CertiK Verification Report For Gate.io

Request Date: 2019-04-11
Revision Date: 2019-04-15

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 Gate.io(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.

Summary

This audit report summarises the smart contract verification service requested by Gate.io. The goal of this security audit is to guarantee that the audited smart contracts are robust enough to avoid any potential security loopholes.
The result of this report is only a reflection of the source code that was determined in this scope, and of the source code at the time of the audit.

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	2	SWC-116

Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120
"tx.origin" for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

Modifiers can be simplified using member functions.

For example, `onlySender` can be implemented as

```
if (!isInArray(msg.sender, senders)) {
    revert();
};
```

`isSigner` and `isSender` can then be simplified or removed from both contracts.

`getNextSequenceId` can be enforced in `verifySignature`

So that `sequenceId` can be expected as the result of `getNextSequenceId`. Then

```
if (sequenceId > (recentSequenceIds[minIndex] + 10000)) {
    revert();
}
```

can use `MAX_SEQUENCE_ID_SIZE` to replace the magic number 10000. However, this is totally optional depending on the usage of sequenceId.

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

- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

Source Code with CertiK Labels

File 23contract.sol

```
pragma solidity ^0.5.4;

contract ERC20Interface {
  function transfer(address _to, uint256 _value) public returns (bool success);
  function balanceOf(address _owner) public view returns (uint256 balance);
}

contract ethMultiSig {
  event Sent(address msgSender, address toAddress, address otherSigner, bytes data,
      uint value, bytes32 hash);
  event Deposited(uint value, address from, bytes data);
  event SetSafeModeActivated(address msgSender);

  address[] public signers;
  bool public safeMode = false;

  uint constant MAX_SEQUENCE_ID_SIZE = 10;
  uint[10] recentSequenceIds;

  /*@CTK ethMultiSig
    @tag assume_completion
    @post allowedSigners.length == 3
    @post __post.signers == allowedSigners
   */
  constructor(address[] memory allowedSigners) public {
    if (allowedSigners.length != 3) {
      assert(false);
      // revert();
    }
    signers = allowedSigners;
  }

  function() external payable {
    if (msg.value > 0) {
      emit Deposited(msg.value, msg.sender, msg.data);
    }
  }

  /*@CTK isSigner
    @post (exists i: uint. (i < signers.length /\ signers[i] == signer)) == __return
   */
  function isSigner(address signer) public view returns (bool) {
    /*@CTK isSigner_forloop
      @inv this == this__pre
      @inv signer == signer__pre
      @inv i <= signers.length
      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
      @inv !__return__pre
      @inv __should_return == __return
      @post __should_return -> (i < signers.length /\ signers[i] == signer)
      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
      @post !__should_return -> (__return == __return__pre)
```

```
    @post !__should_return -> i == signers.length
   */
  for (uint i = 0; i < signers.length; i++) {
    if (signers[i] == signer) {
      return true;
    }
  }
  return false;
}

modifier onlySigner {
  if (!isSigner(msg.sender)) {
    assert(false);
    // revert();
  }
  _;
}

function sendEth(uint value, uint expireTime, bytes memory data, uint sequenceId,
    address payable toAddress, bytes memory signature) public payable onlySigner {
  bytes32 hash = keccak256(abi.encodePacked("ETHER", toAddress, value, data,
      expireTime, sequenceId));
  address otherSigner = verifySignature(toAddress, hash, signature, expireTime,
      sequenceId);
  if( !toAddress.send(value)){
      revert();
  }
  emit Sent(msg.sender, toAddress, otherSigner, data, value, hash);
}

function sendToken(uint value, uint expireTime, uint sequenceId, address toAddress,
    address tokenContractAddress, bytes memory signature) public onlySigner {
  bytes32 hash = keccak256(abi.encodePacked("ERC20", toAddress, value,
      tokenContractAddress, expireTime, sequenceId));
  verifySignature(toAddress, hash, signature, expireTime, sequenceId);
  ERC20Interface instance = ERC20Interface(tokenContractAddress);
  if (!instance.transfer(toAddress, value)) {
      revert();
  }
}

function verifySignature(
    address toAddress,
    bytes32 hash,
    bytes memory signature,
    uint expireTime,
    uint sequenceId
) private returns (address) {
  address otherSigner = recoverFromSignature(hash, signature);
  if (safeMode && !isSigner(toAddress)) {
    revert();
  }
  if (expireTime < block.timestamp) {
    revert();
  }
  insertSequenceId(sequenceId);
  if (!isSigner(otherSigner)) {
    revert();
```

```
    }
    if (otherSigner == msg.sender) {
      revert();
    }
    return otherSigner;
  }

  /*@CTK setSafeMode
    @tag assume_completion
    @post __post.safeMode == true
   */
  function setSafeMode() public onlySigner {
    safeMode = true;
    emit SetSafeModeActivated(msg.sender);
  }

  function recoverFromSignature(bytes32 hash, bytes memory signature) private pure
      returns (address) {
    if (signature.length != 65) {
      revert();
    }
    bytes32 r;
    bytes32 s;
    uint8 v;
    assembly {
      r := mload(add(signature, 32))
      s := mload(add(signature, 64))
      v := and(mload(add(signature, 65)), 255)
    }
    if (v < 27) {
      v += 27;
    }
    return ecrecover(hash, v, r, s);
  }

  function insertSequenceId(uint sequenceId) private onlySigner {
    uint minIndex = 0;
    /*CTK insertSequenceId_forloop
      @pre MAX_SEQUENCE_ID_SIZE == 10
      @pre minIndex == 0
      @inv this == this__pre
      @inv i <= MAX_SEQUENCE_ID_SIZE
      @inv recentSequenceIds[i] == sequenceId -> __has_assertion_failure
     */
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
      if (recentSequenceIds[i] == sequenceId) {
        revert();
      }
      if (recentSequenceIds[i] < recentSequenceIds[minIndex]) {
        minIndex = i;
      }
    }
    if (sequenceId < recentSequenceIds[minIndex]) {
      revert();
    }
    if (sequenceId > (recentSequenceIds[minIndex] + 10000)) {
      revert();
    }
```

```
    recentSequenceIds[minIndex] = sequenceId;
  }

  function getNextSequenceId() public view returns (uint) {
    uint maxSequenceId = 0;
    /*@CTK getNextSequenceId_forloop
      @pre MAX_SEQUENCE_ID_SIZE == 10
      @pre maxSequenceId == 0
      @inv this == this__pre
      @inv i <= MAX_SEQUENCE_ID_SIZE
      @inv !__should_return
      @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
      @post i == MAX_SEQUENCE_ID_SIZE
      @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >=
          recentSequenceIds[i]
     */
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
      if (recentSequenceIds[i] > maxSequenceId) {
        maxSequenceId = recentSequenceIds[i];
      }
    }
    return maxSequenceId + 1;
  }
}
```

