CertiK Audit Report For hCaptcha

Request Date: 2019-05-24
Revision Date: 2019-07-03
Platform Name: Ethereum

Contents

Disclaimer

About CertiK

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

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

CertiK differs from traditional testing approaches by employing Formal Verification to mathematically prove blockchain ecosystem and smart contracts are hacker-resistant and bug-free. CertiK uses this industry-leading technology together with standardized test suites, static analysis and expert manual review to create a full-stack solution for our partners across the blockchain world to secure 1.4B in assets.

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

Exective Summary

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

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

Vulnerability Classification

For every 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.

Testing Summary

PASS

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

Jul 03, 2019

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120

“tx.origin” for authorization	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused variables	Unused variables reduce code quality	0	

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **HMToken.sol**
  `2beb2103decfe1bb07195b0546082ad5f12a38c3f3dec38382eb617ef11c150b`
- **HMTokenInterface.sol**
  `f93b10306b6d53835f10533dc941b4c7e49d2aa4e6149d154fcd36158d3abc6b`
- **Migrations.sol**
  `b2b5280e1b16afb218c37a3083c6e8d2a2c82627ab27f9fe044af6f24e2008a8`
- **SafeMath.sol**
  `95e6ccf2891bd7814fedb6768c99cc61d813814029f16e907d8a8c3c9c5c6c10`

Summary

CertiK team is invited by the hCaptcha team to audit the design and implementations of its to be released ERC20 based smart contract, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checking as well as manual reviews by smart contract experts. We have been actively interacting with client-side engineers when there was any potential loopholes or recommended design changes during the audit process, and hCaptcha team has been actively giving us updates for the source code and feedback about the business logic.

Overall we found the hCaptcha contracts follow good practices, with a reasonable amount of features on top of the ERC20 such as bulk functions to handle a batch of transfer or approve requests. With the final update of source code and delivery of the audit report, we conclude that the contract is 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 seeking multiple opinions, more test coverage, and sandbox deployments before the mainnet release.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File SafeMath.sol

```
pragma solidity 0.5.9;
```

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

INSECURE_COMPILER_VERSION

Line 1 in File HMToken.sol

```
pragma solidity 0.5.9;
```

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

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 add

03, Jul 2019
17.66 ms

Line 27-35 in File SafeMath.sol

```
    /*@CTK "SafeMath add"
      @tag spec
      @tag is_pure
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
```

Line 36-41 in File SafeMath.sol

```
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 2

SafeMath sub

03, Jul 2019
15.48 ms

Line 52-60 in File SafeMath.sol

```
    /*@CTK "SafeMath sub"
      @tag spec
      @tag is_pure
      @post (b > a) == __reverted
      @post !__reverted -> __return == a - b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
```

Line 61-66 in File SafeMath.sol

```
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 3

SafeMath mul zero

📅 03, Jul 2019
⏱ 19.2 ms

Line 77-82 in File SafeMath.sol

```
    /*@CTK "SafeMath mul zero"
      @tag spec
      @tag is_pure
      @pre (a == 0)
      @post __return == 0
    */
```

Line 93-105 in File SafeMath.sol

```
    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, "SafeMath: multiplication overflow");

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 4

SafeMath mul nonzero

📅 03, Jul 2019
⏱ 289.32 ms

Line 83-92 in File SafeMath.sol

```
    /*@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)
    */
```

Line 93-105 in File SafeMath.sol

```
    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, "SafeMath: multiplication overflow");

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 5

SafeMath div

03, Jul 2019
13.18 ms

Line 118-126 in File SafeMath.sol

```
    /*@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)
    */
```

Line 127-134 in File SafeMath.sol

```
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
```

✓ The code meets the specification.

Formal Verification Request 6

SafeMath mod

03, Jul 2019
12.0 ms

Line 147-155 in File SafeMath.sol

```
    /*@CTK "SafeMath mod"
      @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)
    */
```

Line 156-159 in File SafeMath.sol

```
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
```

✓ The code meets the specification.

