CertiK Audit Report For Akropolis

Request Date: 2019-07-23
Revision Date: 2019-08-02
Platform Name: Ethereum

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 Akropolis(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 1.4B in assets.

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

Exective Summary

This report has been prepared as the product of the Smart Contract Audit request by Akropolis. This audit was conducted to discover issues and vulnerabilities in the source code of Akropolis'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

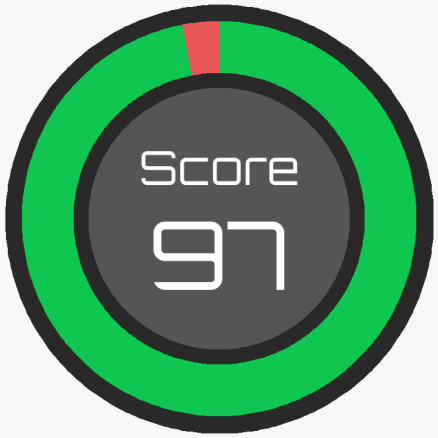
The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilies, but no concern found yet.

Testing Summary

PASS

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

Aug 02, 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.	0	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.	0	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

Source Code SHA-256 Checksum

v2.3 **commit** 410b2bd20b7523f57c1d89942afcdec08f948928

- **TokenTimelock.sol**
  `76757babeed18b0c545e6af3e429bea29abd5ddbda78550013e28c474b664735`
- **TokenVesting.sol**
  `f73ca425ab964e3f63b5aece04e6c70e6fa3c042d4be38cb6a7feea35d581927`
- **AkropolisTimeLock.sol**
  `184e66582567fae45a59adbae53b5fd5672f430dfafba628ddec801926c92813`
- **AkropolisTokenVesting.sol**
  `7b3ec21d952c4fae9cf2fc209dc17c160ef3c4bcfef080016efa639d5e1b80f9`
- **BeneficiaryOperations.sol**
  `ebc95fa9079e679d9b2ac7644be35a4e05929a7d0d62496b8dfbe0c7a3025819`
- **Migrations.sol**
  `1c4e30fd3aa765cb0ee259a29dead71c1c99888dcc7157c25df3405802cf5b09`

Summary

CertiK was chosen by Akropolis to audit the design and implementation of its soon to be released upgradeable time lock and vesting smart contracts. 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.

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.

Discussions

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes. Entries are labeled CRITICAL, MAJOR, MINOR, INFO, DISCUSSION (in decreasing significance order).

v2.3 **commit** 410b2bd20b7523f57c1d89942afcdec08f948928

BeneficiaryOperations:

- INFO `insideCallCount`: Can be defined as `uint8`.

v2.2 **commit** 8322937a511a5530f5213b734143b8c29fd83b9f

BeneficiaryOperations:

- MAJOR `transferBeneficiaryShipWithHowMany()`: Is there assumption that beneficiaries won't inject invalid operations maliciously? Currently it is easy for beneficiaries to perform gas limit DDoS attack by injecting invalid operations and utilize the deletion of `allOperations` in `transferBenficiaryShipWithHowMany()`. E.g. A beneficiary can call these guarded functions with different parameters and easily fill up the `allOperations` to make `transferBeneficiaryShipWithHowMany()` un-runnable. If there is no such prevention mechanism then recommend setting a total operations limit for each beneficiary.
  - (Akropolis) Resolved in `v2.3`.
- MINOR `deleteOperation()`: Recommend using `SafeMath` for the two usage of `allOperations.length - 1` as well.
  - (Akropolis) Resolved in `v2.3`.

v2.1 **commit** 06b3e760ae75d5f532622a06ce7fe6ff5b097414

BeneficiaryOperations:

- INFO `checkHowManyBeneficiaries(), cancelPending()`: Recommend moving the beneficiary check `require(beneficiaryIndex >= 0...)` to above the `uint beneficiaryIndex` declaration for better error reporting.
  - (Akropolis) Resolved in `v2.2`.
- MAJOR `transferBeneficiaryShipWithHowMany()`: The size requirement `require(newBeneficiaries.length <= 256)` needs to be updated to `<= 255` or `< 256`, otherwise new operation at index `0` might be overwritten.
  - (Akropolis) Resolved in `v2.2`.
- MAJOR `transferBeneficiaryShipWithHowMany()`: `votesMaskByOperation` and `votesCountByOperation` need to be cleared together with `allOperationsIndicies`. Otherwise previous cache might be used in `checkHowManyBeneficiaries()`, leading to unexpected operation handling.
  - (Akropolis) Resolved in `v2.2`.
- MINOR `deleteOperation()`: Recommend using `SafeMath` for `allOperations.length--`.
  - (Akropolis) Resolved in `v2.2`.

AkropolisTimeLock, AkropolisVesting:

- MAJOR `transferBeneficiaryShipWithHowMany()`: Call to the function will fail upon input array of size `1` because the use of `beneficiaries[1]`. The argument `beneficiaries[1]` should be `beneficiaries[0]`.
  - (Akropolis) Resolved in `v2.2`.
- MAJOR `transferBeneficiaryShipWithHowMany()`: Upon called with an array size ≥ 1 and `_newHowManyBeneficiariesDecide` ≥ 1, the function will revert with "checkHowManyBeneficiaries: nested beneficiaries modifier check require more beneficiarys"

  The function call fails at `changeBeneficiary(beneficiaries[1])`, due to the check `require(howMany <= insideCallCount)` to be specific. The function is guarded by the initial `howManyBeneficiariesDecide` instead of the new `_newHowManyBeneficiariesDecide`. Therefore the new `_newHowManyBeneficiariesDecide` cannot be larger than the initial `howManyBeneficiariesDecide` in the current implementation. Please see the corresponding entry in `helpers/BeneficiaryOperations.sol` above.

  If `super` is to be called, the `onlyManyBeneficiaries` modifier for the overriding `transferBeneficiaryShipWithHowMany()` in `AkropolisVesting` can be removed.
  - (Akropolis) Resolved in `v2.2`.

v2 **commit** $7f4f4543b08d3749b92839c85e1d77a33d917a37$

BeneficiaryOperations:

- MINOR `checkHowManyBeneficiaries()`, `cancelPending()`: Recommend using `SafeMath` for `beneficiaryIndex` and `operationVotesCount`.
  - (Akropolis) Resolved in `v2.1`.
- INFO `beneficiariesIndices`: If the size `255` and `256` does not make much difference in the Akropolis's voting system, recommend changing `uint` to `uint8` to impose a better restriction on the beneficiaries size.
  - (Akropolis) Resolved in `v2.1`.
- MAJOR `transferBeneficiaryShipWithHowMany()`: It's possible to remove a new operation after `transferBeneficiaryShipWithHowMany()` by utilizing the old `allOperationsIndicies` cache. Recommend clearing `allOperationsIndicies`.
  - (Akropolis) Resolved in `v2.1`.

AkropolisTimeLock:

- MAJOR `changeBeneficiary()`: Recursive call. The `super` should be invoked.

  - (Akropolis) Resolved in `v2.1`.

- INFO `changeBeneficiary()`: Recommend using the pull model.
  - (Akropolis) Resolved in `v2.1`.

AkropolisVesting:

- MAJOR `changeBeneficiary()`: Recursive call. No corresponding method exists in `super` as well.
  - (Akropolis) Resolved in `v2.1`.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File AkropolisTokenVesting.sol

```
1 pragma solidity ^0.5.9;
```

Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File AkropolisTimeLock.sol

```
1 pragma solidity ^0.5.9;
```

Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 7 in File BeneficiaryOperations.sol

```
7 pragma solidity ^0.5.9;
```

Only these compiler versions are safe to compile your code: 0.5.9

INSECURE_COMPILER_VERSION

Line 1 in File TokenTimelock.sol

```
1 pragma solidity ^0.5.9;
```

Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 56 in File TokenTimelock.sol

```
56        require(block.timestamp >= _releaseTime, "TokenTimelock: current time is before
              release time");
```

”block.timestamp” can be influenced by minors to some degree

INSECURE_COMPILER_VERSION

Line 1 in File TokenVesting.sol

```
1 pragma solidity ^0.5.9;
```

Only these compiler versions are safe to compile your code: 0.5.9

TIMESTAMP_DEPENDENCY

Line 166 in File TokenVesting.sol

```
166       if (block.timestamp < _cliff) {
```

”block.timestamp” can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 168 in File TokenVesting.sol

```
168         } else if (block.timestamp >= _start.add(_duration) || _revoked[address(token)
                ]) {
```

"block.timestamp" can be influenced by minors to some degree

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
```

Formal Verification Request 1

AkropolisTokenVesting_transferBeneficiaryShip

02, Aug 2019
528.04 ms

Line 67-70 in File AkropolisTokenVesting.sol

```
67    /*@CTK AkropolisTokenVesting_transferBeneficiaryShip
68      @tag assume_completion
69      @post __post._pendingBeneficiary == beneficiaries[0]
70     */
```

Line 71-74 in File AkropolisTokenVesting.sol

```
71    function transferBeneficiaryShipWithHowMany(address[] memory _newBeneficiaries,
          uint256 _newHowManyBeneficiariesDecide) public {
72        super.transferBeneficiaryShipWithHowMany(_newBeneficiaries,
              _newHowManyBeneficiariesDecide);
73        _setPendingBeneficiary(beneficiaries[0]);
74    }
```

The code meets the specification.

Formal Verification Request 2

AkropolisTokenVesting_changeBeneficiary

02, Aug 2019
212.5 ms

Line 80-84 in File AkropolisTokenVesting.sol

```
80    /*@CTK AkropolisTokenVesting_changeBeneficiary
81      @tag assume_completion
82      @post __post.insideCallSender == insideCallSender
83      @pre __post.insideCallCount <= __post.howManyBeneficiariesDecide
84    */
```

Line 85-87 in File AkropolisTokenVesting.sol

```
85    function changeBeneficiary(address _newBeneficiary) public onlyManyBeneficiaries {
86        _setPendingBeneficiary(_newBeneficiary);
87    }
```

The code meets the specification.

Formal Verification Request 3

If method completes, integer overflow would not happen.

02, Aug 2019
44.09 ms

Line 92 in File AkropolisTokenVesting.sol

```
92    //@CTK NO_OVERFLOW
```

Line 101-105 in File AkropolisTokenVesting.sol

```
101   function claimBeneficiary() public onlyPendingBeneficiary {
102       _changeBeneficiary(_pendingBeneficiary);
103       emit LogBeneficiaryTransfered(_pendingBeneficiary);
104       _pendingBeneficiary = address(0);
105   }
```

✅ The code meets the specification.

Formal Verification Request 4

Buffer overflow / array index out of bound would never happen.

📅 02, Aug 2019
⏱ 0.57 ms

Line 93 in File AkropolisTokenVesting.sol

```
93    //@CTK NO_BUF_OVERFLOW
```

Line 101-105 in File AkropolisTokenVesting.sol