File mncontract.sol

```
pragma solidity ^0.5.4;

contract ERC20Interface {
  function transfer(address _to, uint256 _value) public returns (bool success);
  function balanceOf(address _owner) public view returns (uint256 balance);
}

contract ethMultiSig {
  event Sent(address msgSender, address toAddress, address[] otherSigners, bytes data,
      uint value, bytes32 hash);
  event Deposited(uint value, address from, bytes data);
  event SetSafeModeActivated(address msgSender);

  address[] public signers;
  address[] public senders;
  uint countSigners;
  uint minNeedSigners;
  bool public safeMode = false;

  uint constant MAX_SEQUENCE_ID_SIZE = 10;
  uint[10] recentSequenceIds;

  constructor(address[] memory allowedSigners,address[] memory extraSenders,uint m,
      uint n) public {
    if (allowedSigners.length != n) {
      revert();
    }
    signers = allowedSigners;
    for (uint i = 0; i < signers.length; i++) {
        senders.push(signers[i]);
    }
    for (uint i = 0; i < extraSenders.length; i++) {
```

```
      senders.push(extraSenders[i]);
    }
    countSigners = n;
    minNeedSigners = m;
  }

  function() external payable {
    if (msg.value > 0) {
      emit Deposited(msg.value, msg.sender, msg.data);
    }
  }

  /*@CTK isSigner
    @post (exists i: uint. (i < signers.length /\ signers[i] == signer)) == __return
   */
  function isSigner(address signer) public view returns (bool) {
    /*@CTK isSigner_forloop
      @inv this == this__pre
      @inv signer == signer__pre
      @inv i <= signers.length
      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
      @inv !__return__pre
      @inv __should_return == __return
      @post __should_return -> (i < signers.length /\ signers[i] == signer)
      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
      @post !__should_return -> (__return == __return__pre)
      @post !__should_return -> i == signers.length
     */
    for (uint i = 0; i < signers.length; i++) {
      if (signers[i] == signer) {
        return true;
      }
    }
    return false;
  }

  /*@CTK isSender
    @post (exists i: uint. (i < senders.length /\ senders[i] == sender)) == __return
   */
  function isSender(address sender) public view returns (bool) {
    /*@CTK isSender_forloop
      @inv this == this__pre
      @inv sender == sender__pre
      @inv i <= senders.length
      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
      @inv !__return__pre
      @inv __should_return == __return
      @post __should_return -> (i < senders.length /\ senders[i] == sender)
      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
      @post !__should_return -> (__return == __return__pre)
      @post !__should_return -> i == senders.length
     */
    for (uint i = 0; i < senders.length; i++) {
      if (senders[i] == sender) {
```

```
      return true;
    }
  }
  return false;
}

 modifier onlySender {
  if (!isSender(msg.sender)) {
    assert(false);
    // revert();
  }
  _;
}

function sendEth(uint value, uint expireTime, bytes memory data, uint sequenceId, address payable toAddress, bytes memory signatures) public payable onlySender {
  bytes32 hash = keccak256(abi.encodePacked("ETHER", toAddress, value, data, expireTime, sequenceId));
  address [] memory otherSigners = verifySignature(toAddress, hash, signatures, expireTime, sequenceId);
  if( !toAddress.send(value)){
      revert();
  }
  emit Sent(msg.sender, toAddress, otherSigners, data, value, hash);
}

function sendToken(uint value, uint expireTime, uint sequenceId, address toAddress, address tokenContractAddress, bytes memory signatures) public onlySender {
  bytes32 hash = keccak256(abi.encodePacked("ERC20", toAddress, value, tokenContractAddress, expireTime, sequenceId));
