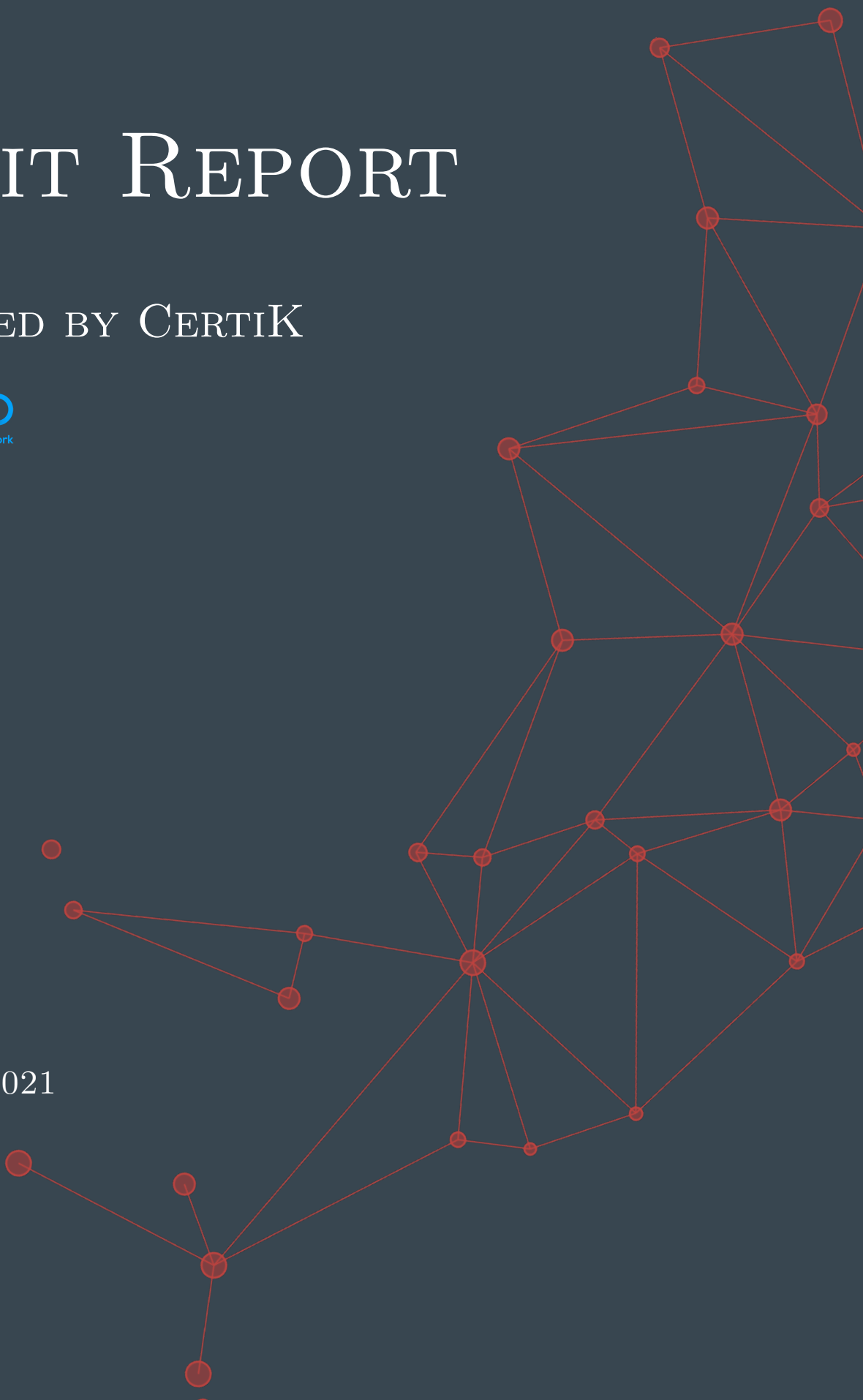
CERTIK

Audit Report

Produced by CertiK

for Idavoll Network



March 1, 2021

CertiK Audit Report For Idavoll Network

Request Date: 2021-02-26
Revision Date: 2021-03-01
Platform Name: Ethereum

Contents

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

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, CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that projects are checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and assessments to each project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality deliverable. For more information: https://certik.io.

Executive Summary

This report has been prepared for Idavoll Network to discover issues and vulnerabilities in the source code of their idavoll.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 practices 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 three buckets based on overall risk levels:

Critical

Code implementation does not match specification, which could result in the loss of funds for contract owner or users.