```
101   function claimBeneficiary() public onlyPendingBeneficiary {
102       _changeBeneficiary(_pendingBeneficiary);
103       emit LogBeneficiaryTransfered(_pendingBeneficiary);
104       _pendingBeneficiary = address(0);
105   }
```

✅ The code meets the specification.

Formal Verification Request 5

Method will not encounter an assertion failure.

📅 02, Aug 2019
⏱ 0.59 ms

Line 94 in File AkropolisTokenVesting.sol

```
94    //@CTK NO_ASF
```

Line 101-105 in File AkropolisTokenVesting.sol

```
101   function claimBeneficiary() public onlyPendingBeneficiary {
102       _changeBeneficiary(_pendingBeneficiary);
103       emit LogBeneficiaryTransfered(_pendingBeneficiary);
104       _pendingBeneficiary = address(0);
105   }
```

✅ The code meets the specification.

Formal Verification Request 6

Vesting_claimBeneficiary

📅 02, Aug 2019
⏱ 4.53 ms

Line 95-100 in File AkropolisTokenVesting.sol

```
95    /*@CTK Vesting_claimBeneficiary
96      @tag assume_completion
97      @post (msg.sender) == _pendingBeneficiary
98      @post __post._beneficiary == (msg.sender)
99      @post __post._pendingBeneficiary == address(0)
100    */
```

Line 101-105 in File AkropolisTokenVesting.sol

```
101   function claimBeneficiary() public onlyPendingBeneficiary {
102       _changeBeneficiary(_pendingBeneficiary);
103       emit LogBeneficiaryTransfered(_pendingBeneficiary);
104       _pendingBeneficiary = address(0);
105   }
```

✅ The code meets the specification.

Formal Verification Request 7

If method completes, integer overflow would not happen.

📅 02, Aug 2019
⏱ 0.56 ms

Line 115 in File AkropolisTokenVesting.sol

```
115   //@CTK NO_OVERFLOW
```

Line 124-127 in File AkropolisTokenVesting.sol

```
124   function _setPendingBeneficiary(address _newBeneficiary) internal
          onlyExistingBeneficiary(_newBeneficiary) {
125       _pendingBeneficiary = _newBeneficiary;
126       emit LogBeneficiaryTransferProposed(_newBeneficiary);
127   }
```

✅ The code meets the specification.

Formal Verification Request 8

Buffer overflow / array index out of bound would never happen.

📅 02, Aug 2019
⏱ 0.52 ms

Line 116 in File AkropolisTokenVesting.sol

```
116   //@CTK NO_BUF_OVERFLOW
```

Line 124-127 in File AkropolisTokenVesting.sol

```
124     function _setPendingBeneficiary(address _newBeneficiary) internal
            onlyExistingBeneficiary(_newBeneficiary) {
125         _pendingBeneficiary = _newBeneficiary;
126         emit LogBeneficiaryTransferProposed(_newBeneficiary);
127     }
```

The code meets the specification.

Formal Verification Request 9

Method will not encounter an assertion failure.

02, Aug 2019
0.53 ms

Line 117 in File AkropolisTokenVesting.sol

```
117     //@CTK NO_ASF
```

Line 124-127 in File AkropolisTokenVesting.sol

```
124     function _setPendingBeneficiary(address _newBeneficiary) internal
            onlyExistingBeneficiary(_newBeneficiary) {
125         _pendingBeneficiary = _newBeneficiary;
126         emit LogBeneficiaryTransferProposed(_newBeneficiary);
127     }
```

The code meets the specification.

Formal Verification Request 10

Vesting_claimBeneficiary

02, Aug 2019
1.14 ms

Line 118-123 in File AkropolisTokenVesting.sol

```
118     /*@CTK Vesting_claimBeneficiary
119       @tag assume_completion
120       @pre beneficiariesIndices[_beneficiary] > 0
121       @post __post._beneficiary == _beneficiary
122       @post __post._pendingBeneficiary == _newBeneficiary
123      */
```

Line 124-127 in File AkropolisTokenVesting.sol

```
124     function _setPendingBeneficiary(address _newBeneficiary) internal
            onlyExistingBeneficiary(_newBeneficiary) {
125         _pendingBeneficiary = _newBeneficiary;
126         emit LogBeneficiaryTransferProposed(_newBeneficiary);
127     }
```

The code meets the specification.

Formal Verification Request 11

AkropolisTimeLock_transferBeneficiaryShip

02, Aug 2019
1080.18 ms

Line 61-64 in File AkropolisTimeLock.sol

```
61      /*@CTK AkropolisTimeLock_transferBeneficiaryShip
62        @tag assume_completion
63        @post __post._pendingBeneficiary == beneficiaries[0]
64       */
```

Line 65-68 in File AkropolisTimeLock.sol

```
65      function transferBeneficiaryShip(address[] memory _newBeneficiaries) public {
66          super.transferBeneficiaryShip(_newBeneficiaries);
67          _setPendingBeneficiary(beneficiaries[0]);
68      }
```

The code meets the specification.

Formal Verification Request 12

AkropolisTimeLock_changeBeneficiary

02, Aug 2019
231.73 ms

Line 74-78 in File AkropolisTimeLock.sol

```
74      /*@CTK AkropolisTimeLock_changeBeneficiary
75        @tag assume_completion
76        @post __post.insideCallSender == insideCallSender
77        @pre __post.insideCallCount <= __post.howManyBeneficiariesDecide
78       */
```

Line 79-81 in File AkropolisTimeLock.sol

```
79      function changeBeneficiary(address _newBeneficiary) public
            onlyManyBeneficiaries {
80          _setPendingBeneficiary(_newBeneficiary);
81      }
```

The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

02, Aug 2019
39.23 ms

Line 86 in File AkropolisTimeLock.sol

```
86      //@CTK NO_OVERFLOW
```

Line 95-99 in File AkropolisTimeLock.sol

```
95      function claimBeneficiary() public onlyPendingBeneficiary {
96          _changeBeneficiary(_pendingBeneficiary);
97          emit LogBeneficiaryTransfered(_pendingBeneficiary);
98          _pendingBeneficiary = address(0);
99      }
```

The code meets the specification.

Formal Verification Request 14

Buffer overflow / array index out of bound would never happen.

02, Aug 2019
0.62 ms

Line 87 in File AkropolisTimeLock.sol

```
87      //@CTK NO_BUF_OVERFLOW
```

Line 95-99 in File AkropolisTimeLock.sol

```
95      function claimBeneficiary() public onlyPendingBeneficiary {
96          _changeBeneficiary(_pendingBeneficiary);
97          emit LogBeneficiaryTransfered(_pendingBeneficiary);
98          _pendingBeneficiary = address(0);
99      }
```

The code meets the specification.

Formal Verification Request 15

Method will not encounter an assertion failure.

02, Aug 2019
0.6 ms

Line 88 in File AkropolisTimeLock.sol

```
88      //@CTK NO_ASF
```

Line 95-99 in File AkropolisTimeLock.sol

```
95      function claimBeneficiary() public onlyPendingBeneficiary {
96          _changeBeneficiary(_pendingBeneficiary);
97          emit LogBeneficiaryTransfered(_pendingBeneficiary);
98          _pendingBeneficiary = address(0);
99      }
```

The code meets the specification.

Formal Verification Request 16

TimeLock_claimBeneficiary

📅 02, Aug 2019
⏱ 3.72 ms

Line 89-94 in File AkropolisTimeLock.sol

```
89        /*@CTK TimeLock_claimBeneficiary
90          @tag assume_completion
91          @post (msg.sender) == _pendingBeneficiary
92          @post __post._beneficiary == (msg.sender)
93          @post __post._pendingBeneficiary == address(0)
94         */
```

Line 95-99 in File AkropolisTimeLock.sol

```
95        function claimBeneficiary() public onlyPendingBeneficiary {
96            _changeBeneficiary(_pendingBeneficiary);
97            emit LogBeneficiaryTransfered(_pendingBeneficiary);
98            _pendingBeneficiary = address(0);
99        }
```

✅ The code meets the specification.

Formal Verification Request 17

If method completes, integer overflow would not happen.

📅 02, Aug 2019
⏱ 0.52 ms

Line 109 in File AkropolisTimeLock.sol

```
109       //@CTK NO_OVERFLOW
```

Line 118-121 in File AkropolisTimeLock.sol

```
118       function _setPendingBeneficiary(address _newBeneficiary) internal
              onlyExistingBeneficiary(_newBeneficiary) {
119           _pendingBeneficiary = _newBeneficiary;
120           emit LogBeneficiaryTransferProposed(_newBeneficiary);
121       }
```

✅ The code meets the specification.

Formal Verification Request 18

Buffer overflow / array index out of bound would never happen.

📅 02, Aug 2019
⏱ 0.5 ms

Line 110 in File AkropolisTimeLock.sol

```
110       //@CTK NO_BUF_OVERFLOW
```

Line 118-121 in File AkropolisTimeLock.sol

```
118         function _setPendingBeneficiary(address _newBeneficiary) internal
                onlyExistingBeneficiary(_newBeneficiary) {
119             _pendingBeneficiary = _newBeneficiary;
120             emit LogBeneficiaryTransferProposed(_newBeneficiary);
121         }
```

The code meets the specification.

Formal Verification Request 19

Method will not encounter an assertion failure.

02, Aug 2019
0.52 ms

Line 111 in File AkropolisTimeLock.sol

```
111         //@CTK NO_ASF
```

Line 118-121 in File AkropolisTimeLock.sol

```
118         function _setPendingBeneficiary(address _newBeneficiary) internal
                onlyExistingBeneficiary(_newBeneficiary) {
119             _pendingBeneficiary = _newBeneficiary;
120             emit LogBeneficiaryTransferProposed(_newBeneficiary);
121         }
```

The code meets the specification.

Formal Verification Request 20

TimeLock_claimBeneficiary

02, Aug 2019
1.22 ms

Line 112-117 in File AkropolisTimeLock.sol

```
112         /*@CTK TimeLock_claimBeneficiary
113           @tag assume_completion
114           @pre beneficiariesIndices[_beneficiary] > 0
115           @post __post._beneficiary == _beneficiary
116           @post __post._pendingBeneficiary == _newBeneficiary
117          */
```

Line 118-121 in File AkropolisTimeLock.sol

```
118         function _setPendingBeneficiary(address _newBeneficiary) internal
                onlyExistingBeneficiary(_newBeneficiary) {
119             _pendingBeneficiary = _newBeneficiary;
120             emit LogBeneficiaryTransferProposed(_newBeneficiary);
121         }
```

The code meets the specification.

Formal Verification Request 21

If method completes, integer overflow would not happen.

02, Aug 2019
4.62 ms

Line 49 in File BeneficiaryOperations.sol

```
49    //@CTK NO_OVERFLOW
```

Line 56-58 in File BeneficiaryOperations.sol

```
56    function isExistBeneficiary(address wallet) public view returns(bool) {
57        return beneficiariesIndices[wallet] > 0;
58    }
```

The code meets the specification.

Formal Verification Request 22

Buffer overflow / array index out of bound would never happen.

02, Aug 2019
0.3 ms

Line 50 in File BeneficiaryOperations.sol

```
50    //@CTK NO_BUF_OVERFLOW
```

Line 56-58 in File BeneficiaryOperations.sol