  verifySignature(toAddress, hash, signatures, expireTime, sequenceId);
  ERC20Interface instance = ERC20Interface(tokenContractAddress);
  if (!instance.transfer(toAddress, value)) {
      revert();
  }
}

function verifySignature(address toAddress, bytes32 hash, bytes memory signatures, uint expireTime, uint sequenceId) private returns (address [] memory) {
  if (safeMode && !isSigner(toAddress)) {
   revert();
  }
  if (expireTime < block.timestamp) {
   revert();
  }
  insertSequenceId(sequenceId);
  uint signatureLength = 65;
  if( signatures.length != minNeedSigners * signatureLength ){
      revert();
  }
  address [] memory otherSigners = new address[](minNeedSigners);
  for (uint i = 0; i < minNeedSigners ; i++) {
    bytes memory curSignatures = new bytes(signatureLength);
    for( uint j = 0 ; j < 65 ; j++){
        curSignatures[j] = signatures[ i*signatureLength + j];
    }
    address otherSigner = recoverFromSignature(hash, curSignatures);
    if (!isSigner(otherSigner)) {
```

```
      revert();
    }
    if( isInArray(otherSigner, otherSigners)){
        revert();
    }
     otherSigners[i] = otherSigner;
  }
  return otherSigners;
}

/*@CTK setSafeMode
  @tag assume_completion
  @post __post.safeMode
 */
function setSafeMode() public onlySender {
  safeMode = true;
  emit SetSafeModeActivated(msg.sender);
}

function recoverFromSignature(bytes32 operationHash, bytes memory signature) private
     pure returns (address) {
  if (signature.length != 65) {
    revert();
  }
  bytes32 r;
  bytes32 s;
  uint8 v;
  assembly {
    r := mload(add(signature, 32))
    s := mload(add(signature, 64))
    v := and(mload(add(signature, 65)), 255)
  }
  if (v < 27) {
    v += 27;
  }
  return ecrecover(operationHash, v, r, s);
}

function insertSequenceId(uint sequenceId) private onlySender {
  uint minIndex = 0;
  for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
    if (recentSequenceIds[i] == sequenceId) {
      revert();
    }
    if (recentSequenceIds[i] < recentSequenceIds[minIndex]) {
      minIndex = i;
    }
  }
  if (sequenceId < recentSequenceIds[minIndex]) {
    revert();
  }
  if (sequenceId > (recentSequenceIds[minIndex] + 10000)) {
   revert();
  }
  recentSequenceIds[minIndex] = sequenceId;
}

function getNextSequenceId() public view returns (uint) {
```

```
    uint maxSequenceId = 0;
    /*@CTK getNextSequenceId_forloop
      @pre MAX_SEQUENCE_ID_SIZE == 10
      @pre maxSequenceId == 0
      @inv this == this__pre
      @inv i <= MAX_SEQUENCE_ID_SIZE
      @inv !__should_return
      @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
      @post i == MAX_SEQUENCE_ID_SIZE
      @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >=
          recentSequenceIds[i]
     */
    for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
      if (recentSequenceIds[i] > maxSequenceId) {
        maxSequenceId = recentSequenceIds[i];
      }
    }
    return maxSequenceId + 1;
  }

  function isInArray(address item, address [] memory items) private pure returns (bool
      ) {
    /*@CTK isInArray_forloop
      @inv this == this__pre
      @inv item == item__pre
      @inv i <= items.length
      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item)
      @inv !__return__pre
      @inv __should_return == __return
      @post __should_return -> (i < items.length /\ items[i] == item)
      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item
          )
      @post !__should_return -> (__return == __return__pre)
      @post !__should_return -> i == items.length
     */
    for (uint i = 0; i < items.length; i++) {
      if (items[i] == item) {
        return true;
      }
    }
    return false;
  }
}
```

