CERTIK

Audit Report

Produced by CertiK

for

Nov 27, 2019

CertiK Audit Report For Metadium

Request Date: 2019-10-27
Revision Date: 2019-11-27
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 Metadium(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

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

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

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

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

Executive Summary

This report has been prepared for Metadium to discover issues and vulnerabilities in the source code of their Gov.sol, GovChecker.sol, GovImp.sol, Migrations.sol, Registry.sol, Staking.sol, AEnvStorage.sol, AGov.sol, BallotEnums.sol, EnvConstants.sol, IBallotStorage.sol, IEnvStorage.sol, IGov.sol, IRegistry.sol, IStaking.sol, OwnedUpgradeabilityProxy.sol, Proxy.sol, UpgradeabilityProxy.sol, BallotStorage.sol, EnvStorage.sol, EnvStorageImp.sol, EternalStorage.sol, Gov.sol, GovImp.sol, Registry.sol, Staking.sol, BallotStorage.sol, EnvStorage.sol and EnvStorageImp.sol smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

Vulnerability Classification

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

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

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

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

Testing Summary

PASS

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

Nov 27, 2019

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by miners to some degree.	5	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.	2	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by miners 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

Introduction

Metadium is an ecosystem built on a public identity blockchain. The Governance features of this smart contract suite let users submit proposals and vote on them. The only things people can vote to do is add or remove members of the governance contract, or to upgrade the governance contract.
As far as staking goes, the Staking contract allows people to lock up certain amounts of the native token. They cannot collect interest on it. The votes in the Governance contract are weighted based on how much the user has staked.

Source Code SHA-256 Checksum

- **Gov.sol**
  `f826cc19e0b4228fa2061cc9c29260c31441f898dc0dfe95404c8c6e94329aa9`
- **GovChecker.sol**
  `6cc382cb0aef827e8d926822026b36b9be36bffc97a31a0fab2f014c1b4b25dd`
- **GovImp.sol**
  `522103b71d7e9637f21d88f853e56c6916778500d548b73f4a3f05fd0570886a`
- **Migrations.sol**
  `75a9bd0836a685d837ddba44f7b53f0098810829b19f8cfb9f1a77393793be17`
- **Registry.sol**
  `ea7c1c646936c3e78509a2a9b1ecc07d995be387d479a144cdccdbed7e64da9a`
- **Staking.sol**
  `edefb7a2b1488d29d9eb1afd35b4b3e43d1288bc2c755212bed620fb42b78d63`
- **AEnvStorage.sol**
  `acf22a579e587431d12b49c39c908c5ec499d7f2df21ea55270feea347a0cfcf`
- **AGov.sol**
  `32d236aad898db012287d5dfabbd52792fe4e3493dffb68dee19f39f78cecf50`
- **BallotEnums.sol**
  `68f58519120f01a9e12049f31bd9f219f180844d824e9cf448e3dacc733bd4e0`
- **EnvConstants.sol**
  `b8928dfb95a26b9b979ba2c4e568a3a560548e44edc390166c6a9db9fe0dde48`
- **IBallotStorage.sol**
  `20fe8308cd4de6913dac56755f9e563f9acd8b2bee1b9128c3e8f28651fa56bc`
- **IEnvStorage.sol**
  `e041178b32332dbfe9e6f4a63adb528853e15dcaf26d6b0ed68db2dae6c702ab`
- **IGov.sol**
  `62db3f79cb3ba68049f21f5473f8743844f38887171a48cb4bd83f576f9877b9`

- **IRegistry.sol**
  `969242797a57f452cbd84233bba6e7a6707eab5cc7f0f3829e039cb7bdb4caba`
- **IStaking.sol**
  `1e921147fb82d676774cf7235225e5baf6def7e742b48f9eda12523bd0ca5d73`
- **OwnedUpgradeabilityProxy.sol**
  `586a716b15d807176a690f4e1c5c36a1474a870fbd0730f4a75c93b0e787a3b7`
- **Proxy.sol**
  `654b241a5d6bed272ff9219d1be239a0faf87e952a65f038fb0cf1e0b2485333`
- **UpgradeabilityProxy.sol**
  `f7f4e9a7de3a55e5d98af6f9b5aa92e4a138910456f9940fa450c39d0de1e018`
- **BallotStorage.sol**
  `02f0ceff7441df45f50feb948560ea601885010e678bbeb11959bb1b95712374`
- **EnvStorage.sol**
  `4212f34f888b6f332b761d74200ac21807d1a49a537efd1c62c4531112023716`
- **EnvStorageImp.sol**
  `2a0d17c0fddf197f62c01c5194928e4ada5820921e70856c349e7a05369bbec0`
- **EternalStorage.sol**
  `d3fa5ed420067bc419d812dee5cb01d6b6cc1de3d0782b665840e81d5a306194`
- **Gov.sol**
  `5da121d462d0fa4db01b502bbc181615bfe50b4b2630e7e829f60d8663edf230`
- **GovImp.sol**
  `bceeae820ebda1f56897500e6f2f0e32b1ed629661e826e59a1ba57ab70e8e6f`
- **Registry.sol**
  `13ca29f0b8325b2bcbe467b48fd0cf72997554c5cb876551cf7b3c692df4398e`
- **Staking.sol**
  `fbc9aaaffe6b86845187a394ab4af7bcfe04efe8a9ab20169253e6abcdfbaa40`
- **BallotStorage.sol**
  `391af4ad2fd107b9de466fab0e92ff07041d5dcb635705fe1f7bce6eac3f3b5e`
- **EnvStorage.sol**
  `c3deaa14344d0f18de5660006c1106b6c5ef4a3a52475574ca3906e0b0056fee`
- **EnvStorageImp.sol**
  `5227458039c6632ccfa5a32e79886e3743ba206042ca6953bbb8dec9785e002b`

Summary

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

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.

Recommendations

Round 3 Review Findings

GovImp.sol

- MINOR Modifier `nonReentrant`: Recommend changing to:

```
modifier nonReentrant() {
            require(!rentrancy_lock);
            rentrancy_lock = true;
            _;
            rentrancy_lock = false;
        }
```

  where default value of `rentrancy_lock` is false, according to Openzeppelin.

- MINOR `block.timestamp` used as a test condition.

BallotStorage.sol

- MINOR In function `createBallotForMember`: not necessary to wrap `_areMemberBallotParamValid` in a require statement.

- MINOR A number of functions are declared as `public` but are not called withing the contract. Consider changing to `external`.

Round 2 Review Findings

GovImp.sol

- CRITICAL It appears that it is only possible to have one ballot open for voting on at a time. This means that once one ballot has closed voting, users must race to submit the next ballot for voting as quickly as possible before some other ballot has blocked off the next chunk of time, potentially up to the maximum defined voting time. In addition to introducing an awkward user experience, it seems like severe DDoS/greifing vulnerabilities are possible, particularly by validating nodes who can

front-run transactions. We did not analyze the game theory for what attacks on the voting mechanism are possible but given that it seems easy to remove this logic and allow multiple ballots to be active at a time we recommend that this action is taken.

  - ✓ Metadium For Phase 1, only one ballot will be open at a time, but the addition of the `notApplicable` ballot state solves this issue. Allowing multiple ballots will be considered in Phase 2.

- MINOR `ballotLength` is redundant with `ballotCount` in the `BallotStorage` contract

  - ✓ Metadium Unchanged; variable is for ease of use.

BallotStorage.sol

- CRITICAL The `duration` field of a ballot never seems to be set. It's not set when a ballot is created, and the `updateBallotDuration` method which sets it seems to never be called. This will mean that every ballot's duration will be zero and voting will be impossible.

  - ✓ Metadium `updateBallotDuration` function will be called from a governance the DApp. A `duration` of zero means that the default duration is used.

- MINOR function `cancelBallot` never used

  - ✓ Metadium This function will be called from the governance DApp.

- MINOR `isFinalized` is redundant with `state`. `isFinalized` is equivalent to `state == BallotEnums.REJECTED || state == BallotEnums.ACCEPTED`

  - ✓ Metadium Unchanged; variable is for ease of use.

Round 1 Review Findings

- MINOR The code should be compiled with the latest version of the Solidity compiler, since there are several (albeit minor) bug fixes that have been implemented since version 0.4.24. Upgrading to version 5.x will introduce breaking changes and likely force some minor code rewriting. An alternative is to simply upgrade to version 0.4.26 which fixes as many bugs as possible in versions 0.4.x without introducing breaking changes.

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

- MINOR Different contracts use different versions of Solidity (some 0.4.16, some 0.4.24)

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

GovImp.sol

- MINOR `getThreshould` seems to be misspelled, should be `getThreshold`

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

- MINOR can get rid of `require(msg.sender != member, ``Cannot add self");` because it's redundant with `require(!isMember(member), ``Already member");`

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

- MINOR Include a check to forbid adding the zero address in `addProposalToAddMember`.

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

- MINOR SafeMath should perhaps be used on lines 228, 230, and 232 to avoid overflow

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

- MINOR The `unlock` function is never used

  - ✓ Metadium Function `transferLockedAndUnlock()` now uses `unlock()`

- MINOR `rewards`, `members`, and `nodes` seem like they all could be arrays rather than mappings. This would also allow for the length-tracking variables to be removed.

  - ✓ Metadium Not changed to minimize modifications

- MINOR The swapping logic in that begins on line 345 `removeMember()` seems uncessary. While swapping is necessary for deletion from an array in constant time, it is not necessary for deleting from a mapping.

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

- MAJOR In `removeMember()`, `rewardIdx[rewards[memberLength]] = 0` seems like it should be `rewardIdx[addr] = 0`. As is, the `rewardIdx` for the deleted member does not seem to actually be cleared, since `rewards[memberLength]` is set to `0x0` in the line immediately before.

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

- MINOR A `payable` fallback function with a `revert` in it similar to the one in `Staking.sol` should be included here, because it is currently possible to send Ether to this contract with no way to withdraw it.

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

Staking.sol

- MINOR It's possible for an `addMember` ballot to be indefinitely prevented from finalizing due to `lock` reverting, if a user has failed to lock up the required funds. This is perhaps fine depending on what the guarantees of the system are.

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

- MINOR The first `require` statement in `lock` is redundant with the second, since `_balance[payee]`is greater than or equal to `availableBalanceOf(payee)`.
  - ✓ Metadium The code is updated and reflected in the latest commit.

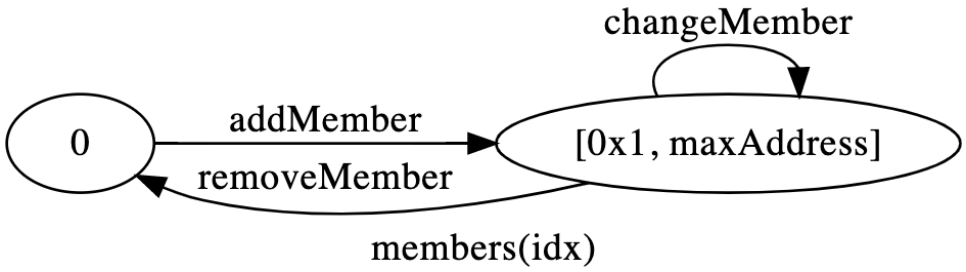
BallotStorage.sol

- MINOR A lot of the `require`s in `createVote` are redundant with the `require`s in `_updateBallotForVote`
  - ✓ Metadium The code is updated and reflected in the latest commit.
- MINOR Functions `createBallotForAddress` and `createBallotForVariable` return `_id` which is neither used nor necessary (it's passed in as an argument and is not modified). `createVote` has a return value that is also never returned.
  - ✓ Metadium The code is updated and reflected in the latest commit.

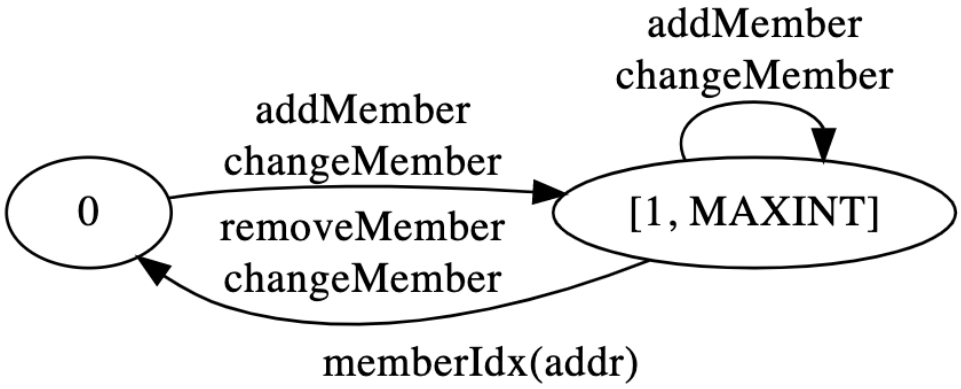
Diagrams

VoteImp.sol

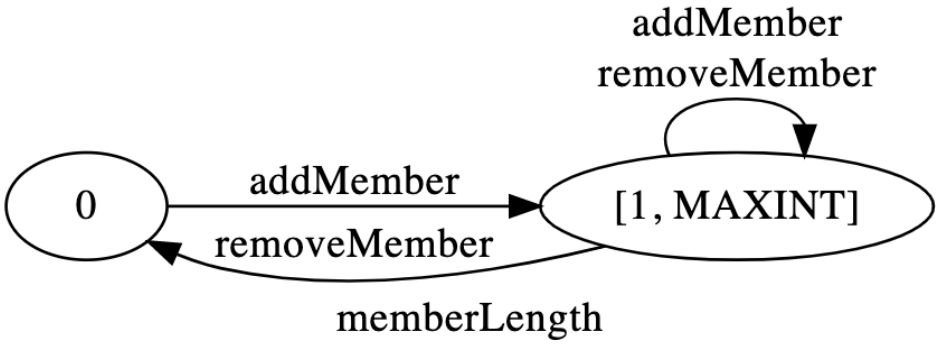
members(idx)



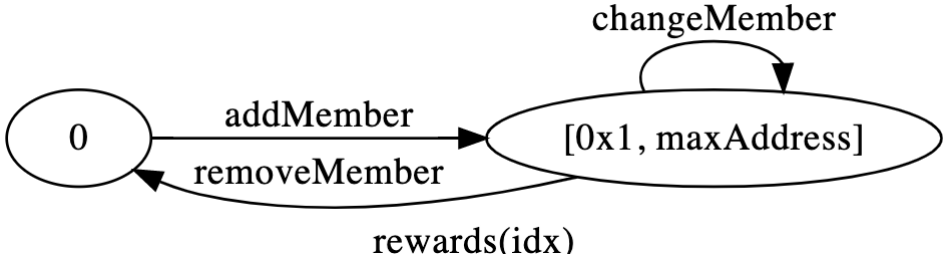
memberIdx(addr)



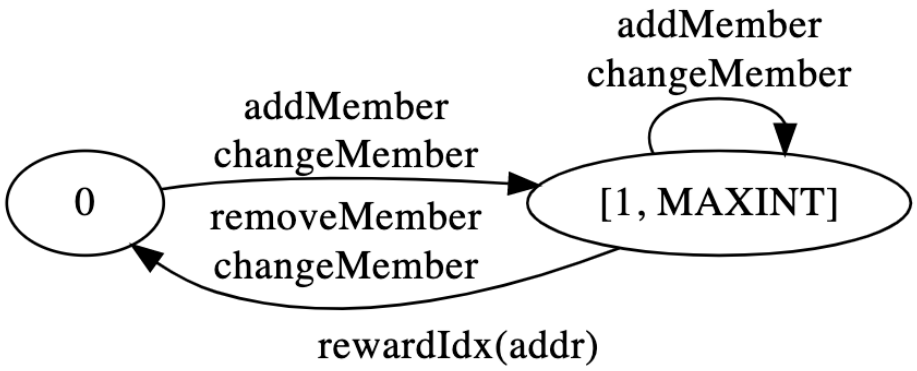
memberLength



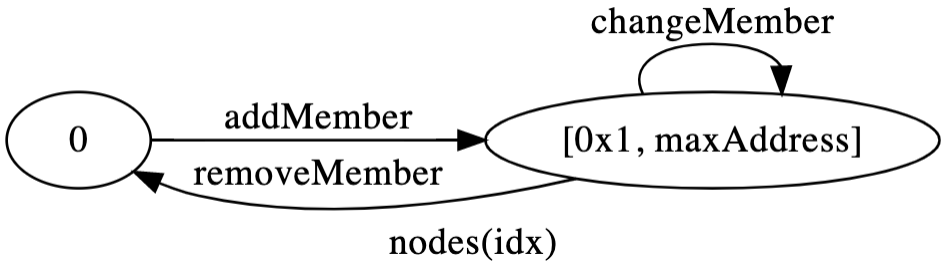
rewards(idx)



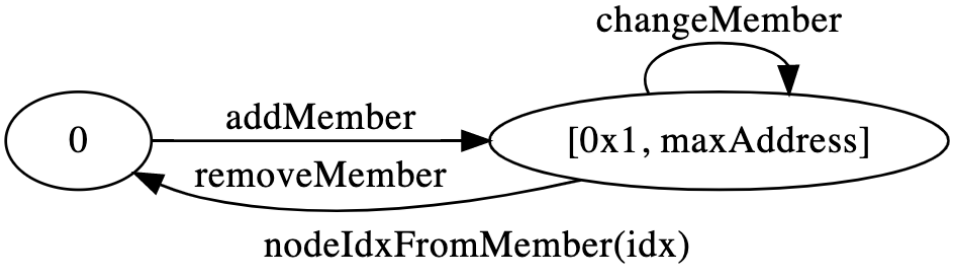
rewardIdx(addr)



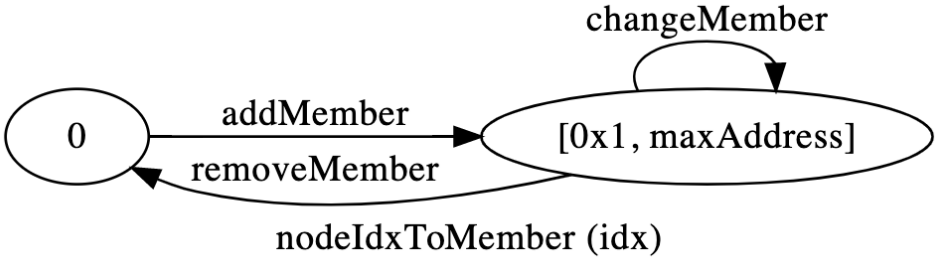
nodes(idx)



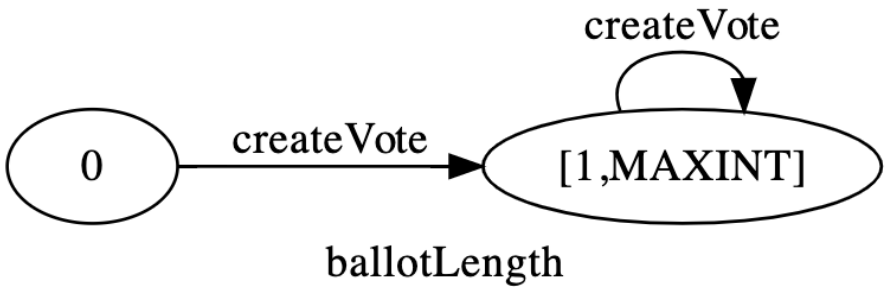
nodeIdxFromMember(idx)



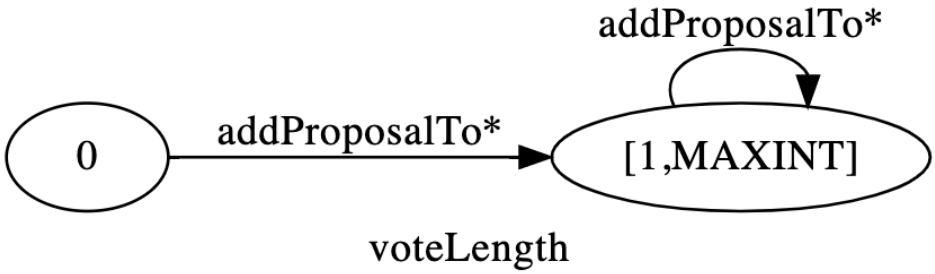
nodeIdxToMember(idx)



ballotLength

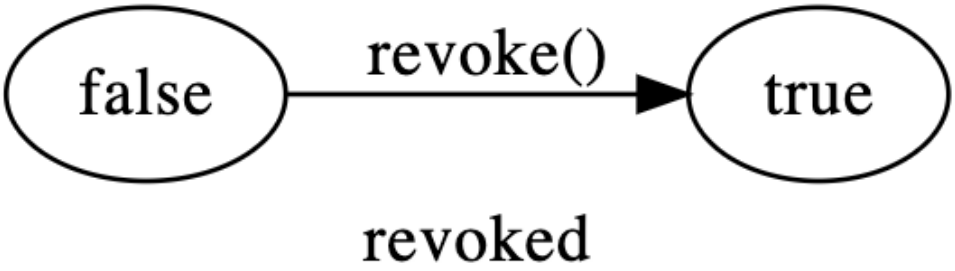


voteLength

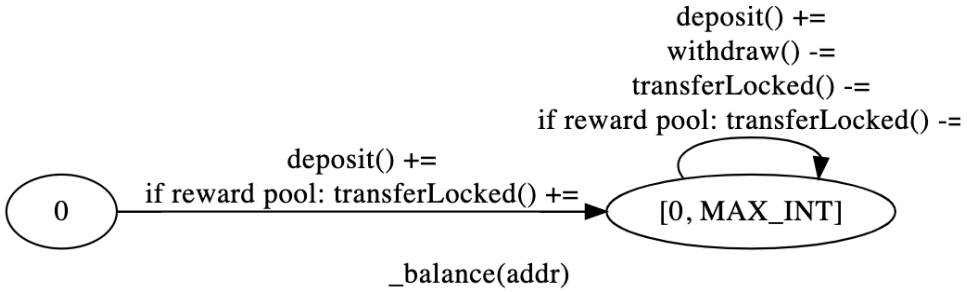


Staking.sol

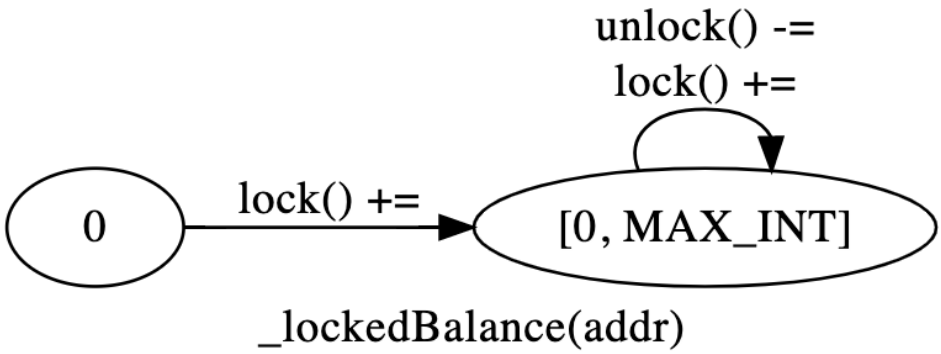
revoked



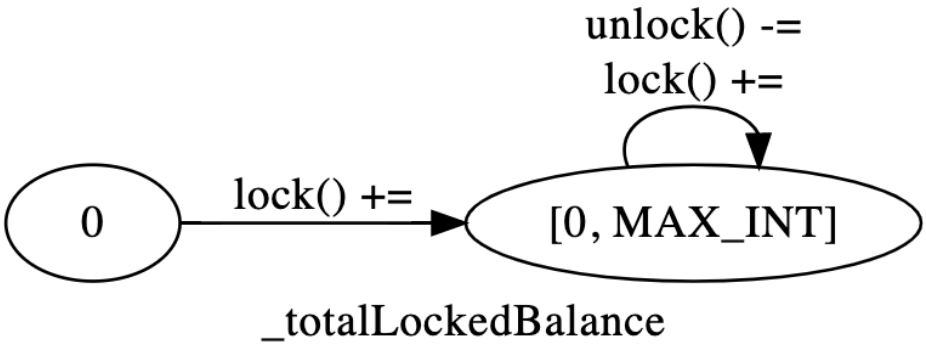
_balance(addr)



_lockedBalance(addr)