```
56    function isExistBeneficiary(address wallet) public view returns(bool) {
57        return beneficiariesIndices[wallet] > 0;
58    }
```

The code meets the specification.

Formal Verification Request 23

Method will not encounter an assertion failure.

02, Aug 2019
0.29 ms

Line 51 in File BeneficiaryOperations.sol

```
51    //@CTK NO_ASF
```

Line 56-58 in File BeneficiaryOperations.sol

```
56    function isExistBeneficiary(address wallet) public view returns(bool) {
57        return beneficiariesIndices[wallet] > 0;
58    }
```

The code meets the specification.

Formal Verification Request 24

isExistBeneficiary

02, Aug 2019
0.3 ms

Line 52-55 in File BeneficiaryOperations.sol

```
52    /*@CTK isExistBeneficiary
53      @tag assume_completion
54      @post __return == (beneficiariesIndices[wallet] > 0)
55     */
```

Line 56-58 in File BeneficiaryOperations.sol

```
56    function isExistBeneficiary(address wallet) public view returns(bool) {
57        return beneficiariesIndices[wallet] > 0;
58    }
```

The code meets the specification.

Formal Verification Request 25

If method completes, integer overflow would not happen.

02, Aug 2019
5.0 ms

Line 60 in File BeneficiaryOperations.sol

```
60    //@CTK NO_OVERFLOW
```

Line 67-69 in File BeneficiaryOperations.sol

```
67    function beneficiariesCount() public view returns(uint) {
68        return beneficiaries.length;
69    }
```

The code meets the specification.

Formal Verification Request 26

Buffer overflow / array index out of bound would never happen.

02, Aug 2019
0.31 ms

Line 61 in File BeneficiaryOperations.sol

```
61    //@CTK NO_BUF_OVERFLOW
```

Line 67-69 in File BeneficiaryOperations.sol

```
67    function beneficiariesCount() public view returns(uint) {
68        return beneficiaries.length;
69    }
```

The code meets the specification.

Formal Verification Request 27

Method will not encounter an assertion failure.

📅 02, Aug 2019
⏱ 0.31 ms

Line 62 in File BeneficiaryOperations.sol

```
62    //@CTK NO_ASF
```

Line 67-69 in File BeneficiaryOperations.sol

```
67    function beneficiariesCount() public view returns(uint) {
68        return beneficiaries.length;
69    }
```

✅ The code meets the specification.

Formal Verification Request 28

beneficiariesCount

📅 02, Aug 2019
⏱ 0.3 ms

Line 63-66 in File BeneficiaryOperations.sol

```
63    /*@CTK beneficiariesCount
64      @tag assume_completion
65      @post __return == beneficiaries.length
66     */
```

Line 67-69 in File BeneficiaryOperations.sol

```
67    function beneficiariesCount() public view returns(uint) {
68        return beneficiaries.length;
69    }
```

✅ The code meets the specification.

Formal Verification Request 29

If method completes, integer overflow would not happen.

📅 02, Aug 2019
⏱ 4.61 ms

Line 71 in File BeneficiaryOperations.sol

```
71    //@CTK NO_OVERFLOW
```

Line 78-80 in File BeneficiaryOperations.sol

```
78    function allOperationsCount() public view returns(uint) {
79        return allOperations.length;
80    }
```

✅ The code meets the specification.

Formal Verification Request 30

Buffer overflow / array index out of bound would never happen.

📅 02, Aug 2019
⏱ 0.33 ms

Line 72 in File BeneficiaryOperations.sol

```
72    //@CTK NO_BUF_OVERFLOW
```

Line 78-80 in File BeneficiaryOperations.sol

```
78    function allOperationsCount() public view returns(uint) {
79        return allOperations.length;
80    }
```

✅ The code meets the specification.

Formal Verification Request 31

Method will not encounter an assertion failure.

📅 02, Aug 2019
⏱ 0.32 ms

Line 73 in File BeneficiaryOperations.sol

```
73    //@CTK NO_ASF
```

Line 78-80 in File BeneficiaryOperations.sol

```
78    function allOperationsCount() public view returns(uint) {
79        return allOperations.length;
80    }
```

✅ The code meets the specification.

Formal Verification Request 32

allOperationsCount

📅 02, Aug 2019
⏱ 0.35 ms

Line 74-77 in File BeneficiaryOperations.sol

```
74    /*@CTK allOperationsCount
75      @tag assume_completion
76      @post __return == allOperations.length
77     */
```

Line 78-80 in File BeneficiaryOperations.sol

```
78    function allOperationsCount() public view returns(uint) {
79        return allOperations.length;
80    }
```

✅ The code meets the specification.

Formal Verification Request 33

_operationLimitByBeneficiaryIndex

02, Aug 2019
4.88 ms

Line 86-89 in File BeneficiaryOperations.sol

```
86    /*@CTK _operationLimitByBeneficiaryIndex
87      @tag assume_completion
88      @post __return == (operationsCountByBeneficiaryIndex[beneficiaryIndex] <= 3)
89     */
```

Line 90-92 in File BeneficiaryOperations.sol

```
90    function _operationLimitByBeneficiaryIndex(uint8 beneficiaryIndex) internal view
          returns(bool) {
91        return (operationsCountByBeneficiaryIndex[beneficiaryIndex] <= 3);
92    }
```

The code meets the specification.

Formal Verification Request 34

_cancelAllPending

02, Aug 2019
27.21 ms

Line 94-97 in File BeneficiaryOperations.sol

```
94    /*@CTK _cancelAllPending
95      @tag assume_completion
96      @post __post.allOperations.length == 0
97     */
```

Line 98-127 in File BeneficiaryOperations.sol

```
98    function _cancelAllPending() internal {
99        /*@CTK loop_cancelAllPending_votes
100         @inv (i <= this.allOperations.length)
101         @post (i == this.allOperations.length)
102         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.allOperationsIndicies[this.
                allOperations[k]] == 0)
103         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesMaskByOperation[this.
                allOperations[k]] == 0)
104         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesCountByOperation[this.
                allOperations[k]] == 0)
105         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.operationsByBeneficiaryIndex[
                this.allOperations[k]] == 0)
106         @post !__should_return
107        */
108       for (uint i = 0; i < allOperations.length; i++) {
109           delete(allOperationsIndicies[allOperations[i]]);
110           delete(votesMaskByOperation[allOperations[i]]);
111           delete(votesCountByOperation[allOperations[i]]);
112           //delete operation->beneficiaryIndex
113           delete(operationsByBeneficiaryIndex[allOperations[i]]);
```

```
114         }
115
116         allOperations.length = 0;
117         //delete operations count for beneficiary
118         /*@CTK loop_cancelAllPending_operationsCount
119           @inv (j <= this.beneficiaries.length)
120           @post (j == this.beneficiaries.length)
121           @inv forall k: uint. (k >= 0 /\ k < j) -> (this.
                  operationsCountByBeneficiaryIndex[k] == 0)
122           @post !__should_return
123          */
124         for (uint8 j = 0; j < beneficiaries.length; j++) {
125             operationsCountByBeneficiaryIndex[j] = 0;
126         }
127     }
```

The code meets the specification.

Formal Verification Request 35

Buffer overflow / array index out of bound would never happen.

02, Aug 2019
17.35 ms

Line 209 in File BeneficiaryOperations.sol

```
209     //@CTK NO_BUF_OVERFLOW
```

Line 217-221 in File BeneficiaryOperations.sol

```
217     constructor() public {
218         beneficiaries.push(msg.sender);
219         beneficiariesIndices[msg.sender] = 1;
220         howManyBeneficiariesDecide = 1;
221     }
```

The code meets the specification.

Formal Verification Request 36

Method will not encounter an assertion failure.

02, Aug 2019
0.42 ms

Line 210 in File BeneficiaryOperations.sol

```
210     //@CTK NO_ASF
```

Line 217-221 in File BeneficiaryOperations.sol

```
217     constructor() public {
218         beneficiaries.push(msg.sender);
219         beneficiariesIndices[msg.sender] = 1;
220         howManyBeneficiariesDecide = 1;
221     }
```

The code meets the specification.

Formal Verification Request 37

BeneficiaryOperations

02, Aug 2019
2.46 ms

Line 211-216 in File BeneficiaryOperations.sol

```
211    /*@CTK BeneficiaryOperations
212      @tag assume_completion
213      @pre beneficiaries[0] == 0
214      @post __post.beneficiariesIndices[msg.sender] == 1
215      @post __post.howManyBeneficiariesDecide == 1
216     */
```

Line 217-221 in File BeneficiaryOperations.sol

```
217    constructor() public {
218        beneficiaries.push(msg.sender);
219        beneficiariesIndices[msg.sender] = 1;
220        howManyBeneficiariesDecide = 1;
221    }
```

The code meets the specification.

Formal Verification Request 38

checkHowManyBeneficiaries_nested

02, Aug 2019
30.5 ms

Line 228-233 in File BeneficiaryOperations.sol

```
228    /*@CTK checkHowManyBeneficiaries_nested
229      @tag assume_completion
230      @pre insideCallSender == msg.sender
231      @post howMany <= insideCallCount
232      @post __return == true
233     */
```

Line 238-282 in File BeneficiaryOperations.sol

```
238    function checkHowManyBeneficiaries(uint howMany) internal returns(bool) {
239        if (insideCallSender == msg.sender) {
240            require(howMany <= insideCallCount, "checkHowManyBeneficiaries: nested
                   beneficiaries modifier check require more beneficiarys");
241            return true;
242        }
243
244        require((isExistBeneficiary(msg.sender) && (beneficiariesIndices[msg.sender] <=
                    beneficiaries.length)), "checkHowManyBeneficiaries: msg.sender is not an
               beneficiary");
245
```

```
246         uint beneficiaryIndex = beneficiariesIndices[msg.sender].sub(1);
247
248         bytes32 operation = keccak256(abi.encodePacked(msg.data,
                beneficiariesGeneration));
249
250         require((votesMaskByOperation[operation] & (2 ** beneficiaryIndex)) == 0, "
                checkHowManyBeneficiaries: beneficiary already voted for the operation");
251         //check limit for operation
252         require(_operationLimitByBeneficiaryIndex(uint8(beneficiaryIndex)), "
                checkHowManyBeneficiaries: operation limit is reached for this beneficiary"
                );
253
254         votesMaskByOperation[operation] |= (2 ** beneficiaryIndex);
255         uint operationVotesCount = votesCountByOperation[operation].add(1);
256         votesCountByOperation[operation] = operationVotesCount;
257
258         if (operationVotesCount == 1) {
259             allOperationsIndicies[operation] = allOperations.length;
260
261             operationsByBeneficiaryIndex[operation] = uint8(beneficiaryIndex);
262
263             operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)] = uint8(
                    operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)].add(1));
264
265             allOperations.push(operation);
266
267
268             emit OperationCreated(operation, howMany, beneficiaries.length, msg.sender)
                    ;
269         }
270         emit OperationUpvoted(operation, operationVotesCount, howMany, beneficiaries.
                length, msg.sender);
271
272         // If enough beneficiaries confirmed the same operation
273         if (votesCountByOperation[operation] == howMany) {
274             deleteOperation(operation);
275             emit OperationPerformed(operation, howMany, beneficiaries.length, msg.
                    sender);
276             return true;
277         }
278         return false;
279     }
```

The code meets the specification.

Formal Verification Request 39

checkHowManyBeneficiaries_root

02, Aug 2019
0.33 ms

Line 234-237 in File BeneficiaryOperations.sol

