CertiK Verification Report For UPT

Request Date: 2019-02-05
Revision Date: 2019-02-11

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 UPT(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 CertiKs prior written consent.

Summary

This is the report for smart contract verification service requestd by UPT. The goal of the audition is to guarantee that verified smart contracts are robust enough to avoid potentially unexpected 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 audit time.

Type of Issues

CertiK smart label engine applied 100% coveraged formal verification labels on the source code, and scanned the code by 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Insecure Compiler Version	Using an fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0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.

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 UniversalProtocolToken.sol

```
pragma solidity ^0.5.0;

import "openzeppelin-solidity/contracts/token/ERC20/ERC20.sol";
import "openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol";

/**
 * @title UniversalProtocolToken
 */

contract UniversalProtocolToken is ERC20, ERC20Detailed {
  uint8 public constant DECIMALS = 18;
  uint256 public constant INITIAL_SUPPLY = (10 ** 10) * (10 ** uint256(DECIMALS));

  /**
   * @dev Constructor that gives beneficiary all of existing tokens.
   */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  constructor (address beneficiary) public ERC20Detailed("Universal Protocol Token", "
      UPT", DECIMALS) {
    _mint(beneficiary, INITIAL_SUPPLY);
  }
}
```

File openzeppelin-solidity/contracts/math/SafeMath.sol

```
pragma solidity ^0.4.24;

/**
 * @title SafeMath
 * @dev Math operations with safety checks that revert on error
 */
library SafeMath {

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

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

```

  return c;
}

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

  return c;
}

/**
* @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater
    than minuend).
*/
/*@CTK "SafeMath sub"
  @post (a < b) == __reverted
  @post !__reverted -> __return == a - b
  @post !__reverted -> !__has_overflow
  @post !(__has_buf_overflow)
  @post !(__has_assertion_failure)
*/
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b <= a);
  uint256 c = a - b;

  return c;
}

/**
* @dev Adds two numbers, reverts on overflow.
*/
/*@CTK "SafeMath add"
  @post (a + b < a || a + b < b) == __reverted
  @post !__reverted -> __return == a + b
  @post !__reverted -> !__has_overflow
  @post !(__has_buf_overflow)
  @post !(__has_assertion_failure)
*/
function add(uint256 a, uint256 b) internal pure returns (uint256) {
  uint256 c = a + b;
  require(c >= a);

  return c;
}

```

```
  /**
  * @dev Divides two numbers and returns the remainder (unsigned integer modulo),
  * reverts when dividing by zero.
  */
  /*@CTK "SafeMath div"
    @post b != 0 -> !__reverted
    @post !__reverted -> __return == a % b
    @post !__reverted -> !__has_overflow
    @post !(__has_buf_overflow)
    @post !(__has_assertion_failure)
  */
  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b != 0);
    return a % b;
  }
}
```

File openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

```
pragma solidity ^0.4.24;

import "./IERC20.sol";
import "../../math/SafeMath.sol";

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/
     master/smart_contract/FirstBloodToken.sol
 */
contract ERC20 is IERC20 {
  using SafeMath for uint256;

  mapping (address => uint256) private _balances;

  mapping (address => mapping (address => uint256)) private _allowed;

  uint256 private _totalSupply;

  /**
  * @dev Total number of tokens in existence
  */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "totalSupply correctness"
    @post __return == _totalSupply
   */
  function totalSupply() public view returns (uint256) {
    return _totalSupply;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param owner The address to query the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
```

```
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "balanceOf correctness"
  @post __return == _balances[owner]
 */
function balanceOf(address owner) public view returns (uint256) {
  return _balances[owner];
}

/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param owner address The address which owns the funds.
 * @param spender address The address which will spend the funds.
 * @return A uint256 specifying the amount of tokens still available for the spender.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "allowance correctness"
  @post __return == _allowed[owner][spender]
 */
function allowance(
  address owner,
  address spender
 )
  public
  view
  returns (uint256)
{
  return _allowed[owner][spender];
}

/**
* @dev Transfer token for a specified address
* @param to The address to transfer to.
* @param value The amount to be transferred.
*/
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "transfer correctness"
  @tag assume_completion
  @post to != 0x0
  @post value <= _balances[msg.sender]
  @post to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender] - value
  @post to != msg.sender -> __post._balances[to] == _balances[to] + value
  @post to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
 */
function transfer(address to, uint256 value) public returns (bool) {
  _transfer(msg.sender, to, value);
  return true;
}

/**
 * @dev Approve the passed address to spend the specified amount of tokens on behalf
```

```
      of msg.sender.
 * Beware that changing an allowance with this method brings the risk that someone may use both the old
 * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
 * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 * @param spender The address which will spend the funds.
 * @param value The amount of tokens to be spent.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "approve correctness"
  @post spender == 0x0 -> __reverted
  @post spender != 0x0 -> __post._allowed[msg.sender][spender] == value
*/
function approve(address spender, uint256 value) public returns (bool) {
  require(spender != address(0));

  _allowed[msg.sender][spender] = value;
  emit Approval(msg.sender, spender, value);
  return true;
}

/**
 * @dev Transfer tokens from one address to another
 * @param from address The address which you want to send tokens from
 * @param to address The address which you want to transfer to
 * @param value uint256 the amount of tokens to be transferred
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "transferFrom correctness"
  @tag assume_completion
  @post to != 0x0
  @post value <= _balances[from] && value <= _allowed[from][msg.sender]
  @post to != from -> __post._balances[from] == _balances[from] - value
  @post to != from -> __post._balances[to] == _balances[to] + value
  @post to == from -> __post._balances[from] == _balances[from]
  @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
 */
function transferFrom(
  address from,
  address to,
  uint256 value
)
  public
  returns (bool)
{
  require(value <= _allowed[from][msg.sender]);