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
```

Static Analysis Request

TIMESTAMP_DEPENDENCY

Line 99 in File 23contract.sol

```
99    if (expireTime < block.timestamp) {
```

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

TIMESTAMP_DEPENDENCY

Line 121 in File mncontract.sol

```
121   if (expireTime < block.timestamp) {
```

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

Formal Verification Request 1

ethMultiSig

15, Apr 2019
37.04 ms

Line 19-23 in File 23contract.sol

```
19 /*@CTK ethMultiSig
20   @tag assume_completion
21   @post allowedSigners.length == 3
22   @post __post.signers == allowedSigners
23  */
```

Line 24-30 in File 23contract.sol

```
24 constructor(address[] memory allowedSigners) public {
25   if (allowedSigners.length != 3) {
26     assert(false);
27     // revert();
28   }
29   signers = allowedSigners;
30 }
```

The code meets the specification

Formal Verification Request 2

isSigner

15, Apr 2019
33.17 ms

Line 38-40 in File 23contract.sol

```
38 /*@CTK isSigner
39   @post (exists i: uint. (i < signers.length /\ signers[i] == signer)) == __return
40  */
```

Line 41-60 in File 23contract.sol

```
41 function isSigner(address signer) public view returns (bool) {
42   /*@CTK isSigner_forloop
43     @inv this == this__pre
44     @inv signer == signer__pre
45     @inv i <= signers.length
46     @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
         signer)
47     @inv !__return__pre
48     @inv __should_return == __return
49     @post __should_return -> (i < signers.length /\ signers[i] == signer)
50     @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
         signer)
51     @post !__should_return -> (__return == __return__pre)
52     @post !__should_return -> i == signers.length
53    */
```

```
  for (uint i = 0; i < signers.length; i++) {
    if (signers[i] == signer) {
      return true;
    }
  }
  return false;
}
```

The code meets the specification

Formal Verification Request 3

setSafeMode

15, Apr 2019
59.46 ms

Line 112-115 in File 23contract.sol

```
/*@CTK setSafeMode
  @tag assume_completion
  @post __post.safeMode == true
 */
```

Line 116-119 in File 23contract.sol

```
function setSafeMode() public onlySigner {
  safeMode = true;
  emit SetSafeModeActivated(msg.sender);
}
```

The code meets the specification

Formal Verification Request 4

isSigner_forloop__Generated

15, Apr 2019
107.42 ms

(Loop) Line 42-53 in File 23contract.sol

```
  /*@CTK isSigner_forloop
    @inv this == this__pre
    @inv signer == signer__pre
    @inv i <= signers.length
    @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
        signer)
    @inv !__return__pre
    @inv __should_return == __return
    @post __should_return -> (i < signers.length /\ signers[i] == signer)
    @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
        signer)
    @post !__should_return -> (__return == __return__pre)
    @post !__should_return -> i == signers.length
   */
```

(Loop) Line 42-58 in File 23contract.sol

```
42    /*@CTK isSigner_forloop
43      @inv this == this__pre
44      @inv signer == signer__pre
45      @inv i <= signers.length
46      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
            signer)
47      @inv !__return__pre
48      @inv __should_return == __return
49      @post __should_return -> (i < signers.length /\ signers[i] == signer)
50      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
            signer)
51      @post !__should_return -> (__return == __return__pre)
52      @post !__should_return -> i == signers.length
53     */
54    for (uint i = 0; i < signers.length; i++) {
55      if (signers[i] == signer) {
56        return true;
57      }
58    }
```

The code meets the specification

Formal Verification Request 5

getNextSequenceId_forloop__Generated

15, Apr 2019
936.05 ms

(Loop) Line 167-176 in File 23contract.sol