```
234     /*@CTK checkHowManyBeneficiaries_root
235       @tag assume_completion
236       @pre insideCallSender != msg.sender
```

```
237     */
```

Line 238-282 in File BeneficiaryOperations.sol

```
238     function checkHowManyBeneficiaries(uint howMany) internal returns(bool) {
239         if (insideCallSender == msg.sender) {
240             require(howMany <= insideCallCount, "checkHowManyBeneficiaries: nested
                    beneficiaries modifier check require more beneficiarys");
241             return true;
242         }
243 
244         require((isExistBeneficiary(msg.sender) && (beneficiariesIndices[msg.sender] <=
                beneficiaries.length)), "checkHowManyBeneficiaries: msg.sender is not an
                beneficiary");
245 
246         uint beneficiaryIndex = beneficiariesIndices[msg.sender].sub(1);
247 
248         bytes32 operation = keccak256(abi.encodePacked(msg.data,
                beneficiariesGeneration));
249 
250         require((votesMaskByOperation[operation] & (2 ** beneficiaryIndex)) == 0, "
                checkHowManyBeneficiaries: beneficiary already voted for the operation");
251         //check limit for operation
252         require(_operationLimitByBeneficiaryIndex(uint8(beneficiaryIndex)), "
                checkHowManyBeneficiaries: operation limit is reached for this beneficiary"
                );
253 
254         votesMaskByOperation[operation] |= (2 ** beneficiaryIndex);
255         uint operationVotesCount = votesCountByOperation[operation].add(1);
256         votesCountByOperation[operation] = operationVotesCount;
257 
258         if (operationVotesCount == 1) {
259             allOperationsIndicies[operation] = allOperations.length;
260 
261             operationsByBeneficiaryIndex[operation] = uint8(beneficiaryIndex);
262 
263             operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)] = uint8(
                    operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)].add(1));
264 
265             allOperations.push(operation);
266 
267 
268             emit OperationCreated(operation, howMany, beneficiaries.length, msg.sender)
                    ;
269         }
270         emit OperationUpvoted(operation, operationVotesCount, howMany, beneficiaries.
                length, msg.sender);
271 
272         // If enough beneficiaries confirmed the same operation
273         if (votesCountByOperation[operation] == howMany) {
274             deleteOperation(operation);
275             emit OperationPerformed(operation, howMany, beneficiaries.length, msg.
                    sender);
276             return true;
277         }
278         return false;
279     }
```

The code meets the specification.

Formal Verification Request 40

deleteOperation

02, Aug 2019
266.74 ms

Line 288-300 in File BeneficiaryOperations.sol

```
288     /*@CTK deleteOperation
289       @tag assume_completion
290       @post __post.votesMaskByOperation[operation] == 0
291       @post __post.votesCountByOperation[operation] == 0
292       @post __post.allOperationsIndicies[operation] == 0
293       @post __post.operationsByBeneficiaryIndex[operation] == 0
294       @post __post.allOperations.length == allOperations.length - 1
295       @post __post.allOperationsIndicies[operation] == 0
296       @inv forall k: uint. (k >= 0 /\ k < allOperations.length) -> __post.
              allOperationsIndicies[__post.allOperations[k]] == k
297       @post __post.operationsCountByBeneficiaryIndex[uint8(
              operationsByBeneficiaryIndex[operation])] <=
              operationsCountByBeneficiaryIndex[uint8(operationsByBeneficiaryIndex[
              operation])]
298       @inv forall k: uint. (k >= 0 /\ k < allOperations.length /\ k !=
              operationsByBeneficiaryIndex[operation]) -> __post.
              operationsCountByBeneficiaryIndex[k] == operationsCountByBeneficiaryIndex[k]
299       @inv forall k: uint. (k >= 0 /\ k < allOperations.length /\ k ==
              operationsByBeneficiaryIndex[operation]) -> __post.
              operationsCountByBeneficiaryIndex[k] == operationsCountByBeneficiaryIndex[k]
              - 1
300      */
```

Line 301-318 in File BeneficiaryOperations.sol

```
301     function deleteOperation(bytes32 operation) internal {
302         uint index = allOperationsIndicies[operation];
303         if (index < allOperations.length - 1) { // Not last
304             allOperations[index] = allOperations[allOperations.length.sub(1)];
305             allOperationsIndicies[allOperations[index]] = index;
306         }
307         allOperations.length = allOperations.length.sub(1);
308 
309         uint8 beneficiaryIndex = uint8(operationsByBeneficiaryIndex[operation]);
310         operationsCountByBeneficiaryIndex[beneficiaryIndex] = uint8(
                operationsCountByBeneficiaryIndex[beneficiaryIndex].sub(1));
311         delete votesMaskByOperation[operation];
312         delete votesCountByOperation[operation];
313         delete allOperationsIndicies[operation];
314         delete operationsByBeneficiaryIndex[operation];
315     }
```

The code meets the specification.

Formal Verification Request 41

cancelPending

02, Aug 2019

984.86 ms

Line 326-330 in File BeneficiaryOperations.sol

```
326   /*@CTK cancelPending
327     @tag assume_completion
328     @post __post.insideCallSender == insideCallSender
329     @post (votesCountByOperation[operation] - __post.votesCountByOperation[operation
          ]) <= 1
330    */
```

Line 331-348 in File BeneficiaryOperations.sol

```
331   function cancelPending(bytes32 operation) public onlyAnyBeneficiary {
332
333       require((isExistBeneficiary(msg.sender) && (beneficiariesIndices[msg.sender] <=
              beneficiaries.length)), "checkHowManyBeneficiaries: msg.sender is not an
             beneficiary");
334
335       uint beneficiaryIndex = beneficiariesIndices[msg.sender].sub(1);
336
337       require((votesMaskByOperation[operation] & (2 ** beneficiaryIndex)) != 0, "
              cancelPending: operation not found for this user");
338       votesMaskByOperation[operation] &= ~(2 ** beneficiaryIndex);
339       uint operationVotesCount = votesCountByOperation[operation].sub(1);
340       votesCountByOperation[operation] = operationVotesCount;
341       emit OperationDownvoted(operation, operationVotesCount, beneficiaries.length,
              msg.sender);
342       if (operationVotesCount == 0) {
343           deleteOperation(operation);
344           emit OperationCancelled(operation, msg.sender);
345       }
346   }
```

The code meets the specification.

Formal Verification Request 42

cancelAllPending

02, Aug 2019
143.37 ms

Line 354-358 in File BeneficiaryOperations.sol

```
354   /*@CTK cancelAllPending
355     @tag assume_completion
356     @post __post.insideCallSender == insideCallSender
357     @post (__post.insideCallCount >= 0) || (__post.insideCallCount <=
          howManyBeneficiariesDecide)
358    */
```

Line 364-366 in File BeneficiaryOperations.sol

```
364   function cancelAllPending() public onlyManyBeneficiaries {
365      _cancelAllPending();
366   }
```

The code meets the specification.

Formal Verification Request 43

cancelAllPending_nested

02, Aug 2019
94.07 ms

Line 359-363 in File BeneficiaryOperations.sol

```
359     /*@CTK cancelAllPending_nested
360       @tag assume_completion
361       @pre insideCallSender != address(0)
362       @post __post.insideCallCount == insideCallCount
363      */
```

Line 364-366 in File BeneficiaryOperations.sol

```
364     function cancelAllPending() public onlyManyBeneficiaries {
365         _cancelAllPending();
366     }
```

The code meets the specification.

Formal Verification Request 44

transferBeneficiaryShipWithHowMany

02, Aug 2019
151.16 ms

Line 381-394 in File BeneficiaryOperations.sol

```
381     /*@CTK transferBeneficiaryShipWithHowMany
382       @tag assume_completion
383       @pre newBeneficiaries.length < 256
384       @post __post.insideCallSender == insideCallSender
385       @post __post.insideCallCount == insideCallCount
386       @post insideCallCount <= howManyBeneficiariesDecide
387       @post newHowManyBeneficiariesDecide <= newBeneficiaries.length
388       @post __post.beneficiariesGeneration == beneficiariesGeneration + 1
389       @post __post.howManyBeneficiariesDecide == newHowManyBeneficiariesDecide
390       @post __post.beneficiaries.length == newBeneficiaries.length
391       @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) -> __post.
              beneficiaries[k] == newBeneficiaries[k]
392       @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) -> __post.
              beneficiariesIndices[newBeneficiaries[k]] == (k + 1)
393       @post __post.allOperations.length == 0
394      */
```

Line 395-430 in File BeneficiaryOperations.sol

```
395     function transferBeneficiaryShipWithHowMany(address[] memory newBeneficiaries,
            uint256 newHowManyBeneficiariesDecide) public onlyManyBeneficiaries {
396         require(newBeneficiaries.length > 0, "transferBeneficiaryShipWithHowMany:
                beneficiaries array is empty");
397         require(newBeneficiaries.length < 256, "transferBeneficiaryshipWithHowMany:
                beneficiaries count is greater then 255");
398         require(newHowManyBeneficiariesDecide > 0, "transferBeneficiaryshipWithHowMany:
                 newHowManybeneficiarysDecide equal to 0");
```

```
399         require(newHowManyBeneficiariesDecide <= newBeneficiaries.length, "
                transferBeneficiaryShipWithHowMany: newHowManybeneficiarysDecide exceeds
                the number of beneficiarys");
400
401         // Reset beneficiaries reverse lookup table
402         /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_clear
403           @inv j <= beneficiaries.length
404           @post j == beneficiaries.length
405           @inv forall k: uint. (k >= 0 /\ k < j) -> this.beneficiariesIndices[this.
                  beneficiaries[k]] == 0
406           @post !__should_return
407          */
408         for (uint j = 0; j < beneficiaries.length; j++) {
409             delete beneficiariesIndices[beneficiaries[j]];
410         }
411         /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_reset
412           @inv i <= newBeneficiaries.length
413           @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) ->
                  newBeneficiaries[k] != address(0)
414           @post i == newBeneficiaries.length
415           @post !__should_return
416          */
417         for (uint i = 0; i < newBeneficiaries.length; i++) {
418             require(newBeneficiaries[i] != address(0), "
                    transferBeneficiaryShipWithHowMany: beneficiaries array contains zero")
                    ;
419             require(beneficiariesIndices[newBeneficiaries[i]] == 0, "
                    transferBeneficiaryShipWithHowMany: beneficiaries array contains
                    duplicates");
420             beneficiariesIndices[newBeneficiaries[i]] = uint8(i.add(1));
421         }
422
423         emit BeneficiaryshipTransferred(beneficiaries, howManyBeneficiariesDecide,
                newBeneficiaries, newHowManyBeneficiariesDecide);
424         beneficiaries = newBeneficiaries;
425         howManyBeneficiariesDecide = newHowManyBeneficiariesDecide;
426
427         _cancelAllPending();
428
429         beneficiariesGeneration++;
430     }
```

The code meets the specification.

Formal Verification Request 45

loop_cancelAllPending_votes__Generated

02, Aug 2019
582.97 ms

(Loop) Line 99-107 in File BeneficiaryOperations.sol