  _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
  _transfer(from, to, value);
  return true;
}
```

```

/**
 * @dev Increase the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To increment
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param spender The address which will spend the funds.
 * @param addedValue The amount of tokens to increase the allowance by.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "increaseAllowance correctness"
  @tag assume_completion
  @post spender != 0x0
  @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] +
      addedValue
*/
function increaseAllowance(
  address spender,
  uint256 addedValue
)
  public
  returns (bool)
{
  require(spender != address(0));

  _allowed[msg.sender][spender] = (
    _allowed[msg.sender][spender].add(addedValue));
  emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
  return true;
}

/**
 * @dev Decrease the amount of tokens that an owner allowed to a spender.
 * approve should be called when allowed_[_spender] == 0. To decrement
 * allowed value is better to use this function to avoid 2 calls (and wait until
 * the first transaction is mined)
 * From MonolithDAO Token.sol
 * @param spender The address which will spend the funds.
 * @param subtractedValue The amount of tokens to decrease the allowance by.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "decreaseAllowance correctness"
  @tag assume_completion
  @post spender != 0x0
  @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] -
      subtractedValue
*/
function decreaseAllowance(
  address spender,
  uint256 subtractedValue
)
  public
  returns (bool)
```

```
{
  require(spender != address(0));

  _allowed[msg.sender][spender] = (
    _allowed[msg.sender][spender].sub(subtractedValue));
  emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
  return true;
}

/**
* @dev Transfer token for a specified addresses
* @param from The address to transfer from.
* @param to The address to transfer to.
* @param value The amount to be transferred.
*/
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "_transfer correctness"
  @tag assume_completion
  @post to != 0x0
  @post value <= _balances[from]
  @post to != from -> __post._balances[from] == _balances[from] - value
  @post to != from -> __post._balances[to] == _balances[to] + value
  @post to == from -> __post._balances[from] == _balances[from]
 */
function _transfer(address from, address to, uint256 value) internal {
  require(value <= _balances[from]);
  require(to != address(0));

  _balances[from] = _balances[from].sub(value);
  _balances[to] = _balances[to].add(value);
  emit Transfer(from, to, value);
}

/**
 * @dev Internal function that mints an amount of the token and assigns it to
 * an account. This encapsulates the modification of balances such that the
 * proper events are emitted.
 * @param account The account that will receive the created tokens.
 * @param value The amount that will be created.
 */
//@CTK NO_OVERFLOW
//@CTK NO_BUF_OVERFLOW
//@CTK NO_ASF
/*@CTK "_mint correctness"
  @tag assume_completion
  @post account != 0x0
  @post __post._balances[account] == _balances[account] + value
  @post __post._totalSupply == _totalSupply + value
 */
function _mint(address account, uint256 value) internal {
  require(account != 0);
  _totalSupply = _totalSupply.add(value);
  _balances[account] = _balances[account].add(value);
  emit Transfer(address(0), account, value);
}

```

```
  /**
   * @dev Internal function that burns an amount of the token of a given
   * account.
   * @param account The account whose tokens will be burnt.
   * @param value The amount that will be burnt.
   */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "_burn correctness"
    @tag assume_completion
    @post account != 0x0
    @post value <= _balances[account]
    @post __post._balances[account] == _balances[account] - value
    @post __post._totalSupply == _totalSupply - value
   */
  function _burn(address account, uint256 value) internal {
    require(account != 0);
    require(value <= _balances[account]);

    _totalSupply = _totalSupply.sub(value);
    _balances[account] = _balances[account].sub(value);
    emit Transfer(account, address(0), value);
  }

  /**
   * @dev Internal function that burns an amount of the token of a given
   * account, deducting from the sender's allowance for said account. Uses the
   * internal burn function.
   * @param account The account whose tokens will be burnt.
   * @param value The amount that will be burnt.
   */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "_burnFrom correctness"
    @tag assume_completion
    @post account != 0x0
    @post value <= _balances[account] && value <= _allowed[account][msg.sender]
    @post __post._balances[account] == _balances[account] - value
    @post __post._totalSupply == _totalSupply - value
    @post __post._allowed[account][msg.sender] == _allowed[account][msg.sender] -
        value
   */
  function _burnFrom(address account, uint256 value) internal {
    require(value <= _allowed[account][msg.sender]);

    // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted
        ,
    // this function needs to emit an event with the updated approval.
    _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
      value);
    _burn(account, value);
  }
}
```

File openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol

```
pragma solidity ^0.4.24;
```

```

import "./IERC20.sol";

/**
 * @title ERC20Detailed token
 * @dev The decimals are only for visualization purposes.
 * All the operations are done using the smallest and indivisible token unit,
 * just as on Ethereum all the operations are done in wei.
 */
contract ERC20Detailed is IERC20 {
  string private _name;
  string private _symbol;
  uint8 private _decimals;

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ERC20Detailed constructor correctness"
    @post __post._name == name
    @post __post._symbol == symbol
    @post __post._decimals == decimals
   */
  constructor(string name, string symbol, uint8 decimals) public {
    _name = name;
    _symbol = symbol;
    _decimals = decimals;
  }

  /**
   * @return the name of the token.
   */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ERC20Detailed name correctness"
    @post __return == _name
   */
  function name() public view returns(string) {
    return _name;
  }

  /**
   * @return the symbol of the token.
   */
  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ERC20Detailed symbol correctness"
    @post __return == _symbol
   */
  function symbol() public view returns(string) {
    return _symbol;
  }

  //@CTK NO_OVERFLOW
  //@CTK NO_BUF_OVERFLOW
  //@CTK NO_ASF
  /*@CTK "ERC20Detailed decimals correctness"
```

