKeyNonce(address _key) external view returns(uint256) {
41          return keyNonce[_key];
42      }
43
44      // *************** Signature ********************** //
45
46      function getSignHash(bytes memory _data, uint256 _nonce) internal view returns(
            bytes32) {
47          // use EIP 191
48          // 0x1900 + this logic address + data + nonce of signing key
49          bytes32 msgHash = keccak256(abi.encodePacked(byte(0x19), byte(0), address(this)
                , _data, _nonce));
50          bytes32 prefixedHash = keccak256(abi.encodePacked(SIGN_HASH_PREFIX, msgHash));
51          return prefixedHash;
52      }
53
54      function verifySig(address _signingKey, bytes memory _signature, bytes32 _signHash
            ) internal pure {
55          address recoveredAddr = recover(_signHash, _signature);
56          require(recoveredAddr == _signingKey, "signature verification failed");
57      }
58
59      /**
60       * @dev Returns the address that signed a hashed message (`hash`) with
61       * `signature`. This address can then be used for verification purposes.
62       *
63       * The `ecrecover` EVM opcode allows for malleable (non-unique) signatures:
64       * this function rejects them by requiring the `s` value to be in the lower
65       * half order, and the `v` value to be either 27 or 28.
66       *
67       * NOTE: This call _does not revert_ if the signature is invalid, or
68       * if the signer is otherwise unable to be retrieved. In those scenarios,
69       * the zero address is returned.
70       *
71       * IMPORTANT: `hash` _must_ be the result of a hash operation for the
```

```
72      * verification to be secure: it is possible to craft signatures that
73      * recover to arbitrary addresses for non-hashed data. A safe way to ensure
74      * this is by receiving a hash of the original message (which may otherwise)
75      * be too long), and then calling {toEthSignedMessageHash} on it.
76      */
77     function recover(bytes32 hash, bytes memory signature) internal pure returns (
            address) {
78         // Check the signature length
79         if (signature.length != 65) {
80             return (address(0));
81         }
82
83         // Divide the signature in r, s and v variables
84         bytes32 r;
85         bytes32 s;
86         uint8 v;
87
88         // ecrecover takes the signature parameters, and the only way to get them
89         // currently is to use assembly.
90         // solhint-disable-next-line no-inline-assembly
91         assembly {
92             r := mload(add(signature, 0x20))
93             s := mload(add(signature, 0x40))
94             v := byte(0, mload(add(signature, 0x60)))
95         }
96
97         // EIP-2 still allows signature malleability for ecrecover(). Remove this
                possibility and make the signature
98         // unique. Appendix F in the Ethereum Yellow paper (https://ethereum.github.io/
                yellowpaper/paper.pdf), defines
99         // the valid range for s in (281): 0 < s < secp256k1n / 2 + 1, and for v in
                (282): v \in {27, 28}. Most
100        // signatures from current libraries generate a unique signature with an s-
                value in the lower half order.
101        //
102        // If your library generates malleable signatures, such as s-values in the
                upper range, calculate a new s-value
103        // with 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141 - s1
                 and flip v from 27 to 28 or
104        // vice versa. If your library also generates signatures with 0/1 for v instead
                 27/28, add 27 to v to accept
105        // these malleable signatures as well.
106        if (uint256(s) > 0
               x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
107            return address(0);
108        }
109
110        if (v != 27 && v != 28) {
111            return address(0);
112        }
113
114        // If the signature is valid (and not malleable), return the signer address
115        return ecrecover(hash, v, r, s);
116    }
117
118    /* get signer address from data
119    * @dev Gets an address encoded as the first argument in transaction data
120    * @param b The byte array that should have an address as first argument
```