```
99          /*@CTK loop_cancelAllPending_votes
100           @inv (i <= this.allOperations.length)
101           @post (i == this.allOperations.length)
```

```
102         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.allOperationsIndicies[this.
                allOperations[k]] == 0)
103         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesMaskByOperation[this.
                allOperations[k]] == 0)
104         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesCountByOperation[this.
                allOperations[k]] == 0)
105         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.operationsByBeneficiaryIndex[
                this.allOperations[k]] == 0)
106         @post !__should_return
107        */
```

(Loop) Line 99-114 in File BeneficiaryOperations.sol

```
99        /*@CTK loop_cancelAllPending_votes
100         @inv (i <= this.allOperations.length)
101         @post (i == this.allOperations.length)
102         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.allOperationsIndicies[this.
                allOperations[k]] == 0)
103         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesMaskByOperation[this.
                allOperations[k]] == 0)
104         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesCountByOperation[this.
                allOperations[k]] == 0)
105         @inv forall k: uint. (k >= 0 /\ k < i) -> (this.operationsByBeneficiaryIndex[
                this.allOperations[k]] == 0)
106         @post !__should_return
107        */
108       for (uint i = 0; i < allOperations.length; i++) {
109           delete(allOperationsIndicies[allOperations[i]]);
110           delete(votesMaskByOperation[allOperations[i]]);
111           delete(votesCountByOperation[allOperations[i]]);
112           //delete operation->beneficiaryIndex
113           delete(operationsByBeneficiaryIndex[allOperations[i]]);
114       }
```

The code meets the specification.

Formal Verification Request 46

loop_cancelAllPending_operationsCount__Generated

02, Aug 2019
149.37 ms

(Loop) Line 118-123 in File BeneficiaryOperations.sol

```
118       /*@CTK loop_cancelAllPending_operationsCount
119         @inv (j <= this.beneficiaries.length)
120         @post (j == this.beneficiaries.length)
121         @inv forall k: uint. (k >= 0 /\ k < j) -> (this.
                operationsCountByBeneficiaryIndex[k] == 0)
122         @post !__should_return
123        */
```

(Loop) Line 118-126 in File BeneficiaryOperations.sol

```
118       /*@CTK loop_cancelAllPending_operationsCount
119         @inv (j <= this.beneficiaries.length)
120         @post (j == this.beneficiaries.length)
```

```
121         @inv forall k: uint. (k >= 0 /\ k < j) -> (this.
                operationsCountByBeneficiaryIndex[k] == 0)
122         @post !__should_return
123        */
124       for (uint8 j = 0; j < beneficiaries.length; j++) {
125          operationsCountByBeneficiaryIndex[j] = 0;
126       }
```

The code meets the specification.

Formal Verification Request 47

loop_transferBeneficiaryShipWithHowMany_beneficiaries_clear__Generated

02, Aug 2019
178.16 ms

(Loop) Line 402-407 in File BeneficiaryOperations.sol

```
402       /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_clear
403         @inv j <= beneficiaries.length
404         @post j == beneficiaries.length
405         @inv forall k: uint. (k >= 0 /\ k < j) -> this.beneficiariesIndices[this.
                beneficiaries[k]] == 0
406         @post !__should_return
407        */
```

(Loop) Line 402-410 in File BeneficiaryOperations.sol

```
402       /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_clear
403         @inv j <= beneficiaries.length
404         @post j == beneficiaries.length
405         @inv forall k: uint. (k >= 0 /\ k < j) -> this.beneficiariesIndices[this.
                beneficiaries[k]] == 0
406         @post !__should_return
407        */
408       for (uint j = 0; j < beneficiaries.length; j++) {
409          delete beneficiariesIndices[beneficiaries[j]];
410       }
```

The code meets the specification.

Formal Verification Request 48

loop_transferBeneficiaryShipWithHowMany_beneficiaries_reset__Generated

02, Aug 2019
91.59 ms

(Loop) Line 411-416 in File BeneficiaryOperations.sol

```
411       /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_reset
412         @inv i <= newBeneficiaries.length
413         @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) ->
                newBeneficiaries[k] != address(0)
414         @post i == newBeneficiaries.length
```

```
415         @post !__should_return
416        */
```

(Loop) Line 411-421 in File BeneficiaryOperations.sol

```
411       /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_reset
412         @inv i <= newBeneficiaries.length
413         @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) ->
                newBeneficiaries[k] != address(0)
414         @post i == newBeneficiaries.length
415         @post !__should_return
416        */
417       for (uint i = 0; i < newBeneficiaries.length; i++) {
418           require(newBeneficiaries[i] != address(0), "
                transferBeneficiaryShipWithHowMany: beneficiaries array contains zero")
                ;
419           require(beneficiariesIndices[newBeneficiaries[i]] == 0, "
                transferBeneficiaryShipWithHowMany: beneficiaries array contains
                duplicates");
420           beneficiariesIndices[newBeneficiaries[i]] = uint8(i.add(1));
421       }
```

The code meets the specification.

Formal Verification Request 49

OpenZeppelin_TokenTimelock_changeBeneficiary

02, Aug 2019
4.61 ms

Line 68-71 in File TokenTimelock.sol

```
68    /*@CTK OpenZeppelin_TokenTimelock_changeBeneficiary
69      @tag assume_completion
70      @post __post._beneficiary == _newBeneficiary
71     */
```

Line 72-74 in File TokenTimelock.sol

```
72    function _changeBeneficiary(address _newBeneficiary) internal {
73        _beneficiary = _newBeneficiary;
74    }
```

The code meets the specification.

Formal Verification Request 50

OpenZeppelin_TokenVesting_changeBeneficiary

02, Aug 2019
5.72 ms

Line 179-182 in File TokenVesting.sol

```
179     /*@CTK OpenZeppelin_TokenVesting_changeBeneficiary
180       @tag assume_completion
181       @post __post._beneficiary == _newBeneficiary
182      */
```

Line 183-185 in File TokenVesting.sol

```
183     function _changeBeneficiary(address _newBeneficiary) internal {
184         _beneficiary = _newBeneficiary;
185     }
```

The code meets the specification.

Source Code with CertiK Labels

File logics/AkropolisTokenVesting.sol

```
1  pragma solidity ^0.5.9;
2
3  import "../openzeppelin/TokenVesting.sol";
4
5  //Beneficieries template
6  import "../helpers/BeneficiaryOperations.sol";
7
8  contract AkropolisTokenVesting is TokenVesting, BeneficiaryOperations {
9
10     IERC20 private token;
11
12     address private _pendingBeneficiary;
13
14     event LogBeneficiaryTransferProposed(address _beneficiary);
15     event LogBeneficiaryTransfered(address _beneficiary);
16
17     constructor (IERC20 _token, uint256 _start, uint256 _cliffDuration, uint256
           _duration) public
18         TokenVesting(msg.sender, _start, _cliffDuration, _duration, false) {
19             token = _token;
20         }
21
22      /**
23      * @notice Transfers vested tokens to beneficiary.
24      */
25
26     function release() public {
27         super.release(token);
28     }
29
30     // MODIFIERS
31     /**
32     * @dev Allows to perform method by existing beneficiary
33     */
34     modifier onlyExistingBeneficiary(address _beneficiary) {
35         require(isExistBeneficiary(_beneficiary), "address is not in beneficiary array"
               );
36         _;
37     }
38
39     /**
40     * @dev Allows to perform method by pending beneficiary
41     */
42
43     modifier onlyPendingBeneficiary {
44         require(msg.sender == _pendingBeneficiary, "Unpermitted operation.");
45         _;
46     }
47
48     function pendingBeneficiary() public view returns (address) {
49         return _pendingBeneficiary;
50     }
51
52      /**
```

```
53          * @dev Allows beneficiaries to change beneficiaryShip and set first beneficiary
                as default
54          * @param _newBeneficiaries defines array of addresses of new beneficiaries
55      */
56      function transferBeneficiaryShip(address[] memory _newBeneficiaries) public {
57          super.transferBeneficiaryShip(_newBeneficiaries);
58          _setPendingBeneficiary(beneficiaries[0]);
59      }
60
61       /**
62          * @dev Allows beneficiaries to change beneficiaryShip and set first beneficiary
                as default
63          * @param _newBeneficiaries defines array of addresses of new beneficiaries
64          * @param _newHowManyBeneficiariesDecide defines how many beneficiaries can
                decide
65      */
66
67      /*@CTK AkropolisTokenVesting_transferBeneficiaryShip
68        @tag assume_completion
69        @post __post._pendingBeneficiary == beneficiaries[0]
70       */
71      function transferBeneficiaryShipWithHowMany(address[] memory _newBeneficiaries,
            uint256 _newHowManyBeneficiariesDecide) public {
72          super.transferBeneficiaryShipWithHowMany(_newBeneficiaries,
                _newHowManyBeneficiariesDecide);
73          _setPendingBeneficiary(beneficiaries[0]);
74      }
75
76      /**
77          * @dev Allows beneficiaries to change beneficiary as default
78           * @param _newBeneficiary defines address of new beneficiary
79      */
80      /*@CTK AkropolisTokenVesting_changeBeneficiary
81        @tag assume_completion
82        @post __post.insideCallSender == insideCallSender
83        @pre __post.insideCallCount <= __post.howManyBeneficiariesDecide
84      */
85      function changeBeneficiary(address _newBeneficiary) public onlyManyBeneficiaries {
86          _setPendingBeneficiary(_newBeneficiary);
87      }
88
89      /**
90          * @dev Claim Beneficiary
91      */
92      //@CTK NO_OVERFLOW
93      //@CTK NO_BUF_OVERFLOW
94      //@CTK NO_ASF
95      /*@CTK Vesting_claimBeneficiary
96        @tag assume_completion
97        @post (msg.sender) == _pendingBeneficiary
98        @post __post._beneficiary == (msg.sender)
99        @post __post._pendingBeneficiary == address(0)
100      */
101     function claimBeneficiary() public onlyPendingBeneficiary {
102         _changeBeneficiary(_pendingBeneficiary);
103         emit LogBeneficiaryTransfered(_pendingBeneficiary);
104         _pendingBeneficiary = address(0);
105     }
```

```
106
107     /*
108      * Internal Functions
109      *
110      */
111     /**
112         * @dev Set pending Beneficiary address
113         * @param _newBeneficiary defines address of new beneficiary
114     */
115     //@CTK NO_OVERFLOW
116     //@CTK NO_BUF_OVERFLOW
117     //@CTK NO_ASF
118     /*@CTK Vesting_claimBeneficiary
119       @tag assume_completion
120       @pre beneficiariesIndices[_beneficiary] > 0
121       @post __post._beneficiary == _beneficiary
122       @post __post._pendingBeneficiary == _newBeneficiary
123      */
124     function _setPendingBeneficiary(address _newBeneficiary) internal
            onlyExistingBeneficiary(_newBeneficiary) {
125         _pendingBeneficiary = _newBeneficiary;
126         emit LogBeneficiaryTransferProposed(_newBeneficiary);
127     }
128 }
```

File logics/AkropolisTimeLock.sol