```
    @post __return == _decimals
   */
  /**
   * @return the number of decimals of the token.
   */
  function decimals() public view returns(uint8) {
    return _decimals;
  }
}
```

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

INSECURE_COMPILER_VERSION

Line 1 in File UniversalProtocolToken.sol

```
1 pragma solidity ^0.5.0;
```

Only these compiler versions are safe to compile your code: 0.5.0, 0.5.1, 0.5.2, 0.5.3

INSECURE_COMPILER_VERSION

Line 1 in File SafeMath.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

INSECURE_COMPILER_VERSION

Line 1 in File ERC20.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

INSECURE_COMPILER_VERSION

Line 1 in File ERC20Detailed.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

Formal Verification Request 1

If method completes, integer overflow would not happen.

11, Feb 2019
255.15 ms

Line 17 in File UniversalProtocolToken.sol

```
17 //@CTK NO_OVERFLOW
```

Line 20-22 in File UniversalProtocolToken.sol

```
20 constructor (address beneficiary) public ERC20Detailed("Universal Protocol Token", "
       UPT", DECIMALS) {
21   _mint(beneficiary, INITIAL_SUPPLY);
22 }
```

The code meets the specification

Formal Verification Request 2

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

11, Feb 2019
7.86 ms

Line 18 in File UniversalProtocolToken.sol

```
18 //@CTK NO_BUF_OVERFLOW
```

Line 20-22 in File UniversalProtocolToken.sol

```
20 constructor (address beneficiary) public ERC20Detailed("Universal Protocol Token", "
       UPT", DECIMALS) {
21   _mint(beneficiary, INITIAL_SUPPLY);
22 }
```

The code meets the specification

Formal Verification Request 3

Method will not encounter an assertion failure.

11, Feb 2019
6.95 ms

Line 19 in File UniversalProtocolToken.sol

```
19 //@CTK NO_ASF
```

Line 20-22 in File UniversalProtocolToken.sol

```
20 constructor (address beneficiary) public ERC20Detailed("Universal Protocol Token", "
       UPT", DECIMALS) {
21   _mint(beneficiary, INITIAL_SUPPLY);
22 }
```

The code meets the specification

Formal Verification Request 4

SafeMath mul

11, Feb 2019
429.79 ms

Line 12-18 in File SafeMath.sol

```
12 /*@CTK "SafeMath mul"
13   @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
14   @post !__reverted -> __return == a * b
15   @post !__reverted == !__has_overflow
16   @post !(__has_buf_overflow)
17   @post !(__has_assertion_failure)
18   */
```

Line 19-31 in File SafeMath.sol

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

The code meets the specification

Formal Verification Request 5

SafeMath div

11, Feb 2019
17.13 ms

Line 36-42 in File SafeMath.sol

```
36 /*@CTK "SafeMath div"
37   @post b != 0 -> !__reverted
38   @post !__reverted -> __return == a / b
39   @post !__reverted -> !__has_overflow
40   @post !(__has_buf_overflow)
41   @post !(__has_assertion_failure)
42 */
```

Line 43-49 in File SafeMath.sol

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

  return c;
}
```

The code meets the specification

Formal Verification Request 6

SafeMath sub

11, Feb 2019
17.4 ms

Line 54-60 in File SafeMath.sol

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

Line 61-66 in File SafeMath.sol

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
  require(b <= a);
  uint256 c = a - b;

  return c;
}
```

The code meets the specification

Formal Verification Request 7

SafeMath add

11, Feb 2019
22.02 ms

Line 71-77 in File SafeMath.sol

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

Line 78-83 in File SafeMath.sol

```
78 function add(uint256 a, uint256 b) internal pure returns (uint256) {
79   uint256 c = a + b;
80   require(c >= a);
81
82   return c;
83 }
```

The code meets the specification

Formal Verification Request 8

SafeMath div

11, Feb 2019
15.42 ms

Line 89-95 in File SafeMath.sol

```
89 /*@CTK "SafeMath div"
90   @post b != 0 -> !__reverted
91   @post !__reverted -> __return == a % b
92   @post !__reverted -> !__has_overflow
93   @post !(__has_buf_overflow)
94   @post !(__has_assertion_failure)
95 */
```

Line 96-99 in File SafeMath.sol

```
96 function mod(uint256 a, uint256 b) internal pure returns (uint256) {
97   require(b != 0);
98   return a % b;
99 }
```

The code meets the specification

Formal Verification Request 9

If method completes, integer overflow would not happen.

11, Feb 2019
6.83 ms

Line 25 in File ERC20.sol

```
25 //@CTK NO_OVERFLOW
```

Line 31-33 in File ERC20.sol

```
31 function totalSupply() public view returns (uint256) {
32   return _totalSupply;
33 }
```

The code meets the specification

Formal Verification Request 10

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

11, Feb 2019
0.43 ms

Line 26 in File ERC20.sol

```
26 //@CTK NO_BUF_OVERFLOW
```

Line 31-33 in File ERC20.sol

```
31 function totalSupply() public view returns (uint256) {
32   return _totalSupply;
33 }
```

The code meets the specification

Formal Verification Request 11

Method will not encounter an assertion failure.

11, Feb 2019
0.43 ms

Line 27 in File ERC20.sol

```
27 //@CTK NO_ASF
```

Line 31-33 in File ERC20.sol

```
31 function totalSupply() public view returns (uint256) {
32   return _totalSupply;
33 }
```

The code meets the specification

Formal Verification Request 12

totalSupply correctness

11, Feb 2019
0.46 ms

Line 28-30 in File ERC20.sol

```
28 /*@CTK "totalSupply correctness"
29   @post __return == _totalSupply
30  */
```

Line 31-33 in File ERC20.sol