Formal Verification Request 7

HMToken

03, Jul 2019
20.66 ms

Line 24-30 in File HMToken.sol

```
    /*@CTK HMToken
      @tag assume_completion
      @post __post.name == _name
      @post __post.decimals == _decimals
      @post __post.symbol == _symbol
      @post __post.balances[msg.sender] == __post.totalSupply
     */
```

Line 31-37 in File HMToken.sol

```
    constructor(uint256 _totalSupply, string memory _name, uint8 _decimals, string
        memory _symbol) public {
        totalSupply = _totalSupply * (10 ** uint256(_decimals));
        name = _name;
        decimals = _decimals;
        symbol = _symbol;
        balances[msg.sender] = totalSupply;
    }
```

✓ The code meets the specification.

Formal Verification Request 8

transfer

03, Jul 2019
158.41 ms

Line 39-48 in File HMToken.sol

```
    /*@CTK transfer
      @tag assume_completion
      @pre msg.sender != _to
      @post _to != address(0)
      @post _to != address(this)
      @post balances[msg.sender] >= _value
      @post (balances[_to] + _value) >= balances[_to]
      @post __post.balances[msg.sender] == balances[msg.sender] - _value
      @post __post.balances[_to] == balances[_to] + _value
     */
```

Line 49-53 in File HMToken.sol

```
    function transfer(address _to, uint256 _value) public returns (bool success) {
        success = transferQuiet(_to, _value);
        require(success, "Transfer didn't succeed");
        return success;
    }
```

✅ The code meets the specification.

Formal Verification Request 9

transferBulk

📅 03, Jul 2019
⏱ 34.42 ms

Line 55-59 in File HMToken.sol

```
    /*@CTK transferBulk
      @tag assume_completion
      @pre _tos.length == _values.length
      @pre _tos.length < BULK_MAX_COUNT
     */
```

Line 60-92 in File HMToken.sol

```
     function transferBulk(address[] memory _tos, uint256[] memory _values, uint256
         _txId) public returns (uint256 _bulkCount) {
        require(_tos.length == _values.length, "Amount of recipients and values don't
            match");
        require(_tos.length < BULK_MAX_COUNT, "Too many recipients");

        uint256 _bulkValue = 0;
        /*@CTK transferBulk_bulkValue_sum
          @inv j <= _tos.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _tos.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
        require(_bulkValue < BULK_MAX_VALUE, "Bulk value too high");

        _bulkCount = 0;
        bool _success;
        /*@CTK transferBulk_transfer
```

```
78           @inv i <= _tos.length
79           @inv _bulkCount >= _bulkCount__pre
80           @post !__should_return
81          */
82        for (uint256 i = 0; i < _tos.length; ++i) {
83            _success = transferQuiet(_tos[i], _values[i]);
84            if (_success) {
85                _bulkCount = _bulkCount.add(1);
86            } else {
87                emit BulkTransferFailure(_txId, _bulkCount);
88            }
89        }
90        emit BulkTransfer(_txId, _bulkCount);
91        return _bulkCount;
92    }
```

The code meets the specification.

Formal Verification Request 10

transferFrom

03, Jul 2019
270.0 ms

Line 94-103 in File HMToken.sol

```
94    /*@CTK transferFrom
95      @tag assume_completion
96      @pre _spender != _to
97      @post balances[_spender] >= _value && allowed[_spender][msg.sender] >= _value
98      @post _to != address(0)
99      @post __post.balances[_spender] == balances[_spender] - _value
100     @post __post.balances[_to] == balances[_to] + _value
101     @post (allowed[_spender][msg.sender] < MAX_UINT256) -> (__post.allowed[_spender
          ][msg.sender] == allowed[_spender][msg.sender] - _value)
102     @post (allowed[_spender][msg.sender] >= MAX_UINT256) -> (__post.allowed[_spender
          ][msg.sender] == allowed[_spender][msg.sender])
103    */
```

Line 104-118 in File HMToken.sol

```
104   function transferFrom(address _spender, address _to, uint256 _value) public
          returns (bool success) {
105       uint256 _allowance = allowed[_spender][msg.sender];
106       require(balances[_spender] >= _value && _allowance >= _value, "Spender balance
              or allowance too low");
107       require(_to != address(0), "Can't send tokens to uninitialized address");
108
109       balances[_spender] = balances[_spender].sub(_value);
110       balances[_to] = balances[_to].add(_value);
111
112       if (_allowance < MAX_UINT256) {
113           allowed[_spender][msg.sender] = allowed[_spender][msg.sender].sub(_value);
114       }
115
116       emit Transfer(_spender, _to, _value);
117       return true;
```

```
    }
```

✓ The code meets the specification.