```
1  pragma solidity ^0.5.9;
2  import "../openzeppelin/TokenTimelock.sol";
3
4  //Beneficieries template
5  import "../helpers/BeneficiaryOperations.sol";
6
7  contract AkropolisTimeLock is TokenTimelock, BeneficiaryOperations {
8
9          address private _pendingBeneficiary;
10
11
12         event LogBeneficiaryTransferProposed(address _beneficiary);
13         event LogBeneficiaryTransfered(address _beneficiary);
14
15         /**
16         * @notice Constructor.
17         * @param _token Address of AKRO token
18         * @param _releaseTime Timestamp date
19         */
20
21         constructor (IERC20 _token, uint256 _releaseTime) public
22             TokenTimelock(_token, msg.sender, _releaseTime) {
23         }
24
25         // MODIFIERS
26         /**
27         * @dev Allows to perform method by existing beneficiary
28         */
29         modifier onlyExistingBeneficiary(address _beneficiary) {
30             require(isExistBeneficiary(_beneficiary), "address is not in beneficiary
                    array");
31              _;
```

```
32        }
33
34        /**
35        * @dev Allows to perform method by pending beneficiary
36        */
37        modifier onlyPendingBeneficiary {
38            require(msg.sender == _pendingBeneficiary, "Unpermitted operation.");
39            _;
40        }
41
42        function pendingBeneficiary() public view returns (address) {
43            return _pendingBeneficiary;
44        }
45
46        /**
47            * @dev Allows beneficiaries to change beneficiaryShip and set first
                    beneficiary as default
48            * @param _newBeneficiaries defines array of addresses of new beneficiaries
49            * @param _newHowManyBeneficiariesDecide defines how many beneficiaries can
                    decide
50        */
51        function transferBeneficiaryShipWithHowMany(address[] memory _newBeneficiaries,
              uint256 _newHowManyBeneficiariesDecide) public {
52            super.transferBeneficiaryShipWithHowMany(_newBeneficiaries,
                  _newHowManyBeneficiariesDecide);
53            _setPendingBeneficiary(beneficiaries[0]);
54        }
55
56         /**
57            * @dev Allows beneficiaries to change beneficiaryShip and set first
                    beneficiary as default
58            * @param _newBeneficiaries defines array of addresses of new beneficiaries
59        */
60
61        /*@CTK AkropolisTimeLock_transferBeneficiaryShip
62          @tag assume_completion
63          @post __post._pendingBeneficiary == beneficiaries[0]
64         */
65        function transferBeneficiaryShip(address[] memory _newBeneficiaries) public {
66            super.transferBeneficiaryShip(_newBeneficiaries);
67            _setPendingBeneficiary(beneficiaries[0]);
68        }
69
70        /**
71            * @dev Allows beneficiaries to change beneficiary as default
72            * @param _newBeneficiary defines address of new beneficiary
73        */
74        /*@CTK AkropolisTimeLock_changeBeneficiary
75          @tag assume_completion
76          @post __post.insideCallSender == insideCallSender
77          @pre __post.insideCallCount <= __post.howManyBeneficiariesDecide
78         */
79        function changeBeneficiary(address _newBeneficiary) public
              onlyManyBeneficiaries {
80            _setPendingBeneficiary(_newBeneficiary);
81        }
82
83        /**
```

```
84              * @dev Claim Beneficiary
85          */
86          //@CTK NO_OVERFLOW
87          //@CTK NO_BUF_OVERFLOW
88          //@CTK NO_ASF
89          /*@CTK TimeLock_claimBeneficiary
90            @tag assume_completion
91            @post (msg.sender) == _pendingBeneficiary
92            @post __post._beneficiary == (msg.sender)
93            @post __post._pendingBeneficiary == address(0)
94           */
95          function claimBeneficiary() public onlyPendingBeneficiary {
96              _changeBeneficiary(_pendingBeneficiary);
97              emit LogBeneficiaryTransfered(_pendingBeneficiary);
98              _pendingBeneficiary = address(0);
99          }
100
101         /*
102         * Internal Functions
103         *
104         */
105         /**
106             * @dev Set pending Beneficiary address
107             * @param _newBeneficiary defines address of new beneficiary
108         */
109         //@CTK NO_OVERFLOW
110         //@CTK NO_BUF_OVERFLOW
111         //@CTK NO_ASF
112         /*@CTK TimeLock_claimBeneficiary
113           @tag assume_completion
114           @pre beneficiariesIndices[_beneficiary] > 0
115           @post __post._beneficiary == _beneficiary
116           @post __post._pendingBeneficiary == _newBeneficiary
117          */
118         function _setPendingBeneficiary(address _newBeneficiary) internal
                onlyExistingBeneficiary(_newBeneficiary) {
119             _pendingBeneficiary = _newBeneficiary;
120             emit LogBeneficiaryTransferProposed(_newBeneficiary);
121         }
122 }
```

File helpers/BeneficiaryOperations.sol

```
1  /*
2    License: MIT
3    Copyright Bitclave, 2018
4    It's modified contract BeneficiaryOperations from https://github.com/bitclave/
         BeneficiaryOperations
5  */
6
7  pragma solidity ^0.5.9;
8
9  import "openzeppelin-solidity/contracts/math/SafeMath.sol";
10
11 contract BeneficiaryOperations {
12
13     using SafeMath for uint256;
14
15     using SafeMath for uint8;
```

```
16      // VARIABLES
17
18      uint256 public beneficiariesGeneration;
19      uint256 public howManyBeneficiariesDecide;
20      address[] public beneficiaries;
21      bytes32[] public allOperations;
22      address internal insideCallSender;
23      uint256 internal insideCallCount;
24
25
26      // Reverse lookup tables for beneficiaries and allOperations
27      mapping(address => uint8) public beneficiariesIndices; // Starts from 1, size 255
28      mapping(bytes32 => uint) public allOperationsIndicies;
29
30
31      // beneficiaries voting mask per operations
32      mapping(bytes32 => uint256) public votesMaskByOperation;
33      mapping(bytes32 => uint256) public votesCountByOperation;
34
35      //operation -> beneficiaryIndex
36      mapping(bytes32 => uint8) internal operationsByBeneficiaryIndex;
37      mapping(uint8 => uint8) internal operationsCountByBeneficiaryIndex;
38      // EVENTS
39
40      event BeneficiaryshipTransferred(address[] previousbeneficiaries, uint
            howManyBeneficiariesDecide, address[] newBeneficiaries, uint
            newHowManybeneficiarysDecide);
41      event OperationCreated(bytes32 operation, uint howMany, uint beneficiariesCount,
            address proposer);
42      event OperationUpvoted(bytes32 operation, uint votes, uint howMany, uint
            beneficiariesCount, address upvoter);
43      event OperationPerformed(bytes32 operation, uint howMany, uint beneficiariesCount,
             address performer);
44      event OperationDownvoted(bytes32 operation, uint votes, uint beneficiariesCount,
            address downvoter);
45      event OperationCancelled(bytes32 operation, address lastCanceller);
46
47      // ACCESSORS
48
49      //@CTK NO_OVERFLOW
50      //@CTK NO_BUF_OVERFLOW
51      //@CTK NO_ASF
52      /*@CTK isExistBeneficiary
53        @tag assume_completion
54        @post __return == (beneficiariesIndices[wallet] > 0)
55       */
56      function isExistBeneficiary(address wallet) public view returns(bool) {
57          return beneficiariesIndices[wallet] > 0;
58      }
59
60      //@CTK NO_OVERFLOW
61      //@CTK NO_BUF_OVERFLOW
62      //@CTK NO_ASF
63      /*@CTK beneficiariesCount
64        @tag assume_completion
65        @post __return == beneficiaries.length
66       */
67      function beneficiariesCount() public view returns(uint) {
```

```
68          return beneficiaries.length;
69      }
70
71      //@CTK NO_OVERFLOW
72      //@CTK NO_BUF_OVERFLOW
73      //@CTK NO_ASF
74      /*@CTK allOperationsCount
75        @tag assume_completion
76        @post __return == allOperations.length
77       */
78      function allOperationsCount() public view returns(uint) {
79          return allOperations.length;
80      }
81
82      /*
83        Internal functions
84      */
85
86      /*@CTK _operationLimitByBeneficiaryIndex
87        @tag assume_completion
88        @post __return == (operationsCountByBeneficiaryIndex[beneficiaryIndex] <= 3)
89       */
90      function _operationLimitByBeneficiaryIndex(uint8 beneficiaryIndex) internal view
            returns(bool) {
91          return (operationsCountByBeneficiaryIndex[beneficiaryIndex] <= 3);
92      }
93
94      /*@CTK _cancelAllPending
95        @tag assume_completion
96        @post __post.allOperations.length == 0
97       */
98      function _cancelAllPending() internal {
99          /*@CTK loop_cancelAllPending_votes
100           @inv (i <= this.allOperations.length)
101           @post (i == this.allOperations.length)
102           @inv forall k: uint. (k >= 0 /\ k < i) -> (this.allOperationsIndicies[this.
                  allOperations[k]] == 0)
103           @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesMaskByOperation[this.
                  allOperations[k]] == 0)
104           @inv forall k: uint. (k >= 0 /\ k < i) -> (this.votesCountByOperation[this.
                  allOperations[k]] == 0)
105           @inv forall k: uint. (k >= 0 /\ k < i) -> (this.operationsByBeneficiaryIndex[
                  this.allOperations[k]] == 0)
106           @post !__should_return
107          */
108         for (uint i = 0; i < allOperations.length; i++) {
109             delete(allOperationsIndicies[allOperations[i]]);
110             delete(votesMaskByOperation[allOperations[i]]);
111             delete(votesCountByOperation[allOperations[i]]);
112             //delete operation->beneficiaryIndex
113             delete(operationsByBeneficiaryIndex[allOperations[i]]);
114         }
115
116         allOperations.length = 0;
117         //delete operations count for beneficiary
118         /*@CTK loop_cancelAllPending_operationsCount
119           @inv (j <= this.beneficiaries.length)
120           @post (j == this.beneficiaries.length)
```