```
31 function totalSupply() public view returns (uint256) {
32   return _totalSupply;
33 }
```

The code meets the specification

Formal Verification Request 13

If method completes, integer overflow would not happen.

11, Feb 2019
10.44 ms

Line 40 in File ERC20.sol

```
40 //@CTK NO_OVERFLOW
```

Line 46-48 in File ERC20.sol

```
46 function balanceOf(address owner) public view returns (uint256) {
47   return _balances[owner];
48 }
```

The code meets the specification

Formal Verification Request 14

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

11, Feb 2019
0.52 ms

Line 41 in File ERC20.sol

```
41 //@CTK NO_BUF_OVERFLOW
```

Line 46-48 in File ERC20.sol

```
46 function balanceOf(address owner) public view returns (uint256) {
47   return _balances[owner];
48 }
```

The code meets the specification

Formal Verification Request 15

Method will not encounter an assertion failure.

11, Feb 2019
0.72 ms

Line 42 in File ERC20.sol

```
42 //@CTK NO_ASF
```

Line 46-48 in File ERC20.sol

```
46 function balanceOf(address owner) public view returns (uint256) {
47   return _balances[owner];
48 }
```

The code meets the specification

Formal Verification Request 16

balanceOf correctness

11, Feb 2019
0.47 ms

Line 43-45 in File ERC20.sol

```
43 /*@CTK "balanceOf correctness"
44   @post __return == _balances[owner]
45  */
```

Line 46-48 in File ERC20.sol

```
46 function balanceOf(address owner) public view returns (uint256) {
47   return _balances[owner];
48 }
```

The code meets the specification

Formal Verification Request 17

If method completes, integer overflow would not happen.

11, Feb 2019
8.59 ms

Line 56 in File ERC20.sol

```
56 //@CTK NO_OVERFLOW
```

Line 62-71 in File ERC20.sol

```
62 function allowance(
63   address owner,
64   address spender
65  )
66   public
67   view
68   returns (uint256)
69 {
70   return _allowed[owner][spender];
71 }
```

The code meets the specification

Formal Verification Request 18

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

11, Feb 2019
0.53 ms

Line 57 in File ERC20.sol

```
57 //@CTK NO_BUF_OVERFLOW
```

Line 62-71 in File ERC20.sol

```
62 function allowance(
63   address owner,
64   address spender
65  )
66   public
67   view
68   returns (uint256)
69 {
70   return _allowed[owner][spender];
71 }
```

The code meets the specification

Formal Verification Request 19

Method will not encounter an assertion failure.

11, Feb 2019
0.45 ms

Line 58 in File ERC20.sol

```
58 //@CTK NO_ASF
```

Line 62-71 in File ERC20.sol

```
62 function allowance(
63   address owner,
64   address spender
65  )
66   public
67   view
68   returns (uint256)
69 {
70   return _allowed[owner][spender];
71 }
```

The code meets the specification

Formal Verification Request 20

allowance correctness

11, Feb 2019
0.44 ms

Line 59-61 in File ERC20.sol

```
59 /*@CTK "allowance correctness"
60   @post __return == _allowed[owner][spender]
61  */
```

Line 62-71 in File ERC20.sol

```
62 function allowance(
63   address owner,
64   address spender
65  )
66   public
67   view
68   returns (uint256)
69 {
70   return _allowed[owner][spender];
71 }
```

The code meets the specification

Formal Verification Request 21

If method completes, integer overflow would not happen.

11, Feb 2019
219.41 ms

Line 78 in File ERC20.sol

```
78 //@CTK NO_OVERFLOW
```

Line 89-92 in File ERC20.sol

```
89 function transfer(address to, uint256 value) public returns (bool) {
90   _transfer(msg.sender, to, value);
91   return true;
92 }
```

The code meets the specification

Formal Verification Request 22

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

11, Feb 2019
12.2 ms

Line 79 in File ERC20.sol

```
79 //@CTK NO_BUF_OVERFLOW
```

Line 89-92 in File ERC20.sol

```
89 function transfer(address to, uint256 value) public returns (bool) {
90   _transfer(msg.sender, to, value);
91   return true;
92 }
```

The code meets the specification

Formal Verification Request 23

Method will not encounter an assertion failure.

11, Feb 2019
14.19 ms

Line 80 in File ERC20.sol

```
80 //@CTK NO_ASF
```

Line 89-92 in File ERC20.sol

```
89 function transfer(address to, uint256 value) public returns (bool) {
90   _transfer(msg.sender, to, value);
91   return true;
92 }
```

The code meets the specification

Formal Verification Request 24

transfer correctness

11, Feb 2019
189.2 ms

Line 81-88 in File ERC20.sol

```
81 /*@CTK "transfer correctness"
82   @tag assume_completion
83   @post to != 0x0
84   @post value <= _balances[msg.sender]
85   @post to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender] -
       value
86   @post to != msg.sender -> __post._balances[to] == _balances[to] + value
87   @post to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
88  */
```

Line 89-92 in File ERC20.sol

```
89 function transfer(address to, uint256 value) public returns (bool) {
90   _transfer(msg.sender, to, value);
91   return true;
92 }
```

The code meets the specification

Formal Verification Request 25

If method completes, integer overflow would not happen.

11, Feb 2019
21.3 ms

Line 103 in File ERC20.sol

```
103 //@CTK NO_OVERFLOW
```

Line 110-116 in File ERC20.sol

```
110 function approve(address spender, uint256 value) public returns (bool) {
111   require(spender != address(0));
112
113   _allowed[msg.sender][spender] = value;
114   emit Approval(msg.sender, spender, value);
115   return true;
116 }
```

The code meets the specification

Formal Verification Request 26

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

11, Feb 2019
0.58 ms

Line 104 in File ERC20.sol