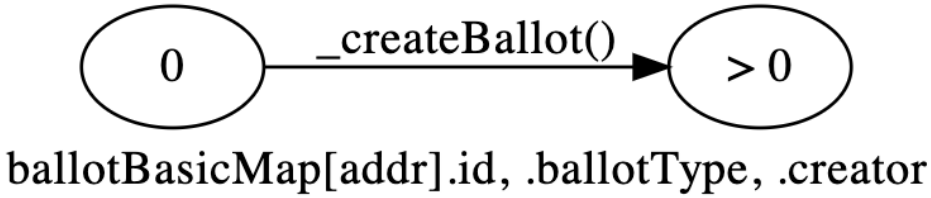


_totalLockedBalance

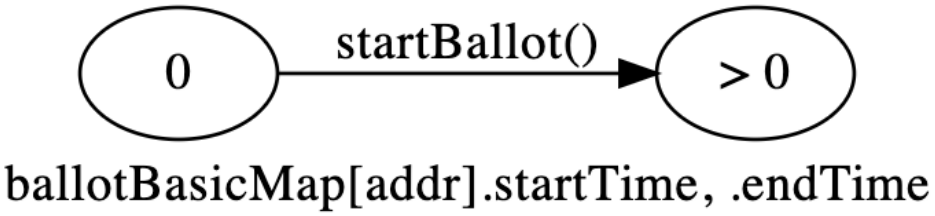


BallotStorage

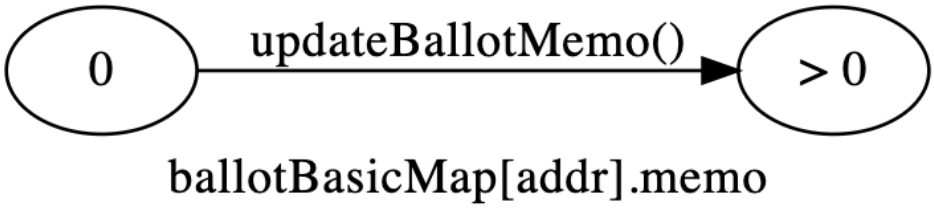
ballotBasicMap[addr].id, .ballotType, .creator



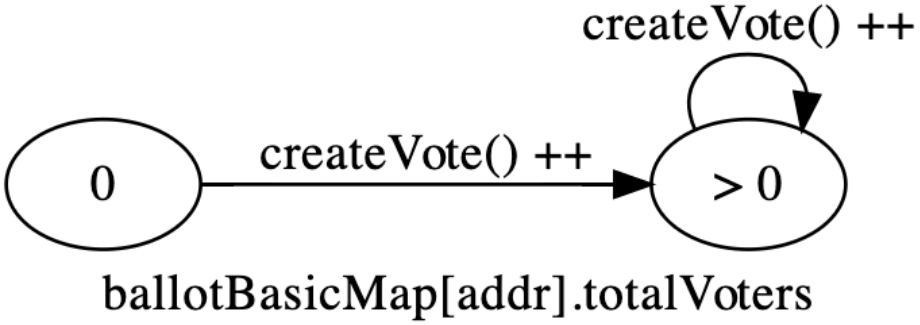
ballotBasicMap[addr].startTime, .endTime



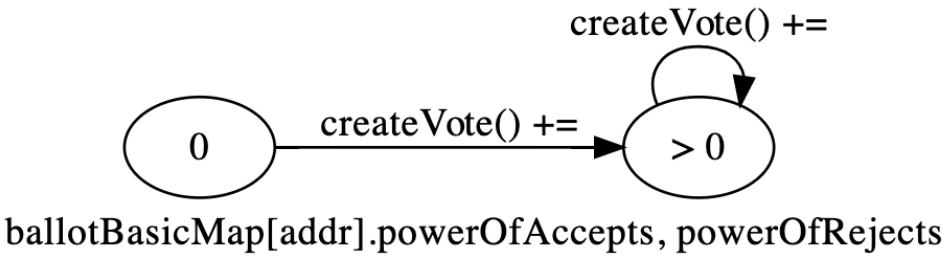
ballotBasicMap[addr].memo



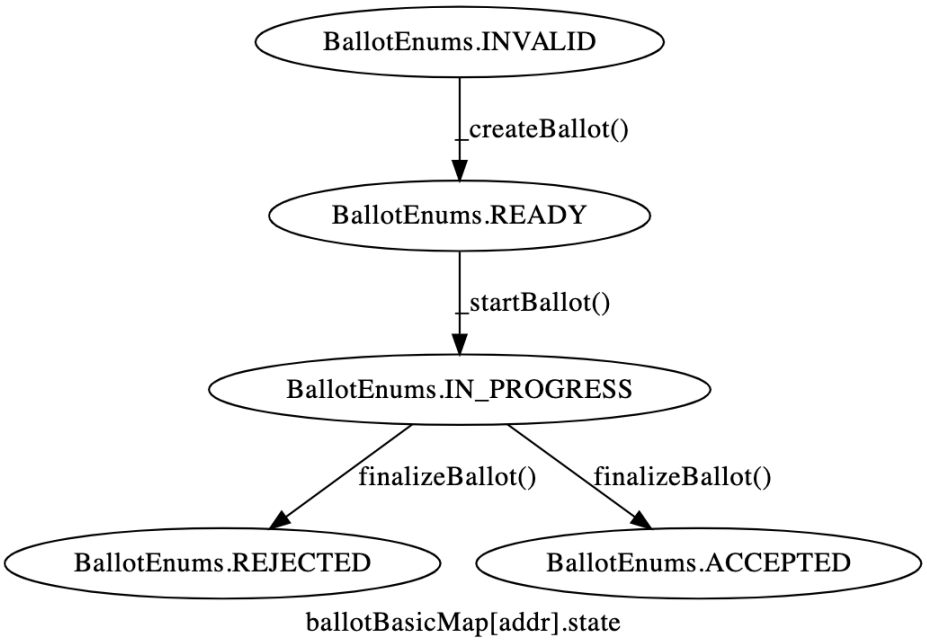
ballotBasicMap[addr].totalVoters



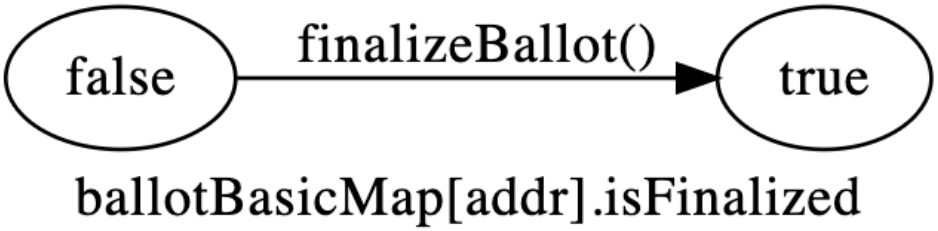
ballotBasicMap[addr].powerOfAccepts, powerOfRejects



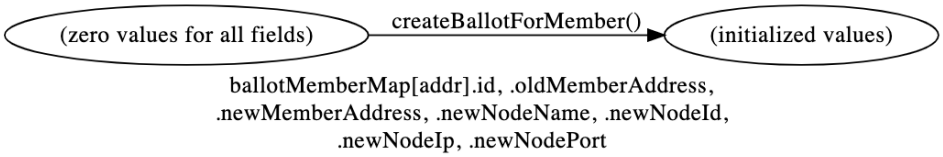
ballotBasicMap[addr].state



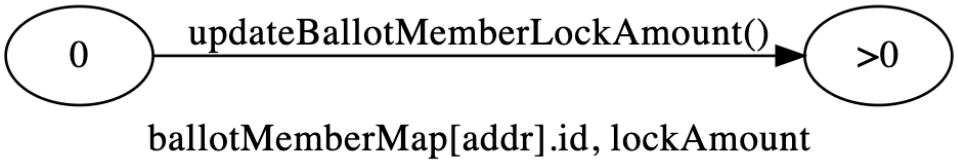
ballotBasicMap[addr].isFinalized



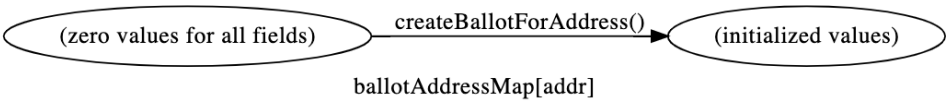
ballotMemberMap[addr].id, .oldMemberAddress, .newMemberAddress, .newNodeName, .newNodeId, .newNodeIp, .newNodePort



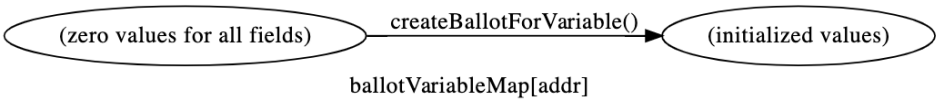
ballotMemberMap[addr].id, lockAmount



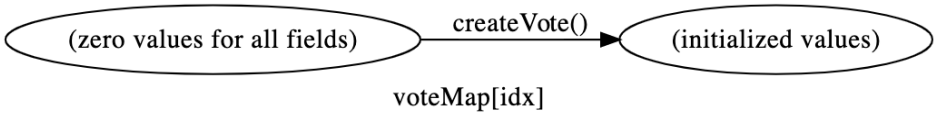
ballotAddressMap[addr]



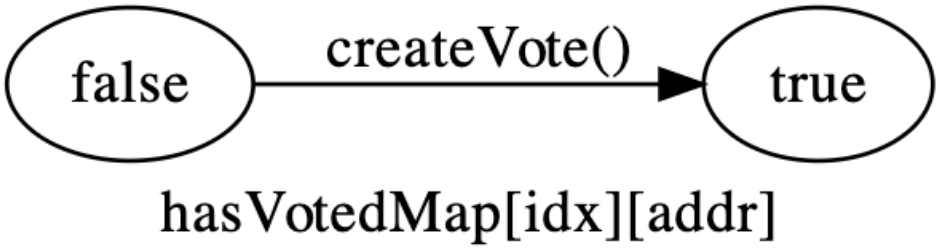
ballotVariableMap[addr]



voteMap[idx]



hasVotedMap[idx][addr]



hasVotedMap[idx][addr]

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File fv.sol

```
1 pragma solidity ^0.4.26;
```

Version to compile has the following bug: 0.4.26: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2

TIMESTAMP_DEPENDENCY

Line 517 in File fv.sol

```
517             if (endTime < block.timestamp) {
```

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

TIMESTAMP_DEPENDENCY

Line 536 in File fv.sol

```
536             startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                    getMinVotingDuration()));
```

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

TIMESTAMP_DEPENDENCY

Line 538 in File fv.sol

```
538             startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                    getMaxVotingDuration()));
```

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

TIMESTAMP_DEPENDENCY

Line 540 in File fv.sol

```
540             startBallot(ballotIdx, block.timestamp, block.timestamp.add(duration));
```

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

TIMESTAMP_DEPENDENCY

Line 1294 in File fv.sol

```
1294         return now;
```

"now" can be influenced by miners to some degree

INSECURE_COMPILER_VERSION

Line 1 in File enum.sol

```
1 pragma solidity ^0.4.18;
```

Version to compile has the following bug: 0.4.18: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.19: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.20: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.21: SignedArrayStorageCopy, ABIEncoderV2StorageArrayDynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder 0.4.22: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, OneOfTwoConstructorsSkipped 0.4.23: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData 0.4.24: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData 0.4.25: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x 0.4.26: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2

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

SafeMath mul zero

27, Nov 2019
20.74 ms

Line 8-13 in File fv.sol

```
8   /*@CTK "SafeMath mul zero"
9       @tag spec
10      @tag is_pure
11      @pre (a == 0)
12      @post __return == 0
13    */
```

Line 24-36 in File fv.sol

```
24  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
25    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
26    // benefit is lost if 'b' is also tested.
27    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
28    if (a == 0) {
29      return 0;
30    }
31
32    uint256 c = a * b;
33    require(c / a == b);
34
35    return c;
36  }
```

The code meets the specification.

Formal Verification Request 2

SafeMath mul nonzero

27, Nov 2019
179.82 ms

Line 14-23 in File fv.sol

```
14    /*@CTK "SafeMath mul nonzero"
15      @tag spec
16      @tag is_pure
17      @pre (a != 0)
18      @post (a * b / a != b) == __reverted
19      @post !__reverted -> __return == a * b
20      @post !__reverted -> !__has_overflow
21      @post !__reverted -> !__has_assertion_failure
22      @post !(__has_buf_overflow)
23    */
```

Line 24-36 in File fv.sol

```
24  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
25    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
26    // benefit is lost if 'b' is also tested.
```

```
27    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
28    if (a == 0) {
29      return 0;
30    }
31
32    uint256 c = a * b;
33    require(c / a == b);
34
35    return c;
36  }
```

The code meets the specification.

Formal Verification Request 3

SafeMath div

27, Nov 2019
15.96 ms

Line 41-49 in File fv.sol

```
41  /*@CTK "SafeMath div"
42      @tag spec
43      @tag is_pure
44      @post (b == 0) == __reverted
45      @post !__reverted -> __return == a / b
46      @post !__reverted -> !__has_overflow
47      @post !__reverted -> !__has_assertion_failure
48      @post !(__has_buf_overflow)
49    */
```

Line 50-56 in File fv.sol

```
50  function div(uint256 a, uint256 b) internal pure returns (uint256) {
51    require(b > 0); // Solidity only automatically asserts when dividing by 0
52    uint256 c = a / b;
53    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
54
55    return c;
56  }
```

The code meets the specification.

Formal Verification Request 4

SafeMath sub

27, Nov 2019
14.96 ms

Line 61-69 in File fv.sol

```
61  /*@CTK "SafeMath sub"
62      @tag spec
63      @tag is_pure
```

```
64      @post (b > a) == __reverted
65      @post !__reverted -> __return == a - b
66      @post !__reverted -> !__has_overflow
67      @post !__reverted -> !__has_assertion_failure
68      @post !(__has_buf_overflow)
69    */
```

Line 70-75 in File fv.sol

```
70  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
71    require(b <= a);
72    uint256 c = a - b;
73
74    return c;
75  }
```

The code meets the specification.

Formal Verification Request 5

SafeMath mod

27, Nov 2019
14.51 ms

Line 91-99 in File fv.sol

```
91  /*@CTK "SafeMath mod"
92      @tag spec
93      @tag is_pure
94      @post (b == 0) == __reverted
95      @post !__reverted -> __return == a % b
96      @post !__reverted -> !__has_overflow
97      @post !__reverted -> !__has_assertion_failure
98      @post !(__has_buf_overflow)
99    */
```

Line 100-103 in File fv.sol

```
100 function mod(uint256 a, uint256 b) internal pure returns (uint256) {
101   require(b != 0);
102   return a % b;
103 }
```

The code meets the specification.

Formal Verification Request 6

isMember

27, Nov 2019
6.49 ms

Line 213-215 in File fv.sol

```
213   /*@CTK isMember
214     @post __return == (memberIdx[addr] != 0)
215   */
```

Line 216 in File fv.sol

```
216     function isMember(address addr) public view returns (bool) { return (memberIdx[
            addr] != 0); }
```

The code meets the specification.

Formal Verification Request 7

getMember

27, Nov 2019
6.3 ms

Line 217-219 in File fv.sol

```
217     /*@CTK getMember
218       @post __return == members[idx]
219     */
```

Line 220 in File fv.sol

```
220     function getMember(uint256 idx) public view returns (address) { return members[idx
            ]; }
```

The code meets the specification.

Formal Verification Request 8

getMemberLength

27, Nov 2019
7.25 ms

Line 221-223 in File fv.sol

```
221     /*@CTK getMemberLength
222       @post __return == memberLength
223     */
```

Line 224 in File fv.sol

```
224     function getMemberLength() public view returns (uint256) { return memberLength; }
```

The code meets the specification.

Formal Verification Request 9

getReward

27, Nov 2019
5.9 ms

Line 225-227 in File fv.sol

```
225     /*@CTK getReward
226       @post __return == rewards[idx]
227     */
```

Line 228 in File fv.sol

```
228     function getReward(uint256 idx) public view returns (address) { return rewards[idx
            ]; }
```

The code meets the specification.

Formal Verification Request 10

getNodeIdxFromMember

27, Nov 2019
6.43 ms

Line 229-231 in File fv.sol

```
229     /*@CTK getNodeIdxFromMember
230       @post __return == nodeIdxFromMember[addr]
231     */
```

Line 232 in File fv.sol

```
232     function getNodeIdxFromMember(address addr) public view returns (uint256) { return
            nodeIdxFromMember[addr]; }
```

The code meets the specification.

Formal Verification Request 11

getMemberFromNodeIdx

27, Nov 2019
7.02 ms

Line 233-235 in File fv.sol

```
233     /*@CTK getMemberFromNodeIdx
234       @post __return == nodeToMember[idx]
235     */
```

Line 236 in File fv.sol

```
236     function getMemberFromNodeIdx(uint256 idx) public view returns (address) { return
            nodeToMember[idx]; }
```

The code meets the specification.

Formal Verification Request 12

getNodeLength

27, Nov 2019
5.79 ms

Line 237-239 in File fv.sol

```
237    /*@CTK getNodeLength
238      @post __return == nodeLength
239    */
```

Line 240 in File fv.sol

```
240    function getNodeLength() public view returns (uint256) { return nodeLength; }
```

The code meets the specification.

Formal Verification Request 13

If method completes, integer overflow would not happen.

27, Nov 2019
11.74 ms

Line 242 in File fv.sol

```
242    //@CTK NO_OVERFLOW
```

Line 245-247 in File fv.sol

```
245    function getNode(uint256 idx) public view returns (bytes name, bytes enode, bytes
           ip, uint port) {
246        return (nodes[idx].name, nodes[idx].enode, nodes[idx].ip, nodes[idx].port);
247    }
```

The code meets the specification.

Formal Verification Request 14

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

27, Nov 2019
0.4 ms

Line 243 in File fv.sol

```
243    //@CTK NO_BUF_OVERFLOW
```

Line 245-247 in File fv.sol

```
245    function getNode(uint256 idx) public view returns (bytes name, bytes enode, bytes
           ip, uint port) {
246        return (nodes[idx].name, nodes[idx].enode, nodes[idx].ip, nodes[idx].port);
247    }
```

The code meets the specification.

Formal Verification Request 15

Method will not encounter an assertion failure.

27, Nov 2019
0.41 ms

Line 244 in File fv.sol

```
244     //@CTK NO_ASF
```

Line 245-247 in File fv.sol

```
245     function getNode(uint256 idx) public view returns (bytes name, bytes enode, bytes
            ip, uint port) {
246         return (nodes[idx].name, nodes[idx].enode, nodes[idx].ip, nodes[idx].port);
247     }
```

The code meets the specification.

Formal Verification Request 16

If method completes, integer overflow would not happen.

27, Nov 2019
6.2 ms

Line 249 in File fv.sol

```
249     //@CTK NO_OVERFLOW
```

Line 252 in File fv.sol

```
252     function getBallotInVoting() public view returns (uint256) { return ballotInVoting
            ; }
```

The code meets the specification.

Formal Verification Request 17

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

27, Nov 2019
0.37 ms

Line 250 in File fv.sol

```
250     //@CTK NO_BUF_OVERFLOW
```

Line 252 in File fv.sol

```
252     function getBallotInVoting() public view returns (uint256) { return ballotInVoting
            ; }
```

The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.

27, Nov 2019
0.42 ms

Line 251 in File fv.sol

```
251     //@CTK NO_ASF
```

Line 252 in File fv.sol

```
252     function getBallotInVoting() public view returns (uint256) { return ballotInVoting
            ; }
```

The code meets the specification.

Formal Verification Request 19

addProposalToAddMember

27, Nov 2019
137.31 ms

Line 266-275 in File fv.sol

```
266     /*@CTK addProposalToAddMember
267       @tag assume_completion
268       @pre member != address(0)
269       @pre name.length > 0
270       @pre ip.length > 0
271       @pre portNlockAmount[0] > 0
272       @pre portNlockAmount[1] > 0
273       @pre memberIdx[member] == 0
274       @post __post.ballotLength == ballotLength + 1
275     */
```

Line 276-311 in File fv.sol

```
276     function addProposalToAddMember(
277         address member,
278         bytes name,
279         bytes enode,
280         bytes ip,
281         uint256[2] portNlockAmount,
282         bytes memo
283     )
284         external
285         returns (uint256 ballotIdx)
286     {
287         require(member != address(0), "Invalid address");
288         require(name.length > 0, "Invalid node name");
289         require(ip.length > 0, "Invalid node IP");
290         require(portNlockAmount[0] > 0, "Invalid node port");
291         require(portNlockAmount[1] > 0, "Invalid lockAmmount");
292         require(!isMember(member), "Already member");
293 
294         ballotIdx = ballotLength.add(1);
```

```
295         createBallotForMember(
296             ballotIdx, // ballot id
297             uint256(BallotTypes.MemberAdd), // ballot type
298             msg.sender, // creator
299             address(0), // old member address
300             member, // new member address
301             name,
302             enode, // new enode
303             ip, // new ip
304             portNlockAmount[0] // new port
305         );
306         updateBallotLock(ballotIdx, portNlockAmount[1]);
307         updateBallotMemo(ballotIdx, memo);
308         ballotLength = ballotIdx;
309     }
```

The code meets the specification.

Formal Verification Request 20

addProposalToRemoveMember

27, Nov 2019
92.08 ms

Line 314-319 in File fv.sol

```
314     /*@CTK addProposalToRemoveMember
315       @pre member != address(0)
316       @pre member == address(0)
317       @pre memberLength > 1
318       @post __post.ballotLength == ballotLength + 1
319     */
```

Line 320-349 in File fv.sol

```
320     function addProposalToRemoveMember(
321         address member,
322         uint256 lockAmount,
323         bytes memo
324     )
325         external
326         returns (uint256 ballotIdx)
327     {
328         require(member != address(0), "Invalid address");
329         require(isMember(member), "Non-member");
330         require(getMemberLength() > 1, "Cannot remove a sole member");
331 
332         ballotIdx = ballotLength.add(1);
333         createBallotForMember(
334             ballotIdx, // ballot id
335             uint256(BallotTypes.MemberRemoval), // ballot type
336             msg.sender, // creator
337             member, // old member address
338             address(0), // new member address
339             new bytes(0), // new name
340             new bytes(0), // new enode
341             new bytes(0), // new ip
```

```
            0 // new port
        );
        updateBallotLock(ballotIdx, lockAmount);
        updateBallotMemo(ballotIdx, memo);
        ballotLength = ballotIdx;
    }
```

The code meets the specification.

Formal Verification Request 21

addProposalToChangeMember

27, Nov 2019
117.05 ms

Line 351-360 in File fv.sol

```
    /*@CTK addProposalToChangeMember
      @pre targetNnewMember[0] != address(0)
      @pre targetNnewMember[1] != address(0)
      @pre nName.length > 0
      @pre nIp.length > 0
      @pre portNlockAmount[0] > 0
      @pre portNlockAmount[1] > 0
      @pre targetNnewMember[0] == address(0)
      @post __post.ballotLength == ballotLength + 1
    */
```

Line 361-397 in File fv.sol

```
    function addProposalToChangeMember(
        address[2] targetNnewMember,
        bytes nName,
        bytes nEnode,
        bytes nIp,
        uint256[2] portNlockAmount,
        bytes memo
    )
        external
        returns (uint256 ballotIdx)
    {
        require(targetNnewMember[0] != address(0), "Invalid old Address");
        require(targetNnewMember[1] != address(0), "Invalid new Address");
        require(nName.length > 0, "Invalid node name");
        require(nIp.length > 0, "Invalid node IP");
        require(portNlockAmount[0] > 0, "Invalid node port");
        require(portNlockAmount[1] > 0, "Invalid lockAmmount");
        require(isMember(targetNnewMember[0]), "Non-member");

        ballotIdx = ballotLength.add(1);
        createBallotForMember(
            ballotIdx, // ballot id
            uint256(BallotTypes.MemberChange), // ballot type
            msg.sender, // creator
            targetNnewMember[0], // old member address
            targetNnewMember[1], // new member address
            nName, //new Name
```

```
388             nEnode, // new enode
389             nIp, // new ip
390             portNlockAmount[0] // new port
391         );
392         updateBallotLock(ballotIdx, portNlockAmount[1]);
393         updateBallotMemo(ballotIdx, memo);
394         ballotLength = ballotIdx;
395     }
```

The code meets the specification.

Formal Verification Request 22

addProposalToChangeGov

27, Nov 2019
54.57 ms

Line 399-403 in File fv.sol

```
399     /*@CTK "addProposalToChangeGov"
400       @tag assume_completion
401       @pre newGovAddr != address(0)
402       @post __post.ballotLength == ballotLength + 1
403     */
```

Line 405-428 in File fv.sol

```
405     function addProposalToChangeGov(
406         address newGovAddr,
407         bytes memo
408     )
409         external
410         returns (uint256 ballotIdx)
411     {
412         require(newGovAddr != address(0), "Implementation cannot be zero");
413         require(newGovAddr != implementation(), "Same contract address");
414 
415         ballotIdx = ballotLength.add(1);
416         IBallotStorage(getBallotStorageAddress()).createBallotForAddress(
417             ballotLength.add(1), // ballot id
418             uint256(BallotTypes.GovernanceChange), // ballot type
419             msg.sender, // creator
420             newGovAddr // new governance address
421         );
422         updateBallotMemo(ballotIdx, memo);
423         ballotLength = ballotIdx;
424     }
```

The code meets the specification.

Formal Verification Request 23

If method completes, integer overflow would not happen.

27, Nov 2019
2.66 ms

Line 404 in File fv.sol

```
404     //@CTK NO_OVERFLOW
```

Line 405-428 in File fv.sol

```
405     function addProposalToChangeGov(
406         address newGovAddr,
407         bytes memo
408     )
409         external
410         returns (uint256 ballotIdx)
411     {
412         require(newGovAddr != address(0), "Implementation cannot be zero");
413         require(newGovAddr != implementation(), "Same contract address");
414 
415         ballotIdx = ballotLength.add(1);
416         IBallotStorage(getBallotStorageAddress()).createBallotForAddress(
417             ballotLength.add(1), // ballot id
418             uint256(BallotTypes.GovernanceChange), // ballot type
419             msg.sender, // creator
420             newGovAddr // new governance address
421         );
422         updateBallotMemo(ballotIdx, memo);
423         ballotLength = ballotIdx;
424     }
```

The code meets the specification.