Formal Verification Request 11

approve

03, Jul 2019
19.46 ms

Line 120-124 in File HMToken.sol

```
    /*@CTK approve
      @tag assume_completion
      @post _spender != address(0)
      @post __post.allowed[msg.sender][_spender] == _value
     */
```

Line 125-131 in File HMToken.sol

```
    function approve(address _spender, uint256 _value) public returns (bool success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value); //solhint-disable-line indent, no-
            unused-vars
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 12

approveBulk

03, Jul 2019
33.51 ms

Line 133-137 in File HMToken.sol

```
    /*@CTK approveBulk
      @tag assume_completion
      @pre _spenders.length == _values.length
      @pre _spenders.length < BULK_MAX_COUNT
     */
```

Line 138-170 in File HMToken.sol

```
     function approveBulk(address[] memory _spenders, uint256[] memory _values,
         uint256 _txId) public returns (uint256 _bulkCount) {
        require(_spenders.length == _values.length, "Amount of spenders and values don'
            t match");
        require(_spenders.length < BULK_MAX_COUNT, "Too many spenders");

        uint256 _bulkValue = 0;
        /*@CTK approveBulk_bulkValue_sum
```

```
          @inv j <= _spenders.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _spenders.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
        require(_bulkValue < BULK_MAX_VALUE, "Bulk value too high");

        _bulkCount = 0;
        bool _success;
        /*@CTK approveBulk_approve
          @inv i <= _spenders.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
        for (uint256 i = 0; i < _spenders.length; ++i) {
            _success = increaseApproval(_spenders[i], _values[i]);
            if (_success) {
                _bulkCount = _bulkCount.add(1);
            } else {
                emit BulkApprovalFailure(_txId, _bulkCount);
            }
        }
        emit BulkApproval(_txId, _bulkCount);
        return _bulkCount;
    }
```

✓ The code meets the specification.

Formal Verification Request 13

increaseApproval

03, Jul 2019
304.54 ms

Line 172-178 in File HMToken.sol

```
    /*@CTK increaseApproval
      @tag assume_completion
      @pre allowed[msg.sender][_spender] + _delta < MAX_UINT256
      @post _spender != address(0)
      @post __post.allowed[msg.sender][_spender] ==
            allowed[msg.sender][_spender] + _delta
     */
```

Line 179-190 in File HMToken.sol

```
    function increaseApproval(address _spender, uint256 _delta) public returns (bool
        success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        uint256 _oldValue = allowed[msg.sender][_spender];
        if (_oldValue.add(_delta) < _oldValue || _oldValue.add(_delta) >= MAX_UINT256)
            { // Truncate upon overflow.
            allowed[msg.sender][_spender] = MAX_UINT256.sub(1);
        } else {
            allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_delta);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 14

decreaseApproval

03, Jul 2019
90.77 ms

Line 192-200 in File HMToken.sol

```
    /*@CTK decreaseApproval
      @tag assume_completion
      @post _spender != address(0)
      @post _delta > allowed[msg.sender][_spender] ->
            __post.allowed[msg.sender][_spender] == 0
      @post _delta <= allowed[msg.sender][_spender]
            -> __post.allowed[msg.sender][_spender] ==
            allowed[msg.sender][_spender] - _delta
     */
```

Line 201-212 in File HMToken.sol

```
    function decreaseApproval(address _spender, uint256 _delta) public returns (bool
        success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        uint256 _oldValue = allowed[msg.sender][_spender];
        if (_delta > _oldValue) { // Truncate upon overflow.
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = allowed[msg.sender][_spender].sub(_delta);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 15

allowance

03, Jul 2019
4.49 ms

Line 214-216 in File HMToken.sol

```
    /*@CTK allowance
      @post remaining == allowed[_owner][_spender]
     */
```

Line 217-219 in File HMToken.sol

```
function allowance(address _owner, address _spender) public view returns (uint256
    remaining) {
    return allowed[_owner][_spender];
}
```

✓ The code meets the specification.

Formal Verification Request 16

balanceOf

03, Jul 2019
4.51 ms

Line 221-223 in File HMToken.sol