```
104 //@CTK NO_BUF_OVERFLOW
```

Line 110-116 in File ERC20.sol

```
110 function approve(address spender, uint256 value) public returns (bool) {
111   require(spender != address(0));
112
113   _allowed[msg.sender][spender] = value;
114   emit Approval(msg.sender, spender, value);
115   return true;
116 }
```

The code meets the specification

Formal Verification Request 27

Method will not encounter an assertion failure.

11, Feb 2019
0.54 ms

Line 105 in File ERC20.sol

```
105 //@CTK NO_ASF
```

Line 110-116 in File ERC20.sol

```
110 function approve(address spender, uint256 value) public returns (bool) {
111   require(spender != address(0));
112
113   _allowed[msg.sender][spender] = value;
114   emit Approval(msg.sender, spender, value);
115   return true;
116 }
```

The code meets the specification

Formal Verification Request 28

approve correctness

11, Feb 2019
1.97 ms

Line 106-109 in File ERC20.sol

```
106 /*@CTK "approve correctness"
107   @post spender == 0x0 -> __reverted
108   @post spender != 0x0 -> __post._allowed[msg.sender][spender] == value
109 */
```

Line 110-116 in File ERC20.sol

```
110 function approve(address spender, uint256 value) public returns (bool) {
111   require(spender != address(0));
112
113   _allowed[msg.sender][spender] = value;
114   emit Approval(msg.sender, spender, value);
115   return true;
116 }
```

The code meets the specification

Formal Verification Request 29

If method completes, integer overflow would not happen.

11, Feb 2019
155.81 ms

Line 124 in File ERC20.sol

```
124 //@CTK NO_OVERFLOW
```

Line 136-149 in File ERC20.sol

```
136 function transferFrom(
137   address from,
138   address to,
139   uint256 value
140 )
141   public
142   returns (bool)
143 {
144   require(value <= _allowed[from][msg.sender]);
145
146   _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
147   _transfer(from, to, value);
148   return true;
149 }
```

The code meets the specification

Formal Verification Request 30

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

11, Feb 2019
8.71 ms

Line 125 in File ERC20.sol

```
125 //@CTK NO_BUF_OVERFLOW
```

Line 136-149 in File ERC20.sol

```
136 function transferFrom(
137   address from,
138   address to,
139   uint256 value
140 )
141   public
142   returns (bool)
143 {
144   require(value <= _allowed[from][msg.sender]);
145
146   _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
147   _transfer(from, to, value);
148   return true;
149 }
```

The code meets the specification

Formal Verification Request 31

Method will not encounter an assertion failure.

11, Feb 2019
9.21 ms

Line 126 in File ERC20.sol

```
126 //@CTK NO_ASF
```

Line 136-149 in File ERC20.sol

```
136 function transferFrom(
137   address from,
138   address to,
139   uint256 value
140 )
141   public
142   returns (bool)
143 {
144   require(value <= _allowed[from][msg.sender]);
145
```

```
146     _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
147     _transfer(from, to, value);
148     return true;
149   }
```

The code meets the specification

Formal Verification Request 32

transferFrom correctness

11, Feb 2019
222.84 ms

Line 127-135 in File ERC20.sol

```
127   /*@CTK "transferFrom correctness"
128     @tag assume_completion
129     @post to != 0x0
130     @post value <= _balances[from] && value <= _allowed[from][msg.sender]
131     @post to != from -> __post._balances[from] == _balances[from] - value
132     @post to != from -> __post._balances[to] == _balances[to] + value
133     @post to == from -> __post._balances[from] == _balances[from]
134     @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
135    */
```

Line 136-149 in File ERC20.sol

```
136   function transferFrom(
137     address from,
138     address to,
139     uint256 value
140   )
141     public
142     returns (bool)
143   {
144     require(value <= _allowed[from][msg.sender]);
145
146     _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
147     _transfer(from, to, value);
148     return true;
149   }
```

The code meets the specification

Formal Verification Request 33

If method completes, integer overflow would not happen.

11, Feb 2019
56.42 ms

Line 160 in File ERC20.sol

```
160   //@CTK NO_OVERFLOW
```

Line 168-181 in File ERC20.sol

```
168 function increaseAllowance(
169   address spender,
170   uint256 addedValue
171 )
172   public
173   returns (bool)
174 {
175   require(spender != address(0));
176
177   _allowed[msg.sender][spender] = (
178     _allowed[msg.sender][spender].add(addedValue));
179   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
180   return true;
181 }
```

✅ The code meets the specification

Formal Verification Request 34

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

📅 11, Feb 2019
⏱ 0.87 ms

Line 161 in File ERC20.sol

```
161 //@CTK NO_BUF_OVERFLOW
```

Line 168-181 in File ERC20.sol

```
168 function increaseAllowance(
169   address spender,
170   uint256 addedValue
171 )
172   public
173   returns (bool)
174 {
175   require(spender != address(0));
176
177   _allowed[msg.sender][spender] = (
178     _allowed[msg.sender][spender].add(addedValue));
179   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
180   return true;
181 }
```

✅ The code meets the specification

Formal Verification Request 35

Method will not encounter an assertion failure.

📅 11, Feb 2019
⏱ 0.84 ms

Line 162 in File ERC20.sol

```
162 //@CTK NO_ASF
```

Line 168-181 in File ERC20.sol

```
168 function increaseAllowance(
169   address spender,
170   uint256 addedValue
171 )
172   public
173   returns (bool)
174 {
175   require(spender != address(0));
176
177   _allowed[msg.sender][spender] = (
178     _allowed[msg.sender][spender].add(addedValue));
179   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
180   return true;
181 }
```

The code meets the specification

Formal Verification Request 36

increaseAllowance correctness

11, Feb 2019
2.87 ms

Line 163-167 in File ERC20.sol