```
121         @inv forall k: uint. (k >= 0 /\ k < j) -> (this.
                operationsCountByBeneficiaryIndex[k] == 0)
122         @post !__should_return
123         */
124         for (uint8 j = 0; j < beneficiaries.length; j++) {
125             operationsCountByBeneficiaryIndex[j] = 0;
126         }
127     }
128
129
130     // MODIFIERS
131
132     /**
133     * @dev Allows to perform method by any of the beneficiaries
134     */
135     modifier onlyAnyBeneficiary {
136         if (checkHowManyBeneficiaries(1)) {
137             bool update = (insideCallSender == address(0));
138             if (update) {
139                 insideCallSender = msg.sender;
140                 insideCallCount = 1;
141             }
142             _;
143             if (update) {
144                 insideCallSender = address(0);
145                 insideCallCount = 0;
146             }
147         }
148     }
149
150     /**
151     * @dev Allows to perform method only after many beneficiaries call it with the
            same arguments
152     */
153     modifier onlyManyBeneficiaries {
154         if (checkHowManyBeneficiaries(howManyBeneficiariesDecide)) {
155             bool update = (insideCallSender == address(0));
156             if (update) {
157                 insideCallSender = msg.sender;
158                 insideCallCount = howManyBeneficiariesDecide;
159             }
160             _;
161             if (update) {
162                 insideCallSender = address(0);
163                 insideCallCount = 0;
164             }
165         }
166     }
167
168     /**
169     * @dev Allows to perform method only after all beneficiaries call it with the same
            arguments
170     */
171     modifier onlyAllBeneficiaries {
172         if (checkHowManyBeneficiaries(beneficiaries.length)) {
173             bool update = (insideCallSender == address(0));
174             if (update) {
175                 insideCallSender = msg.sender;
```

```
176                 insideCallCount = beneficiaries.length;
177             }
178             _;
179             if (update) {
180                 insideCallSender = address(0);
181                 insideCallCount = 0;
182             }
183         }
184     }
185
186     /**
187     * @dev Allows to perform method only after some beneficiaries call it with the
            same arguments
188     */
189     modifier onlySomeBeneficiaries(uint howMany) {
190         require(howMany > 0, "onlySomeBeneficiaries: howMany argument is zero");
191         require(howMany <= beneficiaries.length, "onlySomeBeneficiaries: howMany
                argument exceeds the number of Beneficiaries");
192
193         if (checkHowManyBeneficiaries(howMany)) {
194             bool update = (insideCallSender == address(0));
195             if (update) {
196                 insideCallSender = msg.sender;
197                 insideCallCount = howMany;
198             }
199             _;
200             if (update) {
201                 insideCallSender = address(0);
202                 insideCallCount = 0;
203             }
204         }
205     }
206
207     // CONSTRUCTOR
208
209     //@CTK NO_BUF_OVERFLOW
210     //@CTK NO_ASF
211     /*@CTK BeneficiaryOperations
212       @tag assume_completion
213       @pre beneficiaries[0] == 0
214       @post __post.beneficiariesIndices[msg.sender] == 1
215       @post __post.howManyBeneficiariesDecide == 1
216      */
217     constructor() public {
218         beneficiaries.push(msg.sender);
219         beneficiariesIndices[msg.sender] = 1;
220         howManyBeneficiariesDecide = 1;
221     }
222
223     // INTERNAL METHODS
224
225     /**
226      * @dev onlyManybeneficiaries modifier helper
227      */
228     /*@CTK checkHowManyBeneficiaries_nested
229       @tag assume_completion
230       @pre insideCallSender == msg.sender
231       @post howMany <= insideCallCount
```

```
232      @post __return == true
233     */
234    /*@CTK checkHowManyBeneficiaries_root
235      @tag assume_completion
236      @pre insideCallSender != msg.sender
237     */
238    function checkHowManyBeneficiaries(uint howMany) internal returns(bool) {
239        if (insideCallSender == msg.sender) {
240            require(howMany <= insideCallCount, "checkHowManyBeneficiaries: nested
                   beneficiaries modifier check require more beneficiarys");
241            return true;
242        }
243
244        require((isExistBeneficiary(msg.sender) && (beneficiariesIndices[msg.sender] <=
                beneficiaries.length)), "checkHowManyBeneficiaries: msg.sender is not an
               beneficiary");
245
246        uint beneficiaryIndex = beneficiariesIndices[msg.sender].sub(1);
247
248        bytes32 operation = keccak256(abi.encodePacked(msg.data,
               beneficiariesGeneration));
249
250        require((votesMaskByOperation[operation] & (2 ** beneficiaryIndex)) == 0, "
               checkHowManyBeneficiaries: beneficiary already voted for the operation");
251        //check limit for operation
252        require(_operationLimitByBeneficiaryIndex(uint8(beneficiaryIndex)), "
               checkHowManyBeneficiaries: operation limit is reached for this beneficiary"
               );
253
254        votesMaskByOperation[operation] |= (2 ** beneficiaryIndex);
255        uint operationVotesCount = votesCountByOperation[operation].add(1);
256        votesCountByOperation[operation] = operationVotesCount;
257
258        if (operationVotesCount == 1) {
259            allOperationsIndicies[operation] = allOperations.length;
260
261            operationsByBeneficiaryIndex[operation] = uint8(beneficiaryIndex);
262
263            operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)] = uint8(
                   operationsCountByBeneficiaryIndex[uint8(beneficiaryIndex)].add(1));
264
265            allOperations.push(operation);
266
267
268            emit OperationCreated(operation, howMany, beneficiaries.length, msg.sender)
                   ;
269        }
270        emit OperationUpvoted(operation, operationVotesCount, howMany, beneficiaries.
               length, msg.sender);
271
272        // If enough beneficiaries confirmed the same operation
273        if (votesCountByOperation[operation] == howMany) {
274            deleteOperation(operation);
275            emit OperationPerformed(operation, howMany, beneficiaries.length, msg.
                   sender);
276            return true;
277        }
278
```

```
279         return false;
280     }
281 
282     /**
283     * @dev Used to delete cancelled or performed operation
284     * @param operation defines which operation to delete
285     */
286     /*@CTK deleteOperation
287       @tag assume_completion
288       @post __post.votesMaskByOperation[operation] == 0
289       @post __post.votesCountByOperation[operation] == 0
290       @post __post.allOperationsIndicies[operation] == 0
291       @post __post.operationsByBeneficiaryIndex[operation] == 0
292       @post __post.allOperations.length == allOperations.length - 1
293       @post __post.allOperationsIndicies[operation] == 0
294       @inv forall k: uint. (k >= 0 /\ k < allOperations.length) -> __post.
              allOperationsIndicies[__post.allOperations[k]] == k
295       @post __post.operationsCountByBeneficiaryIndex[uint8(
              operationsByBeneficiaryIndex[operation])] <=
              operationsCountByBeneficiaryIndex[uint8(operationsByBeneficiaryIndex[
              operation])]
296       @inv forall k: uint. (k >= 0 /\ k < allOperations.length /\ k !=
              operationsByBeneficiaryIndex[operation]) -> __post.
              operationsCountByBeneficiaryIndex[k] == operationsCountByBeneficiaryIndex[k]
297       @inv forall k: uint. (k >= 0 /\ k < allOperations.length /\ k ==
              operationsByBeneficiaryIndex[operation]) -> __post.
              operationsCountByBeneficiaryIndex[k] == operationsCountByBeneficiaryIndex[k]
              - 1
298      */
299     function deleteOperation(bytes32 operation) internal {
300         uint index = allOperationsIndicies[operation];
301         if (index < allOperations.length - 1) { // Not last
302             allOperations[index] = allOperations[allOperations.length.sub(1)];
303             allOperationsIndicies[allOperations[index]] = index;
304         }
305         allOperations.length = allOperations.length.sub(1);
306 
307         uint8 beneficiaryIndex = uint8(operationsByBeneficiaryIndex[operation]);
308         operationsCountByBeneficiaryIndex[beneficiaryIndex] = uint8(
              operationsCountByBeneficiaryIndex[beneficiaryIndex].sub(1));
309 
310         delete votesMaskByOperation[operation];
311         delete votesCountByOperation[operation];
312         delete allOperationsIndicies[operation];
313         delete operationsByBeneficiaryIndex[operation];
314     }
315 
316     // PUBLIC METHODS
317 
318     /**
319     * @dev Allows beneficiaries to change their mind by cancelling
            votesMaskByOperation operations
320     * @param operation defines which operation to delete
321     */
322     /*@CTK cancelPending
323       @tag assume_completion
324       @post __post.insideCallSender == insideCallSender
325       @post (votesCountByOperation[operation] - __post.votesCountByOperation[operation
```

```
        ]) <= 1
326     */
327     function cancelPending(bytes32 operation) public onlyAnyBeneficiary {
328
329         require((isExistBeneficiary(msg.sender) && (beneficiariesIndices[msg.sender] <=
                beneficiaries.length)), "checkHowManyBeneficiaries: msg.sender is not an
                beneficiary");
330
331         uint beneficiaryIndex = beneficiariesIndices[msg.sender].sub(1);
332
333         require((votesMaskByOperation[operation] & (2 ** beneficiaryIndex)) != 0, "
                cancelPending: operation not found for this user");
334         votesMaskByOperation[operation] &= ~(2 ** beneficiaryIndex);
335         uint operationVotesCount = votesCountByOperation[operation].sub(1);
336         votesCountByOperation[operation] = operationVotesCount;
337         emit OperationDownvoted(operation, operationVotesCount, beneficiaries.length,
                msg.sender);
338         if (operationVotesCount == 0) {
339             deleteOperation(operation);
340             emit OperationCancelled(operation, msg.sender);
341         }
342     }
343
344     /**
345     * @dev Allows beneficiaries to change their mind by cancelling all operations
346     */
347
348     /*@CTK cancelAllPending
349       @tag assume_completion
350       @post __post.insideCallSender == insideCallSender
351       @post (__post.insideCallCount >= 0) || (__post.insideCallCount <=
              howManyBeneficiariesDecide)
352      */
353     /*@CTK cancelAllPending_nested
354       @tag assume_completion
355       @pre insideCallSender != address(0)
356       @post __post.insideCallCount == insideCallCount
357      */
358     function cancelAllPending() public onlyManyBeneficiaries {
359        _cancelAllPending();
360     }
361
362     /**
363     * @dev Allows beneficiaries to change beneficiariesship
364     * @param newBeneficiaries defines array of addresses of new beneficiaries
365     */
366     function transferBeneficiaryShip(address[] memory newBeneficiaries) public {
367         transferBeneficiaryShipWithHowMany(newBeneficiaries, newBeneficiaries.length);
368     }
369
370     /**
371     * @dev Allows beneficiaries to change beneficiaryShip
372     * @param newBeneficiaries defines array of addresses of new beneficiaries
373     * @param newHowManyBeneficiariesDecide defines how many beneficiaries can decide
374     */
375     /*@CTK transferBeneficiaryShipWithHowMany
376       @tag assume_completion
377       @pre newBeneficiaries.length < 256
```