Formal Verification Request 24

addProposalToChangeEnv

27, Nov 2019
56.56 ms

Line 431-435 in File fv.sol

```
431     /*@CTK "addProposalToChangeEnv"
432      @tag assume_completion
433      @pre 1 <= envType && envType <= 6
434      @post __post.ballotLength == ballotLength + 1
435     */
```

Line 437-468 in File fv.sol

```
437     function addProposalToChangeEnv(
438         bytes32 envName,
439         uint256 envType,
440         bytes envVal,
441         bytes memo
442     )
443         external
444         returns (uint256 ballotIdx)
445     {
446         // require(envName != 0, "Invalid name");
447         require(uint256(VariableTypes.Int) <= envType && envType <= uint256(
            VariableTypes.String), "Invalid type");
```

```
448         //ctk start hacking
449         require(1 <= envType && envType <= 6, "Invalid type");
450         //ctk end hacking
451
452
453         ballotIdx = ballotLength.add(1);
454         __ballotStoragecreateBallotForVariable(
455             ballotIdx, // ballot id
456             uint256(BallotTypes.EnvValChange), // ballot type
457             msg.sender, // creator
458             envName, // env name
459             envType, // env type
460             envVal // env value
461         );
462         updateBallotMemo(ballotIdx, memo);
463         ballotLength = ballotIdx;
464     }
```

✅ The code meets the specification.

Formal Verification Request 25

If method completes, integer overflow would not happen.

📅 27, Nov 2019
⏱ 3.49 ms

Line 436 in File fv.sol

```
436     //@CTK NO_OVERFLOW
```

Line 437-468 in File fv.sol

```
437     function addProposalToChangeEnv(
438         bytes32 envName,
439         uint256 envType,
440         bytes envVal,
441         bytes memo
442     )
443         external
444         returns (uint256 ballotIdx)
445     {
446         // require(envName != 0, "Invalid name");
447         require(uint256(VariableTypes.Int) <= envType && envType <= uint256(
                VariableTypes.String), "Invalid type");
448         //ctk start hacking
449         require(1 <= envType && envType <= 6, "Invalid type");
450         //ctk end hacking
451
452
453         ballotIdx = ballotLength.add(1);
454         __ballotStoragecreateBallotForVariable(
455             ballotIdx, // ballot id
456             uint256(BallotTypes.EnvValChange), // ballot type
457             msg.sender, // creator
458             envName, // env name
459             envType, // env type
460             envVal // env value
```

```
461         );
462         updateBallotMemo(ballotIdx, memo);
463         ballotLength = ballotIdx;
464     }
```

The code meets the specification.

Formal Verification Request 26

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

27, Nov 2019
22510.1 ms

Line 469 in File fv.sol

```
469     //@CTK NO_BUF_OVERFLOW
```

Line 470-491 in File fv.sol

```
470     function vote(uint256 ballotIdx, bool approval) private nonReentrant {
471         // Check if some ballot is in progress
472         checkUnfinalized(ballotIdx);
473 
474         // Check if the ballot can be voted
475         uint256 ballotType = checkVotable(ballotIdx);
476 
477         // Vote
478         createVote(ballotIdx, approval);
479 
480         // Finalize
481        // (, uint256 accept, uint256 reject) = getBallotVotingInfo(ballotIdx);
482 
483         BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
484         uint256 accept = tBallot.powerOfAccepts;
485         uint256 reject = tBallot.powerOfRejects;
486 
487         uint256 threshold = getThreshold();
488         if (accept >= threshold || reject >= threshold) {
489             finalizeVote(ballotIdx, ballotType, accept > reject);
490         }
491     }
```

The code meets the specification.

Formal Verification Request 27

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

27, Nov 2019
18.93 ms

Line 511 in File fv.sol

```
511     //@CTK NO_BUF_OVERFLOW
```

Line 512-525 in File fv.sol

```
512     function checkUnfinalized(uint256 ballotIdx) private {
513         if (ballotInVoting != 0) {
514             uint256 state = __ballotStorageballotBasicMap[ballotInVoting].state;
515             uint256 endTime = __ballotStorageballotBasicMap[ballotInVoting].endTime;
516             if (state == uint256(BallotStates.InProgress)) {
517                 if (endTime < block.timestamp) {
518                     finalizeBallot(ballotInVoting, uint256(BallotStates.Rejected));
519                     ballotInVoting = 0;
520                 } else if (ballotIdx != ballotInVoting) {
521                     require(false, "Now in voting with different ballot");
522                 }
523             }
524         }
525     }
```

The code meets the specification.

Formal Verification Request 28

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

27, Nov 2019
45.22 ms

Line 527 in File fv.sol

```
527     //@CTK NO_BUF_OVERFLOW
```

Line 528-549 in File fv.sol

```
528     function checkVotable(uint256 ballotIdx) private returns (uint256) {
529         BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
530         uint256 ballotType = tBallot.ballotType;
531         uint256 state = tBallot.state;
532 
533         if (state == uint256(BallotStates.Ready)) {
534             uint256 duration = __ballotStorageballotBasicMap[ballotInVoting].duration;
535             if (duration < getMinVotingDuration()) {
536                 startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                        getMinVotingDuration()));
537             } else if (getMaxVotingDuration() < duration) {
538                 startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                        getMaxVotingDuration()));
539             } else {
540                 startBallot(ballotIdx, block.timestamp, block.timestamp.add(duration));
541             }
542             ballotInVoting = ballotIdx;
543         } else if (state == uint256(BallotStates.InProgress)) {
544             // Nothing to do
545         } else {
546             require(false, "Expired");
547         }
548         return ballotType;
549     }
```

The code meets the specification.

Formal Verification Request 29

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

27, Nov 2019
94.62 ms

Line 551 in File fv.sol

```
551     //@CTK NO_BUF_OVERFLOW
```

Line 552-573 in File fv.sol

```
552     function createVote(uint256 ballotIdx, bool approval) private {
553         uint256 voteIdx = voteLength.add(1);
554         uint256 weight = __stakingcalcVotingWeightWithScaleFactor(msg.sender, 1e4);
555         if (approval) {
556             __ballotStoragecreateVote(
557                 voteIdx,
558                 ballotIdx,
559                 msg.sender,
560                 uint256(DecisionTypes.Accept),
561                 weight
562             );
563         } else {
564             __ballotStoragecreateVote(
565                 voteIdx,
566                 ballotIdx,
567                 msg.sender,
568                 uint256(DecisionTypes.Reject),
569                 weight
570             );
571         }
572         voteLength = voteIdx;
573     }
```

The code meets the specification.

Formal Verification Request 30

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

27, Nov 2019
2951.16 ms

Line 575 in File fv.sol

```
575     //@CTK NO_BUF_OVERFLOW
```

Line 576-598 in File fv.sol

```
576     function finalizeVote(uint256 ballotIdx, uint256 ballotType, bool isAccepted)
            private {
577         uint256 ballotState = uint256(BallotStates.Rejected);
578         if (isAccepted) {
579             ballotState = uint256(BallotStates.Accepted);
580             if (ballotType == uint256(BallotTypes.MemberAdd)) {
581                 if (!addMember(ballotIdx)){
582                     ballotState = uint256(BallotStates.NotApplicable);
```

```
583                 }
584             } else if (ballotType == uint256(BallotTypes.MemberRemoval)) {
585                 removeMember(ballotIdx);
586             } else if (ballotType == uint256(BallotTypes.MemberChange)) {
587                 if(!changeMember(ballotIdx)){
588                     ballotState = uint256(BallotStates.NotApplicable);
589                 }
590             } else if (ballotType == uint256(BallotTypes.GovernanceChange)) {
591                 changeGov(ballotIdx);
592             } else if (ballotType == uint256(BallotTypes.EnvValChange)) {
593                 applyEnv(ballotIdx);
594             }
595         }
596         finalizeBallot(ballotIdx, ballotState);
597         ballotInVoting = 0;
598     }
```

✅ The code meets the specification.

Formal Verification Request 31

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

📅 27, Nov 2019
⏱ 5.44 ms

Line 600 in File fv.sol

```
600     //@CTK NO_BUF_OVERFLOW
```

Line 601-611 in File fv.sol

```
601     function fromValidBallot(uint256 ballotIdx, uint256 targetType) private view {
602         BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
603         uint256 ballotType = tBallot.ballotType;
604         uint256 state = tBallot.state;
605         require(ballotType == targetType, "Invalid voting type");
606         require(state == uint(BallotStates.InProgress), "Invalid voting state");
607         BallotBasic memory tBallot2 = __ballotStorageballotBasicMap[ballotIdx];
608         uint256 accept = tBallot2.powerOfAccepts;
609         uint256 reject = tBallot2.powerOfRejects;
610         require(accept >= getThreshold() || reject >= getThreshold(), "Not yet
                finalized");
611     }
```

✅ The code meets the specification.

Formal Verification Request 32

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

📅 27, Nov 2019
⏱ 65.31 ms

Line 612 in File fv.sol

```
612     //@CTK NO_BUF_OVERFLOW
```

Line 613-664 in File fv.sol

```
613     function addMember(uint256 ballotIdx) private returns (bool) {
614         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberAdd));
615 
616         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
617         address addr = __ballotStorageballotMemberMap[ballotIdx].newMemberAddress;
618         bytes memory name = __ballotStorageballotMemberMap[ballotIdx].newNodeName;
619         bytes memory enode = __ballotStorageballotMemberMap[ballotIdx].newNodeId;
620         bytes memory ip = __ballotStorageballotMemberMap[ballotIdx].newNodeIp;
621         uint port = __ballotStorageballotMemberMap[ballotIdx].newNodePort;
622         uint256 lockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
623 
624         if (isMember(addr)) {
625             emit NotApplicable(ballotIdx, "Already a member");
626             return true; // Already member. it is abnormal case, but passed.
627         }
628 
629         // Lock
630         // require(getMinStaking() <= lockAmount && lockAmount <= getMaxStaking(), "
                Invalid lock amount");
631         if( lockAmount < getMinStaking() || getMaxStaking() < lockAmount ){
632             emit NotApplicable(ballotIdx, "Invalid lock amount");
633             return false;
634         }
635 
636         if(__stakingavailableBalanceOf(addr) < lockAmount){
637             emit NotApplicable(ballotIdx, "Insufficient balance that can be locked");
638             return false;
639         }
640         lock(addr, lockAmount);
641 
642         // Add voting and reward member
643         uint256 nMemIdx = memberLength.add(1);
644         members[nMemIdx] = addr;
645         memberIdx[addr] = nMemIdx;
646         rewards[nMemIdx] = addr;
647         rewardIdx[addr] = nMemIdx;
648 
649         // Add node
650         uint256 nNodeIdx = nodeLength.add(1);
651         //Node storage node = nodes[nNodeIdx];
652 
653         nodes[nNodeIdx].enode = enode;
654         nodes[nNodeIdx].ip = ip;
655         nodes[nNodeIdx].port = port;
656         nodeToMember[nNodeIdx] = addr;
657         nodeIdxFromMember[addr] = nNodeIdx;
658         nodes[nNodeIdx].name = name;
659         memberLength = nMemIdx;
660         nodeLength = nNodeIdx;
661         modifiedBlock = block.number;
662         emit MemberAdded(addr);
663         return true;
664     }
```

The code meets the specification.

Formal Verification Request 33

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

27, Nov 2019
191.12 ms

Line 665 in File fv.sol

```
665     //@CTK NO_BUF_OVERFLOW
```

Line 666-719 in File fv.sol

```
666     function removeMember(uint256 ballotIdx) private {
667         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberRemoval));
668 
669 
670         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
671         address addr = __ballotStorageballotMemberMap[ballotIdx].oldMemberAddress;
672         uint256 unlockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
673 
674         if (!isMember(addr)) {
675             emit NotApplicable(ballotIdx, "Not already a member");
676             return; // Non-member. it is abnormal case, but passed
677         }
678 
679         // Remove voting and reward member
680         uint256 removeIdx = memberIdx[addr];
681         address endAddr = members[memberLength];
682         if (memberIdx[addr] != memberLength) {
683             (members[removeIdx], members[memberLength],memberIdx[addr], memberIdx[
                endAddr] ) = (members[memberLength],address(0), 0, memberIdx[addr] );
684             removeIdx = rewardIdx[addr];
685             endAddr = rewards[memberLength];
686             (rewards[removeIdx], rewards[memberLength],rewardIdx[addr], rewardIdx[
                endAddr]) = (rewards[memberLength], address(0),0, rewardIdx[addr]);
687         } else {
688             members[memberLength] = address(0);
689             memberIdx[addr] = 0;
690             rewards[memberLength] = address(0);
691             rewardIdx[addr] = 0;
692         }
693 
694         memberLength = memberLength.sub(1);
695 
696         // Remove node
697         if (nodeIdxFromMember[addr] != nodeLength) {
698             removeIdx = nodeIdxFromMember[addr];
699             endAddr = nodeToMember[nodeLength];
700 
701             //Node storage node = nodes[removeIdx];
702             nodes[removeIdx].name = nodes[nodeLength].name;
703             nodes[removeIdx].enode = nodes[nodeLength].enode;
704             nodes[removeIdx].ip = nodes[nodeLength].ip;
705             nodes[removeIdx].port = nodes[nodeLength].port;
706 
707             nodeToMember[removeIdx] = endAddr;
708             nodeIdxFromMember[endAddr]=removeIdx;
709         }
```

```
710         nodeToMember[nodeLength] = address(0);
711         nodeIdxFromMember[addr] = 0;
712         delete nodes[nodeLength];
713         nodeLength = nodeLength.sub(1);
714         modifiedBlock = block.number;
715         // Unlock and transfer remained to governance
716         transferLockedAndUnlock(addr, unlockAmount);
717
718         emit MemberRemoved(addr);
719     }
```

The code meets the specification.

Formal Verification Request 34

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

27, Nov 2019
93.55 ms

Line 720 in File fv.sol

```
720     //@CTK NO_BUF_OVERFLOW
```

Line 721-788 in File fv.sol

```
721     function changeMember(uint256 ballotIdx) private returns (bool) {
722         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberChange));
723
724         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
725         address addr = __ballotStorageballotMemberMap[ballotIdx].oldMemberAddress;
726         address nAddr = __ballotStorageballotMemberMap[ballotIdx].newMemberAddress;
727         bytes memory name = __ballotStorageballotMemberMap[ballotIdx].newNodeName;
728         bytes memory enode = __ballotStorageballotMemberMap[ballotIdx].newNodeId;
729         bytes memory ip = __ballotStorageballotMemberMap[ballotIdx].newNodeIp;
730         uint port = __ballotStorageballotMemberMap[ballotIdx].newNodePort;
731         uint256 lockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
732
733         if (!isMember(addr)) {
734             emit NotApplicable(ballotIdx, "Old address is not a member");
735             return false; // Non-member. it is abnormal case.
736         }
737
738         if (addr != nAddr) {
739             if (isMember(nAddr)) {
740                 emit NotApplicable(ballotIdx, "new address is already a member");
741                 return false; // already member. it is abnormal case.
742             }
743
744             // Lock
745             // require(getMinStaking() <= lockAmount && lockAmount <= getMaxStaking(),
                    "Invalid lock amount");
746
747             if( lockAmount < getMinStaking() || getMaxStaking() < lockAmount ){
748                 emit NotApplicable(ballotIdx, "Invalid lock amount");
749                 return false;
750             }
751
```

```
752             if(__stakingavailableBalanceOf(nAddr) < lockAmount){
753                 emit NotApplicable(ballotIdx, "Insufficient balance that can be locked"
                        );
754                 return false;
755             }
756             lock(nAddr, lockAmount);
757 
758             // Change member
759             members[memberIdx[addr]] = nAddr;
760             memberIdx[nAddr] = memberIdx[addr];
761             rewards[memberIdx[addr]] = nAddr;
762             rewardIdx[nAddr] = rewardIdx[addr];
763             memberIdx[addr] = 0;
764 
765             return true;
766         }
767 
768         // Change node
769         uint256 nodeIdx = nodeIdxFromMember[addr];
770         //Node storage node = nodes[nodeIdx];
771         nodes[nodeIdx].name = name;
772         nodes[nodeIdx].enode = enode;
773         nodes[nodeIdx].ip = ip;
774         nodes[nodeIdx].port = port;
775         modifiedBlock = block.number;
776         if (addr != nAddr) {
777             nodeToMember[nodeIdx] = nAddr;
778             nodeIdxFromMember[nAddr] = nodeIdx;
779             nodeIdxFromMember[addr] = 0;
780 
781             // Unlock and transfer remained to governance
782             transferLockedAndUnlock(addr, lockAmount);
783 
784             emit MemberChanged(addr, nAddr);
785         } else {
786             emit MemberUpdated(addr);
787         }
788     }
```

The code meets the specification.

Formal Verification Request 35

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

27, Nov 2019
7.97 ms

Line 789 in File fv.sol

```
789     //@CTK NO_BUF_OVERFLOW
```

Line 790-798 in File fv.sol

```
790     function changeGov(uint256 ballotIdx) private {
791         fromValidBallot(ballotIdx, uint256(BallotTypes.GovernanceChange));
792 
793         address newImp = __ballotStoragegetBallotAddress(ballotIdx);
```

```
794         if (newImp != address(0)) {
795             //setImplementation(newImp);
796             modifiedBlock = block.number;
797         }
798     }
```

The code meets the specification.

Formal Verification Request 36

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

27, Nov 2019
6.17 ms

Line 799 in File fv.sol

```
799     //@CTK NO_BUF_OVERFLOW
```

Line 800-831 in File fv.sol

```
800     function applyEnv(uint256 ballotIdx) private {
801         fromValidBallot(ballotIdx, uint256(BallotTypes.EnvValChange));
802 
803 
804        // BallotVariable storage tBallot = __ballotStorageballotVariableMap[ballotIdx
               ];
805         bytes32 envVariableName = __ballotStorageballotVariableMap[ballotIdx].
                envVariableName;
806         uint256 envVariableType = __ballotStorageballotVariableMap[ballotIdx].
                envVariableType;
807         bytes memory envVariableValue = __ballotStorageballotVariableMap[ballotIdx].
                envVariableValue;
808 
809 /*
810         IEnvStorage envStorage = IEnvStorage(getEnvStorageAddress());
811         uint256 uintType = uint256(VariableTypes.Uint);
812         if (envKey == BLOCKS_PER_NAME && envType == uintType) {
813             envStorage.setBlocksPerByBytes(envVal);
814         } else if (envKey == BALLOT_DURATION_MIN_NAME && envType == uintType) {
815             envStorage.setBallotDurationMinByBytes(envVal);
816         } else if (envKey == BALLOT_DURATION_MAX_NAME && envType == uintType) {
817             envStorage.setBallotDurationMaxByBytes(envVal);
818         } else if (envKey == STAKING_MIN_NAME && envType == uintType) {
819             envStorage.setStakingMinByBytes(envVal);
820         } else if (envKey == STAKING_MAX_NAME && envType == uintType) {
821             envStorage.setStakingMaxByBytes(envVal);
822         } else if (envKey == GAS_PRICE_NAME && envType == uintType) {
823             envStorage.setGasPriceByBytes(envVal);
824         } else if (envKey == MAX_IDLE_BLOCK_INTERVAL_NAME && envType == uintType) {
825             envStorage.setMaxIdleBlockIntervalByBytes(envVal);
826         }*/
827 
828         modifiedBlock = block.number;
829 
830         emit EnvChanged(envKey, envType, envVal);
831     }
```

The code meets the specification.

Formal Verification Request 37

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

27, Nov 2019
903.09 ms

Line 834 in File fv.sol

```
834     //@CTK NO_BUF_OVERFLOW
```

Line 835-859 in File fv.sol

```
835     function createBallotForMember(
836         uint256 id,
837         uint256 bType,
838         address creator,
839         address oAddr,
840         address nAddr,
841         bytes name,
842         bytes enode,
843         bytes ip,
844         uint port
845     )
846         private
847     {
848         __ballotStoragecreateBallotForMember(
849             id, // ballot id
850             bType, // ballot type
851             creator, // creator
852             oAddr, // old member address
853             nAddr, // new member address
854             name, // new name
855             enode, // new enode
856             ip, // new ip
857             port // new port
858         );
859     }
```

The code meets the specification.

Formal Verification Request 38

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

27, Nov 2019
24.83 ms

Line 901 in File fv.sol

```
901     //@CTK NO_BUF_OVERFLOW
```

Line 902-908 in File fv.sol

```
902     function transferLockedAndUnlock(address addr, uint256 unlockAmount) private {
903         uint256 locked = __stakinglockedBalanceOf(addr);
904         if (locked > unlockAmount) {
905             __stakingtransferLocked(addr, locked.sub(unlockAmount));
906         }
907         unlock(addr, unlockAmount);
908     }
```

The code meets the specification.

Formal Verification Request 39

__stakingcontstuctor

27, Nov 2019
22.5 ms

Line 927-929 in File fv.sol

```
927     /*@CTK __stakingcontstuctor
928       @post __post.__staking_totalLockedBalance == 0
929     */
```

Line 932-972 in File fv.sol

```
932     function __stakingconstructor(address registry, bytes data) public {
933         __staking_totalLockedBalance = 0;
934      //   setRegistry(registry);
935 
936         // data is only for test purpose
937         if (data.length == 0)
938             return;
939 
940         // []{address, amount}
941         address addr;
942         uint amount;
943         uint ix;
944         uint eix;
945         assembly {
946             ix := add(data, 0x20)
947         }
948         eix = ix + data.length;
949         /*#CTK loop
950           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_balance[addr] == mload(ix)
951           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_lockedBalance[addr] == mload(ix)
952         */
953         while (ix < eix) {
954             assembly {
955                 amount := mload(ix)
956             }
957             addr = address(amount);
958             ix += 0x20;
959             require(ix < eix);
960             assembly {
961                 amount := mload(ix)
962             }
```

```
963             ix += 0x20;
964
965             __staking_balance[addr] = amount;
966             __staking_lockedBalance[addr] = amount;
967         }
968     }
```

The code meets the specification.

Formal Verification Request 40

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

27, Nov 2019
0.47 ms

Line 930 in File fv.sol

```
930     //@CTK NO_BUF_OVERFLOW
```

Line 932-972 in File fv.sol

```
932     function __stakingconstructor(address registry, bytes data) public {
933         __staking_totalLockedBalance = 0;
934      //   setRegistry(registry);
935
936         // data is only for test purpose
937         if (data.length == 0)
938             return;
939
940         // []{address, amount}
941         address addr;
942         uint amount;
943         uint ix;
944         uint eix;
945         assembly {
946             ix := add(data, 0x20)
947         }
948         eix = ix + data.length;
949         /*#CTK loop
950           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_balance[addr] == mload(ix)
951           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_lockedBalance[addr] == mload(ix)
952         */
953         while (ix < eix) {
954             assembly {
955                 amount := mload(ix)
956             }
957             addr = address(amount);
958             ix += 0x20;
959             require(ix < eix);
960             assembly {
961                 amount := mload(ix)
962             }
963             ix += 0x20;
964
965             __staking_balance[addr] = amount;
```

```
966             __staking_lockedBalance[addr] = amount;
967         }
968     }
```

The code meets the specification.

Formal Verification Request 41

If method completes, integer overflow would not happen.

27, Nov 2019
0.44 ms

Line 931 in File fv.sol

```
931     //@CTK NO_OVERFLOW
```

Line 932-972 in File fv.sol

```
932     function __stakingconstructor(address registry, bytes data) public {
933         __staking_totalLockedBalance = 0;
934      //   setRegistry(registry);
935 
936         // data is only for test purpose
937         if (data.length == 0)
938             return;
939 
940         // []{address, amount}
941         address addr;
942         uint amount;
943         uint ix;
944         uint eix;
945         assembly {
946             ix := add(data, 0x20)
947         }
948         eix = ix + data.length;
949         /*#CTK loop
950           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_balance[addr] == mload(ix)
951           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_lockedBalance[addr] == mload(ix)
952         */
953         while (ix < eix) {
954             assembly {
955                 amount := mload(ix)
956             }
957             addr = address(amount);
958             ix += 0x20;
959             require(ix < eix);
960             assembly {
961                 amount := mload(ix)
962             }
963             ix += 0x20;
964 
965             __staking_balance[addr] = amount;
966             __staking_lockedBalance[addr] = amount;
967         }
968     }
```

The code meets the specification.

Formal Verification Request 42

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

27, Nov 2019
86.57 ms

Line 983 in File fv.sol

```
983     //@CTK NO_BUF_OVERFLOW
```

Line 989-995 in File fv.sol

```
989     function __stakingdeposit() external nonReentrant /* notRevoked */ payable {
990         require(msg.value > 0, "Deposit amount should be greater than zero");
991 
992         __staking_balance[msg.sender] = __staking_balance[msg.sender].add(msg.value);
993 
994         emit Staked(msg.sender, msg.value, __staking_balance[msg.sender],
                __stakingavailableBalanceOf(msg.sender));
995     }
```

The code meets the specification.