```
163 /*@CTK "increaseAllowance correctness"
164   @tag assume_completion
165   @post spender != 0x0
166   @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] +
        addedValue
167 */
```

Line 168-181 in File ERC20.sol

```
168 function increaseAllowance(
169   address spender,
170   uint256 addedValue
171 )
172   public
173   returns (bool)
174 {
175   require(spender != address(0));
176
177   _allowed[msg.sender][spender] = (
178     _allowed[msg.sender][spender].add(addedValue));
179   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
180   return true;
181 }
```

The code meets the specification

Formal Verification Request 37

If method completes, integer overflow would not happen.

11, Feb 2019
47.23 ms

Line 192 in File ERC20.sol

```
192 //@CTK NO_OVERFLOW
```

Line 200-213 in File ERC20.sol

```
200 function decreaseAllowance(
201   address spender,
202   uint256 subtractedValue
203 )
204   public
205   returns (bool)
206 {
207   require(spender != address(0));
208
209   _allowed[msg.sender][spender] = (
210     _allowed[msg.sender][spender].sub(subtractedValue));
211   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
212   return true;
213 }
```

The code meets the specification

Formal Verification Request 38

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

11, Feb 2019
0.99 ms

Line 193 in File ERC20.sol

```
193 //@CTK NO_BUF_OVERFLOW
```

Line 200-213 in File ERC20.sol

```
200 function decreaseAllowance(
201   address spender,
202   uint256 subtractedValue
203 )
204   public
205   returns (bool)
206 {
207   require(spender != address(0));
208
209   _allowed[msg.sender][spender] = (
210     _allowed[msg.sender][spender].sub(subtractedValue));
211   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
212   return true;
213 }
```

The code meets the specification

Formal Verification Request 39

Method will not encounter an assertion failure.

11, Feb 2019
0.82 ms

Line 194 in File ERC20.sol

```
194 //@CTK NO_ASF
```

Line 200-213 in File ERC20.sol

```
200 function decreaseAllowance(
201   address spender,
202   uint256 subtractedValue
203 )
204   public
205   returns (bool)
206 {
207   require(spender != address(0));
208
209   _allowed[msg.sender][spender] = (
210     _allowed[msg.sender][spender].sub(subtractedValue));
211   emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
212   return true;
213 }
```

The code meets the specification

Formal Verification Request 40

decreaseAllowance correctness

11, Feb 2019
3.09 ms

Line 195-199 in File ERC20.sol

```
195 /*@CTK "decreaseAllowance correctness"
196   @tag assume_completion
197   @post spender != 0x0
198   @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] -
        subtractedValue
199 */
```

Line 200-213 in File ERC20.sol

```
200 function decreaseAllowance(
201   address spender,
202   uint256 subtractedValue
203 )
204   public
```

```
  returns (bool)
{
  require(spender != address(0));

  _allowed[msg.sender][spender] = (
    _allowed[msg.sender][spender].sub(subtractedValue));
  emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
  return true;
}
```

The code meets the specification

Formal Verification Request 41

If method completes, integer overflow would not happen.

11, Feb 2019
10.39 ms

Line 221 in File ERC20.sol

```
221 //@CTK NO_OVERFLOW
```

Line 232-239 in File ERC20.sol

```
232 function _transfer(address from, address to, uint256 value) internal {
233   require(value <= _balances[from]);
234   require(to != address(0));
235
236   _balances[from] = _balances[from].sub(value);
237   _balances[to] = _balances[to].add(value);
238   emit Transfer(from, to, value);
239 }
```

The code meets the specification

Formal Verification Request 42

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

11, Feb 2019
8.84 ms

Line 222 in File ERC20.sol

```
222 //@CTK NO_BUF_OVERFLOW
```

Line 232-239 in File ERC20.sol

```
232 function _transfer(address from, address to, uint256 value) internal {
233   require(value <= _balances[from]);
234   require(to != address(0));
235
236   _balances[from] = _balances[from].sub(value);
237   _balances[to] = _balances[to].add(value);
238   emit Transfer(from, to, value);
239 }
```

The code meets the specification

Formal Verification Request 43

Method will not encounter an assertion failure.

11, Feb 2019
8.55 ms

Line 223 in File ERC20.sol

```
223 //@CTK NO_ASF
```

Line 232-239 in File ERC20.sol

```
232 function _transfer(address from, address to, uint256 value) internal {
233   require(value <= _balances[from]);
234   require(to != address(0));
235
236   _balances[from] = _balances[from].sub(value);
237   _balances[to] = _balances[to].add(value);
238   emit Transfer(from, to, value);
239 }
```

The code meets the specification

Formal Verification Request 44

_transfer correctness

11, Feb 2019
197.53 ms

Line 224-231 in File ERC20.sol

```
224 /*@CTK "_transfer correctness"
225   @tag assume_completion
226   @post to != 0x0
227   @post value <= _balances[from]
228   @post to != from -> __post._balances[from] == _balances[from] - value
229   @post to != from -> __post._balances[to] == _balances[to] + value
230   @post to == from -> __post._balances[from] == _balances[from]
231  */
```

Line 232-239 in File ERC20.sol

```
232 function _transfer(address from, address to, uint256 value) internal {
233   require(value <= _balances[from]);
234   require(to != address(0));
235
236   _balances[from] = _balances[from].sub(value);
237   _balances[to] = _balances[to].add(value);
238   emit Transfer(from, to, value);
239 }
```

The code meets the specification

Formal Verification Request 45

If method completes, integer overflow would not happen.

11, Feb 2019
98.62 ms

Line 248 in File ERC20.sol