```
167    /*@CTK getNextSequenceId_forloop
168      @pre MAX_SEQUENCE_ID_SIZE == 10
169      @pre maxSequenceId == 0
170      @inv this == this__pre
171      @inv i <= MAX_SEQUENCE_ID_SIZE
172      @inv !__should_return
173      @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
174      @post i == MAX_SEQUENCE_ID_SIZE
175      @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >=
             recentSequenceIds[i]
176     */
```

(Loop) Line 167-181 in File 23contract.sol

```
167    /*@CTK getNextSequenceId_forloop
168      @pre MAX_SEQUENCE_ID_SIZE == 10
169      @pre maxSequenceId == 0
170      @inv this == this__pre
171      @inv i <= MAX_SEQUENCE_ID_SIZE
172      @inv !__should_return
173      @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
174      @post i == MAX_SEQUENCE_ID_SIZE
175      @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >=
             recentSequenceIds[i]
```

```
    */
  for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
    if (recentSequenceIds[i] > maxSequenceId) {
      maxSequenceId = recentSequenceIds[i];
    }
  }
```

The code meets the specification

Formal Verification Request 6

isSigner

15, Apr 2019
24.93 ms

Line 43-45 in File mncontract.sol

```
43 /*@CTK isSigner
44   @post (exists i: uint. (i < signers.length /\ signers[i] == signer)) == __return
45  */
```

Line 46-65 in File mncontract.sol

```
46 function isSigner(address signer) public view returns (bool) {
47   /*@CTK isSigner_forloop
48     @inv this == this__pre
49     @inv signer == signer__pre
50     @inv i <= signers.length
51     @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
         signer)
52     @inv !__return__pre
53     @inv __should_return == __return
54     @post __should_return -> (i < signers.length /\ signers[i] == signer)
55     @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
         signer)
56     @post !__should_return -> (__return == __return__pre)
57     @post !__should_return -> i == signers.length
58    */
59   for (uint i = 0; i < signers.length; i++) {
60     if (signers[i] == signer) {
61       return true;
62     }
63   }
64   return false;
65 }
```

The code meets the specification

Formal Verification Request 7

isSender

15, Apr 2019
24.98 ms

Line 67-69 in File mncontract.sol

```
/*@CTK isSender
  @post (exists i: uint. (i < senders.length /\ senders[i] == sender)) == __return
 */
```

Line 70-89 in File mncontract.sol

```
function isSender(address sender) public view returns (bool) {
  /*@CTK isSender_forloop
    @inv this == this__pre
    @inv sender == sender__pre
    @inv i <= senders.length
    @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
        sender)
    @inv !__return__pre
    @inv __should_return == __return
    @post __should_return -> (i < senders.length /\ senders[i] == sender)
    @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
        sender)
    @post !__should_return -> (__return == __return__pre)
    @post !__should_return -> i == senders.length
   */
  for (uint i = 0; i < senders.length; i++) {
    if (senders[i] == sender) {
      return true;
    }
  }
  return false;
}
```

✅ The code meets the specification

Formal Verification Request 8

setSafeMode

📅 15, Apr 2019
⏱ 53.86 ms

Line 147-150 in File mncontract.sol

```
/*@CTK setSafeMode
  @tag assume_completion
  @post __post.safeMode
 */
```

Line 151-154 in File mncontract.sol

```
function setSafeMode() public onlySender {
  safeMode = true;
  emit SetSafeModeActivated(msg.sender);
}
```

✅ The code meets the specification

Formal Verification Request 9

isSigner_forloop__Generated

15, Apr 2019
99.54 ms

(Loop) Line 47-58 in File mncontract.sol

```
47    /*@CTK isSigner_forloop
48      @inv this == this__pre
49      @inv signer == signer__pre
50      @inv i <= signers.length
51      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
52      @inv !__return__pre
53      @inv __should_return == __return
54      @post __should_return -> (i < signers.length /\ signers[i] == signer)
55      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
56      @post !__should_return -> (__return == __return__pre)
57      @post !__should_return -> i == signers.length
58     */
```

(Loop) Line 47-63 in File mncontract.sol