Formal Verification Request 43

__stakingdeposit

27, Nov 2019
7.43 ms

Line 984-988 in File fv.sol

```
984     /*@CTK __stakingdeposit
985       @tag assume_completion
986       @pre msg.value > 0
987       @post __post.__staking_balance[msg.sender] == __staking_balance[msg.sender] +
              msg.value
988     */
```

Line 989-995 in File fv.sol

```
989     function __stakingdeposit() external nonReentrant /* notRevoked */ payable {
990         require(msg.value > 0, "Deposit amount should be greater than zero");
991 
992         __staking_balance[msg.sender] = __staking_balance[msg.sender].add(msg.value);
993 
994         emit Staked(msg.sender, msg.value, __staking_balance[msg.sender],
                __stakingavailableBalanceOf(msg.sender));
995     }
```

The code meets the specification.

Formal Verification Request 44

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

27, Nov 2019
68.97 ms

Line 1001 in File fv.sol

```
1001    //@CTK NO_BUF_OVERFLOW
```

Line 1008-1020 in File fv.sol

```
1008    function __stakingwithdraw(uint256 amount) external nonReentrant /*notRevoked*/ {
1009        require(amount > 0, "Amount should be bigger than zero");
1010        require(amount <= __stakingavailableBalanceOf(msg.sender), "Withdraw amount
                should be equal or less than balance");
1011
1012        __staking_balance[msg.sender] = __staking_balance[msg.sender].sub(amount);
1013        msg.sender.transfer(amount);
1014
1015        emit Unstaked(msg.sender, amount, __staking_balance[msg.sender],
                __stakingavailableBalanceOf(msg.sender));
1016    }
```

The code meets the specification.

Formal Verification Request 45

__stakingwithdraw

27, Nov 2019
6.58 ms

Line 1002-1007 in File fv.sol

```
1002    /*@CTK __stakingwithdraw
1003      @tag assume_completion
1004      @pre amount > 0
1005      @pre amount <= __staking_balance[msg.sender] - __staking_lockedBalance[msg.
              sender]
1006      @post __post.__staking_balance[msg.sender] == __staking_balance[msg.sender] -
              amount
1007    */
```

Line 1008-1020 in File fv.sol

```
1008    function __stakingwithdraw(uint256 amount) external nonReentrant /*notRevoked*/ {
1009        require(amount > 0, "Amount should be bigger than zero");
1010        require(amount <= __stakingavailableBalanceOf(msg.sender), "Withdraw amount
                should be equal or less than balance");
1011
1012        __staking_balance[msg.sender] = __staking_balance[msg.sender].sub(amount);
1013        msg.sender.transfer(amount);
1014
1015        emit Unstaked(msg.sender, amount, __staking_balance[msg.sender],
                __stakingavailableBalanceOf(msg.sender));
1016    }
```

The code meets the specification.

Formal Verification Request 46

__stakinglock

27, Nov 2019
261.68 ms

Line 1023-1025 in File fv.sol

```
1023    /*@CTK __stakinglock
1024      @post __post.__staking_totalLockedBalance - __post.__staking_lockedBalance[payee
              ] == __staking_totalLockedBalance - __staking_lockedBalance[payee]
1025     */
```

Line 1028-1036 in File fv.sol

```
1028    function __stakinglock(address payee, uint256 lockAmount) private {
1029        if (lockAmount == 0) return;
1030        require(__stakingavailableBalanceOf(payee) >= lockAmount, "Insufficient balance
                that can be locked");
1031
1032        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].add(lockAmount)
                ;
1033        __staking_totalLockedBalance = __staking_totalLockedBalance.add(lockAmount);
1034
1035        emit Locked(payee, lockAmount, _balance[payee], __stakingavailableBalanceOf(
                payee));
1036    }
```

The code meets the specification.

Formal Verification Request 47

If method completes, integer overflow would not happen.

27, Nov 2019
15.14 ms

Line 1026 in File fv.sol

```
1026    //@CTK NO_OVERFLOW
```

Line 1028-1036 in File fv.sol

```
1028    function __stakinglock(address payee, uint256 lockAmount) private {
1029        if (lockAmount == 0) return;
1030        require(__stakingavailableBalanceOf(payee) >= lockAmount, "Insufficient balance
                that can be locked");
1031
1032        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].add(lockAmount)
                ;
1033        __staking_totalLockedBalance = __staking_totalLockedBalance.add(lockAmount);
1034
1035        emit Locked(payee, lockAmount, _balance[payee], __stakingavailableBalanceOf(
                payee));
1036    }
```

The code meets the specification.

Formal Verification Request 48

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

27, Nov 2019
11.63 ms

Line 1027 in File fv.sol

```
1027    //@CTK NO_BUF_OVERFLOW
```

Line 1028-1036 in File fv.sol

```
1028    function __stakinglock(address payee, uint256 lockAmount) private {
1029        if (lockAmount == 0) return;
1030        require(__stakingavailableBalanceOf(payee) >= lockAmount, "Insufficient balance
                 that can be locked");
1031
1032        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].add(lockAmount)
                ;
1033        __staking_totalLockedBalance = __staking_totalLockedBalance.add(lockAmount);
1034
1035        emit Locked(payee, lockAmount, _balance[payee], __stakingavailableBalanceOf(
                payee));
1036    }
```

The code meets the specification.

Formal Verification Request 49

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

27, Nov 2019
0.99 ms

Line 1044 in File fv.sol

```
1044    //@CTK NO_BUF_OVERFLOW
```

Line 1051-1057 in File fv.sol

```
1051    function __stakingtransferLocked(address from, uint256 amount) private {
1052        __staking_balance[from] = __staking_balance[from].sub(amount);
1053        address rewardPool = address(0x0);//getRewardPoolAddress();
1054        __staking_balance[rewardPool] = __staking_balance[rewardPool].add(amount);
1055
1056        emit TransferLocked(from, amount, _balance[from], __stakingavailableBalanceOf(
                from));
1057    }
```

The code meets the specification.

Formal Verification Request 50

__stakingtransferLocked

27, Nov 2019
24.65 ms

Line 1045-1050 in File fv.sol

```
1045    /*@CTK __stakingtransferLocked
1046      @tag assume_completion
1047      @pre from != 0x0
1048      @post __post.__staking_balance[from] == __staking_balance[from] - amount
1049      @post __post.__staking_balance[0x0] == __staking_balance[0x0] + amount
1050    */
```

Line 1051-1057 in File fv.sol

```
1051    function __stakingtransferLocked(address from, uint256 amount) private {
1052        __staking_balance[from] = __staking_balance[from].sub(amount);
1053        address rewardPool = address(0x0);//getRewardPoolAddress();
1054        __staking_balance[rewardPool] = __staking_balance[rewardPool].add(amount);
1055
1056        emit TransferLocked(from, amount, _balance[from], __stakingavailableBalanceOf(
                from));
1057    }
```

The code meets the specification.

Formal Verification Request 51

If method completes, integer overflow would not happen.

27, Nov 2019
5.81 ms

Line 1064 in File fv.sol

```
1064    //@CTK NO_OVERFLOW
```

Line 1066-1073 in File fv.sol

```
1066    function __stakingunlock(address payee, uint256 unlockAmount) private {
1067        if (unlockAmount == 0) return;
1068        // require(_lockedBalance[payee] >= unlockAmount, "Unlock amount should be
                equal or less than balance locked");
1069        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].sub(
                unlockAmount);
1070        __staking_totalLockedBalance = __staking_totalLockedBalance.sub(unlockAmount);
1071
1072        emit Unlocked(payee, unlockAmount, _balance[payee], __stakingavailableBalanceOf
                (payee));
1073    }
```

The code meets the specification.

Formal Verification Request 52

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

27, Nov 2019
6.54 ms

Line 1065 in File fv.sol

```
1065    //@CTK NO_BUF_OVERFLOW
```

Line 1066-1073 in File fv.sol

```
1066    function __stakingunlock(address payee, uint256 unlockAmount) private {
1067        if (unlockAmount == 0) return;
1068        // require(_lockedBalance[payee] >= unlockAmount, "Unlock amount should be
                equal or less than balance locked");
1069        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].sub(
                unlockAmount);
1070        __staking_totalLockedBalance = __staking_totalLockedBalance.sub(unlockAmount);
1071
1072        emit Unlocked(payee, unlockAmount, _balance[payee], __stakingavailableBalanceOf
                (payee));
1073    }
```

The code meets the specification.

Formal Verification Request 53

If method completes, integer overflow would not happen.

27, Nov 2019
5.92 ms

Line 1074 in File fv.sol

```
1074    //@CTK NO_OVERFLOW
```

Line 1076-1078 in File fv.sol

```
1076    function __stakingbalanceOf(address payee) public view returns (uint256) {
1077        return __staking_balance[payee];
1078    }
```

The code meets the specification.

Formal Verification Request 54

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

27, Nov 2019
0.37 ms

Line 1075 in File fv.sol

```
1075    //@CTK NO_BUF_OVERFLOW
```

Line 1076-1078 in File fv.sol

```
1076    function __stakingbalanceOf(address payee) public view returns (uint256) {
1077        return __staking_balance[payee];
1078    }
```

The code meets the specification.

Formal Verification Request 55

If method completes, integer overflow would not happen.

27, Nov 2019
0.39 ms

Line 1079 in File fv.sol

```
1079    //@CTK NO_OVERFLOW
```

Line 1081-1083 in File fv.sol

```
1081    function __stakinglockedBalanceOf(address payee) public view returns (uint256) {
1082        return __staking_lockedBalance[payee];
1083    }
```

The code meets the specification.

Formal Verification Request 56

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

27, Nov 2019
0.38 ms

Line 1080 in File fv.sol

```
1080    //@CTK NO_BUF_OVERFLOW
```

Line 1081-1083 in File fv.sol

```
1081    function __stakinglockedBalanceOf(address payee) public view returns (uint256) {
1082        return __staking_lockedBalance[payee];
1083    }
```

The code meets the specification.

Formal Verification Request 57

If method completes, integer overflow would not happen.

27, Nov 2019
1.33 ms

Line 1084 in File fv.sol

```
1084    //@CTK NO_OVERFLOW
```

Line 1086-1088 in File fv.sol

```
1086    function __stakingavailableBalanceOf(address payee) public view returns (uint256)
            {
1087        return __staking_balance[payee].sub(__staking_lockedBalance[payee]);
1088    }
```

The code meets the specification.

Formal Verification Request 58

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

27, Nov 2019
0.48 ms

Line 1085 in File fv.sol

```
1085    //@CTK NO_BUF_OVERFLOW
```

Line 1086-1088 in File fv.sol

```
1086    function __stakingavailableBalanceOf(address payee) public view returns (uint256)
            {
1087        return __staking_balance[payee].sub(__staking_lockedBalance[payee]);
1088    }
```

The code meets the specification.

Formal Verification Request 59

If method completes, integer overflow would not happen.

27, Nov 2019
54.31 ms

Line 1094 in File fv.sol

```
1094     //@CTK NO_OVERFLOW
```

Line 1096-1098 in File fv.sol

```
1096    function calcVotingWeight(address payee) public view returns (uint256) {
1097        return __stakingcalcVotingWeightWithScaleFactor(payee, 1e2);
1098    }
```

The code meets the specification.

Formal Verification Request 60

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

27, Nov 2019
3.5 ms

Line 1095 in File fv.sol

```
1095     //@CTK NO_BUF_OVERFLOW
```

Line 1096-1098 in File fv.sol

```
1096    function calcVotingWeight(address payee) public view returns (uint256) {
1097        return __stakingcalcVotingWeightWithScaleFactor(payee, 1e2);
1098    }
```

The code meets the specification.

Formal Verification Request 61

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

27, Nov 2019
3.17 ms

Line 1108 in File fv.sol

```
1108    //@CTK NO_BUF_OVERFLOW
```

Line 1115-1118 in File fv.sol

```
1115    function __stakingcalcVotingWeightWithScaleFactor(address payee, uint32 factor)
            public view returns (uint256) {
1116        if (__staking_lockedBalance[payee] == 0 || factor == 0) return 0;
1117        return __staking_lockedBalance[payee].mul(factor).div(
                __staking_totalLockedBalance);
1118    }
```

The code meets the specification.

Formal Verification Request 62

__stakingcalcVotingWeightWithScaleFactor

27, Nov 2019
203.96 ms

Line 1109-1114 in File fv.sol

```
1109    /*@CTK __stakingcalcVotingWeightWithScaleFactor
1110      @tag assume_completion
1111      @post __staking_lockedBalance[payee] == 0 || factor == 0 -> __return == 0
1112      @post __staking_lockedBalance[payee] != 0 && factor != 0 -> __return ==
              __staking_lockedBalance[payee] * factor / __staking_totalLockedBalance
1113
1114    */
```

Line 1115-1118 in File fv.sol

```
1115    function __stakingcalcVotingWeightWithScaleFactor(address payee, uint32 factor)
            public view returns (uint256) {
1116        if (__staking_lockedBalance[payee] == 0 || factor == 0) return 0;
1117        return __staking_lockedBalance[payee].mul(factor).div(
                __staking_totalLockedBalance);
1118    }
```

The code meets the specification.

Formal Verification Request 63

Method will not encounter an assertion failure.

27, Nov 2019
6.09 ms

Line 1119 in File fv.sol

```
1119    //@CTK NO_ASF
```

Line 1122-1124 in File fv.sol

```
1122    function __stakingisRevoked() public view returns (bool) {
1123        return __stakingrevoked;
1124    }
```

The code meets the specification.

Formal Verification Request 64

If method completes, integer overflow would not happen.

27, Nov 2019
0.39 ms

Line 1120 in File fv.sol

```
1120    //@CTK NO_OVERFLOW
```

Line 1122-1124 in File fv.sol

```
1122    function __stakingisRevoked() public view returns (bool) {
1123        return __stakingrevoked;
1124    }
```

The code meets the specification.

Formal Verification Request 65

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

27, Nov 2019
0.38 ms

Line 1121 in File fv.sol

```
1121    //@CTK NO_BUF_OVERFLOW
```

Line 1122-1124 in File fv.sol

```
1122    function __stakingisRevoked() public view returns (bool) {
1123        return __stakingrevoked;
1124    }
```

The code meets the specification.

Formal Verification Request 66

__ballotStoragegetMinVotingDuration

27, Nov 2019
5.42 ms

Line 1277-1279 in File fv.sol

```
1277    /*@CTK __ballotStoragegetMinVotingDuration
1278      @post __return == 500000000
1279    */
```

Line 1280-1282 in File fv.sol

```
1280    function __ballotStoragegetMinVotingDuration() public view returns (uint256) {
1281        return 500000000;
1282    }
```

The code meets the specification.

Formal Verification Request 67

__ballotStoragegetMaxVotingDuration

27, Nov 2019
5.39 ms

Line 1283-1285 in File fv.sol

```
1283    /*@CTK __ballotStoragegetMaxVotingDuration
1284      @post __return == 800000000
1285    */
```

Line 1286-1288 in File fv.sol

```
1286    function __ballotStoragegetMaxVotingDuration() public view returns (uint256) {
1287        return 800000000;
1288    }
```

The code meets the specification.

Formal Verification Request 68

__ballotStoragegetTime

27, Nov 2019
0.38 ms

Line 1290-1292 in File fv.sol

```
1290    /*@CTK __ballotStoragegetTime
1291      @post __return == now
1292    */
```

Line 1293-1295 in File fv.sol

```
1293    function __ballotStoragegetTime() public view returns (uint256) {
1294        return now;
1295    }
```

The code meets the specification.

Formal Verification Request 69

__ballotStoragegetPreviousBallotStorage

27, Nov 2019
5.69 ms

Line 1296-1298 in File fv.sol

```
1296    /*@CTK __ballotStoragegetPreviousBallotStorage
1297      @post __return == __ballotStoragepreviousBallotStorage
1298    */
```

Line 1299-1301 in File fv.sol

```
1299    function __ballotStoragegetPreviousBallotStorage() public view returns (address) {
1300        return __ballotStoragepreviousBallotStorage;
1301    }
```

The code meets the specification.

Formal Verification Request 70

__ballotStoragegetBallotCount

27, Nov 2019
5.56 ms

Line 1306-1308 in File fv.sol

```
1306    /*@CTK __ballotStoragegetBallotCount
1307      @post __return == __ballotStorageballotCount
1308    */
```

Line 1309-1311 in File fv.sol

```
1309    function __ballotStoragegetBallotCount() public view returns (uint256) {
1310        return __ballotStorageballotCount;
1311    }
```

The code meets the specification.

Formal Verification Request 71

If method completes, integer overflow would not happen.

27, Nov 2019
15.89 ms

Line 1316 in File fv.sol

```
1316    //@CTK NO_OVERFLOW
```

Line 1318-1344 in File fv.sol

```
1318    function __ballotStoragegetBallotBasic(uint256 _id) public view returns (
1319        uint256 startTime,
1320        uint256 endTime,
1321        uint256 ballotType,
1322        address creator,
1323        bytes memo,
1324        uint256 totalVoters,
1325        uint256 powerOfAccepts,
1326        uint256 powerOfRejects,
1327        uint256 state,
1328        bool isFinalized,
1329        uint256 duration
1330    )
1331    {
1332        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1333        startTime = tBallot.startTime;
1334        endTime = tBallot.endTime;
1335        ballotType = tBallot.ballotType;
1336        creator = tBallot.creator;
1337        memo = tBallot.memo;
1338        totalVoters = tBallot.totalVoters;
1339        powerOfAccepts = tBallot.powerOfAccepts;
1340        powerOfRejects = tBallot.powerOfRejects;
1341        state = tBallot.state;
1342        isFinalized = tBallot.isFinalized;
1343        duration = tBallot.duration;
1344    }
```

✅ The code meets the specification.

Formal Verification Request 72

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

📅 27, Nov 2019
⏱ 0.43 ms

Line 1317 in File fv.sol

```
1317    //@CTK NO_BUF_OVERFLOW
```

Line 1318-1344 in File fv.sol

```
1318    function __ballotStoragegetBallotBasic(uint256 _id) public view returns (
1319        uint256 startTime,
1320        uint256 endTime,
1321        uint256 ballotType,
1322        address creator,
1323        bytes memo,
1324        uint256 totalVoters,
1325        uint256 powerOfAccepts,
1326        uint256 powerOfRejects,
1327        uint256 state,
1328        bool isFinalized,
```

```
1329        uint256 duration
1330    )
1331    {
1332        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1333        startTime = tBallot.startTime;
1334        endTime = tBallot.endTime;
1335        ballotType = tBallot.ballotType;
1336        creator = tBallot.creator;
1337        memo = tBallot.memo;
1338        totalVoters = tBallot.totalVoters;
1339        powerOfAccepts = tBallot.powerOfAccepts;
1340        powerOfRejects = tBallot.powerOfRejects;
1341        state = tBallot.state;
1342        isFinalized = tBallot.isFinalized;
1343        duration = tBallot.duration;
1344    }
```

The code meets the specification.

Formal Verification Request 73

If method completes, integer overflow would not happen.

27, Nov 2019
14.41 ms

Line 1348 in File fv.sol

```
1348    //@CTK NO_OVERFLOW
```

Line 1350-1368 in File fv.sol

```
1350    function __ballotStoragegetBallotMember(uint256 _id) public view returns (
1351        address oldMemberAddress,
1352        address newMemberAddress,
1353        bytes newNodeName, // name
1354        bytes newNodeId, // admin.nodeInfo.id is 512 bit public key
1355        bytes newNodeIp,
1356        uint256 newNodePort,
1357        uint256 lockAmount
1358    )
1359    {
1360        //BallotMember storage tBallot = __ballotStorageballotMemberMap[_id];
1361        oldMemberAddress = __ballotStorageballotMemberMap[_id].oldMemberAddress;
1362        newMemberAddress = __ballotStorageballotMemberMap[_id].newMemberAddress;
1363        newNodeName = __ballotStorageballotMemberMap[_id].newNodeName;
1364        newNodeId = __ballotStorageballotMemberMap[_id].newNodeId;
1365        newNodeIp = __ballotStorageballotMemberMap[_id].newNodeIp;
1366        newNodePort = __ballotStorageballotMemberMap[_id].newNodePort;
1367        lockAmount = __ballotStorageballotMemberMap[_id].lockAmount;
1368    }
```

The code meets the specification.

Formal Verification Request 74

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

27, Nov 2019
0.4 ms

Line 1349 in File fv.sol

```
1349    //@CTK NO_BUF_OVERFLOW
```

Line 1350-1368 in File fv.sol

```
1350    function __ballotStoragegetBallotMember(uint256 _id) public view returns (
1351        address oldMemberAddress,
1352        address newMemberAddress,
1353        bytes newNodeName, // name
1354        bytes newNodeId, // admin.nodeInfo.id is 512 bit public key
1355        bytes newNodeIp,
1356        uint256 newNodePort,
1357        uint256 lockAmount
1358    )
1359    {
1360        //BallotMember storage tBallot = __ballotStorageballotMemberMap[_id];
1361        oldMemberAddress = __ballotStorageballotMemberMap[_id].oldMemberAddress;
1362        newMemberAddress = __ballotStorageballotMemberMap[_id].newMemberAddress;
1363        newNodeName = __ballotStorageballotMemberMap[_id].newNodeName;
1364        newNodeId = __ballotStorageballotMemberMap[_id].newNodeId;
1365        newNodeIp = __ballotStorageballotMemberMap[_id].newNodeIp;
1366        newNodePort = __ballotStorageballotMemberMap[_id].newNodePort;
1367        lockAmount = __ballotStorageballotMemberMap[_id].lockAmount;
1368    }
```

The code meets the specification.

Formal Verification Request 75

If method completes, integer overflow would not happen.

27, Nov 2019
0.37 ms

Line 1369 in File fv.sol

```
1369    //@CTK NO_OVERFLOW
```

Line 1374-1380 in File fv.sol

```
1374    function __ballotStoragegetBallotAddress(uint256 _id) public view returns (
1375        address newGovernanceAddress
1376    )
1377    {
1378        //BallotAddress storage tBallot = __ballotStorageballotAddressMap[_id];
1379        newGovernanceAddress = __ballotStorageballotAddressMap[_id].
                newGovernanceAddress;
1380    }
```

The code meets the specification.

Formal Verification Request 76

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

27, Nov 2019
0.36 ms

Line 1370 in File fv.sol

```
1370    //@CTK NO_BUF_OVERFLOW
```

Line 1374-1380 in File fv.sol

```
1374    function __ballotStoragegetBallotAddress(uint256 _id) public view returns (
1375        address newGovernanceAddress
1376    )
1377    {
1378        //BallotAddress storage tBallot = __ballotStorageballotAddressMap[_id];
1379        newGovernanceAddress = __ballotStorageballotAddressMap[_id].
                newGovernanceAddress;
1380    }
```

The code meets the specification.

Formal Verification Request 77

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

27, Nov 2019
7.97 ms

Line 1384 in File fv.sol

```
1384    //@CTK NO_BUF_OVERFLOW
```

Line 1386-1396 in File fv.sol

```
1386    function __ballotStoragegetBallotVariable(uint256 _id) public view returns (
1387        bytes32 envVariableName,
1388        uint256 envVariableType,
1389        bytes envVariableValue
1390    )
1391    {
1392        //BallotVariable storage tBallot = __ballotStorageballotVariableMap[_id];
1393        envVariableName = __ballotStorageballotVariableMap[_id].envVariableName;
1394        envVariableType = __ballotStorageballotVariableMap[_id].envVariableType;
1395        envVariableValue = __ballotStorageballotVariableMap[_id].envVariableValue;
1396    }
```

The code meets the specification.

Formal Verification Request 78

If method completes, integer overflow would not happen.

27, Nov 2019
0.39 ms

Line 1385 in File fv.sol

```
1385    //@CTK NO_OVERFLOW
```

Line 1386-1396 in File fv.sol

```
1386    function __ballotStoragegetBallotVariable(uint256 _id) public view returns (
1387        bytes32 envVariableName,
1388        uint256 envVariableType,
1389        bytes envVariableValue
1390    )
1391    {
1392        //BallotVariable storage tBallot = __ballotStorageballotVariableMap[_id];
1393        envVariableName = __ballotStorageballotVariableMap[_id].envVariableName;
1394        envVariableType = __ballotStorageballotVariableMap[_id].envVariableType;
1395        envVariableValue = __ballotStorageballotVariableMap[_id].envVariableValue;
1396    }
```

The code meets the specification.