```
121     * @returns a The address retrieved from the array
122     */
123     function getSignerAddress(bytes memory _b) internal pure returns (address _a) {
124         require(_b.length >= 36, "invalid bytes");
125         // solium-disable-next-line security/no-inline-assembly
126         assembly {
127             let mask := 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
128             _a := and(mask, mload(add(_b, 36)))
129             // b = {length:32}{method sig:4}{address:32}{...}
130             // 36 is the offset of the first parameter of the data, if encoded properly
                .
131             // 32 bytes for the length of the bytes array, and the first 4 bytes for
                the function signature.
132             // 32 bytes is the length of the bytes array!!!!
133         }
134     }
135
136     // get method id, first 4 bytes of data
137     function getMethodId(bytes memory _b) internal pure returns (bytes4 _a) {
138         require(_b.length >= 4, "invalid data");
139         // solium-disable-next-line security/no-inline-assembly
140         assembly {
141             // 32 bytes is the length of the bytes array
142             _a := mload(add(_b, 32))
143         }
144     }
145
146     // _nonce is timestamp in microsecond(1/1000000 second)
147     function checkAndUpdateNonce(address _account, uint256 _nonce, uint256 _index)
            internal {
148         // check operation key status
149         if (_index > 0) {
150             require(accountStorage.getKeyStatus(_account, _index) != 1, "frozen key");
151         }
152         address key = accountStorage.getKeyData(_account, _index);
153         require(_nonce > keyNonce[key], "nonce too small");
154         require(SafeMath.div(_nonce, 1000000) <= now + 86400, "nonce too big"); //
              86400=24*3600 seconds
155
156         keyNonce[key] = _nonce;
157     }
158 }
```

File testUtils/MyToken.sol

```
1  pragma solidity ^0.5.0;
2
3  // import "openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol";
4  import "openzeppelin-solidity/contracts/token/ERC20/ERC20Mintable.sol";
5
6  contract MyToken is ERC20Mintable {
7    string private _name;
8      string private _symbol;
9      uint8 private _decimals;
10   uint256 public val;
11
12   constructor(string memory name, string memory symbol, uint8 decimals/*, address
         account, uint256 amount*/) public {
13         _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 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);
```

```
19      event Invoked(address indexed module, address indexed target, uint indexed value
            , bytes data);
20      event Received(uint indexed value, address indexed sender, bytes data);
21
22      event AccountInit(address indexed account);
23
24      modifier allowAuthorizedLogicContractsCallsOnly {
25          require(LogicManager(manager).isAuthorized(msg.sender), "not an authorized
                logic");
26          _;
27      }
28
29      function init(address _manager, address _accountStorage, address[] calldata
            _logics, address[] calldata _keys, address[] calldata _backups)
30          external
31      {
32          require(manager == address(0), "Account: account already initialized");
33          require(_manager != address(0) && _accountStorage != address(0), "Account:
                address is null");
34          manager = _manager;
35
36          for (uint i = 0; i < _logics.length; i++) {
37              address logic = _logics[i];
38              require(LogicManager(manager).isAuthorized(logic), "must be authorized
                    logic");
39
40              BaseLogic(logic).initAccount(this);
41          }
42
43          AccountStorage(_accountStorage).initAccount(this, _keys, _backups);
44
45          emit AccountInit(address(this));
46      }
47
48      function invoke(address _target, uint _value, bytes calldata _data)
49          external
50          allowAuthorizedLogicContractsCallsOnly
51      {
52          // solium-disable-next-line security/no-call-value
53          //(bool success,_) = _target.call.value(_value)(_data);
54          //require(success, "call to target failed");
55          emit Invoked(msg.sender, _target, _value, _data);
56      }
57
58      /**
59      * @dev Enables a static method by specifying the target module to which the call
                must be delegated.
60      * @param _module The target module.
61      * @param _method The static method signature.
62      */
63      function enableStaticCall(address _module, bytes4 _method) external
            allowAuthorizedLogicContractsCallsOnly {
64          enabled[_method] = _module;
65          emit EnabledStaticCall(_module, _method);
66      }
67
68       /**
69       * @dev This method makes it possible for the wallet to comply to interfaces
```