Medium

Code implementation does not match the specification under certain conditions, which could affect the security standard by loss of access control.

Low

Code implementation does not follow best practices, or uses suboptimal design patterns, which could 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.*

Mar 01, 2021

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow/ Underflow	An overflow/underflow occurs when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker can 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.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic if there are publicly disclosed bugs and issues that affect the current compiler version used.	1	SWC-102 SWC-103
Insecure Randomness	Using block attributes to generate random numbers is unreliable, 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

Title	Description	Issues	SWC ID
Delegatecall to Untrusted Callee	Calling untrusted contracts is very dangerous, so 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, meaning a malicious user can 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.	0	SWC-111
Unused Variables	Unused variables reduce code quality	0	SWC-131

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Review Notes

Source Code

`https://etherscan.io/address/0x92ec47df1aa167806dfa4916d9cfb99da6953b8f#code`

Source Code SHA-256 Checksum

- **[idavoll.sol].sol**
  `a69da2ec32e74ab63af9e7effc7f5dcac9a5b3ec30fad8dbb4fe78102cdecf87`

Summary

CertiK worked closely with Idavoll Network to audit the design and implementation of its soon-to-be released smart contract. To ensure comprehensive protection, the source code was 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 best practices.

Overall, we found Idavoll Network's 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, continually improve the codebase, and perform additional tests before the mainnet release.

Recommendations

Items in this section are not critical to the overall functionality of Idavoll Network's smart contracts; however, we leave it to the client's discretion to decide whether to address them before the final deployment of source codes. Recommendations are labeled CRITICAL, MAJOR, MINOR, INFO, and DISCUSSION in decreasing significance level.

idavoll.sol

- INFO `add()`,`sub()` – Consider replacing `assert` by `require`.
- MINOR `constructor()` – If the `_moveAddr` is not the `msg.sender` address, the contract will be deployed failed. Consider changing like this: `emit Transfer(adress(0x0), _moveAddr, totalSupply);`
- MINOR `transfer()`, `transferFrom()` – Consider adding zero check for the address `_to`.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File idavoll.sol

```
1 pragma solidity ^0.5.17;
```

Version to compile has the following bug:
0.5.17: EmptyByteArrayCopy, DynamicArrayCleanup, MissingEscapingInFormatting, ImplicitConstructorCallvalueCheck, TupleAssignmentMultiStackSlotComponents, MemoryArrayCreationOverflow

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_sub

01, Mar 2021
13.94 ms

Line 4-12 in File idavoll.sol

```
4    /*@CTK SafeMath_sub
5    @tag spec
6    @tag is_pure
7    @post __reverted == __has_assertion_failure
8    @post __has_overflow == true -> __has_assertion_failure == true
9    @post !__reverted -> __return == a - b
10   @post (a < b) == __has_assertion_failure
11   @post !__has_buf_overflow
12 */
```

Line 13-16 in File idavoll.sol

```
13   function sub(uint256 a, uint256 b) internal pure returns (uint256) {
14     assert(b <= a);
15     return a - b;
16   }
```

The code meets the specification.

Formal Verification Request 2

SafeMath_add

01, Mar 2021
16.16 ms

Line 17-25 in File idavoll.sol

```
17   /*@CTK SafeMath_add
18   @tag spec
19   @tag is_pure
20   @post __reverted == __has_assertion_failure
21   @post __has_assertion_failure == __has_overflow
22   @post !__reverted -> __return == a + b
23   @post ((a + b < a) || (a + b < b)) == __has_assertion_failure
24   @post !__has_buf_overflow
25 */
```

Line 26-30 in File idavoll.sol

```
26   function add(uint256 a, uint256 b) internal pure returns (uint256) {
27     uint256 c = a + b;
28     assert(c >= a);
29     return c;
30   }
```

✅ The code meets the specification.

Formal Verification Request 3

idavoll_constructor

📅 01, Mar 2021
⏱ 96.1 ms

Line 46-50 in File idavoll.sol

```
46    /*@CTK idavoll_constructor
47      @tag assume_completion
48      @pre _moveAddr != address(0)
49      @post __post.balances[_moveAddr] == totalSupply
50    */
```

Line 51-55 in File idavoll.sol

```
51    constructor(address _moveAddr) public {
52      require(_moveAddr != address(0), "_moveAddress is a zero address");
53      balances[_moveAddr] = totalSupply;
54      transfer(_moveAddr, totalSupply);
55    }
```

✅ The code meets the specification.