Formal Verification Request 79

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

27, Nov 2019
36.99 ms

Line 1405 in File fv.sol

```
1405    //@CTK NO_BUF_OVERFLOW
```

Line 1406-1442 in File fv.sol

```
1406    function __ballotStoragecreateBallotForMember(
1407        uint256 _id,
1408        uint256 _ballotType,
1409        address _creator,
1410        address _oldMemberAddress,
1411        address _newMemberAddress,
1412        bytes _newNodeName, // name
1413        bytes _newNodeId, // admin.nodeInfo.id is 512 bit public key
1414        bytes _newNodeIp,
1415        uint _newNodePort
1416    )
1417        private
1418        notDisabled
1419    {
1420        require(
1421            __ballotStorage_areMemberBallotParamValid(
1422                _ballotType,
1423                _oldMemberAddress,
1424                _newMemberAddress,
1425                _newNodeName,
1426                _newNodeId,
1427                _newNodeIp,
1428                _newNodePort
1429            ),
1430            "Invalid Parameter"
```

```
1431        );
1432        __ballotStorage_createBallot(_id, _ballotType, _creator);
1433        BallotMember memory newBallot;
1434        newBallot.id = _id;
1435        newBallot.oldMemberAddress = _oldMemberAddress;
1436        newBallot.newMemberAddress = _newMemberAddress;
1437        newBallot.newNodeName = _newNodeName;
1438        newBallot.newNodeId = _newNodeId;
1439        newBallot.newNodeIp = _newNodeIp;
1440        newBallot.newNodePort = _newNodePort;
1441        __ballotStorageballotMemberMap[_id] = newBallot;
1442    }
```

The code meets the specification.

Formal Verification Request 80

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

27, Nov 2019
150.12 ms

Line 1446 in File fv.sol

```
1446    //@CTK NO_BUF_OVERFLOW
```

Line 1447-1464 in File fv.sol

```
1447    function __ballotStoragecreateBallotForAddress(
1448        uint256 _id,
1449        uint256 _ballotType,
1450        address _creator,
1451        address _newGovernanceAddress
1452    )
1453        private
1454        notDisabled
1455    {
1456        require(_ballotType == uint256(BallotTypes.GovernanceChange), "Invalid Ballot
                Type");
1457        require(_newGovernanceAddress != address(0), "Invalid Parameter");
1458
1459        __ballotStorage_createBallot(_id, _ballotType, _creator);
1460        BallotAddress memory newBallot;
1461        newBallot.id = _id;
1462        newBallot.newGovernanceAddress = _newGovernanceAddress;
1463        __ballotStorageballotAddressMap[_id] = newBallot;
1464    }
```

The code meets the specification.

Formal Verification Request 81

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

27, Nov 2019
240.46 ms

Line 1467 in File fv.sol

```
1467    //@CTK NO_BUF_OVERFLOW
```

Line 1468-1492 in File fv.sol

```
1468    function __ballotStoragecreateBallotForVariable(
1469        uint256 _id,
1470        uint256 _ballotType,
1471        address _creator,
1472        bytes32 _envVariableName,
1473        uint256 _envVariableType,
1474        bytes _envVariableValue
1475    )
1476        private
1477    //    notDisabled
1478        returns (uint256)
1479    {
1480
1481        require(
1482            __ballotStorage_areVariableBallotParamValid(_ballotType, _envVariableName,
                    _envVariableType, _envVariableValue),
1483            "Invalid Parameter"
1484        );
1485        __ballotStorage_createBallot(_id, _ballotType, _creator);
1486        BallotVariable memory newBallot;
1487        newBallot.id = _id;
1488        newBallot.envVariableName = _envVariableName;
1489        newBallot.envVariableType = _envVariableType;
1490        newBallot.envVariableValue = _envVariableValue;
1491        __ballotStorageballotVariableMap[_id] = newBallot;
1492    }
```

The code meets the specification.

Formal Verification Request 82

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

27, Nov 2019
32.17 ms

Line 1493 in File fv.sol

```
1493    //@CTK NO_BUF_OVERFLOW
```

Line 1494-1521 in File fv.sol

```
1494    function __ballotStoragecreateVote(
1495        uint256 _voteId,
1496        uint256 _ballotId,
1497        address _voter,
1498        uint256 _decision,
1499        uint256 _power
1500    )
1501        private
1502       // notDisabled
1503    {
```

```
1504        // Check decision type
1505        /*
1506        require((_decision == uint256(DecisionTypes.Accept))
1507            || (_decision == uint256(DecisionTypes.Reject)), "Invalid decision");*/
1508        // Check if ballot exists
1509        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1510        // Check if vote exists
1511        require(__ballotStoragevoteMap[_voteId].voteId != _voteId, "already existed
                voteId");
1512        // Check if voted
1513        require(!__ballotStoragehasVotedMap[_ballotId][_voter], "already voted");
1514      /* require(ballotBasicMap[_ballotId].state
1515            == uint256(BallotStates.InProgress), "Not InProgress State");
1516 */
1517        __ballotStoragevoteMap[_voteId] = Vote(_voteId, _ballotId, _voter, _decision,
                _power, __ballotStoragegetTime());
1518        __ballotStorage_updateBallotForVote(_ballotId, _voter, _decision, _power);
1519
1520        emit Voted(_voteId, _ballotId, _voter, _decision);
1521    }
```

The code meets the specification.

Formal Verification Request 83

__ballotStoragestartBallot

27, Nov 2019
12.63 ms

Line 1523-1531 in File fv.sol

```
1523    /*@CTK __ballotStoragestartBallot
1524      @pre _startTime > 0 && _endTime > 0
1525      @pre _endTime > _startTime
1526      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1527      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1528      @post __post.__ballotStorageballotBasicMap[_ballotId].startTime == _startTime
1529      @post __post.__ballotStorageballotBasicMap[_ballotId].endTime == _endTime
1530      @post __post.__ballotStorageballotBasicMap[_ballotId].state == 2
1531    */
```

Line 1532-1550 in File fv.sol

```
1532    function __ballotStoragestartBallot(
1533        uint256 _ballotId,
1534        uint256 _startTime,
1535        uint256 _endTime
1536    )
1537        private
1538      // notDisabled
1539      // onlyValidTime(_startTime, _endTime)
1540    {
1541        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1542        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
```

```
1543        // require(__ballotStorageballotBasicMap[_ballotId].state == uint256(
                BallotStates.Ready), "Not Ready State");
1544
1545        //  BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1546          __ballotStorageballotBasicMap[_ballotId].startTime = _startTime;
1547          __ballotStorageballotBasicMap[_ballotId].endTime = _endTime;
1548          __ballotStorageballotBasicMap[_ballotId].state = uint256(BallotStates.
                  InProgress);
1549          emit BallotStarted(_ballotId, _startTime, _endTime);
1550      }
```

The code meets the specification.

Formal Verification Request 84

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

27, Nov 2019
46.48 ms

Line 1554 in File fv.sol

```
1554      //@CTK NO_BUF_OVERFLOW
```

Line 1560-1572 in File fv.sol

```
1560      function __ballotStorageupdateBallotMemo(
1561          uint256 _ballotId,
1562          bytes _memo
1563      )
1564          private
1565          notDisabled
1566      {
1567          require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                  Ballot");
1568          require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                   finalized");
1569          //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1570          __ballotStorageballotBasicMap[_ballotId].memo = _memo;
1571          emit BallotUpdated (_ballotId, msg.sender);
1572      }
```

The code meets the specification.

Formal Verification Request 85

__ballotStorageupdateBallotMemo

27, Nov 2019
6.24 ms

Line 1555-1559 in File fv.sol

```
1555      /*@CTK __ballotStorageupdateBallotMemo
1556        @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1557        @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
```

```
1558      @post __post.__ballotStorageballotBasicMap[_ballotId].memo == _memo
1559    */
```

Line 1560-1572 in File fv.sol

```
1560    function __ballotStorageupdateBallotMemo(
1561        uint256 _ballotId,
1562        bytes _memo
1563    )
1564        private
1565        notDisabled
1566    {
1567        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1568        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1569        //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1570        __ballotStorageballotBasicMap[_ballotId].memo = _memo;
1571        emit BallotUpdated (_ballotId, msg.sender);
1572    }
```

The code meets the specification.

Formal Verification Request 86

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

27, Nov 2019
57.1 ms

Line 1575 in File fv.sol

```
1575    //@CTK NO_BUF_OVERFLOW
```

Line 1582-1597 in File fv.sol

```
1582    function __ballotStorageupdateBallotDuration(
1583        uint256 _ballotId,
1584        uint256 _duration
1585    )
1586        private
1587      // notDisabled
1588      // onlyValidDuration(_duration)
1589    {
1590        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1591        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1592        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1593
1594        //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1595        __ballotStorageballotBasicMap[_ballotId].duration = _duration;
1596        emit BallotUpdated (_ballotId, msg.sender);
1597    }
```

The code meets the specification.

Formal Verification Request 87

__ballotStorageupdateBallotDuration

27, Nov 2019
7.62 ms

Line 1576-1581 in File fv.sol

```
1576    /*@CTK __ballotStorageupdateBallotDuration
1577      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1578      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1579      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1580      @post __post.__ballotStorageballotBasicMap[_ballotId].duration == _duration
1581    */
```

Line 1582-1597 in File fv.sol

```
1582    function __ballotStorageupdateBallotDuration(
1583        uint256 _ballotId,
1584        uint256 _duration
1585    )
1586        private
1587      // notDisabled
1588      // onlyValidDuration(_duration)
1589    {
1590        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
              Ballot");
1591        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
               finalized");
1592        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
              Ready), "Not Ready State");
1593
1594        //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1595        __ballotStorageballotBasicMap[_ballotId].duration = _duration;
1596        emit BallotUpdated (_ballotId, msg.sender);
1597    }
```

The code meets the specification.

Formal Verification Request 88

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

27, Nov 2019
70.9 ms

Line 1600 in File fv.sol

```
1600    //@CTK NO_BUF_OVERFLOW
```

Line 1609-1623 in File fv.sol

```
1609    function __ballotStorageupdateBallotMemberLockAmount(
1610        uint256 _ballotId,
1611        uint256 _lockAmount
1612    )
1613        private
1614        notDisabled
```

```
1615    {
1616        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1617        require(__ballotStorageballotMemberMap[_ballotId].id == _ballotId, "not existed
                 BallotMember");
1618        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1619        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1620       // BallotMember storage _ballot = __ballotStorageballotMemberMap[_ballotId];
1621        __ballotStorageballotMemberMap[_ballotId].lockAmount = _lockAmount;
1622        emit BallotUpdated (_ballotId, msg.sender);
1623    }
```

The code meets the specification.

Formal Verification Request 89

__ballotStorageupdateBallotMemberLockAmount

27, Nov 2019
9.06 ms

Line 1601-1608 in File fv.sol

```
1601    /*@CTK __ballotStorageupdateBallotMemberLockAmount
1602      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1603      @pre __ballotStorageballotMemberMap[_ballotId].id == _ballotId
1604      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1605      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1606
1607      @post __post.__ballotStorageballotMemberMap[_ballotId].lockAmount == _lockAmount
1608    */
```

Line 1609-1623 in File fv.sol

```
1609    function __ballotStorageupdateBallotMemberLockAmount(
1610        uint256 _ballotId,
1611        uint256 _lockAmount
1612    )
1613        private
1614        notDisabled
1615    {
1616        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1617        require(__ballotStorageballotMemberMap[_ballotId].id == _ballotId, "not existed
                 BallotMember");
1618        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1619        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1620       // BallotMember storage _ballot = __ballotStorageballotMemberMap[_ballotId];
1621        __ballotStorageballotMemberMap[_ballotId].lockAmount = _lockAmount;
1622        emit BallotUpdated (_ballotId, msg.sender);
1623    }
```

The code meets the specification.

Formal Verification Request 90

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

27, Nov 2019
62.88 ms

Line 1627 in File fv.sol

```
1627    //@CTK NO_BUF_OVERFLOW
```

Line 1634-1642 in File fv.sol

```
1634    function __ballotStoragecancelBallot(uint256 _ballotId) private notDisabled {
1635        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1636        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1637
1638        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1639       // BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1640        __ballotStorageballotBasicMap[_ballotId].state = uint256(BallotStates.Canceled)
                ;
1641        emit BallotCanceled (_ballotId);
1642    }
```

The code meets the specification.

Formal Verification Request 91

__ballotStorageupdateBallotDuration

27, Nov 2019
7.04 ms

Line 1628-1633 in File fv.sol

```
1628    /*@CTK __ballotStorageupdateBallotDuration
1629      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1630      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1631      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1632      @post __post.__ballotStorageballotBasicMap[_ballotId].state == 5
1633    */
```

Line 1634-1642 in File fv.sol

```
1634    function __ballotStoragecancelBallot(uint256 _ballotId) private notDisabled {
1635        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1636        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1637
1638        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1639       // BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1640        __ballotStorageballotBasicMap[_ballotId].state = uint256(BallotStates.Canceled)
                ;
1641        emit BallotCanceled (_ballotId);
```

```
1642    }
```

The code meets the specification.

Formal Verification Request 92

__ballotStoragefinalizeBallot

27, Nov 2019
8.43 ms

Line 1646-1652 in File fv.sol

```
1646    /*@CTK "__ballotStoragefinalizeBallot"
1647      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1648      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1649      @pre (_ballotState == uint256(BallotStates.Accepted)) || (_ballotState ==
              uint256(BallotStates.Rejected)) || (_ballotState == uint256(BallotStates.
              NotApplicable))
1650      @post __post.__ballotStorageballotBasicMap[_ballotId].state == _ballotState
1651      @post __post.__ballotStorageballotBasicMap[_ballotId].isFinalized
1652    */
```

Line 1653-1664 in File fv.sol

```
1653    function __ballotStoragefinalizeBallot(uint256 _ballotId, uint256 _ballotState)
            private notDisabled {
1654        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1655        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1656        require((_ballotState == uint256(BallotStates.Accepted))
1657            || (_ballotState == uint256(BallotStates.Rejected))
1658            || (_ballotState == uint256(BallotStates.NotApplicable)), "Invalid Ballot
                    State");
1659
1660       // BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1661        __ballotStorageballotBasicMap[_ballotId].state = _ballotState;
1662        __ballotStorageballotBasicMap[_ballotId].isFinalized = true;
1663        emit BallotFinalized (_ballotId, _ballotState);
1664    }
```

The code meets the specification.

Formal Verification Request 93

If method completes, integer overflow would not happen.

27, Nov 2019
24.4 ms

Line 1670 in File fv.sol

```
1670    //@CTK NO_OVERFLOW
```

Line 1672-1689 in File fv.sol

```
1672    function __ballotStoragegetVote(uint256 _voteId) public view returns (
1673        uint256 voteId,
1674        uint256 ballotId,
1675        address voter,
1676        uint256 decision,
1677        uint256 power,
1678        uint256 time
1679    )
1680    {
1681        require(__ballotStoragevoteMap[_voteId].voteId == _voteId, "not existed voteId"
                );
1682        Vote memory _vote = __ballotStoragevoteMap[_voteId];
1683        voteId = _vote.voteId;
1684        ballotId = _vote.ballotId;
1685        voter = _vote.voter;
1686        decision = _vote.decision;
1687        power = _vote.power;
1688        time = _vote.time;
1689    }
```

The code meets the specification.

Formal Verification Request 94

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

27, Nov 2019
0.52 ms

Line 1671 in File fv.sol

```
1671    //@CTK NO_BUF_OVERFLOW
```

Line 1672-1689 in File fv.sol

```
1672    function __ballotStoragegetVote(uint256 _voteId) public view returns (
1673        uint256 voteId,
1674        uint256 ballotId,
1675        address voter,
1676        uint256 decision,
1677        uint256 power,
1678        uint256 time
1679    )
1680    {
1681        require(__ballotStoragevoteMap[_voteId].voteId == _voteId, "not existed voteId"
                );
1682        Vote memory _vote = __ballotStoragevoteMap[_voteId];
1683        voteId = _vote.voteId;
1684        ballotId = _vote.ballotId;
1685        voter = _vote.voter;
1686        decision = _vote.decision;
1687        power = _vote.power;
1688        time = _vote.time;
1689    }
```

The code meets the specification.

Formal Verification Request 95

If method completes, integer overflow would not happen.

27, Nov 2019
7.39 ms

Line 1691 in File fv.sol

```
1691    //@CTK NO_OVERFLOW
```

Line 1693-1703 in File fv.sol

```
1693    function __ballotStoragegetBallotPeriod(uint256 _id) public view returns (
1694        uint256 startTime,
1695        uint256 endTime,
1696        uint256 duration
1697    )
1698    {
1699        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1700        startTime = tBallot.startTime;
1701        endTime = tBallot.endTime;
1702        duration = tBallot.duration;
1703    }
```

The code meets the specification.

Formal Verification Request 96

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

27, Nov 2019
0.38 ms

Line 1692 in File fv.sol

```
1692    //@CTK NO_BUF_OVERFLOW
```

Line 1693-1703 in File fv.sol

```
1693    function __ballotStoragegetBallotPeriod(uint256 _id) public view returns (
1694        uint256 startTime,
1695        uint256 endTime,
1696        uint256 duration
1697    )
1698    {
1699        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1700        startTime = tBallot.startTime;
1701        endTime = tBallot.endTime;
1702        duration = tBallot.duration;
1703    }
```

The code meets the specification.

Formal Verification Request 97

If method completes, integer overflow would not happen.

27, Nov 2019
7.34 ms

Line 1705 in File fv.sol

```
1705    //@CTK NO_OVERFLOW
```

Line 1707-1718 in File fv.sol

```
1707    function __ballotStoragegetBallotVotingInfo(uint256 _id) public view returns (
1708        uint256 totalVoters,
1709        uint256 powerOfAccepts,
1710        uint256 powerOfRejects
1711
1712    )
1713    {
1714        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1715        totalVoters = tBallot.totalVoters;
1716        powerOfAccepts = tBallot.powerOfAccepts;
1717        powerOfRejects = tBallot.powerOfRejects;
1718    }
```

The code meets the specification.

Formal Verification Request 98

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

27, Nov 2019
0.41 ms

Line 1706 in File fv.sol

```
1706    //@CTK NO_BUF_OVERFLOW
```

Line 1707-1718 in File fv.sol

```
1707    function __ballotStoragegetBallotVotingInfo(uint256 _id) public view returns (
1708        uint256 totalVoters,
1709        uint256 powerOfAccepts,
1710        uint256 powerOfRejects
1711
1712    )
1713    {
1714        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1715        totalVoters = tBallot.totalVoters;
1716        powerOfAccepts = tBallot.powerOfAccepts;
1717        powerOfRejects = tBallot.powerOfRejects;
1718    }
```

The code meets the specification.

Formal Verification Request 99

If method completes, integer overflow would not happen.

27, Nov 2019
7.18 ms

Line 1720 in File fv.sol

```
1720    //@CTK NO_OVERFLOW
```

Line 1722-1732 in File fv.sol

```
1722    function __ballotStoragegetBallotState(uint256 _id) public view returns (
1723        uint256 ballotType,
1724        uint256 state,
1725        bool isFinalized
1726    )
1727    {
1728        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1729        ballotType = tBallot.ballotType;
1730        state = tBallot.state;
1731        isFinalized = tBallot.isFinalized;
1732    }
```

The code meets the specification.

Formal Verification Request 100

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

27, Nov 2019
0.42 ms

Line 1721 in File fv.sol

```
1721    //@CTK NO_BUF_OVERFLOW
```

Line 1722-1732 in File fv.sol

```
1722    function __ballotStoragegetBallotState(uint256 _id) public view returns (
1723        uint256 ballotType,
1724        uint256 state,
1725        bool isFinalized
1726    )
1727    {
1728        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1729        ballotType = tBallot.ballotType;
1730        state = tBallot.state;
1731        isFinalized = tBallot.isFinalized;
1732    }
```

The code meets the specification.

Formal Verification Request 101

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

27, Nov 2019
1.18 ms

Line 1735 in File fv.sol

```
1735    //@CTK NO_BUF_OVERFLOW
```

Line 1736-1758 in File fv.sol

```
1736    function __ballotStorage_createBallot(
1737        uint256 _id,
1738        uint256 _ballotType,
1739        address _creator
1740    )
1741        internal
1742    {
1743
1744        require(__ballotStorageballotBasicMap[_id].id != _id, "Already existed ballot")
                ;
1745
1746        BallotBasic memory newBallot;
1747
1748        newBallot.id = _id;
1749        newBallot.ballotType = _ballotType;
1750        newBallot.creator = _creator;
1751       // newBallot.memo = _memo;
1752        newBallot.state = uint256(BallotStates.Ready);
1753        newBallot.isFinalized = false;
1754       //   newBallot.duration = _duration;
1755         __ballotStorageballotBasicMap[_id] = newBallot;
1756        __ballotStorageballotCount = __ballotStorageballotCount.add(1);
1757        emit BallotCreated(_id, _ballotType, _creator);
1758    }
```

The code meets the specification.

Formal Verification Request 102

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

27, Nov 2019
12.52 ms

Line 1761 in File fv.sol

```
1761    //@CTK NO_BUF_OVERFLOW
```

Line 1762-1800 in File fv.sol

```
1762    function __ballotStorage_areMemberBallotParamValid(
1763        uint256 _ballotType,
1764        address _oldMemberAddress,
1765        address _newMemberAddress,
1766        bytes _newName,
1767        bytes _newNodeId, // admin.nodeInfo.id is 512 bit public key
```

```
1768        bytes _newNodeIp,
1769        uint _newNodePort
1770    )
1771        internal
1772        pure
1773        returns (bool)
1774    {
1775        require((_ballotType >= uint256(BallotTypes.MemberAdd))
1776            && (_ballotType <= uint256(BallotTypes.MemberChange)), "Invalid Ballot Type
                ");
1777
1778        if (_ballotType == uint256(BallotTypes.MemberRemoval)){
1779            require(_oldMemberAddress != address(0), "Invalid old member address");
1780            require(_newMemberAddress == address(0), "Invalid new member address");
1781            require(_newName.length == 0, "Invalid new node name");
1782            require(_newNodeId.length == 0, "Invalid new node id");
1783            require(_newNodeIp.length == 0, "Invalid new node IP");
1784            require(_newNodePort == 0, "Invalid new node Port");
1785        }else {
1786            require(_newName.length > 0, "Invalid new node name");
1787            require(_newNodeId.length == 64, "Invalid new node id");
1788            require(_newNodeIp.length > 0, "Invalid new node IP");
1789            require(_newNodePort > 0, "Invalid new node Port");
1790            if (_ballotType == uint256(BallotTypes.MemberAdd)) {
1791                require(_oldMemberAddress == address(0), "Invalid old member address");
1792                require(_newMemberAddress != address(0), "Invalid new member address");
1793            } else if (_ballotType == uint256(BallotTypes.MemberChange)) {
1794                require(_oldMemberAddress != address(0), "Invalid old member address");
1795                require(_newMemberAddress != address(0), "Invalid new member address");
1796            }
1797        }
1798
1799        return true;
1800    }
```

The code meets the specification.

Formal Verification Request 103

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

27, Nov 2019
5.88 ms

Line 1803 in File fv.sol

```
1803    //@CTK NO_BUF_OVERFLOW
```

Line 1804-1821 in File fv.sol

```
1804    function __ballotStorage_areVariableBallotParamValid(
1805        uint256 _ballotType,
1806        bytes32 _envVariableName,
1807        uint256 _envVariableType,
1808        bytes _envVariableValue
1809    )
1810        internal
1811        pure
```

```
1812        returns (bool)
1813    {
1814          require(_ballotType == uint256(BallotTypes.EnvValChange), "Invalid Ballot
                Type");
1815    //    require(_envVariableName > 0, "Invalid environment variable name");
1816          require(_envVariableType >= uint256(VariableTypes.Int), "Invalid environment
                variable Type");
1817          require(_envVariableType <= uint256(VariableTypes.String), "Invalid
                environment variable Type");
1818          require(_envVariableValue.length > 0, "Invalid environment variable value");
1819
1820        return true;
1821    }
```

The code meets the specification.

Formal Verification Request 104

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

27, Nov 2019
14.71 ms

Line 1825 in File fv.sol

```
1825    //@CTK NO_BUF_OVERFLOW
```

Line 1833-1855 in File fv.sol

```
1833    function __ballotStorage_updateBallotForVote(
1834        uint256 _ballotId,
1835        address _voter,
1836        uint256 _decision,
1837        uint256 _power
1838    )
1839        internal
1840    {
1841
1842        //1.get ballotBasic
1843        BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1844        //2.
1845        __ballotStoragehasVotedMap[_ballotId][_voter] = true;
1846        //3. update totalVoters
1847        __ballotStorageballotBasicMap[_ballotId].totalVoters =
                __ballotStorageballotBasicMap[_ballotId].totalVoters.add(1);
1848        //4. Update power of accept/reject
1849        if (_decision == uint256(DecisionTypes.Accept)){
1850            __ballotStorageballotBasicMap[_ballotId].powerOfAccepts =
                    __ballotStorageballotBasicMap[_ballotId].powerOfAccepts.add(_power);
1851        }
1852        else {
1853            __ballotStorageballotBasicMap[_ballotId].powerOfRejects =
                    __ballotStorageballotBasicMap[_ballotId].powerOfRejects.add(_power);
1854        }
1855    }
```

The code meets the specification.