```
         expecting the wallet to
70     * implement specific static methods. It delegates the static call to a target
         contract if the data corresponds
71     * to an enabled method, or logs the call otherwise.
72     */
73    function() external payable {
74        if(msg.data.length > 0) {
75            address logic = enabled[msg.sig];
76            if(logic == address(0)) {
77                emit Received(msg.value, msg.sender, msg.data);
78            }
79            else {
80                require(LogicManager(manager).isAuthorized(logic), "must be an
                      authorized logic for static call");
81                // solium-disable-next-line security/no-inline-assembly
82                assembly {
83                    calldatacopy(0, 0, calldatasize())
84                    let result := staticcall(gas, logic, 0, calldatasize(), 0, 0)
85                    returndatacopy(0, 0, returndatasize())
86                    switch result
87                    case 0 {revert(0, returndatasize())}
88                    default {return (0, returndatasize())}
89                }
90            }
91        }
92    }
93 }
```

File AccountCreator.sol

```
1  pragma solidity ^0.5.4;
2
3  import "./utils/MultiOwned.sol";
4  import "./Account.sol";
5  import "./AccountProxy.sol";
6
7  contract AccountCreator is MultiOwned {
8
9      address public logicManager;
10     address public accountStorage;
11     address public accountImpl;
12     address[] public logics;
13
14     // *************** Events *************************** //
15     event AccountCreated(address indexed wallet, address[] keys, address[] backups);
16     event Closed(address indexed sender);
17
18     // *************** Constructor ********************** //
19     constructor(address _mgr, address _storage, address _accountImpl, address[]
           memory _logics) public {
20         logicManager = _mgr;
21         accountStorage = _storage;
22         accountImpl = _accountImpl;
23         logics = _logics;
24     }
25
26     // *************** External Functions ********************* //
27
28     function createAccount(address[] calldata _keys, address[] calldata _backups)
```

```
        external onlyMultiOwners {
29          AccountProxy accountProxy = new AccountProxy(accountImpl);
30          Account(address(accountProxy)).init(logicManager, accountStorage, logics,
                _keys, _backups);
31
32          emit AccountCreated(address(accountProxy), _keys, _backups);
33      }
34
35      // *************** Suicide ********************** //
36
37      function close() external onlyMultiOwners {
38          selfdestruct(msg.sender);
39          emit Closed(msg.sender);
40      }
41  }
```

File AccountProxy.sol

```
1   pragma solidity ^0.5.4;
2
3   contract AccountProxy {
4
5       address implementation;
6
7       event Received(uint indexed value, address indexed sender, bytes data);
8
9       constructor(address _implementation) public {
10          implementation = _implementation;
11      }
12
13      function() external payable {
14
15          if(msg.data.length == 0 && msg.value > 0) {
16              emit Received(msg.value, msg.sender, msg.data);
17          }
18          else {
19              /// solium-disable-next-line security/no-inline-assembly
20              assembly {
21                  let target := sload(0)
22                  calldatacopy(0, 0, calldatasize())
23                  let result := delegatecall(gas, target, 0, calldatasize(), 0, 0)
24                  returndatacopy(0, 0, returndatasize())
25                  switch result
26                  case 0 {revert(0, returndatasize())}
27                  default {return (0, returndatasize())}
28              }
29          }
30      }
31  }
```

File AccountStorage.sol

```
1   pragma solidity ^0.5.4;
2
3   import "./Account.sol";
4   import "./LogicManager.sol";
5
6
7   contract AccountStorage {
8
```

```
9       modifier allowAccountCallsOnly(Account _account) {
10          require(msg.sender == address(_account), "caller must be account");
11          _;
12      }
13
14      modifier allowAuthorizedLogicContractsCallsOnly(address payable _account) {
15          require(LogicManager(Account(_account).manager()).isAuthorized(msg.sender), "
                not an authorized logic");
16          _;
17      }
18
19      struct KeyItem {
20          address pubKey;
21          uint256 status;
22      }
23
24      struct BackupAccount {
25          address backup;
26          uint256 effectiveDate;//means not effective until this timestamp
27          uint256 expiryDate;//means effective until this timestamp
28      }
29
30      struct DelayItem {
31          bytes32 hash;
32          uint256 dueTime;
33      }
34
35      struct Proposal {
36          bytes32 hash;
37          address[] approval;
38      }
39
40      // account => quantity of operation keys (index >= 1)
41      mapping (address => uint256) operationKeyCount;
42
43      // account => index => KeyItem
44      mapping (address => mapping(uint256 => KeyItem)) keyData;
45
46      // account => index => backup account
47      mapping (address => mapping(uint256 => BackupAccount)) backupData;
48
49      /* account => actionId => DelayItem
50
51         delayData applies to these 4 actions:
52         changeAdminKey, changeAllOperationKeys, unfreeze, changeAdminKeyByBackup
53      */
54      mapping (address => mapping(bytes4 => DelayItem)) delayData;
55
56      // client account => proposer account => proposed actionId => Proposal
57      mapping (address => mapping(address => mapping(bytes4 => Proposal)))
            proposalData;
58
59      // *************** keyCount ********************** //
60
61      function getOperationKeyCount(address _account) external view returns(uint256) {
62          return operationKeyCount[_account];
63      }
64
```