```
/*@CTK balanceOf
  @post balance == __post.balances[_owner]
 */
```

Line 224-226 in File HMToken.sol

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

✓ The code meets the specification.

Formal Verification Request 17

transferQuiet

03, Jul 2019
48.93 ms

Line 229-238 in File HMToken.sol

```
/*@CTK transferQuiet
  @pre msg.sender != _to
  @pre _to != address(0)
  @pre _to != address(this)
  @pre balances[msg.sender] >= _value
  @pre (balances[_to] + _value) >= balances[_to]
  @post __post.balances[msg.sender] == balances[msg.sender] - _value
  @post __post.balances[_to] == balances[_to] + _value
  @post success
 */
```

Line 239-250 in File HMToken.sol

```
function transferQuiet(address _to, uint256 _value) internal returns (bool success
    ) {
    if (_to == address(0)) return false; // Preclude burning tokens to
        uninitialized address.
    if (_to == address(this)) return false; // Preclude sending tokens to the
        contract.
        if (balances[msg.sender] < _value) return false; // Preclude transfering more
            than sender's balance.
        if (balances[_to] + _value < balances[_to]) return false; // Handle overflow
            here in order to avoid reverts from SafeMath.

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);

        emit Transfer(msg.sender, _to, _value);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 18

transferBulk_bulkValue_sum__Generated

📅 03, Jul 2019
⏱ 48.11 ms

(Loop) Line 65-69 in File HMToken.sol

```
        /*@CTK transferBulk_bulkValue_sum
          @inv j <= _tos.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
```

(Loop) Line 65-72 in File HMToken.sol

```
        /*@CTK transferBulk_bulkValue_sum
          @inv j <= _tos.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _tos.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
```

✓ The code meets the specification.

Formal Verification Request 19

transferBulk_transfer__Generated

📅 03, Jul 2019
⏱ 159.61 ms

(Loop) Line 77-81 in File HMToken.sol

```
        /*@CTK transferBulk_transfer
          @inv i <= _tos.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
```

(Loop) Line 77-89 in File HMToken.sol

```
        /*@CTK transferBulk_transfer
          @inv i <= _tos.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
        for (uint256 i = 0; i < _tos.length; ++i) {
            _success = transferQuiet(_tos[i], _values[i]);
            if (_success) {
                _bulkCount = _bulkCount.add(1);
            } else {
                emit BulkTransferFailure(_txId, _bulkCount);
            }
        }
```

✓ The code meets the specification.

Formal Verification Request 20

approveBulk_bulkValue_sum__Generated

03, Jul 2019
42.13 ms

(Loop) Line 143-147 in File HMToken.sol

```
        /*@CTK approveBulk_bulkValue_sum
          @inv j <= _spenders.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
```

(Loop) Line 143-150 in File HMToken.sol

```
        /*@CTK approveBulk_bulkValue_sum
          @inv j <= _spenders.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _spenders.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
```

✓ The code meets the specification.

Formal Verification Request 21

approveBulk_approve__Generated

03, Jul 2019
146.76 ms

(Loop) Line 155-159 in File HMToken.sol

```
        /*@CTK approveBulk_approve
          @inv i <= _spenders.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
```

(Loop) Line 155-167 in File HMToken.sol

```
        /*@CTK approveBulk_approve
          @inv i <= _spenders.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
        for (uint256 i = 0; i < _spenders.length; ++i) {
            _success = increaseApproval(_spenders[i], _values[i]);
            if (_success) {
                _bulkCount = _bulkCount.add(1);
            } else {
                emit BulkApprovalFailure(_txId, _bulkCount);
            }
        }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File SafeMath.sol

```
pragma solidity 0.5.9;


/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    /*@CTK "SafeMath add"
      @tag spec
      @tag is_pure
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    /*@CTK "SafeMath sub"
      @tag spec
      @tag is_pure
      @post (b > a) == __reverted
      @post !__reverted -> __return == a - b
      @post !__reverted -> !__has_overflow
      @post !__reverted -> !__has_assertion_failure
      @post !(__has_buf_overflow)
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot 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, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be 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) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
         modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    /*@CTK "SafeMath mod"
      @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 mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
```

File HMToken.sol

```
pragma solidity 0.5.9;