Formal Verification Request 105

__ballotStorage_updateBallotForVote

27, Nov 2019
75.18 ms

Line 1826-1832 in File fv.sol

```
1826    /*@CTK __ballotStorage_updateBallotForVote
1827      @tag assume_completion
1828      @post __post.__ballotStoragehasVotedMap[_ballotId][_voter] == true
1829      @post __post.__ballotStorageballotBasicMap[_ballotId].totalVoters ==
              __ballotStorageballotBasicMap[_ballotId].totalVoters + 1
1830      @post _decision == 1 -> __post.__ballotStorageballotBasicMap[_ballotId].
              powerOfAccepts == __ballotStorageballotBasicMap[_ballotId].powerOfAccepts +
              _power
1831      @post _decision != 1 -> __post.__ballotStorageballotBasicMap[_ballotId].
              powerOfRejects == __ballotStorageballotBasicMap[_ballotId].powerOfRejects +
              _power
1832    */
```

Line 1833-1855 in File fv.sol

```
1833    function __ballotStorage_updateBallotForVote(
1834        uint256 _ballotId,
1835        address _voter,
1836        uint256 _decision,
1837        uint256 _power
1838    )
1839        internal
1840    {
1841
1842        //1.get ballotBasic
1843        BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1844        //2.
1845        __ballotStoragehasVotedMap[_ballotId][_voter] = true;
1846        //3. update totalVoters
1847        __ballotStorageballotBasicMap[_ballotId].totalVoters =
              __ballotStorageballotBasicMap[_ballotId].totalVoters.add(1);
1848        //4. Update power of accept/reject
1849        if (_decision == uint256(DecisionTypes.Accept)){
1850            __ballotStorageballotBasicMap[_ballotId].powerOfAccepts =
                  __ballotStorageballotBasicMap[_ballotId].powerOfAccepts.add(_power);
1851        }
1852        else {
1853            __ballotStorageballotBasicMap[_ballotId].powerOfRejects =
                  __ballotStorageballotBasicMap[_ballotId].powerOfRejects.add(_power);
1854        }
1855    }
```

The code meets the specification.

Formal Verification Request 106

Method will not encounter an assertion failure.

26, Nov 2019

⏱ 53.3 ms

Line 13 in File enum.sol

```
13  //@CTK NO_ASF
```

Line 16-26 in File enum.sol

```
16  function test_enum() {
17    assert(Status.Pending == Status.Pending);
18    assert(Status.Pending != Status.Success);
19
20    Status x = Status.Pending;
21    Status y = Status.Success;
22    assert(x != y);
23
24    Status z = Status.Success;
25    assert(y == z);
26  }
```

✅ The code meets the specification.

Formal Verification Request 107

If method completes, integer overflow would not happen.

📅 26, Nov 2019
⏱ 1.1 ms

Line 14 in File enum.sol

```
14  //@CTK NO_OVERFLOW
```

Line 16-26 in File enum.sol

```
16  function test_enum() {
17    assert(Status.Pending == Status.Pending);
18    assert(Status.Pending != Status.Success);
19
20    Status x = Status.Pending;
21    Status y = Status.Success;
22    assert(x != y);
23
24    Status z = Status.Success;
25    assert(y == z);
26  }
```

✅ The code meets the specification.

Formal Verification Request 108

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

📅 26, Nov 2019
⏱ 1.07 ms

Line 15 in File enum.sol

```
15  //@CTK NO_BUF_OVERFLOW
```

Line 16-26 in File enum.sol

```
16  function test_enum() {
17    assert(Status.Pending == Status.Pending);
18    assert(Status.Pending != Status.Success);
19
20    Status x = Status.Pending;
21    Status y = Status.Success;
22    assert(x != y);
23
24    Status z = Status.Success;
25    assert(y == z);
26  }
```

✅ The code meets the specification.

Formal Verification Request 109

test_enum

📅 26, Nov 2019
⏱ 13.26 ms

Line 28-31 in File enum.sol

```
28  /*@CTK test_enum
29    @post __return == (input == Status.Success)
30    @tag no_overflow
31    */
```

Line 32-34 in File enum.sol

```
32  function check_success(Status input) returns (bool) {
33    return input == Status.Success;
34  }
```

✅ The code meets the specification.

Source Code with CertiK Labels

File fv.sol

```
pragma solidity ^0.4.26;

library SafeMath {

  /**
  * @dev Multiplies two numbers, reverts on overflow.
  */
  /*@CTK "SafeMath mul zero"
      @tag spec
      @tag is_pure
      @pre (a == 0)
      @post __return == 0
    */
    /*@CTK "SafeMath mul nonzero"
      @tag spec
      @tag is_pure
      @pre (a != 0)
      @post (a * b / a != b) == __reverted
      @post !__reverted -> __return == a * b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (a == 0) {
      return 0;
    }

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

    return c;
  }

  /**
  * @dev Integer division of two numbers truncating the quotient, reverts on division
      by zero.
  */
  /*@CTK "SafeMath div"
      @tag spec
      @tag is_pure
      @post (b == 0) == __reverted
      @post !__reverted -> __return == a / b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b > 0); // Solidity only automatically asserts when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
```

```
54
55      return c;
56    }
57
58    /**
59    * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater
          than minuend).
60    */
61    /*@CTK "SafeMath sub"
62        @tag spec
63        @tag is_pure
64        @post (b > a) == __reverted
65        @post !__reverted -> __return == a - b
66        @post !__reverted -> !__has_overflow
67        @post !__reverted -> !__has_assertion_failure
68        @post !(__has_buf_overflow)
69      */
70    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
71      require(b <= a);
72      uint256 c = a - b;
73
74      return c;
75    }
76
77    /**
78    * @dev Adds two numbers, reverts on overflow.
79    */
80    function add(uint256 a, uint256 b) internal pure returns (uint256) {
81      uint256 c = a + b;
82      require(c >= a);
83
84      return c;
85    }
86
87    /**
88    * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
89    * reverts when dividing by zero.
90    */
91    /*@CTK "SafeMath mod"
92        @tag spec
93        @tag is_pure
94        @post (b == 0) == __reverted
95        @post !__reverted -> __return == a % b
96        @post !__reverted -> !__has_overflow
97        @post !__reverted -> !__has_assertion_failure
98        @post !(__has_buf_overflow)
99      */
100   function mod(uint256 a, uint256 b) internal pure returns (uint256) {
101     require(b != 0);
102     return a % b;
103   }
104 }
105
106
107 contract GovImp {
108
109     using SafeMath for uint256;
110
```

```
111     uint256 private _guardCounter;
112
113     enum BallotStates {
114         Invalid,
115         Ready,          // create Ballot but not start
116         InProgress,     // in voting
117         Accepted,       // ballot was passed
118         Rejected,       // ballot was rejected.
119         Canceled,       // ballot was canceled.
120         NotApplicable    // ballot's result was not Applicable.
121     }
122
123     enum DecisionTypes {
124         Invalid,
125         Accept,
126         Reject
127     }
128
129     enum BallotTypes {
130         Invalid,
131         MemberAdd, // new Member Address, new Node id, new Node ip, new Node port
132         MemberRemoval, // old Member Address
133         MemberChange,    // Old Member Address, New Member Address, new Node id, New
                Node ip, new Node port
134         GovernanceChange, // new Governace Impl Address
135         EnvValChange   // Env variable name, type , value
136     }
137
138     bytes32 public constant BLOCKS_PER_NAME = keccak256("blocksPer");
139     // uint256 public constant BLOCKS_PER_TYPE = uint256(VariableTypes.Uint);
140
141     bytes32 public constant BALLOT_DURATION_MIN_NAME = keccak256("ballotDurationMin");
142     // uint256 public constant BALLOT_DURATION_MIN_TYPE = uint256(VariableTypes.Uint);
143
144     bytes32 public constant BALLOT_DURATION_MAX_NAME = keccak256("ballotDurationMax");
145     // uint256 public constant BALLOT_DURATION_MAX_TYPE = uint256(VariableTypes.Uint);
146
147     bytes32 public constant STAKING_MIN_NAME = keccak256("stakingMin");
148     // uint256 public constant STAKING_MIN_TYPE = uint256(VariableTypes.Uint);
149
150     bytes32 public constant STAKING_MAX_NAME = keccak256("stakingMax");
151     // uint256 public constant STAKING_MAX_TYPE = uint256(VariableTypes.Uint);
152
153     bytes32 public constant GAS_PRICE_NAME = keccak256("gasPrice");
154     // uint256 public constant GAS_PRICE_TYPE = uint256(VariableTypes.Uint);
155
156     bytes32 public constant MAX_IDLE_BLOCK_INTERVAL_NAME = keccak256("
            MaxIdleBlockInterval");
157     // uint256 public constant MAX_IDLE_BLOCK_INTERVAL_TYPE = uint256(VariableTypes.
            Uint);
158
159     enum VariableTypes {
160         Invalid,
161         Int,
162         Uint,
163         Address,
164         Bytes32,
165         Bytes,
```

```
166         String
167     }
168
169     // bytes32 internal constant TEST_INT = keccak256("TEST_INT");
170     // bytes32 internal constant TEST_ADDRESS = keccak256("TEST_ADDRESS");
171     // bytes32 internal constant TEST_BYTES32 = keccak256("TEST_BYTES32");
172     // bytes32 internal constant TEST_BYTES = keccak256("TEST_BYTES");
173     // bytes32 internal constant TEST_STRING = keccak256("TEST_STRING");
174     modifier nonReentrant() {
175         _guardCounter += 1;
176         uint256 localCounter = _guardCounter;
177         _;
178         require(localCounter == _guardCounter);
179     }
180
181
182     uint public modifiedBlock;
183
184     // For voting member
185     mapping(uint256 => address) internal members;
186     mapping(address => uint256) internal memberIdx;
187     uint256 internal memberLength;
188
189     // For reward member
190     mapping(uint256 => address) internal rewards;
191     mapping(address => uint256) internal rewardIdx;
192
193     // For enode
194     struct Node {
195         bytes name;
196         bytes enode;
197         bytes ip;
198         uint port;
199     }
200
201     mapping(uint256 => Node) internal nodes;
202     mapping(address => uint256) internal nodeIdxFromMember;
203     mapping(uint256 => address) internal nodeToMember;
204     uint256 internal nodeLength;
205
206     // For ballot
207     uint256 public ballotLength;
208     uint256 public voteLength;
209     uint256 internal ballotInVoting;
210
211     constructor() public {}
212
213     /*@CTK isMember
214       @post __return == (memberIdx[addr] != 0)
215     */
216     function isMember(address addr) public view returns (bool) { return (memberIdx[
            addr] != 0); }
217     /*@CTK getMember
218       @post __return == members[idx]
219     */
220     function getMember(uint256 idx) public view returns (address) { return members[idx
            ]; }
221     /*@CTK getMemberLength
```

```
222       @post __return == memberLength
223     */
224     function getMemberLength() public view returns (uint256) { return memberLength; }
225     /*@CTK getReward
226       @post __return == rewards[idx]
227     */
228     function getReward(uint256 idx) public view returns (address) { return rewards[idx
            ]; }
229     /*@CTK getNodeIdxFromMember
230       @post __return == nodeIdxFromMember[addr]
231     */
232     function getNodeIdxFromMember(address addr) public view returns (uint256) { return
             nodeIdxFromMember[addr]; }
233     /*@CTK getMemberFromNodeIdx
234       @post __return == nodeToMember[idx]
235     */
236     function getMemberFromNodeIdx(uint256 idx) public view returns (address) { return
            nodeToMember[idx]; }
237     /*@CTK getNodeLength
238       @post __return == nodeLength
239     */
240     function getNodeLength() public view returns (uint256) { return nodeLength; }
241
242     //@CTK NO_OVERFLOW
243     //@CTK NO_BUF_OVERFLOW
244     //@CTK NO_ASF
245     function getNode(uint256 idx) public view returns (bytes name, bytes enode, bytes
            ip, uint port) {
246         return (nodes[idx].name, nodes[idx].enode, nodes[idx].ip, nodes[idx].port);
247     }
248
249     //@CTK NO_OVERFLOW
250     //@CTK NO_BUF_OVERFLOW
251     //@CTK NO_ASF
252     function getBallotInVoting() public view returns (uint256) { return ballotInVoting
            ; }
253
254     event MemberAdded(address indexed addr);
255     event MemberRemoved(address indexed addr);
256     event MemberChanged(address indexed oldAddr, address indexed newAddr);
257     event EnvChanged(bytes32 envName, uint256 envType, bytes envVal);
258     event MemberUpdated(address indexed addr);
259     // added for case that ballot's result could not be applicable.
260     event NotApplicable(uint256 indexed ballotId, string reason);
261
262     function () public payable {
263         revert();
264     }
265
266     /*@CTK addProposalToAddMember
267       @tag assume_completion
268       @pre member != address(0)
269       @pre name.length > 0
270       @pre ip.length > 0
271       @pre portNlockAmount[0] > 0
272       @pre portNlockAmount[1] > 0
273       @pre memberIdx[member] == 0
274       @post __post.ballotLength == ballotLength + 1
```

```
275     */
276     function addProposalToAddMember(
277         address member,
278         bytes name,
279         bytes enode,
280         bytes ip,
281         uint256[2] portNlockAmount,
282         bytes memo
283     )
284         external
285         returns (uint256 ballotIdx)
286     {
287         require(member != address(0), "Invalid address");
288         require(name.length > 0, "Invalid node name");
289         require(ip.length > 0, "Invalid node IP");
290         require(portNlockAmount[0] > 0, "Invalid node port");
291         require(portNlockAmount[1] > 0, "Invalid lockAmmount");
292         require(!isMember(member), "Already member");
293 
294         ballotIdx = ballotLength.add(1);
295         createBallotForMember(
296             ballotIdx, // ballot id
297             uint256(BallotTypes.MemberAdd), // ballot type
298             msg.sender, // creator
299             address(0), // old member address
300             member, // new member address
301             name,
302             enode, // new enode
303             ip, // new ip
304             portNlockAmount[0] // new port
305         );
306         updateBallotLock(ballotIdx, portNlockAmount[1]);
307         updateBallotMemo(ballotIdx, memo);
308         ballotLength = ballotIdx;
309     }
310 
311 
312     /*@CTK addProposalToRemoveMember
313       @pre member != address(0)
314       @pre member == address(0)
315       @pre memberLength > 1
316       @post __post.ballotLength == ballotLength + 1
317     */
318     function addProposalToRemoveMember(
319         address member,
320         uint256 lockAmount,
321         bytes memo
322     )
323         external
324         returns (uint256 ballotIdx)
325     {
326         require(member != address(0), "Invalid address");
327         require(isMember(member), "Non-member");
328         require(getMemberLength() > 1, "Cannot remove a sole member");
329 
330         ballotIdx = ballotLength.add(1);
331         createBallotForMember(
332             ballotIdx, // ballot id
```

```
            uint256(BallotTypes.MemberRemoval), // ballot type
            msg.sender, // creator
            member, // old member address
            address(0), // new member address
            new bytes(0), // new name
            new bytes(0), // new enode
            new bytes(0), // new ip
            0 // new port
        );
        updateBallotLock(ballotIdx, lockAmount);
        updateBallotMemo(ballotIdx, memo);
        ballotLength = ballotIdx;
    }

    /*@CTK addProposalToChangeMember
      @pre targetNnewMember[0] != address(0)
      @pre targetNnewMember[1] != address(0)
      @pre nName.length > 0
      @pre nIp.length > 0
      @pre portNlockAmount[0] > 0
      @pre portNlockAmount[1] > 0
      @pre targetNnewMember[0] == address(0)
      @post __post.ballotLength == ballotLength + 1
    */
    function addProposalToChangeMember(
        address[2] targetNnewMember,
        bytes nName,
        bytes nEnode,
        bytes nIp,
        uint256[2] portNlockAmount,
        bytes memo
    )
        external
        returns (uint256 ballotIdx)
    {
        require(targetNnewMember[0] != address(0), "Invalid old Address");
        require(targetNnewMember[1] != address(0), "Invalid new Address");
        require(nName.length > 0, "Invalid node name");
        require(nIp.length > 0, "Invalid node IP");
        require(portNlockAmount[0] > 0, "Invalid node port");
        require(portNlockAmount[1] > 0, "Invalid lockAmmount");
        require(isMember(targetNnewMember[0]), "Non-member");

        ballotIdx = ballotLength.add(1);
        createBallotForMember(
            ballotIdx, // ballot id
            uint256(BallotTypes.MemberChange), // ballot type
            msg.sender, // creator
            targetNnewMember[0], // old member address
            targetNnewMember[1], // new member address
            nName, //new Name
            nEnode, // new enode
            nIp, // new ip
            portNlockAmount[0] // new port
        );
        updateBallotLock(ballotIdx, portNlockAmount[1]);
        updateBallotMemo(ballotIdx, memo);
        ballotLength = ballotIdx;
```

```
391     }
392
393     /*@CTK "addProposalToChangeGov"
394       @tag assume_completion
395       @pre newGovAddr != address(0)
396       @post __post.ballotLength == ballotLength + 1
397     */
398     //@CTK NO_OVERFLOW
399     function addProposalToChangeGov(
400         address newGovAddr,
401         bytes memo
402     )
403         external
404         returns (uint256 ballotIdx)
405     {
406         require(newGovAddr != address(0), "Implementation cannot be zero");
407         require(newGovAddr != implementation(), "Same contract address");
408
409         ballotIdx = ballotLength.add(1);
410         IBallotStorage(getBallotStorageAddress()).createBallotForAddress(
411             ballotLength.add(1), // ballot id
412             uint256(BallotTypes.GovernanceChange), // ballot type
413             msg.sender, // creator
414             newGovAddr // new governance address
415         );
416         updateBallotMemo(ballotIdx, memo);
417         ballotLength = ballotIdx;
418     }
419
420
421     /*@CTK "addProposalToChangeEnv"
422      @tag assume_completion
423      @pre 1 <= envType && envType <= 6
424      @post __post.ballotLength == ballotLength + 1
425     */
426     //@CTK NO_OVERFLOW
427     function addProposalToChangeEnv(
428         bytes32 envName,
429         uint256 envType,
430         bytes envVal,
431         bytes memo
432     )
433         external
434         returns (uint256 ballotIdx)
435     {
436         // require(envName != 0, "Invalid name");
437         require(uint256(VariableTypes.Int) <= envType && envType <= uint256(
                VariableTypes.String), "Invalid type");
438         //ctk start hacking
439         require(1 <= envType && envType <= 6, "Invalid type");
440         //ctk end hacking
441
442
443         ballotIdx = ballotLength.add(1);
444         __ballotStoragecreateBallotForVariable(
445             ballotIdx, // ballot id
446             uint256(BallotTypes.EnvValChange), // ballot type
447             msg.sender, // creator
```

```
448             envName, // env name
449             envType, // env type
450             envVal // env value
451         );
452         updateBallotMemo(ballotIdx, memo);
453         ballotLength = ballotIdx;
454     }
455     //@CTK NO_BUF_OVERFLOW
456     function vote(uint256 ballotIdx, bool approval) private nonReentrant {
457         // Check if some ballot is in progress
458         checkUnfinalized(ballotIdx);
459 
460         // Check if the ballot can be voted
461         uint256 ballotType = checkVotable(ballotIdx);
462 
463         // Vote
464         createVote(ballotIdx, approval);
465 
466         // Finalize
467        // (, uint256 accept, uint256 reject) = getBallotVotingInfo(ballotIdx);
468 
469         BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
470         uint256 accept = tBallot.powerOfAccepts;
471         uint256 reject = tBallot.powerOfRejects;
472 
473         uint256 threshold = getThreshold();
474         if (accept >= threshold || reject >= threshold) {
475             finalizeVote(ballotIdx, ballotType, accept > reject);
476         }
477     }
478 
479     function getMinStaking() public view returns (uint256) {
480         return 100;//__stakinggetEnvStorageAddress().getStakingMin();
481     }
482 
483     function getMaxStaking() public view returns (uint256) {
484         return 200;//__stakinggetEnvStorageAddress().getStakingMax();
485     }
486 
487     function getMinVotingDuration() public view returns (uint256) {
488         return 200;//__stakinggetEnvStorageAddress().getBallotDurationMin();
489     }
490 
491     function getMaxVotingDuration() public view returns (uint256) {
492         return 300;//__stakinggetEnvStorageAddress().getBallotDurationMax();
493     }
494 
495     function getThreshold() public pure returns (uint256) { return 5100; } // 51% from
          5100 of 10000
496 
497     //@CTK NO_BUF_OVERFLOW
498     function checkUnfinalized(uint256 ballotIdx) private {
499         if (ballotInVoting != 0) {
500             uint256 state = __ballotStorageballotBasicMap[ballotInVoting].state;
501             uint256 endTime = __ballotStorageballotBasicMap[ballotInVoting].endTime;
502             if (state == uint256(BallotStates.InProgress)) {
503                 if (endTime < block.timestamp) {
504                     finalizeBallot(ballotInVoting, uint256(BallotStates.Rejected));
```

```
                ballotInVoting = 0;
            } else if (ballotIdx != ballotInVoting) {
                require(false, "Now in voting with different ballot");
            }
        }
    }

    //@CTK NO_BUF_OVERFLOW
    function checkVotable(uint256 ballotIdx) private returns (uint256) {
        BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
        uint256 ballotType = tBallot.ballotType;
        uint256 state = tBallot.state;

        if (state == uint256(BallotStates.Ready)) {
            uint256 duration = __ballotStorageballotBasicMap[ballotInVoting].duration;
            if (duration < getMinVotingDuration()) {
                startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                    getMinVotingDuration()));
            } else if (getMaxVotingDuration() < duration) {
                startBallot(ballotIdx, block.timestamp, block.timestamp.add(
                    getMaxVotingDuration()));
            } else {
                startBallot(ballotIdx, block.timestamp, block.timestamp.add(duration));
            }
            ballotInVoting = ballotIdx;
        } else if (state == uint256(BallotStates.InProgress)) {
            // Nothing to do
        } else {
            require(false, "Expired");
        }
        return ballotType;
    }

    //@CTK NO_BUF_OVERFLOW
    function createVote(uint256 ballotIdx, bool approval) private {
        uint256 voteIdx = voteLength.add(1);
        uint256 weight = __stakingcalcVotingWeightWithScaleFactor(msg.sender, 1e4);
        if (approval) {
            __ballotStoragecreateVote(
                voteIdx,
                ballotIdx,
                msg.sender,
                uint256(DecisionTypes.Accept),
                weight
            );
        } else {
            __ballotStoragecreateVote(
                voteIdx,
                ballotIdx,
                msg.sender,
                uint256(DecisionTypes.Reject),
                weight
            );
        }
        voteLength = voteIdx;
    }