```
47    /*@CTK isSigner_forloop
48      @inv this == this__pre
49      @inv signer == signer__pre
50      @inv i <= signers.length
51      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
52      @inv !__return__pre
53      @inv __should_return == __return
54      @post __should_return -> (i < signers.length /\ signers[i] == signer)
55      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> signers[j] !=
          signer)
56      @post !__should_return -> (__return == __return__pre)
57      @post !__should_return -> i == signers.length
58     */
59    for (uint i = 0; i < signers.length; i++) {
60      if (signers[i] == signer) {
61        return true;
62      }
63    }
```

The code meets the specification

Formal Verification Request 10

isSender_forloop__Generated

15, Apr 2019
117.46 ms

(Loop) Line 71-82 in File mncontract.sol

```
71    /*@CTK isSender_forloop
72      @inv this == this__pre
73      @inv sender == sender__pre
74      @inv i <= senders.length
75      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
76      @inv !__return__pre
77      @inv __should_return == __return
78      @post __should_return -> (i < senders.length /\ senders[i] == sender)
79      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
80      @post !__should_return -> (__return == __return__pre)
81      @post !__should_return -> i == senders.length
82     */
```

(Loop) Line 71-87 in File mncontract.sol

```
71    /*@CTK isSender_forloop
72      @inv this == this__pre
73      @inv sender == sender__pre
74      @inv i <= senders.length
75      @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
76      @inv !__return__pre
77      @inv __should_return == __return
78      @post __should_return -> (i < senders.length /\ senders[i] == sender)
79      @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> senders[j] !=
          sender)
80      @post !__should_return -> (__return == __return__pre)
81      @post !__should_return -> i == senders.length
82     */
83    for (uint i = 0; i < senders.length; i++) {
84      if (senders[i] == sender) {
85        return true;
86      }
87    }
```

✅ The code meets the specification

Formal Verification Request 11

getNextSequenceId_forloop__Generated

📅 15, Apr 2019
⏱ 966.78 ms

(Loop) Line 195-204 in File mncontract.sol

```
195   /*@CTK getNextSequenceId_forloop
196     @pre MAX_SEQUENCE_ID_SIZE == 10
197     @pre maxSequenceId == 0
198     @inv this == this__pre
199     @inv i <= MAX_SEQUENCE_ID_SIZE
200     @inv !__should_return
201     @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
202     @post i == MAX_SEQUENCE_ID_SIZE
```

```
    @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >= recentSequenceIds[i]
   */
```

(Loop) Line 195-209 in File mncontract.sol

```
  /*@CTK getNextSequenceId_forloop
    @pre MAX_SEQUENCE_ID_SIZE == 10
    @pre maxSequenceId == 0
    @inv this == this__pre
    @inv i <= MAX_SEQUENCE_ID_SIZE
    @inv !__should_return
    @inv forall j: uint. (j >= 0 /\ j < i) -> maxSequenceId >= recentSequenceIds[j]
    @post i == MAX_SEQUENCE_ID_SIZE
    @post forall i: uint. (i >= 0 /\ i < MAX_SEQUENCE_ID_SIZE) -> maxSequenceId >= recentSequenceIds[i]
   */
  for (uint i = 0; i < MAX_SEQUENCE_ID_SIZE; i++) {
    if (recentSequenceIds[i] > maxSequenceId) {
      maxSequenceId = recentSequenceIds[i];
    }
  }
```

✅ The code meets the specification

Formal Verification Request 12

isInArray_forloop__Generated

📅 15, Apr 2019
⏱ 115.56 ms

(Loop) Line 214-225 in File mncontract.sol

```
  /*@CTK isInArray_forloop
    @inv this == this__pre
    @inv item == item__pre
    @inv i <= items.length
    @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item)
    @inv !__return__pre
    @inv __should_return == __return
    @post __should_return -> (i < items.length /\ items[i] == item)
    @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item)
    @post !__should_return -> (__return == __return__pre)
    @post !__should_return -> i == items.length
   */
```

(Loop) Line 214-230 in File mncontract.sol

```
  /*@CTK isInArray_forloop
    @inv this == this__pre
    @inv item == item__pre
    @inv i <= items.length
    @inv !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item)
    @inv !__return__pre
    @inv __should_return == __return
    @post __should_return -> (i < items.length /\ items[i] == item)
```

```
    @post !__should_return -> (forall j: uint. (j >= 0 /\ j < i) -> items[j] != item
        )
    @post !__should_return -> (__return == __return__pre)
    @post !__should_return -> i == items.length
   */
  for (uint i = 0; i < items.length; i++) {
    if (items[i] == item) {
      return true;
    }
  }
```

The code meets the specification