```
248 //@CTK NO_OVERFLOW
```

Line 257-262 in File ERC20.sol

```
257 function _mint(address account, uint256 value) internal {
258   require(account != 0);
259   _totalSupply = _totalSupply.add(value);
260   _balances[account] = _balances[account].add(value);
261   emit Transfer(address(0), account, value);
262 }
```

The code meets the specification

Formal Verification Request 46

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

11, Feb 2019
4.87 ms

Line 249 in File ERC20.sol

```
249 //@CTK NO_BUF_OVERFLOW
```

Line 257-262 in File ERC20.sol

```
257 function _mint(address account, uint256 value) internal {
258   require(account != 0);
259   _totalSupply = _totalSupply.add(value);
260   _balances[account] = _balances[account].add(value);
261   emit Transfer(address(0), account, value);
262 }
```

The code meets the specification

Formal Verification Request 47

Method will not encounter an assertion failure.

11, Feb 2019
3.39 ms

Line 250 in File ERC20.sol

```
250 //@CTK NO_ASF
```

Line 257-262 in File ERC20.sol

```
257 function _mint(address account, uint256 value) internal {
258   require(account != 0);
259   _totalSupply = _totalSupply.add(value);
260   _balances[account] = _balances[account].add(value);
261   emit Transfer(address(0), account, value);
262 }
```

The code meets the specification

Formal Verification Request 48

_mint correctness

11, Feb 2019
28.52 ms

Line 251-256 in File ERC20.sol

```
251 /*@CTK "_mint correctness"
252   @tag assume_completion
253   @post account != 0x0
254   @post __post._balances[account] == _balances[account] + value
255   @post __post._totalSupply == _totalSupply + value
256  */
```

Line 257-262 in File ERC20.sol

```
257 function _mint(address account, uint256 value) internal {
258   require(account != 0);
259   _totalSupply = _totalSupply.add(value);
260   _balances[account] = _balances[account].add(value);
261   emit Transfer(address(0), account, value);
262 }
```

The code meets the specification

Formal Verification Request 49

If method completes, integer overflow would not happen.

11, Feb 2019
95.61 ms

Line 270 in File ERC20.sol

```
270 //@CTK NO_OVERFLOW
```

Line 280-287 in File ERC20.sol

```
280 function _burn(address account, uint256 value) internal {
281   require(account != 0);
282   require(value <= _balances[account]);
283
284   _totalSupply = _totalSupply.sub(value);
285   _balances[account] = _balances[account].sub(value);
286   emit Transfer(account, address(0), value);
287 }
```

The code meets the specification

Formal Verification Request 50

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

11, Feb 2019
7.37 ms

Line 271 in File ERC20.sol

```
271 //@CTK NO_BUF_OVERFLOW
```

Line 280-287 in File ERC20.sol

```
280 function _burn(address account, uint256 value) internal {
281   require(account != 0);
282   require(value <= _balances[account]);
283
284   _totalSupply = _totalSupply.sub(value);
285   _balances[account] = _balances[account].sub(value);
286   emit Transfer(account, address(0), value);
287 }
```

The code meets the specification

Formal Verification Request 51

Method will not encounter an assertion failure.

11, Feb 2019
7.07 ms

Line 272 in File ERC20.sol

```
272 //@CTK NO_ASF
```

Line 280-287 in File ERC20.sol

```
280 function _burn(address account, uint256 value) internal {
281   require(account != 0);
282   require(value <= _balances[account]);
283
284   _totalSupply = _totalSupply.sub(value);
285   _balances[account] = _balances[account].sub(value);
286   emit Transfer(account, address(0), value);
287 }
```

The code meets the specification

Formal Verification Request 52

_burn correctness

11, Feb 2019
162.14 ms

Line 273-279 in File ERC20.sol

```
/*@CTK "_burn correctness"
  @tag assume_completion
  @post account != 0x0
  @post value <= _balances[account]
  @post __post._balances[account] == _balances[account] - value
  @post __post._totalSupply == _totalSupply - value
 */
```

Line 280-287 in File ERC20.sol

```
function _burn(address account, uint256 value) internal {
  require(account != 0);
  require(value <= _balances[account]);

  _totalSupply = _totalSupply.sub(value);
  _balances[account] = _balances[account].sub(value);
  emit Transfer(account, address(0), value);
}
```

The code meets the specification

Formal Verification Request 53

If method completes, integer overflow would not happen.

11, Feb 2019
140.72 ms

Line 296 in File ERC20.sol

```
//@CTK NO_OVERFLOW
```

Line 307-315 in File ERC20.sol

```
function _burnFrom(address account, uint256 value) internal {
  require(value <= _allowed[account][msg.sender]);

  // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted
      ,
  // this function needs to emit an event with the updated approval.
  _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
    value);
  _burn(account, value);
}
```

The code meets the specification

Formal Verification Request 54

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

11, Feb 2019
8.01 ms

Line 297 in File ERC20.sol

```
297 //@CTK NO_BUF_OVERFLOW
```

Line 307-315 in File ERC20.sol

```
307 function _burnFrom(address account, uint256 value) internal {
308   require(value <= _allowed[account][msg.sender]);
309
310   // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted
      ,
311   // this function needs to emit an event with the updated approval.
312   _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
313     value);
314   _burn(account, value);
315 }
```

The code meets the specification

Formal Verification Request 55

Method will not encounter an assertion failure.

11, Feb 2019
7.83 ms

Line 298 in File ERC20.sol

```
298 //@CTK NO_ASF
```

Line 307-315 in File ERC20.sol

```
307 function _burnFrom(address account, uint256 value) internal {
308   require(value <= _allowed[account][msg.sender]);
309
310   // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted
      ,
311   // this function needs to emit an event with the updated approval.
312   _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
313     value);
314   _burn(account, value);
315 }
```

The code meets the specification

Formal Verification Request 56

_burnFrom correctness

11, Feb 2019
278.25 ms

Line 299-306 in File ERC20.sol