```

```
561     //@CTK NO_BUF_OVERFLOW
562     function finalizeVote(uint256 ballotIdx, uint256 ballotType, bool isAccepted)
            private {
563         uint256 ballotState = uint256(BallotStates.Rejected);
564         if (isAccepted) {
565             ballotState = uint256(BallotStates.Accepted);
566             if (ballotType == uint256(BallotTypes.MemberAdd)) {
567                 if (!addMember(ballotIdx)){
568                     ballotState = uint256(BallotStates.NotApplicable);
569                 }
570             } else if (ballotType == uint256(BallotTypes.MemberRemoval)) {
571                 removeMember(ballotIdx);
572             } else if (ballotType == uint256(BallotTypes.MemberChange)) {
573                 if(!changeMember(ballotIdx)){
574                     ballotState = uint256(BallotStates.NotApplicable);
575                 }
576             } else if (ballotType == uint256(BallotTypes.GovernanceChange)) {
577                 changeGov(ballotIdx);
578             } else if (ballotType == uint256(BallotTypes.EnvValChange)) {
579                 applyEnv(ballotIdx);
580             }
581         }
582         finalizeBallot(ballotIdx, ballotState);
583         ballotInVoting = 0;
584     }
585
586     //@CTK NO_BUF_OVERFLOW
587     function fromValidBallot(uint256 ballotIdx, uint256 targetType) private view {
588         BallotBasic memory tBallot = __ballotStorageballotBasicMap[ballotIdx];
589         uint256 ballotType = tBallot.ballotType;
590         uint256 state = tBallot.state;
591         require(ballotType == targetType, "Invalid voting type");
592         require(state == uint(BallotStates.InProgress), "Invalid voting state");
593         BallotBasic memory tBallot2 = __ballotStorageballotBasicMap[ballotIdx];
594         uint256 accept = tBallot2.powerOfAccepts;
595         uint256 reject = tBallot2.powerOfRejects;
596         require(accept >= getThreshold() || reject >= getThreshold(), "Not yet
                finalized");
597     }
598     //@CTK NO_BUF_OVERFLOW
599     function addMember(uint256 ballotIdx) private returns (bool) {
600         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberAdd));
601
602         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
603         address addr = __ballotStorageballotMemberMap[ballotIdx].newMemberAddress;
604         bytes memory name = __ballotStorageballotMemberMap[ballotIdx].newNodeName;
605         bytes memory enode = __ballotStorageballotMemberMap[ballotIdx].newNodeId;
606         bytes memory ip = __ballotStorageballotMemberMap[ballotIdx].newNodeIp;
607         uint port = __ballotStorageballotMemberMap[ballotIdx].newNodePort;
608         uint256 lockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
609
610         if (isMember(addr)) {
611             emit NotApplicable(ballotIdx, "Already a member");
612             return true; // Already member. it is abnormal case, but passed.
613         }
614
615         // Lock
```

```
616         // require(getMinStaking() <= lockAmount && lockAmount <= getMaxStaking(), "
                Invalid lock amount");
617         if( lockAmount < getMinStaking() || getMaxStaking() < lockAmount ){
618             emit NotApplicable(ballotIdx, "Invalid lock amount");
619             return false;
620         }
621 
622         if(__stakingavailableBalanceOf(addr) < lockAmount){
623             emit NotApplicable(ballotIdx, "Insufficient balance that can be locked");
624             return false;
625         }
626         lock(addr, lockAmount);
627 
628         // Add voting and reward member
629         uint256 nMemIdx = memberLength.add(1);
630         members[nMemIdx] = addr;
631         memberIdx[addr] = nMemIdx;
632         rewards[nMemIdx] = addr;
633         rewardIdx[addr] = nMemIdx;
634 
635         // Add node
636         uint256 nNodeIdx = nodeLength.add(1);
637         //Node storage node = nodes[nNodeIdx];
638 
639         nodes[nNodeIdx].enode = enode;
640         nodes[nNodeIdx].ip = ip;
641         nodes[nNodeIdx].port = port;
642         nodeToMember[nNodeIdx] = addr;
643         nodeIdxFromMember[addr] = nNodeIdx;
644         nodes[nNodeIdx].name = name;
645         memberLength = nMemIdx;
646         nodeLength = nNodeIdx;
647         modifiedBlock = block.number;
648         emit MemberAdded(addr);
649         return true;
650     }
651     //@CTK NO_BUF_OVERFLOW
652     function removeMember(uint256 ballotIdx) private {
653         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberRemoval));
654 
655 
656         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
657         address addr = __ballotStorageballotMemberMap[ballotIdx].oldMemberAddress;
658         uint256 unlockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
659 
660         if (!isMember(addr)) {
661             emit NotApplicable(ballotIdx, "Not already a member");
662             return; // Non-member. it is abnormal case, but passed
663         }
664 
665         // Remove voting and reward member
666         uint256 removeIdx = memberIdx[addr];
667         address endAddr = members[memberLength];
668         if (memberIdx[addr] != memberLength) {
669             (members[removeIdx], members[memberLength],memberIdx[addr], memberIdx[
                    endAddr] ) = (members[memberLength],address(0), 0, memberIdx[addr] );
670             removeIdx = rewardIdx[addr];
671             endAddr = rewards[memberLength];
```

```
672             (rewards[removeIdx], rewards[memberLength],rewardIdx[addr], rewardIdx[
                    endAddr]) = (rewards[memberLength], address(0),0, rewardIdx[addr]);
673         } else {
674             members[memberLength] = address(0);
675             memberIdx[addr] = 0;
676             rewards[memberLength] = address(0);
677             rewardIdx[addr] = 0;
678         }
679
680         memberLength = memberLength.sub(1);
681
682         // Remove node
683         if (nodeIdxFromMember[addr] != nodeLength) {
684             removeIdx = nodeIdxFromMember[addr];
685             endAddr = nodeToMember[nodeLength];
686
687             //Node storage node = nodes[removeIdx];
688             nodes[removeIdx].name = nodes[nodeLength].name;
689             nodes[removeIdx].enode = nodes[nodeLength].enode;
690             nodes[removeIdx].ip = nodes[nodeLength].ip;
691             nodes[removeIdx].port = nodes[nodeLength].port;
692
693             nodeToMember[removeIdx] = endAddr;
694             nodeIdxFromMember[endAddr]=removeIdx;
695         }
696         nodeToMember[nodeLength] = address(0);
697         nodeIdxFromMember[addr] = 0;
698         delete nodes[nodeLength];
699         nodeLength = nodeLength.sub(1);
700         modifiedBlock = block.number;
701         // Unlock and transfer remained to governance
702         transferLockedAndUnlock(addr, unlockAmount);
703
704         emit MemberRemoved(addr);
705     }
706     //@CTK NO_BUF_OVERFLOW
707     function changeMember(uint256 ballotIdx) private returns (bool) {
708         fromValidBallot(ballotIdx, uint256(BallotTypes.MemberChange));
709
710         //BallotMember storage tBallot = __ballotStorageballotMemberMap[ballotIdx];
711         address addr = __ballotStorageballotMemberMap[ballotIdx].oldMemberAddress;
712         address nAddr = __ballotStorageballotMemberMap[ballotIdx].newMemberAddress;
713         bytes memory name = __ballotStorageballotMemberMap[ballotIdx].newNodeName;
714         bytes memory enode = __ballotStorageballotMemberMap[ballotIdx].newNodeId;
715         bytes memory ip = __ballotStorageballotMemberMap[ballotIdx].newNodeIp;
716         uint port = __ballotStorageballotMemberMap[ballotIdx].newNodePort;
717         uint256 lockAmount = __ballotStorageballotMemberMap[ballotIdx].lockAmount;
718
719         if (!isMember(addr)) {
720             emit NotApplicable(ballotIdx, "Old address is not a member");
721             return false; // Non-member. it is abnormal case.
722         }
723
724         if (addr != nAddr) {
725             if (isMember(nAddr)) {
726                 emit NotApplicable(ballotIdx, "new address is already a member");
727                 return false; // already member. it is abnormal case.
728             }
```

```
729
730             // Lock
731             // require(getMinStaking() <= lockAmount && lockAmount <= getMaxStaking(),
                    "Invalid lock amount");
732
733             if( lockAmount < getMinStaking() || getMaxStaking() < lockAmount ){
734                 emit NotApplicable(ballotIdx, "Invalid lock amount");
735                 return false;
736             }
737
738             if(__stakingavailableBalanceOf(nAddr) < lockAmount){
739                 emit NotApplicable(ballotIdx, "Insufficient balance that can be locked"
                        );
740                 return false;
741             }
742             lock(nAddr, lockAmount);
743
744             // Change member
745             members[memberIdx[addr]] = nAddr;
746             memberIdx[nAddr] = memberIdx[addr];
747             rewards[memberIdx[addr]] = nAddr;
748             rewardIdx[nAddr] = rewardIdx[addr];
749             memberIdx[addr] = 0;
750
751             return true;
752         }
753
754         // Change node
755         uint256 nodeIdx = nodeIdxFromMember[addr];
756         //Node storage node = nodes[nodeIdx];
757         nodes[nodeIdx].name = name;
758         nodes[nodeIdx].enode = enode;
759         nodes[nodeIdx].ip = ip;
760         nodes[nodeIdx].port = port;
761         modifiedBlock = block.number;
762         if (addr != nAddr) {
763             nodeToMember[nodeIdx] = nAddr;
764             nodeIdxFromMember[nAddr] = nodeIdx;
765             nodeIdxFromMember[addr] = 0;
766
767             // Unlock and transfer remained to governance
768             transferLockedAndUnlock(addr, lockAmount);
769
770             emit MemberChanged(addr, nAddr);
771         } else {
772             emit MemberUpdated(addr);
773         }
774     }
775     //@CTK NO_BUF_OVERFLOW
776     function changeGov(uint256 ballotIdx) private {
777         fromValidBallot(ballotIdx, uint256(BallotTypes.GovernanceChange));
778
779         address newImp = __ballotStoragegetBallotAddress(ballotIdx);
780         if (newImp != address(0)) {
781             //setImplementation(newImp);
782             modifiedBlock = block.number;
783         }
784     }
```

```
785     //@CTK NO_BUF_OVERFLOW
786     function applyEnv(uint256 ballotIdx) private {
787         fromValidBallot(ballotIdx, uint256(BallotTypes.EnvValChange));
788 
789 
790        // BallotVariable storage tBallot = __ballotStorageballotVariableMap[ballotIdx
               ];
791         bytes32 envVariableName = __ballotStorageballotVariableMap[ballotIdx].
                envVariableName;
792         uint256 envVariableType = __ballotStorageballotVariableMap[ballotIdx].
                envVariableType;
793         bytes memory envVariableValue = __ballotStorageballotVariableMap[ballotIdx].
                envVariableValue;
794 
795 /*
796         IEnvStorage envStorage = IEnvStorage(getEnvStorageAddress());
797         uint256 uintType = uint256(VariableTypes.Uint);
798         if (envKey == BLOCKS_PER_NAME && envType == uintType) {
799             envStorage.setBlocksPerByBytes(envVal);
800         } else if (envKey == BALLOT_DURATION_MIN_NAME && envType == uintType) {
801             envStorage.setBallotDurationMinByBytes(envVal);
802         } else if (envKey == BALLOT_DURATION_MAX_NAME && envType == uintType) {
803             envStorage.setBallotDurationMaxByBytes(envVal);
804         } else if (envKey == STAKING_MIN_NAME && envType == uintType) {
805             envStorage.setStakingMinByBytes(envVal);
806         } else if (envKey == STAKING_MAX_NAME && envType == uintType) {
807             envStorage.setStakingMaxByBytes(envVal);
808         } else if (envKey == GAS_PRICE_NAME && envType == uintType) {
809             envStorage.setGasPriceByBytes(envVal);
810         } else if (envKey == MAX_IDLE_BLOCK_INTERVAL_NAME && envType == uintType) {
811             envStorage.setMaxIdleBlockIntervalByBytes(envVal);
812         }*/
813 
814         modifiedBlock = block.number;
815 
816         emit EnvChanged(envKey, envType, envVal);
817     }
818 
819     //------------------- Code reduction for creation gas
820     //@CTK NO_BUF_OVERFLOW
821     function createBallotForMember(
822         uint256 id,
823         uint256 bType,
824         address creator,
825         address oAddr,
826         address nAddr,
827         bytes name,
828         bytes enode,
829         bytes ip,
830         uint port
831     )
832         private
833     {
834         __ballotStoragecreateBallotForMember(
835             id, // ballot id
836             bType, // ballot type
837             creator, // creator
838             oAddr, // old member address
```

```
            nAddr, // new member address
            name, // new name
            enode, // new enode
            ip, // new ip
            port // new port
        );
    }

    function updateBallotLock(uint256 id, uint256 amount) private {
        __ballotStorageupdateBallotMemberLockAmount(id, amount);
    }

    function updateBallotMemo(uint256 id, bytes memo) private {
        __ballotStorageupdateBallotMemo(id, memo);
    }

    function startBallot(uint256 id, uint256 s, uint256 e) private {
        __ballotStoragestartBallot(id, s, e);
    }

    function finalizeBallot(uint256 id, uint256 state) private {
        __ballotStoragefinalizeBallot(id, state);
    }

    function getBallotState(uint256 id) private view returns (uint256, uint256, bool)
        {
        return __ballotStoragegetBallotState(id);
    }

    function getBallotPeriod(uint256 id) private view returns (uint256, uint256,
        uint256) {
        return __ballotStoragegetBallotPeriod(id);
    }

    function getBallotVotingInfo(uint256 id) private view returns (uint256, uint256,
        uint256) {
        return __ballotStoragegetBallotVotingInfo(id);
    }

    function getBallotMember(uint256 id) private view returns (address, address, bytes
        , bytes, bytes, uint256, uint256) {
        return __ballotStoragegetBallotMember(id);
    }

    function lock(address addr, uint256 amount) private {
        __stakinglock(addr, amount);
    }

    function unlock(address addr, uint256 amount) private {
        __stakingunlock(addr, amount);
    }

    //@CTK NO_BUF_OVERFLOW
    function transferLockedAndUnlock(address addr, uint256 unlockAmount) private {
        uint256 locked = __stakinglockedBalanceOf(addr);
        if (locked > unlockAmount) {
            __stakingtransferLocked(addr, locked.sub(unlockAmount));
        }
```

```
893         unlock(addr, unlockAmount);
894     }
895     //------------------- Code reduction end
896
897     /*
898         STAKING CONTRACT CODE BEGINS HERE
899     */
900
901     mapping(address => uint256) private __staking_balance;
902     mapping(address => uint256) private __staking_lockedBalance;
903     uint256 private __staking_totalLockedBalance;
904     bool private __stakingrevoked = false;
905
906     event Staked(address indexed payee, uint256 amount, uint256 total, uint256
            available);
907     event Unstaked(address indexed payee, uint256 amount, uint256 total, uint256
            available);
908     event Locked(address indexed payee, uint256 amount, uint256 total, uint256
            available);
909     event Unlocked(address indexed payee, uint256 amount, uint256 total, uint256
            available);
910     event TransferLocked(address indexed payee, uint256 amount, uint256 total, uint256
             available);
911     event Revoked(address indexed owner, uint256 amount);
912
913     /*@CTK __stakingcontstuctor
914       @post __post.__staking_totalLockedBalance == 0
915     */
916     //@CTK NO_BUF_OVERFLOW
917     //@CTK NO_OVERFLOW
918     function __stakingconstructor(address registry, bytes data) public {
919         __staking_totalLockedBalance = 0;
920      //   setRegistry(registry);
921
922         // data is only for test purpose
923         if (data.length == 0)
924             return;
925
926         // []{address, amount}
927         address addr;
928         uint amount;
929         uint ix;
930         uint eix;
931         assembly {
932             ix := add(data, 0x20)
933         }
934         eix = ix + data.length;
935         /*#CTK loop
936           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_balance[addr] == mload(ix)
937           @post forall address(mload(ix)) != address(mload(ix))__pre: __post.
                __staking_lockedBalance[addr] == mload(ix)
938         */
939         while (ix < eix) {
940             assembly {
941                 amount := mload(ix)
942             }
943             addr = address(amount);
```

```
            ix += 0x20;
            require(ix < eix);
            assembly {
                amount := mload(ix)
            }
            ix += 0x20;

            __staking_balance[addr] = amount;
            __staking_lockedBalance[addr] = amount;
        }
    }

    /**
     * @dev Deposit from a sender.
     */
     modifier nonReentrant() {
         _guardCounter += 1;
         uint256 localCounter = _guardCounter;
         _;
         require(localCounter == _guardCounter);
     }
    //@CTK NO_BUF_OVERFLOW
    /*@CTK __stakingdeposit
      @tag assume_completion
      @pre msg.value > 0
      @post __post.__staking_balance[msg.sender] == __staking_balance[msg.sender] +
          msg.value
    */
    function __stakingdeposit() external nonReentrant /* notRevoked */ payable {
        require(msg.value > 0, "Deposit amount should be greater than zero");

        __staking_balance[msg.sender] = __staking_balance[msg.sender].add(msg.value);

        emit Staked(msg.sender, msg.value, __staking_balance[msg.sender],
            __stakingavailableBalanceOf(msg.sender));
    }

    /**
     * @dev Withdraw for a sender.
     * @param amount The amount of funds will be withdrawn and transferred to.
     */
    //@CTK NO_BUF_OVERFLOW
    /*@CTK __stakingwithdraw
      @tag assume_completion
      @pre amount > 0
      @pre amount <= __staking_balance[msg.sender] - __staking_lockedBalance[msg.
          sender]
      @post __post.__staking_balance[msg.sender] == __staking_balance[msg.sender] -
          amount
    */
    function __stakingwithdraw(uint256 amount) external nonReentrant /*notRevoked*/ {
        require(amount > 0, "Amount should be bigger than zero");
        require(amount <= __stakingavailableBalanceOf(msg.sender), "Withdraw amount
            should be equal or less than balance");

        __staking_balance[msg.sender] = __staking_balance[msg.sender].sub(amount);
        msg.sender.transfer(amount);