Formal Verification Request 4

balanceOf

📅 01, Mar 2021
⏱ 8.2 ms

Line 56-58 in File idavoll.sol

```
56    /*@CTK balanceOf
57      @post __return == balances[_owner]
58    */
```

Line 59-61 in File idavoll.sol

```
59    function balanceOf(address _owner) public view returns (uint256) {
60      return balances[_owner];
61    }
```

✅ The code meets the specification.

Formal Verification Request 5

transfer

📅 01, Mar 2021
⏱ 35.37 ms

Line 62-69 in File idavoll.sol

```
62      /*@CTK transfer
63        @tag assume_completion
64        @pre balances[msg.sender] >= _value
65        @pre _to != address(0)
66        @post ((msg.sender) == _to) -> __post.balances[msg.sender] ==
    ↪  balances[msg.sender]
67        @post ((msg.sender) != _to) -> __post.balances[msg.sender] ==
    ↪  balances[msg.sender] - _value
68        @post ((msg.sender) != _to) -> __post.balances[_to] == balances[_to] +
    ↪  _value
69      */
```

Line 70-77 in File idavoll.sol

```
70      function transfer(address _to, uint256 _value) public returns (bool) {
71        //require (_to != address(0), "not enough balance !");
72        require((balances[msg.sender] >= _value), "not enough balance !");
73        balances[msg.sender] = balances[msg.sender].sub(_value);
74        balances[_to] = balances[_to].add(_value);
75        emit Transfer(msg.sender, _to, _value);
76        return true;
77      }
```

✅ The code meets the specification.

Formal Verification Request 6

transferFrom

📅 01, Mar 2021
⏱ 128.37 ms

Line 78-85 in File idavoll.sol

```
78      /*@CTK transferFrom
79        @tag assume_completion
80        @pre balances[_from] >= _value
81        @pre allowed[_from][msg.sender] >= _value
82        @post (_from == _to) -> __post.balances[_from] == balances[_from]
83        @post (_from != _to) -> __post.balances[_from] == balances[_from] -
    ↪  _value
84        @post (_from != _to) -> __post.balances[_to] == balances[_to] + _value
85      */
```

Line 86-94 in File idavoll.sol

```
86    function transferFrom(address _from, address _to, uint256 _value) public
      ↪ returns (bool) {
87      //require (_to != address(0), "not enough balance !");
88      require(balances[_from] >= _value && allowed[_from][msg.sender] >=
      ↪ _value, "not enough allowed balance");
89      allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
90      balances[_from] = balances[_from].sub(_value);
91      balances[_to] = balances[_to].add(_value);
92      emit Transfer(_from, _to, _value);
93      return true;
94    }
```

The code meets the specification.

Formal Verification Request 7

batchTransfer

01, Mar 2021
16.02 ms

Line 112-115 in File idavoll.sol

```
112   /*@CTK batchTransfer
113     @tag assume_completion
114     @pre _users.length == _amounts.length
115    */
```

Line 116-131 in File idavoll.sol

```
116   function batchTransfer(
117       address payable[] memory _users,
118       uint256[] memory _amounts
119   )
120       public
121       returns (bool)
122   {
123       require(_users.length == _amounts.length,"not same length");
124       /*@CTK batchTransfer__loop
125         @tag assume_completion
126     */
127       for(uint8 i = 0; i < _users.length; i++){
128           transfer(_users[i], _amounts[i]);
129       }
130       return true;
131   }
```

The code meets the specification.

Formal Verification Request 8

batchTransfer___loop___Generated

01, Mar 2021
52.19 ms

(Loop) Line 124-126 in File idavoll.sol

```
124         /*@CTK batchTransfer__loop
125           @tag assume_completion
126         */
```

(Loop) Line 124-129 in File idavoll.sol

```
124         /*@CTK batchTransfer__loop
125           @tag assume_completion
126         */
127         for(uint8 i = 0; i < _users.length; i++){
128             transfer(_users[i], _amounts[i]);
129         }
```

The code meets the specification.