/*
Implements EIP20 token standard: https://github.com/ethereum/EIPs/issues/20
.*/

import "./SafeMath.sol";
import "./HMTokenInterface.sol";


contract HMToken is HMTokenInterface {
    using SafeMath for uint256;
    uint256 private constant MAX_UINT256 = 2**256 - 1;
    uint256 private constant BULK_MAX_VALUE = 1000000000 * (10 ** 18);
    uint32 private constant BULK_MAX_COUNT = 100;

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

    string public name;
    uint8 public decimals;
    string public symbol;

    /*@CTK HMToken
      @tag assume_completion
      @post __post.name == _name
      @post __post.decimals == _decimals
      @post __post.symbol == _symbol
      @post __post.balances[msg.sender] == __post.totalSupply
     */
    constructor(uint256 _totalSupply, string memory _name, uint8 _decimals, string
        memory _symbol) public {
        totalSupply = _totalSupply * (10 ** uint256(_decimals));
        name = _name;
        decimals = _decimals;
        symbol = _symbol;
        balances[msg.sender] = totalSupply;
    }

    /*@CTK transfer
      @tag assume_completion
      @pre msg.sender != _to
      @post _to != address(0)
      @post _to != address(this)
      @post balances[msg.sender] >= _value
      @post (balances[_to] + _value) >= balances[_to]
      @post __post.balances[msg.sender] == balances[msg.sender] - _value
      @post __post.balances[_to] == balances[_to] + _value
     */
    function transfer(address _to, uint256 _value) public returns (bool success) {
        success = transferQuiet(_to, _value);
        require(success, "Transfer didn't succeed");
        return success;
    }

    /*@CTK transferBulk
      @tag assume_completion
      @pre _tos.length == _values.length
      @pre _tos.length < BULK_MAX_COUNT
     */
     function transferBulk(address[] memory _tos, uint256[] memory _values, uint256
         _txId) public returns (uint256 _bulkCount) {
        require(_tos.length == _values.length, "Amount of recipients and values don't
            match");
        require(_tos.length < BULK_MAX_COUNT, "Too many recipients");

        uint256 _bulkValue = 0;
        /*@CTK transferBulk_bulkValue_sum
          @inv j <= _tos.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _tos.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
        require(_bulkValue < BULK_MAX_VALUE, "Bulk value too high");

        _bulkCount = 0;
        bool _success;
        /*@CTK transferBulk_transfer
          @inv i <= _tos.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
        for (uint256 i = 0; i < _tos.length; ++i) {
            _success = transferQuiet(_tos[i], _values[i]);
            if (_success) {
                _bulkCount = _bulkCount.add(1);
            } else {
                emit BulkTransferFailure(_txId, _bulkCount);
            }
        }
        emit BulkTransfer(_txId, _bulkCount);
        return _bulkCount;
    }

    /*@CTK transferFrom
      @tag assume_completion
      @pre _spender != _to
      @post balances[_spender] >= _value && allowed[_spender][msg.sender] >= _value
      @post _to != address(0)
      @post __post.balances[_spender] == balances[_spender] - _value
      @post __post.balances[_to] == balances[_to] + _value
      @post (allowed[_spender][msg.sender] < MAX_UINT256) -> (__post.allowed[_spender
          ][msg.sender] == allowed[_spender][msg.sender] - _value)
      @post (allowed[_spender][msg.sender] >= MAX_UINT256) -> (__post.allowed[_spender
          ][msg.sender] == allowed[_spender][msg.sender])
     */
    function transferFrom(address _spender, address _to, uint256 _value) public
        returns (bool success) {
        uint256 _allowance = allowed[_spender][msg.sender];
        require(balances[_spender] >= _value && _allowance >= _value, "Spender balance
            or allowance too low");
        require(_to != address(0), "Can't send tokens to uninitialized address");

        balances[_spender] = balances[_spender].sub(_value);
        balances[_to] = balances[_to].add(_value);

        if (_allowance < MAX_UINT256) {
            allowed[_spender][msg.sender] = allowed[_spender][msg.sender].sub(_value);
        }

        emit Transfer(_spender, _to, _value);
        return true;
    }

    /*@CTK approve
      @tag assume_completion
      @post _spender != address(0)
      @post __post.allowed[msg.sender][_spender] == _value
     */
    function approve(address _spender, uint256 _value) public returns (bool success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        allowed[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value); //solhint-disable-line indent, no-
            unused-vars
        return true;
    }