```
378       @post __post.insideCallSender == insideCallSender
379       @post __post.insideCallCount == insideCallCount
380       @post insideCallCount <= howManyBeneficiariesDecide
381       @post newHowManyBeneficiariesDecide <= newBeneficiaries.length
382       @post __post.beneficiariesGeneration == beneficiariesGeneration + 1
383       @post __post.howManyBeneficiariesDecide == newHowManyBeneficiariesDecide
384       @post __post.beneficiaries.length == newBeneficiaries.length
385       @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) -> __post.
              beneficiaries[k] == newBeneficiaries[k]
386       @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) -> __post.
              beneficiariesIndices[newBeneficiaries[k]] == (k + 1)
387       @post __post.allOperations.length == 0
388      */
389     function transferBeneficiaryShipWithHowMany(address[] memory newBeneficiaries,
          uint256 newHowManyBeneficiariesDecide) public onlyManyBeneficiaries {
390         require(newBeneficiaries.length > 0, "transferBeneficiaryShipWithHowMany:
              beneficiaries array is empty");
391         require(newBeneficiaries.length < 256, "transferBeneficiaryshipWithHowMany:
              beneficiaries count is greater then 255");
392         require(newHowManyBeneficiariesDecide > 0, "transferBeneficiaryshipWithHowMany:
               newHowManybeneficiarysDecide equal to 0");
393         require(newHowManyBeneficiariesDecide <= newBeneficiaries.length, "
              transferBeneficiaryShipWithHowMany: newHowManybeneficiarysDecide exceeds
              the number of beneficiarys");
394
395         // Reset beneficiaries reverse lookup table
396         /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_clear
397           @inv j <= beneficiaries.length
398           @post j == beneficiaries.length
399           @inv forall k: uint. (k >= 0 /\ k < j) -> this.beneficiariesIndices[this.
                beneficiaries[k]] == 0
400           @post !__should_return
401          */
402         for (uint j = 0; j < beneficiaries.length; j++) {
403             delete beneficiariesIndices[beneficiaries[j]];
404         }
405         /*@CTK loop_transferBeneficiaryShipWithHowMany_beneficiaries_reset
406           @inv i <= newBeneficiaries.length
407           @inv forall k: uint. (k >= 0 /\ k < newBeneficiaries.length) ->
                newBeneficiaries[k] != address(0)
408           @post i == newBeneficiaries.length
409           @post !__should_return
410          */
411         for (uint i = 0; i < newBeneficiaries.length; i++) {
412             require(newBeneficiaries[i] != address(0), "
                  transferBeneficiaryShipWithHowMany: beneficiaries array contains zero")
                  ;
413             require(beneficiariesIndices[newBeneficiaries[i]] == 0, "
                  transferBeneficiaryShipWithHowMany: beneficiaries array contains
                  duplicates");
414             beneficiariesIndices[newBeneficiaries[i]] = uint8(i.add(1));
415         }
416
417         emit BeneficiaryshipTransferred(beneficiaries, howManyBeneficiariesDecide,
              newBeneficiaries, newHowManyBeneficiariesDecide);
418         beneficiaries = newBeneficiaries;
419         howManyBeneficiariesDecide = newHowManyBeneficiariesDecide;
420
```

```
421         _cancelAllPending();
422
423         beneficiariesGeneration++;
424     }
425 }
```

File openzeppelin/TokenTimelock.sol

```
1  pragma solidity ^0.5.9;
2
3  import "openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol";
4
5  /**
6   * @title TokenTimelock
7   * @dev TokenTimelock is a token holder contract that will allow a
8   * beneficiary to extract the tokens after a given release time.
9   */
10 contract TokenTimelock {
11     using SafeERC20 for IERC20;
12
13     // ERC20 basic token contract being held
14     IERC20 private _token;
15
16     // beneficiary of tokens after they are released
17     address private _beneficiary;
18
19     // timestamp when token release is enabled
20     uint256 private _releaseTime;
21
22     constructor (IERC20 token, address beneficiary, uint256 releaseTime) public {
23         // solhint-disable-next-line not-rely-on-time
24         require(releaseTime > block.timestamp, "TokenTimelock: release time is before
               current time");
25         _token = token;
26         _beneficiary = beneficiary;
27         _releaseTime = releaseTime;
28     }
29
30     /**
31      * @return the token being held.
32      */
33     function token() public view returns (IERC20) {
34         return _token;
35     }
36
37     /**
38      * @return the beneficiary of the tokens.
39      */
40     function beneficiary() public view returns (address) {
41         return _beneficiary;
42     }
43
44     /**
45      * @return the time when the tokens are released.
46      */
47     function releaseTime() public view returns (uint256) {
48         return _releaseTime;
49     }
50
```

```
51      /**
52       * @notice Transfers tokens held by timelock to beneficiary.
53       */
54      function release() public {
55          // solhint-disable-next-line not-rely-on-time
56          require(block.timestamp >= _releaseTime, "TokenTimelock: current time is before
                release time");
57
58          uint256 amount = _token.balanceOf(address(this));
59          require(amount > 0, "TokenTimelock: no tokens to release");
60
61          _token.safeTransfer(_beneficiary, amount);
62      }
63
64       /**
65       * @return change the beneficiary of tokens
66       */
67
68      /*@CTK OpenZeppelin_TokenTimelock_changeBeneficiary
69        @tag assume_completion
70        @post __post._beneficiary == _newBeneficiary
71       */
72      function _changeBeneficiary(address _newBeneficiary) internal {
73          _beneficiary = _newBeneficiary;
74      }
75  }
```

File openzeppelin/TokenVesting.sol

```
1   pragma solidity ^0.5.9;
2
3   import "openzeppelin-solidity/contracts/token/ERC20/SafeERC20.sol";
4   import "openzeppelin-solidity/contracts/ownership/Ownable.sol";
5   import "openzeppelin-solidity/contracts/math/SafeMath.sol";
6
7   /**
8    * @title TokenVesting
9    * @dev A token holder contract that can release its token balance gradually like a
10   * typical vesting scheme, with a cliff and vesting period. Optionally revocable by
         the
11   * owner.
12   */
13  contract TokenVesting is Ownable {
14      // The vesting schedule is time-based (i.e. using block timestamps as opposed to e
            .g. block numbers), and is
15      // therefore sensitive to timestamp manipulation (which is something miners can do
            , to a certain degree). Therefore,
16      // it is recommended to avoid using short time durations (less than a minute).
            Typical vesting schemes, with a
17      // cliff period of a year and a duration of four years, are safe to use.
18      // solhint-disable not-rely-on-time
19
20      using SafeMath for uint256;
21      using SafeERC20 for IERC20;
22
23      event TokensReleased(address token, uint256 amount);
24      event TokenVestingRevoked(address token);
25
26      // beneficiary of tokens after they are released
```

```
27      address private _beneficiary;
28
29      // Durations and timestamps are expressed in UNIX time, the same units as block.
            timestamp.
30      uint256 private _cliff;
31      uint256 private _start;
32      uint256 private _duration;
33
34      bool private _revocable;
35
36      mapping (address => uint256) private _released;
37      mapping (address => bool) private _revoked;
38
39      /**
40       * @dev Creates a vesting contract that vests its balance of any ERC20 token to
             the
41       * beneficiary, gradually in a linear fashion until start + duration. By then all
42       * of the balance will have vested.
43       * @param beneficiary address of the beneficiary to whom vested tokens are
             transferred
44       * @param cliffDuration duration in seconds of the cliff in which tokens will
             begin to vest
45       * @param start the time (as Unix time) at which point vesting starts
46       * @param duration duration in seconds of the period in which the tokens will vest
47       * @param revocable whether the vesting is revocable or not
48       */
49      constructor (address beneficiary, uint256 start, uint256 cliffDuration, uint256
            duration, bool revocable) public {
50          require(beneficiary != address(0), "TokenVesting: beneficiary is the zero
                address");
51          // solhint-disable-next-line max-line-length
52          require(cliffDuration <= duration, "TokenVesting: cliff is longer than duration
                ");
53          require(duration > 0, "TokenVesting: duration is 0");
54          // solhint-disable-next-line max-line-length
55          require(start.add(duration) > block.timestamp, "TokenVesting: final time is
                before current time");
56
57          _beneficiary = beneficiary;
58          _revocable = revocable;
59          _duration = duration;
60          _cliff = start.add(cliffDuration);
61          _start = start;
62      }
63
64      /**
65       * @return the beneficiary of the tokens.
66       */
67      function beneficiary() public view returns (address) {
68          return _beneficiary;
69      }
70
71      /**
72       * @return the cliff time of the token vesting.
73       */
74      function cliff() public view returns (uint256) {
75          return _cliff;
76      }
```

```
77
78      /**
79       * @return the start time of the token vesting.
80       */
81      function start() public view returns (uint256) {
82          return _start;
83      }
84
85      /**
86       * @return the duration of the token vesting.
87       */
88      function duration() public view returns (uint256) {
89          return _duration;
90      }
91
92      /**
93       * @return true if the vesting is revocable.
94       */
95      function revocable() public view returns (bool) {
96          return _revocable;
97      }
98
99      /**
100      * @return the amount of the token released.
101      */
102     function released(address token) public view returns (uint256) {
103         return _released[token];
104     }
105
106     /**
107      * @return true if the token is revoked.
108      */
109     function revoked(address token) public view returns (bool) {
110         return _revoked[token];
111     }
112
113     /**
114      * @notice Transfers vested tokens to beneficiary.
115      * @param token ERC20 token which is being vested
116      */
117     function release(IERC20 token) public {
118         uint256 unreleased = _releasableAmount(token);
119
120         require(unreleased > 0, "TokenVesting: no tokens are due");
121
122         _released[address(token)] = _released[address(token)].add(unreleased);
123
124         token.safeTransfer(_beneficiary, unreleased);
125
126         emit TokensReleased(address(token), unreleased);
127     }
128
129     /**
130      * @notice Allows the owner to revoke the vesting. Tokens already vested
131      * remain in the contract, the rest are returned to the owner.
132      * @param token ERC20 token which is being vested
133      */
134     function revoke(IERC20 token) public onlyOwner {
```

```
135         require(_revocable, "TokenVesting: cannot revoke");
136         require(!_revoked[address(token)], "TokenVesting: token already revoked");
137 
138         uint256 balance = token.balanceOf(address(this));
139 
140         uint256 unreleased = _releasableAmount(token);
141         uint256 refund = balance.sub(unreleased);
142 
143         _revoked[address(token)] = true;
144 
145         token.safeTransfer(owner(), refund);
146 
147         emit TokenVestingRevoked(address(token));
148     }
149 
150     /**
151      * @dev Calculates the amount that has already vested but hasn't been released yet
152      * @param token ERC20 token which is being vested
153      */
154     function _releasableAmount(IERC20 token) private view returns (uint256) {
155         return _vestedAmount(token).sub(_released[address(token)]);
156     }
157 
158     /**
159      * @dev Calculates the amount that has already vested.
160      * @param token ERC20 token which is being vested
161      */
162     function _vestedAmount(IERC20 token) private view returns (uint256) {
163         uint256 currentBalance = token.balanceOf(address(this));
164         uint256 totalBalance = currentBalance.add(_released[address(token)]);
165 
166         if (block.timestamp < _cliff) {
167             return 0;
168         } else if (block.timestamp >= _start.add(_duration) || _revoked[address(token)]) {
169             return totalBalance;
170         } else {
171             return totalBalance.mul(block.timestamp.sub(_start)).div(_duration);
172         }
173     }
174 
175      /**
176      * @return change the beneficiary of tokens
177      */
178 
179     /*@CTK OpenZeppelin_TokenVesting_changeBeneficiary
180       @tag assume_completion
181       @post __post._beneficiary == _newBeneficiary
182      */
183     function _changeBeneficiary(address _newBeneficiary) internal {
184         _beneficiary = _newBeneficiary;
185     }
186 }
```