Source Code with CertiK Labels

idavoll.sol

```
pragma solidity ^0.5.17;
// SPDX-License-Identifier: MIT
  library SafeMath {
    /*@CTK SafeMath_sub
    @tag spec
    @tag is_pure
    @post __reverted == __has_assertion_failure
    @post __has_overflow == true -> __has_assertion_failure == true
    @post !__reverted -> __return == a - b
    @post (a < b) == __has_assertion_failure
    @post !__has_buf_overflow
  */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
      assert(b <= a);
      return a - b;
    }
    /*@CTK SafeMath_add
    @tag spec
    @tag is_pure
    @post __reverted == __has_assertion_failure
    @post __has_assertion_failure == __has_overflow
    @post !__reverted -> __return == a + b
    @post ((a + b < a) || (a + b < b)) == __has_assertion_failure
    @post !__has_buf_overflow
  */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
      uint256 c = a + b;
      assert(c >= a);
      return c;
    }
  }
  contract idavoll {
    using SafeMath for uint256;

    string public constant name = "Idavoll Network";
    string public constant symbol = "IDV";
    uint256 public constant decimals = 18;
    uint256 public constant totalSupply = 2000000000*10**decimals;

    mapping (address => uint256) private balances;
    mapping (address => mapping (address => uint256)) private allowed;

    event Transfer(address indexed _from, address indexed _to, uint256
↪   _value);
```

```
44      event Approval(address indexed _owner, address indexed _spender, uint256
    ↪   _value);
45
46      /*@CTK idavoll_constructor
47        @tag assume_completion
48        @pre _moveAddr != address(0)
49        @post __post.balances[_moveAddr] == totalSupply
50      */
51      constructor(address _moveAddr) public {
52        require(_moveAddr != address(0), "_moveAddress is a zero address");
53        balances[_moveAddr] = totalSupply;
54        transfer(_moveAddr, totalSupply);
55      }
56      /*@CTK balanceOf
57        @post __return == balances[_owner]
58      */
59      function balanceOf(address _owner) public view returns (uint256) {
60        return balances[_owner];
61      }
62      /*@CTK transfer
63        @tag assume_completion
64        @pre balances[msg.sender] >= _value
65        @pre _to != address(0)
66        @post ((msg.sender) == _to) -> __post.balances[msg.sender] ==
    ↪   balances[msg.sender]
67        @post ((msg.sender) != _to) -> __post.balances[msg.sender] ==
    ↪   balances[msg.sender] - _value
68        @post ((msg.sender) != _to) -> __post.balances[_to] == balances[_to] +
    ↪   _value
69      */
70      function transfer(address _to, uint256 _value) public returns (bool) {
71        //require (_to != address(0), "not enough balance !");
72        require((balances[msg.sender] >= _value), "not enough balance !");
73        balances[msg.sender] = balances[msg.sender].sub(_value);
74        balances[_to] = balances[_to].add(_value);
75        emit Transfer(msg.sender, _to, _value);
76        return true;
77      }
78      /*@CTK transferFrom
79        @tag assume_completion
80        @pre balances[_from] >= _value
81        @pre allowed[_from][msg.sender] >= _value
82        @post (_from == _to) -> __post.balances[_from] == balances[_from]
83        @post (_from != _to) -> __post.balances[_from] == balances[_from] -
    ↪   _value
84        @post (_from != _to) -> __post.balances[_to] == balances[_to] + _value
85      */
```

```
86      function transferFrom(address _from, address _to, uint256 _value) public
        ↪ returns (bool) {
87        //require (_to != address(0), "not enough balance !");
88        require(balances[_from] >= _value && allowed[_from][msg.sender] >=
        ↪ _value, "not enough allowed balance");
89        allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
90        balances[_from] = balances[_from].sub(_value);
91        balances[_to] = balances[_to].add(_value);
92        emit Transfer(_from, _to, _value);
93        return true;
94      }
95      /*CTK approve
96        @tag assume_completion
97        @post __post.allowed[msg.sender][_spender] == _value
98      */
99      function approve(address _spender, uint256 _value) public returns (bool)
        ↪ {
100       allowed[msg.sender][_spender] = _value;
101       emit Approval(msg.sender, _spender, _value);
102       return true;
103     }
104     /*CTK approve
105       @tag assume_completion
106       @post __return == allowed[_owner][_spender]
107     */
108     function allowance(address _owner, address _spender) public view returns
        ↪ (uint256) {
109       return allowed[_owner][_spender];
110     }
111
112     /*@CTK batchTransfer
113       @tag assume_completion
114       @pre _users.length == _amounts.length
115      */
116     function batchTransfer(
117         address payable[] memory _users,
118         uint256[] memory _amounts
119     )
120         public
121         returns (bool)
122     {
123         require(_users.length == _amounts.length,"not same length");
124         /*@CTK batchTransfer__loop
125           @tag assume_completion
126       */
127         for(uint8 i = 0; i < _users.length; i++){
128             transfer(_users[i], _amounts[i]);
129         }
```

```
130             return true;
131         }
132     }
```