    /*@CTK approveBulk
      @tag assume_completion
      @pre _spenders.length == _values.length
      @pre _spenders.length < BULK_MAX_COUNT
     */
     function approveBulk(address[] memory _spenders, uint256[] memory _values,
         uint256 _txId) public returns (uint256 _bulkCount) {
        require(_spenders.length == _values.length, "Amount of spenders and values don’
            t match");
        require(_spenders.length < BULK_MAX_COUNT, "Too many spenders");

        uint256 _bulkValue = 0;
        /*@CTK approveBulk_bulkValue_sum
          @inv j <= _spenders.length
          @inv _bulkValue >= _bulkValue__pre
          @post !__should_return
         */
        for (uint256 j = 0; j < _spenders.length; ++j) {
            _bulkValue = _bulkValue.add(_values[j]);
        }
        require(_bulkValue < BULK_MAX_VALUE, "Bulk value too high");

        _bulkCount = 0;
        bool _success;
        /*@CTK approveBulk_approve
          @inv i <= _spenders.length
          @inv _bulkCount >= _bulkCount__pre
          @post !__should_return
         */
        for (uint256 i = 0; i < _spenders.length; ++i) {
            _success = increaseApproval(_spenders[i], _values[i]);
            if (_success) {
                _bulkCount = _bulkCount.add(1);
            } else {
                emit BulkApprovalFailure(_txId, _bulkCount);
            }
        }
        emit BulkApproval(_txId, _bulkCount);
        return _bulkCount;
    }

    /*@CTK increaseApproval
      @tag assume_completion
      @pre allowed[msg.sender][_spender] + _delta < MAX_UINT256
      @post _spender != address(0)
      @post __post.allowed[msg.sender][_spender] ==
            allowed[msg.sender][_spender] + _delta
     */
    function increaseApproval(address _spender, uint256 _delta) public returns (bool
        success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        uint256 _oldValue = allowed[msg.sender][_spender];
        if (_oldValue.add(_delta) < _oldValue || _oldValue.add(_delta) >= MAX_UINT256)
            { // Truncate upon overflow.
            allowed[msg.sender][_spender] = MAX_UINT256.sub(1);
        } else {
            allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_delta);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    /*@CTK decreaseApproval
      @tag assume_completion
      @post _spender != address(0)
      @post _delta > allowed[msg.sender][_spender] ->
            __post.allowed[msg.sender][_spender] == 0
      @post _delta <= allowed[msg.sender][_spender]
            -> __post.allowed[msg.sender][_spender] ==
            allowed[msg.sender][_spender] - _delta
     */
    function decreaseApproval(address _spender, uint256 _delta) public returns (bool
        success) {
        require(_spender != address(0), "Token spender is an uninitialized address");

        uint256 _oldValue = allowed[msg.sender][_spender];
        if (_delta > _oldValue) { // Truncate upon overflow.
            allowed[msg.sender][_spender] = 0;
        } else {
            allowed[msg.sender][_spender] = allowed[msg.sender][_spender].sub(_delta);
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
        return true;
    }

    /*@CTK allowance
      @post remaining == allowed[_owner][_spender]
     */
    function allowance(address _owner, address _spender) public view returns (uint256
        remaining) {
        return allowed[_owner][_spender];
    }

    /*@CTK balanceOf
      @post balance == __post.balances[_owner]
     */
    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }

    // Like transfer, but fails quietly.
    /*@CTK transferQuiet
      @pre msg.sender != _to
      @pre _to != address(0)
      @pre _to != address(this)
      @pre balances[msg.sender] >= _value
      @pre (balances[_to] + _value) >= balances[_to]
      @post __post.balances[msg.sender] == balances[msg.sender] - _value
      @post __post.balances[_to] == balances[_to] + _value
      @post success
     */
    function transferQuiet(address _to, uint256 _value) internal returns (bool success
        ) {
        if (_to == address(0)) return false; // Preclude burning tokens to
            uninitialized address.
        if (_to == address(this)) return false; // Preclude sending tokens to the
            contract.
        if (balances[msg.sender] < _value) return false; // Preclude transfering more
            than sender's balance.
        if (balances[_to] + _value < balances[_to]) return false; // Handle overflow
            here in order to avoid reverts from SafeMath.

        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);

        emit Transfer(msg.sender, _to, _value);
        return true;
    }
}
```