```
65      function increaseKeyCount(address payable _account) external
            allowAuthorizedLogicContractsCallsOnly(_account) {
66          operationKeyCount[_account] = operationKeyCount[_account] + 1;
67      }
68
69      // *************** keyData ********************** //
70
71      function getKeyData(address _account, uint256 _index) public view returns(
            address) {
72          KeyItem memory item = keyData[_account][_index];
73          return item.pubKey;
74      }
75
76      function setKeyData(address payable _account, uint256 _index, address _key)
            external allowAuthorizedLogicContractsCallsOnly(_account) {
77          require(_key != address(0), "invalid _key value");
78          KeyItem storage item = keyData[_account][_index];
79          item.pubKey = _key;
80      }
81
82      // *************** keyStatus ********************** //
83
84      function getKeyStatus(address _account, uint256 _index) external view returns(
            uint256) {
85          KeyItem memory item = keyData[_account][_index];
86          return item.status;
87      }
88
89      function setKeyStatus(address payable _account, uint256 _index, uint256 _status)
             external allowAuthorizedLogicContractsCallsOnly(_account) {
90          KeyItem storage item = keyData[_account][_index];
91          item.status = _status;
92      }
93
94      // *************** backupData ********************** //
95
96      function getBackupAddress(address _account, uint256 _index) external view
            returns(address) {
97          BackupAccount memory b = backupData[_account][_index];
98          return b.backup;
99      }
100
101     function getBackupEffectiveDate(address _account, uint256 _index) external view
            returns(uint256) {
102         BackupAccount memory b = backupData[_account][_index];
103         return b.effectiveDate;
104     }
105
106     function getBackupExpiryDate(address _account, uint256 _index) external view
            returns(uint256) {
107         BackupAccount memory b = backupData[_account][_index];
108         return b.expiryDate;
109     }
110
111     function setBackup(address payable _account, uint256 _index, address _backup,
            uint256 _effective, uint256 _expiry)
112         external
113         allowAuthorizedLogicContractsCallsOnly(_account)
```

```
114         {
115             BackupAccount storage b = backupData[_account][_index];
116             b.backup = _backup;
117             b.effectiveDate = _effective;
118             b.expiryDate = _expiry;
119         }
120 
121         function setBackupExpiryDate(address payable _account, uint256 _index, uint256
                _expiry)
122             external
123             allowAuthorizedLogicContractsCallsOnly(_account)
124         {
125             BackupAccount storage b = backupData[_account][_index];
126             b.expiryDate = _expiry;
127         }
128 
129         function clearBackupData(address payable _account, uint256 _index) external
                allowAuthorizedLogicContractsCallsOnly(_account) {
130             delete backupData[_account][_index];
131         }
132 
133         // *************** delayData ********************** //
134 
135         function getDelayDataHash(address payable _account, bytes4 _actionId) external
                view returns(bytes32) {
136             DelayItem memory item = delayData[_account][_actionId];
137             return item.hash;
138         }
139 
140         function getDelayDataDueTime(address payable _account, bytes4 _actionId)
                external view returns(uint256) {
141             DelayItem memory item = delayData[_account][_actionId];
142             return item.dueTime;
143         }
144 
145         function setDelayData(address payable _account, bytes4 _actionId, bytes32 _hash,
                 uint256 _dueTime) external allowAuthorizedLogicContractsCallsOnly(_account)
                {
146             DelayItem storage item = delayData[_account][_actionId];
147             item.hash = _hash;
148             item.dueTime = _dueTime;
149         }
150 
151         function clearDelayData(address payable _account, bytes4 _actionId) external
                allowAuthorizedLogicContractsCallsOnly(_account) {
152             delete delayData[_account][_actionId];
153         }
154 
155         // *************** proposalData ********************** //
156 
157         function getProposalDataHash(address _client, address _proposer, bytes4
                _actionId) external view returns(bytes32) {
158             Proposal memory p = proposalData[_client][_proposer][_actionId];
159             return p.hash;
160         }
161 
162         function getProposalDataApproval(address _client, address _proposer, bytes4
                _actionId) external view returns(address[] memory) {
```