```

```
997         emit Unstaked(msg.sender, amount, __staking_balance[msg.sender],
                __stakingavailableBalanceOf(msg.sender));
998     }
999
1000
1001    /*@CTK __stakinglock
1002      @post __post.__staking_totalLockedBalance - __post.__staking_lockedBalance[payee
              ] == __staking_totalLockedBalance - __staking_lockedBalance[payee]
1003     */
1004    //@CTK NO_OVERFLOW
1005    //@CTK NO_BUF_OVERFLOW
1006    function __stakinglock(address payee, uint256 lockAmount) private {
1007        if (lockAmount == 0) return;
1008        require(__stakingavailableBalanceOf(payee) >= lockAmount, "Insufficient balance
                 that can be locked");
1009
1010        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].add(lockAmount)
                ;
1011        __staking_totalLockedBalance = __staking_totalLockedBalance.add(lockAmount);
1012
1013        emit Locked(payee, lockAmount, _balance[payee], __stakingavailableBalanceOf(
                payee));
1014    }
1015
1016    /**
1017     * @dev Transfer locked funds to governance
1018     * @param from The address whose funds will be transfered.
1019     * @param amount The amount of funds will be transfered.
1020     */
1021
1022    //@CTK NO_BUF_OVERFLOW
1023    /*@CTK __stakingtransferLocked
1024      @tag assume_completion
1025      @pre from != 0x0
1026      @post __post.__staking_balance[from] == __staking_balance[from] - amount
1027      @post __post.__staking_balance[0x0] == __staking_balance[0x0] + amount
1028    */
1029    function __stakingtransferLocked(address from, uint256 amount) private {
1030        __staking_balance[from] = __staking_balance[from].sub(amount);
1031        address rewardPool = address(0x0);//getRewardPoolAddress();
1032        __staking_balance[rewardPool] = __staking_balance[rewardPool].add(amount);
1033
1034        emit TransferLocked(from, amount, _balance[from], __stakingavailableBalanceOf(
                from));
1035    }
1036
1037    /**
1038     * @dev Unlock fund
1039     * @param payee The address whose funds will be unlocked.
1040     * @param unlockAmount The amount of funds will be unlocked.
1041     */
1042    //@CTK NO_OVERFLOW
1043    //@CTK NO_BUF_OVERFLOW
1044    function __stakingunlock(address payee, uint256 unlockAmount) private {
1045        if (unlockAmount == 0) return;
1046        // require(_lockedBalance[payee] >= unlockAmount, "Unlock amount should be
                equal or less than balance locked");
```

```
1047        __staking_lockedBalance[payee] = __staking_lockedBalance[payee].sub(
                unlockAmount);
1048        __staking_totalLockedBalance = __staking_totalLockedBalance.sub(unlockAmount);
1049
1050        emit Unlocked(payee, unlockAmount, _balance[payee], __stakingavailableBalanceOf
                (payee));
1051    }
1052    //@CTK NO_OVERFLOW
1053    //@CTK NO_BUF_OVERFLOW
1054    function __stakingbalanceOf(address payee) public view returns (uint256) {
1055        return __staking_balance[payee];
1056    }
1057    //@CTK NO_OVERFLOW
1058    //@CTK NO_BUF_OVERFLOW
1059    function __stakinglockedBalanceOf(address payee) public view returns (uint256) {
1060        return __staking_lockedBalance[payee];
1061    }
1062    //@CTK NO_OVERFLOW
1063    //@CTK NO_BUF_OVERFLOW
1064    function __stakingavailableBalanceOf(address payee) public view returns (uint256)
            {
1065        return __staking_balance[payee].sub(__staking_lockedBalance[payee]);
1066    }
1067
1068    /**
1069     * @dev Calculate voting weight which range between 0 and 100.
1070     * @param payee The address whose funds were locked.
1071     */
1072     //@CTK NO_OVERFLOW
1073     //@CTK NO_BUF_OVERFLOW
1074    function calcVotingWeight(address payee) public view returns (uint256) {
1075        return __stakingcalcVotingWeightWithScaleFactor(payee, 1e2);
1076    }
1077
1078    /**
1079     * @dev Calculate voting weight with a scale factor.
1080     * @param payee The address whose funds were locked.
1081     * @param factor The scale factor for weight. For instance:
1082     *              if 1e1, result range is between 0 ~ 10
1083     *              if 1e2, result range is between 0 ~ 100
1084     *              if 1e3, result range is between 0 ~ 1000
1085     */
1086    //@CTK NO_BUF_OVERFLOW
1087    /*@CTK __stakingcalcVotingWeightWithScaleFactor
1088      @tag assume_completion
1089      @post __staking_lockedBalance[payee] == 0 || factor == 0 -> __return == 0
1090      @post __staking_lockedBalance[payee] != 0 && factor != 0 -> __return ==
              __staking_lockedBalance[payee] * factor / __staking_totalLockedBalance
1091
1092    */
1093    function __stakingcalcVotingWeightWithScaleFactor(address payee, uint32 factor)
            public view returns (uint256) {
1094        if (__staking_lockedBalance[payee] == 0 || factor == 0) return 0;
1095        return __staking_lockedBalance[payee].mul(factor).div(
                __staking_totalLockedBalance);
1096    }
1097    //@CTK NO_ASF
1098    //@CTK NO_OVERFLOW
```

```
1099    //@CTK NO_BUF_OVERFLOW
1100    function __stakingisRevoked() public view returns (bool) {
1101        return __stakingrevoked;
1102    }
1103
1104    modifier notRevoked(){
1105        require(!revoked, "Is revoked");
1106        _;
1107    }
1108
1109    /**
1110     * @dev Allows the owner to revoke the staking. Funds already staked are returned
            to the owner
1111     */
1112     /*
1113    function __stakingrevoke() public onlyOwner notRevoked {
1114        address contractOwner = owner();
1115        uint256 balance = address(this).balance;
1116
1117        require(balance > 0);
1118
1119        contractOwner.transfer(balance);
1120        __stakingrevoked = true;
1121
1122        emit Revoked(contractOwner, balance);
1123    } */
1124
1125    /*
1126        BallotStorage contract code begins here
1127    */
1128    struct BallotBasic {
1129        //Ballot ID
1130        uint256 id;
1131        uint256 startTime;
1132        uint256 endTime;
1133        uint256 ballotType;
1134        address creator;
1135        bytes memo;
1136        uint256 totalVoters;
1137        uint256 powerOfAccepts;
1138        uint256 powerOfRejects;
1139        uint256 state;
1140        bool isFinalized;
1141        uint256 duration;
1142
1143    }
1144
1145    //For MemberAdding/MemberRemoval/MemberSwap
1146    struct BallotMember {
1147        uint256 id;
1148        address oldMemberAddress;
1149        address newMemberAddress;
1150        bytes newNodeName; // name
1151        bytes newNodeId; // admin.nodeInfo.id is 512 bit public key
1152        bytes newNodeIp;
1153        uint256 newNodePort;
1154        uint256 lockAmount;
1155    }
```

```
    //For GovernanceChange
    struct BallotAddress {
        uint256 id;
        address newGovernanceAddress;
    }

    //For EnvValChange
    struct BallotVariable {
        //Ballot ID
        uint256 id;
        bytes32 envVariableName;
        uint256 envVariableType;
        bytes envVariableValue;
    }

    struct Vote {
        uint256 voteId;
        uint256 ballotId;
        address voter;
        uint256 decision;
        uint256 power;
        uint256 time;
    }

    event BallotCreated(
        uint256 indexed ballotId,
        uint256 indexed ballotType,
        address indexed creator
    );

    event BallotStarted(
        uint256 indexed ballotId,
        uint256 indexed startTime,
        uint256 indexed endTime
    );

    event Voted(
        uint256 indexed voteid,
        uint256 indexed ballotId,
        address indexed voter,
        uint256 decision
    );

    event BallotFinalized(
        uint256 indexed ballotId,
        uint256 state
    );

    event BallotCanceled (
        uint256 indexed ballotId
    );

    event BallotUpdated (
        uint256 indexed ballotId,
        address indexed updatedBy
    );
```

```
1214    mapping(uint=>BallotBasic) internal __ballotStorageballotBasicMap;
1215    mapping(uint=>BallotMember) internal __ballotStorageballotMemberMap;
1216    mapping(uint=>BallotAddress) internal __ballotStorageballotAddressMap;
1217    mapping(uint=>BallotVariable) internal __ballotStorageballotVariableMap;
1218
1219    mapping(uint=>Vote) internal __ballotStoragevoteMap;
1220    mapping(uint=>mapping(address=>bool)) internal __ballotStoragehasVotedMap;
1221
1222    address internal __ballotStoragepreviousBallotStorage;
1223
1224    uint256 internal __ballotStorageballotCount = 0;
1225
1226    function __ballotStorageconstructor(address _registry) public {
1227   //     __ballotStoragesetRegistry(_registry);
1228    }
1229
1230    modifier onlyValidTime(uint256 _startTime, uint256 _endTime) {
1231        require(_startTime > 0 && _endTime > 0, "start or end is 0");
1232        require(_endTime > _startTime, "start >= end"); // && _startTime > getTime()
1233        //uint256 diffTime = _endTime.sub(_startTime);
1234        // require(diffTime > minBallotDuration());
1235        // require(diffTime <= maxBallotDuration());
1236        _;
1237    }
1238
1239    modifier onlyValidDuration(uint256 _duration){
1240        require(getMinVotingDuration() <= _duration, "Under min value of duration");
1241        require(_duration <= getMaxVotingDuration(), "Over max value of duration");
1242        _;
1243    }
1244
1245    modifier onlyGovOrCreator(uint256 _ballotId) {
1246        require((getGovAddress() == msg.sender) || (ballotBasicMap[_ballotId].creator
            == msg.sender), "No Permission");
1247        _;
1248    }
1249
1250    modifier notDisabled() {
1251        //require(address(this) == getBallotStorageAddress(), "Is Disabled");
1252        _;
1253    }
1254
1255    /*@CTK __ballotStoragegetMinVotingDuration
1256      @post __return == 500000000
1257    */
1258    function __ballotStoragegetMinVotingDuration() public view returns (uint256) {
1259        return 500000000;
1260    }
1261    /*@CTK __ballotStoragegetMaxVotingDuration
1262      @post __return == 800000000
1263    */
1264    function __ballotStoragegetMaxVotingDuration() public view returns (uint256) {
1265        return 800000000;
1266    }
1267
1268    /*@CTK __ballotStoragegetTime
1269      @post __return == now
1270    */
```

```
1271    function __ballotStoragegetTime() public view returns (uint256) {
1272        return now;
1273    }
1274    /*@CTK __ballotStoragegetPreviousBallotStorage
1275      @post __return == __ballotStoragepreviousBallotStorage
1276    */
1277    function __ballotStoragegetPreviousBallotStorage() public view returns (address) {
1278        return __ballotStoragepreviousBallotStorage;
1279    }
1280
1281    function __ballotStorageisDisabled() public view returns (bool) {
1282        return (address(this) != getBallotStorageAddress());
1283    }
1284    /*@CTK __ballotStoragegetBallotCount
1285      @post __return == __ballotStorageballotCount
1286    */
1287    function __ballotStoragegetBallotCount() public view returns (uint256) {
1288        return __ballotStorageballotCount;
1289    }
1290
1291    /*CTK __ballotStoragegetBallotBasic
1292      @post __return == __ballotStorageballotBasicMap[_id].startTime,
          __ballotStorageballotBasicMap[_id].endTime, __ballotStorageballotBasicMap[_id
          ].ballotType, __ballotStorageballotBasicMap[_id].creator,
          __ballotStorageballotBasicMap[_id].memo, __ballotStorageballotBasicMap[_id].
          totalVoters, __ballotStorageballotBasicMap[_id].powerOfAccepts,
          __ballotStorageballotBasicMap[_id].powerOfRejects,
          __ballotStorageballotBasicMap[_id].state, __ballotStorageballotBasicMap[_id].
          isFinalized, __ballotStorageballotBasicMap[_id].duration
1293    */
1294    //@CTK NO_OVERFLOW
1295    //@CTK NO_BUF_OVERFLOW
1296    function __ballotStoragegetBallotBasic(uint256 _id) public view returns (
1297        uint256 startTime,
1298        uint256 endTime,
1299        uint256 ballotType,
1300        address creator,
1301        bytes memo,
1302        uint256 totalVoters,
1303        uint256 powerOfAccepts,
1304        uint256 powerOfRejects,
1305        uint256 state,
1306        bool isFinalized,
1307        uint256 duration
1308    )
1309    {
1310        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1311        startTime = tBallot.startTime;
1312        endTime = tBallot.endTime;
1313        ballotType = tBallot.ballotType;
1314        creator = tBallot.creator;
1315        memo = tBallot.memo;
1316        totalVoters = tBallot.totalVoters;
1317        powerOfAccepts = tBallot.powerOfAccepts;
1318        powerOfRejects = tBallot.powerOfRejects;
1319        state = tBallot.state;
1320        isFinalized = tBallot.isFinalized;
1321        duration = tBallot.duration;
```

```
1322    }
1323    /*@CTK __ballotStoragegetBallotMember
1324      @post __return == __ballotStorageballotMemberMap[_id].oldMemberAddress,
              __ballotStorageballotMemberMap[_id].newMemberAddress,
              __ballotStorageballotMemberMap[_id].newNodeName,
              __ballotStorageballotMemberMap[_id].newNodeId, __ballotStorageballotMemberMap
              [_id].newNodeIp, __ballotStorageballotMemberMap[_id].newNodePort,
              __ballotStorageballotMemberMap[_id].lockAmount;
1325    */
1326    //@CTK NO_OVERFLOW
1327    //@CTK NO_BUF_OVERFLOW
1328    function __ballotStoragegetBallotMember(uint256 _id) public view returns (
1329        address oldMemberAddress,
1330        address newMemberAddress,
1331        bytes newNodeName, // name
1332        bytes newNodeId, // admin.nodeInfo.id is 512 bit public key
1333        bytes newNodeIp,
1334        uint256 newNodePort,
1335        uint256 lockAmount
1336    )
1337    {
1338        //BallotMember storage tBallot = __ballotStorageballotMemberMap[_id];
1339        oldMemberAddress = __ballotStorageballotMemberMap[_id].oldMemberAddress;
1340        newMemberAddress = __ballotStorageballotMemberMap[_id].newMemberAddress;
1341        newNodeName = __ballotStorageballotMemberMap[_id].newNodeName;
1342        newNodeId = __ballotStorageballotMemberMap[_id].newNodeId;
1343        newNodeIp = __ballotStorageballotMemberMap[_id].newNodeIp;
1344        newNodePort = __ballotStorageballotMemberMap[_id].newNodePort;
1345        lockAmount = __ballotStorageballotMemberMap[_id].lockAmount;
1346    }
1347    //@CTK NO_OVERFLOW
1348    //@CTK NO_BUF_OVERFLOW
1349    /*@CTK __ballotStoragegetBallotAddress
1350      @post __return == __ballotStorageballotAddressMap[_id].newGovernanceAddress
1351    */
1352    function __ballotStoragegetBallotAddress(uint256 _id) public view returns (
1353        address newGovernanceAddress
1354    )
1355    {
1356        //BallotAddress storage tBallot = __ballotStorageballotAddressMap[_id];
1357        newGovernanceAddress = __ballotStorageballotAddressMap[_id].
              newGovernanceAddress;
1358    }
1359    /*@CTK __ballotStoragegetBallotVariable
1360      @post __return == __ballotStorageballotVariableMap[_id].envVariableName,
              __ballotStorageballotVariableMap[_id].envVariableType,
              __ballotStorageballotVariableMap[_id].envVariableValue;
1361    */
1362    //@CTK NO_BUF_OVERFLOW
1363    //@CTK NO_OVERFLOW
1364    function __ballotStoragegetBallotVariable(uint256 _id) public view returns (
1365        bytes32 envVariableName,
1366        uint256 envVariableType,
1367        bytes envVariableValue
1368    )
1369    {
1370        //BallotVariable storage tBallot = __ballotStorageballotVariableMap[_id];
1371        envVariableName = __ballotStorageballotVariableMap[_id].envVariableName;
```

```
1372        envVariableType = __ballotStorageballotVariableMap[_id].envVariableType;
1373        envVariableValue = __ballotStorageballotVariableMap[_id].envVariableValue;
1374    }
1375
1376 /*
1377    function __ballotStoragesetPreviousBallotStorage(address _address) public
           onlyOwner {
1378        require(_address != address(0), "Invalid address");
1379        __ballotStoragepreviousBallotStorage = _address;
1380    }*/
1381
1382    //For MemberAdding/MemberRemoval/MemberSwap
1383    //@CTK NO_BUF_OVERFLOW
1384    function __ballotStoragecreateBallotForMember(
1385        uint256 _id,
1386        uint256 _ballotType,
1387        address _creator,
1388        address _oldMemberAddress,
1389        address _newMemberAddress,
1390        bytes _newNodeName, // name
1391        bytes _newNodeId, // admin.nodeInfo.id is 512 bit public key
1392        bytes _newNodeIp,
1393        uint _newNodePort
1394    )
1395        private
1396        notDisabled
1397    {
1398        require(
1399            __ballotStorage_areMemberBallotParamValid(
1400                _ballotType,
1401                _oldMemberAddress,
1402                _newMemberAddress,
1403                _newNodeName,
1404                _newNodeId,
1405                _newNodeIp,
1406                _newNodePort
1407            ),
1408            "Invalid Parameter"
1409        );
1410        __ballotStorage_createBallot(_id, _ballotType, _creator);
1411        BallotMember memory newBallot;
1412        newBallot.id = _id;
1413        newBallot.oldMemberAddress = _oldMemberAddress;
1414        newBallot.newMemberAddress = _newMemberAddress;
1415        newBallot.newNodeName = _newNodeName;
1416        newBallot.newNodeId = _newNodeId;
1417        newBallot.newNodeIp = _newNodeIp;
1418        newBallot.newNodePort = _newNodePort;
1419        __ballotStorageballotMemberMap[_id] = newBallot;
1420    }
1421
1422
1423
1424    //@CTK NO_BUF_OVERFLOW
1425    function __ballotStoragecreateBallotForAddress(
1426        uint256 _id,
1427        uint256 _ballotType,
1428        address _creator,
```

```
1429        address _newGovernanceAddress
1430    )
1431        private
1432        notDisabled
1433    {
1434        require(_ballotType == uint256(BallotTypes.GovernanceChange), "Invalid Ballot
                Type");
1435        require(_newGovernanceAddress != address(0), "Invalid Parameter");
1436
1437        __ballotStorage_createBallot(_id, _ballotType, _creator);
1438        BallotAddress memory newBallot;
1439        newBallot.id = _id;
1440        newBallot.newGovernanceAddress = _newGovernanceAddress;
1441        __ballotStorageballotAddressMap[_id] = newBallot;
1442    }
1443
1444
1445    //@CTK NO_BUF_OVERFLOW
1446    function __ballotStoragecreateBallotForVariable(
1447        uint256 _id,
1448        uint256 _ballotType,
1449        address _creator,
1450        bytes32 _envVariableName,
1451        uint256 _envVariableType,
1452        bytes _envVariableValue
1453    )
1454        private
1455    //    notDisabled
1456        returns (uint256)
1457    {
1458
1459        require(
1460            __ballotStorage_areVariableBallotParamValid(_ballotType, _envVariableName,
                    _envVariableType, _envVariableValue),
1461            "Invalid Parameter"
1462        );
1463        __ballotStorage_createBallot(_id, _ballotType, _creator);
1464        BallotVariable memory newBallot;
1465        newBallot.id = _id;
1466        newBallot.envVariableName = _envVariableName;
1467        newBallot.envVariableType = _envVariableType;
1468        newBallot.envVariableValue = _envVariableValue;
1469        __ballotStorageballotVariableMap[_id] = newBallot;
1470    }
1471    //@CTK NO_BUF_OVERFLOW
1472    function __ballotStoragecreateVote(
1473        uint256 _voteId,
1474        uint256 _ballotId,
1475        address _voter,
1476        uint256 _decision,
1477        uint256 _power
1478    )
1479        private
1480       // notDisabled
1481    {
1482        // Check decision type
1483        /*
1484        require((_decision == uint256(DecisionTypes.Accept))
```

```
1485            || (_decision == uint256(DecisionTypes.Reject)), "Invalid decision");*/
1486        // Check if ballot exists
1487        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1488        // Check if vote exists
1489        require(__ballotStoragevoteMap[_voteId].voteId != _voteId, "already existed
                voteId");
1490        // Check if voted
1491        require(!__ballotStoragehasVotedMap[_ballotId][_voter], "already voted");
1492      /*  require(ballotBasicMap[_ballotId].state
1493            == uint256(BallotStates.InProgress), "Not InProgress State");
1494  */
1495        __ballotStoragevoteMap[_voteId] = Vote(_voteId, _ballotId, _voter, _decision,
                _power, __ballotStoragegetTime());
1496        __ballotStorage_updateBallotForVote(_ballotId, _voter, _decision, _power);
1497
1498        emit Voted(_voteId, _ballotId, _voter, _decision);
1499    }
1500
1501    /*@CTK __ballotStoragestartBallot
1502      @pre _startTime > 0 && _endTime > 0
1503      @pre _endTime > _startTime
1504      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1505      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1506      @post __post.__ballotStorageballotBasicMap[_ballotId].startTime == _startTime
1507      @post __post.__ballotStorageballotBasicMap[_ballotId].endTime == _endTime
1508      @post __post.__ballotStorageballotBasicMap[_ballotId].state == 2
1509    */
1510    function __ballotStoragestartBallot(
1511        uint256 _ballotId,
1512        uint256 _startTime,
1513        uint256 _endTime
1514    )
1515        private
1516      //  notDisabled
1517      //  onlyValidTime(_startTime, _endTime)
1518    {
1519        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1520        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1521      //  require(__ballotStorageballotBasicMap[_ballotId].state == uint256(
                BallotStates.Ready), "Not Ready State");
1522
1523     //   BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1524        __ballotStorageballotBasicMap[_ballotId].startTime = _startTime;
1525        __ballotStorageballotBasicMap[_ballotId].endTime = _endTime;
1526        __ballotStorageballotBasicMap[_ballotId].state = uint256(BallotStates.
                InProgress);
1527        emit BallotStarted(_ballotId, _startTime, _endTime);
1528    }
1529
1530
1531
1532    //@CTK NO_BUF_OVERFLOW
1533    /*@CTK __ballotStorageupdateBallotMemo
1534      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1535      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
```

```
1536      @post __post.__ballotStorageballotBasicMap[_ballotId].memo == _memo
1537    */
1538    function __ballotStorageupdateBallotMemo(
1539        uint256 _ballotId,
1540        bytes _memo
1541    )
1542        private
1543        notDisabled
1544    {
1545        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1546        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1547        //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1548        __ballotStorageballotBasicMap[_ballotId].memo = _memo;
1549        emit BallotUpdated (_ballotId, msg.sender);
1550    }
1551
1552
1553    //@CTK NO_BUF_OVERFLOW
1554    /*@CTK __ballotStorageupdateBallotDuration
1555      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1556      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1557      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1558      @post __post.__ballotStorageballotBasicMap[_ballotId].duration == _duration
1559    */
1560    function __ballotStorageupdateBallotDuration(
1561        uint256 _ballotId,
1562        uint256 _duration
1563    )
1564        private
1565      // notDisabled
1566      // onlyValidDuration(_duration)
1567    {
1568        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1569        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1570        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1571
1572        //BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1573        __ballotStorageballotBasicMap[_ballotId].duration = _duration;
1574        emit BallotUpdated (_ballotId, msg.sender);
1575    }
1576
1577
1578    //@CTK NO_BUF_OVERFLOW
1579    /*@CTK __ballotStorageupdateBallotMemberLockAmount
1580      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1581      @pre __ballotStorageballotMemberMap[_ballotId].id == _ballotId
1582      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1583      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1584
1585      @post __post.__ballotStorageballotMemberMap[_ballotId].lockAmount == _lockAmount
1586    */
1587    function __ballotStorageupdateBallotMemberLockAmount(
1588        uint256 _ballotId,
```

```
1589        uint256 _lockAmount
1590    )
1591        private
1592        notDisabled
1593    {
1594        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1595        require(__ballotStorageballotMemberMap[_ballotId].id == _ballotId, "not existed
                 BallotMember");
1596        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1597        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1598       // BallotMember storage _ballot = __ballotStorageballotMemberMap[_ballotId];
1599        __ballotStorageballotMemberMap[_ballotId].lockAmount = _lockAmount;
1600        emit BallotUpdated (_ballotId, msg.sender);
1601    }
1602
1603    // cancel ballot info
1604
1605    //@CTK NO_BUF_OVERFLOW
1606    /*@CTK __ballotStorageupdateBallotDuration
1607      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1608      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1609      @pre __ballotStorageballotBasicMap[_ballotId].state == 1
1610      @post __post.__ballotStorageballotBasicMap[_ballotId].state == 5
1611    */
1612    function __ballotStoragecancelBallot(uint256 _ballotId) private notDisabled {
1613        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1614        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
1615
1616        require(__ballotStorageballotBasicMap[_ballotId].state == uint256(BallotStates.
                Ready), "Not Ready State");
1617       // BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1618        __ballotStorageballotBasicMap[_ballotId].state = uint256(BallotStates.Canceled)
                ;
1619        emit BallotCanceled (_ballotId);
1620    }
1621
1622    // finalize ballot info
1623
1624    /*@CTK "__ballotStoragefinalizeBallot"
1625      @pre __ballotStorageballotBasicMap[_ballotId].id == _ballotId
1626      @pre __ballotStorageballotBasicMap[_ballotId].isFinalized == false
1627      @pre (_ballotState == uint256(BallotStates.Accepted)) || (_ballotState ==
             uint256(BallotStates.Rejected)) || (_ballotState == uint256(BallotStates.
             NotApplicable))
1628      @post __post.__ballotStorageballotBasicMap[_ballotId].state == _ballotState
1629      @post __post.__ballotStorageballotBasicMap[_ballotId].isFinalized
1630    */
1631    function __ballotStoragefinalizeBallot(uint256 _ballotId, uint256 _ballotState)
            private notDisabled {
1632        require(__ballotStorageballotBasicMap[_ballotId].id == _ballotId, "not existed
                Ballot");
1633        require(__ballotStorageballotBasicMap[_ballotId].isFinalized == false, "already
                 finalized");
```

```
1634        require((_ballotState == uint256(BallotStates.Accepted))
1635            || (_ballotState == uint256(BallotStates.Rejected))
1636            || (_ballotState == uint256(BallotStates.NotApplicable)), "Invalid Ballot
                    State");
1637
1638        // BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
1639        __ballotStorageballotBasicMap[_ballotId].state = _ballotState;
1640        __ballotStorageballotBasicMap[_ballotId].isFinalized = true;
1641        emit BallotFinalized (_ballotId, _ballotState);
1642    }
1643
1644    function __ballotStoragehasAlreadyVoted(uint56 _ballotId, address _voter) public
            view returns (bool) {
1645        return __ballotStoragehasVotedMap[_ballotId][_voter];
1646    }
1647
1648    //@CTK NO_OVERFLOW
1649    //@CTK NO_BUF_OVERFLOW
1650    function __ballotStoragegetVote(uint256 _voteId) public view returns (
1651        uint256 voteId,
1652        uint256 ballotId,
1653        address voter,
1654        uint256 decision,
1655        uint256 power,
1656        uint256 time
1657    )
1658    {
1659        require(__ballotStoragevoteMap[_voteId].voteId == _voteId, "not existed voteId"
                );
1660        Vote memory _vote = __ballotStoragevoteMap[_voteId];
1661        voteId = _vote.voteId;
1662        ballotId = _vote.ballotId;
1663        voter = _vote.voter;
1664        decision = _vote.decision;
1665        power = _vote.power;
1666        time = _vote.time;
1667    }
1668
1669    //@CTK NO_OVERFLOW
1670    //@CTK NO_BUF_OVERFLOW
1671    function __ballotStoragegetBallotPeriod(uint256 _id) public view returns (
1672        uint256 startTime,
1673        uint256 endTime,
1674        uint256 duration
1675    )
1676    {
1677        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
1678        startTime = tBallot.startTime;
1679        endTime = tBallot.endTime;
1680        duration = tBallot.duration;
1681    }
1682
1683    //@CTK NO_OVERFLOW
1684    //@CTK NO_BUF_OVERFLOW
1685    function __ballotStoragegetBallotVotingInfo(uint256 _id) public view returns (
1686        uint256 totalVoters,
1687        uint256 powerOfAccepts,
1688        uint256 powerOfRejects
```

```
    )
    {
        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
        totalVoters = tBallot.totalVoters;
        powerOfAccepts = tBallot.powerOfAccepts;
        powerOfRejects = tBallot.powerOfRejects;
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function __ballotStoragegetBallotState(uint256 _id) public view returns (
        uint256 ballotType,
        uint256 state,
        bool isFinalized
    )
    {
        BallotBasic memory tBallot = __ballotStorageballotBasicMap[_id];
        ballotType = tBallot.ballotType;
        state = tBallot.state;
        isFinalized = tBallot.isFinalized;
    }


    //@CTK NO_BUF_OVERFLOW
    function __ballotStorage_createBallot(
        uint256 _id,
        uint256 _ballotType,
        address _creator
    )
        internal
    {

        require(__ballotStorageballotBasicMap[_id].id != _id, "Already existed ballot")
            ;

        BallotBasic memory newBallot;

        newBallot.id = _id;
        newBallot.ballotType = _ballotType;
        newBallot.creator = _creator;
       // newBallot.memo = _memo;
        newBallot.state = uint256(BallotStates.Ready);
        newBallot.isFinalized = false;
       //   newBallot.duration = _duration;
         __ballotStorageballotBasicMap[_id] = newBallot;
        __ballotStorageballotCount = __ballotStorageballotCount.add(1);
        emit BallotCreated(_id, _ballotType, _creator);
    }


    //@CTK NO_BUF_OVERFLOW
    function __ballotStorage_areMemberBallotParamValid(
        uint256 _ballotType,
        address _oldMemberAddress,
        address _newMemberAddress,
        bytes _newName,
        bytes _newNodeId, // admin.nodeInfo.id is 512 bit public key
```

```
        bytes _newNodeIp,
        uint _newNodePort
    )
        internal
        pure
        returns (bool)
    {
        require((_ballotType >= uint256(BallotTypes.MemberAdd))
            && (_ballotType <= uint256(BallotTypes.MemberChange)), "Invalid Ballot Type
                ");

        if (_ballotType == uint256(BallotTypes.MemberRemoval)){
            require(_oldMemberAddress != address(0), "Invalid old member address");
            require(_newMemberAddress == address(0), "Invalid new member address");
            require(_newName.length == 0, "Invalid new node name");
            require(_newNodeId.length == 0, "Invalid new node id");
            require(_newNodeIp.length == 0, "Invalid new node IP");
            require(_newNodePort == 0, "Invalid new node Port");
        }else {
            require(_newName.length > 0, "Invalid new node name");
            require(_newNodeId.length == 64, "Invalid new node id");
            require(_newNodeIp.length > 0, "Invalid new node IP");
            require(_newNodePort > 0, "Invalid new node Port");
            if (_ballotType == uint256(BallotTypes.MemberAdd)) {
                require(_oldMemberAddress == address(0), "Invalid old member address");
                require(_newMemberAddress != address(0), "Invalid new member address");
            } else if (_ballotType == uint256(BallotTypes.MemberChange)) {
                require(_oldMemberAddress != address(0), "Invalid old member address");
                require(_newMemberAddress != address(0), "Invalid new member address");
            }
        }

        return true;
    }


    //@CTK NO_BUF_OVERFLOW
    function __ballotStorage_areVariableBallotParamValid(
        uint256 _ballotType,
        bytes32 _envVariableName,
        uint256 _envVariableType,
        bytes _envVariableValue
    )
        internal
        pure
        returns (bool)
    {
          require(_ballotType == uint256(BallotTypes.EnvValChange), "Invalid Ballot
              Type");
    //    require(_envVariableName > 0, "Invalid environment variable name");
          require(_envVariableType >= uint256(VariableTypes.Int), "Invalid environment
              variable Type");
          require(_envVariableType <= uint256(VariableTypes.String), "Invalid
              environment variable Type");
          require(_envVariableValue.length > 0, "Invalid environment variable value");

        return true;
    }
```

```
    // update ballot

    //@CTK NO_BUF_OVERFLOW
    /*@CTK __ballotStorage_updateBallotForVote
      @tag assume_completion
      @post __post.__ballotStoragehasVotedMap[_ballotId][_voter] == true
      @post __post.__ballotStorageballotBasicMap[_ballotId].totalVoters ==
          __ballotStorageballotBasicMap[_ballotId].totalVoters + 1
      @post _decision == 1 -> __post.__ballotStorageballotBasicMap[_ballotId].
          powerOfAccepts == __ballotStorageballotBasicMap[_ballotId].powerOfAccepts +
          _power
      @post _decision != 1 -> __post.__ballotStorageballotBasicMap[_ballotId].
          powerOfRejects == __ballotStorageballotBasicMap[_ballotId].powerOfRejects +
          _power
    */
    function __ballotStorage_updateBallotForVote(
        uint256 _ballotId,
        address _voter,
        uint256 _decision,
        uint256 _power
    )
        internal
    {

        //1.get ballotBasic
        BallotBasic storage _ballot = __ballotStorageballotBasicMap[_ballotId];
        //2.
        __ballotStoragehasVotedMap[_ballotId][_voter] = true;
        //3. update totalVoters
        __ballotStorageballotBasicMap[_ballotId].totalVoters =
            __ballotStorageballotBasicMap[_ballotId].totalVoters.add(1);
        //4. Update power of accept/reject
        if (_decision == uint256(DecisionTypes.Accept)){
            __ballotStorageballotBasicMap[_ballotId].powerOfAccepts =
                __ballotStorageballotBasicMap[_ballotId].powerOfAccepts.add(_power);
        }
        else {
            __ballotStorageballotBasicMap[_ballotId].powerOfRejects =
                __ballotStorageballotBasicMap[_ballotId].powerOfRejects.add(_power);
        }
    }
}
```

File enum.sol

```
pragma solidity ^0.4.18;

contract TestEnum {
  enum Status {
    Pending,
    Success,
    Failure
  }

  uint public index;
  mapping (uint256 => Status) public status_map;

  //@CTK NO_ASF
```

```
14    //@CTK NO_OVERFLOW
15    //@CTK NO_BUF_OVERFLOW
16    function test_enum() {
17      assert(Status.Pending == Status.Pending);
18      assert(Status.Pending != Status.Success);
19
20      Status x = Status.Pending;
21      Status y = Status.Success;
22      assert(x != y);
23
24      Status z = Status.Success;
25      assert(y == z);
26    }
27
28    /*@CTK test_enum
29      @post __return == (input == Status.Success)
30      @tag no_overflow
31      */
32    function check_success(Status input) returns (bool) {
33      return input == Status.Success;
34    }
35  }
```

CERTIK

Building Fully Trustworthy Smart Contracts and Blockchain Ecosystems