```
163         Proposal memory p = proposalData[_client][_proposer][_actionId];
164         return p.approval;
165     }
166 
167     function setProposalData(address payable _client, address _proposer, bytes4
        _actionId, bytes32 _hash, address _approvedBackup)
168         external
169         allowAuthorizedLogicContractsCallsOnly(_client)
170     {
171         Proposal storage p = proposalData[_client][_proposer][_actionId];
172         if (p.hash > 0) {
173             if (p.hash == _hash) {
174                 for (uint256 i = 0; i < p.approval.length; i++) {
175                     require(p.approval[i] != _approvedBackup, "backup already exists")
                            ;
176                 }
177                 p.approval.push(_approvedBackup);
178             } else {
179                 p.hash = _hash;
180                 p.approval.length = 0;
181             }
182         } else {
183             p.hash = _hash;
184             p.approval.push(_approvedBackup);
185         }
186     }
187 
188     function clearProposalData(address payable _client, address _proposer, bytes4
        _actionId) external allowAuthorizedLogicContractsCallsOnly(_client) {
189         delete proposalData[_client][_proposer][_actionId];
190     }
191 
192 
193     // *************** init ********************** //
194     function initAccount(Account _account, address[] calldata _keys, address[]
        calldata _backups)
195         external
196         allowAccountCallsOnly(_account)
197     {
198         require(getKeyData(address(_account), 0) == address(0), "AccountStorage:
            account already initialized!");
199         require(_keys.length > 0, "empty keys array");
200 
201         operationKeyCount[address(_account)] = _keys.length - 1;
202 
203         for (uint256 index = 0; index < _keys.length; index++) {
204             address _key = _keys[index];
205             require(_key != address(0), "_key cannot be 0x0");
206             KeyItem storage item = keyData[address(_account)][index];
207             item.pubKey = _key;
208             item.status = 0;
209         }
210 
211         // avoid backup duplication if _backups.length > 1
212         // normally won't check duplication, in most cases only one initial backup
            when initialization
213         if (_backups.length > 1) {
214             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) {
```

```
40          return authorized[_logic];
41      }
42
43      function submitUpdate(address _logic, bool _value) external onlyOwner {
44          pending storage p = pendingLogics[_logic];
45          p.value = _value;
46          p.dueTime = now + pendingTime;
47          emit UpdateLogicSubmitted(_logic, _value);
48      }
49
50      function updateLogic(address _logic, bool _value) internal {
51          if (authorized[_logic] != _value) {
52              if(_value) {
53                  logicCount += 1;
54                  authorized[_logic] = true;
55              }
56              else {
57                  logicCount -= 1;
58                  require(logicCount > 0, "must have at least one logic module");
59                  delete authorized[_logic];
60              }
61              emit UpdateLogicDone(_logic, _value);
62          }
63      }
64
65      function triggerUpdateLogic(address _logic) external {
66          pending memory p = pendingLogics[_logic];
67          require(p.dueTime > 0, "pending logic not found");
68          require(p.dueTime <= now, "too early to trigger updateLogic");
69          updateLogic(_logic, p.value);
70          delete pendingLogics[_logic];
71      }
72  }
```

CERTIK

Building Fully Trustworthy Smart Contract and Blockchain Ecosystems