```
/*@CTK "_burnFrom correctness"
  @tag assume_completion
  @post account != 0x0
  @post value <= _balances[account] && value <= _allowed[account][msg.sender]
  @post __post._balances[account] == _balances[account] - value
  @post __post._totalSupply == _totalSupply - value
  @post __post._allowed[account][msg.sender] == _allowed[account][msg.sender] -
      value
 */
```

Line 307-315 in File ERC20.sol

```
function _burnFrom(address account, uint256 value) internal {
  require(value <= _allowed[account][msg.sender]);

  // Should https://github.com/OpenZeppelin/zeppelin-solidity/issues/707 be accepted
      ,
  // this function needs to emit an event with the updated approval.
  _allowed[account][msg.sender] = _allowed[account][msg.sender].sub(
    value);
  _burn(account, value);
}
```

The code meets the specification

Formal Verification Request 57

If method completes, integer overflow would not happen.

11, Feb 2019
12.16 ms

Line 16 in File ERC20Detailed.sol

```
//@CTK NO_OVERFLOW
```

Line 24-28 in File ERC20Detailed.sol

```
constructor(string name, string symbol, uint8 decimals) public {
  _name = name;
  _symbol = symbol;
  _decimals = decimals;
}
```

The code meets the specification

Formal Verification Request 58

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

11, Feb 2019
0.45 ms

Line 17 in File ERC20Detailed.sol

```
17 //@CTK NO_BUF_OVERFLOW
```

Line 24-28 in File ERC20Detailed.sol

```
24 constructor(string name, string symbol, uint8 decimals) public {
25   _name = name;
26   _symbol = symbol;
27   _decimals = decimals;
28 }
```

The code meets the specification

Formal Verification Request 59

Method will not encounter an assertion failure.

11, Feb 2019
0.41 ms

Line 18 in File ERC20Detailed.sol

```
18 //@CTK NO_ASF
```

Line 24-28 in File ERC20Detailed.sol

```
24 constructor(string name, string symbol, uint8 decimals) public {
25   _name = name;
26   _symbol = symbol;
27   _decimals = decimals;
28 }
```

The code meets the specification

Formal Verification Request 60

ERC20Detailed constructor correctness

11, Feb 2019
0.83 ms

Line 19-23 in File ERC20Detailed.sol

```
19 /*@CTK "ERC20Detailed constructor correctness"
20   @post __post._name == name
21   @post __post._symbol == symbol
22   @post __post._decimals == decimals
23  */
```

Line 24-28 in File ERC20Detailed.sol

```
24  constructor(string name, string symbol, uint8 decimals) public {
25    _name = name;
26    _symbol = symbol;
27    _decimals = decimals;
28  }
```

✅ The code meets the specification

Formal Verification Request 61

If method completes, integer overflow would not happen.

📅 11, Feb 2019
⏱ 6.86 ms

Line 33 in File ERC20Detailed.sol

```
33  //@CTK NO_OVERFLOW
```

Line 39-41 in File ERC20Detailed.sol

```
39  function name() public view returns(string) {
40    return _name;
41  }
```

✅ The code meets the specification

Formal Verification Request 62

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

📅 11, Feb 2019
⏱ 0.42 ms

Line 34 in File ERC20Detailed.sol

```
34  //@CTK NO_BUF_OVERFLOW
```

Line 39-41 in File ERC20Detailed.sol

```
39  function name() public view returns(string) {
40    return _name;
41  }
```

✅ The code meets the specification

Formal Verification Request 63

Method will not encounter an assertion failure.

📅 11, Feb 2019
⏱ 1.16 ms

Line 35 in File ERC20Detailed.sol

```
35 //@CTK NO_ASF
```

Line 39-41 in File ERC20Detailed.sol

```
39 function name() public view returns(string) {
40   return _name;
41 }
```

✅ The code meets the specification

Formal Verification Request 64

ERC20Detailed name correctness

📅 11, Feb 2019
⏱ 0.46 ms

Line 36-38 in File ERC20Detailed.sol

```
36 /*@CTK "ERC20Detailed name correctness"
37   @post __return == _name
38  */
```

Line 39-41 in File ERC20Detailed.sol

```
39 function name() public view returns(string) {
40   return _name;
41 }
```

✅ The code meets the specification

Formal Verification Request 65

If method completes, integer overflow would not happen.

📅 11, Feb 2019
⏱ 6.52 ms

Line 46 in File ERC20Detailed.sol

```
46 //@CTK NO_OVERFLOW
```

Line 52-54 in File ERC20Detailed.sol

```
52 function symbol() public view returns(string) {
53   return _symbol;
54 }
```

✅ The code meets the specification

Formal Verification Request 66

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

11, Feb 2019
0.44 ms

Line 47 in File ERC20Detailed.sol

```
47 //@CTK NO_BUF_OVERFLOW
```

Line 52-54 in File ERC20Detailed.sol

```
52 function symbol() public view returns(string) {
53   return _symbol;
54 }
```

The code meets the specification

Formal Verification Request 67

Method will not encounter an assertion failure.

11, Feb 2019
0.44 ms

Line 48 in File ERC20Detailed.sol

```
48 //@CTK NO_ASF
```

Line 52-54 in File ERC20Detailed.sol

```
52 function symbol() public view returns(string) {
53   return _symbol;
54 }
```

The code meets the specification

Formal Verification Request 68

ERC20Detailed symbol correctness

11, Feb 2019
0.44 ms

Line 49-51 in File ERC20Detailed.sol

```
49 /*@CTK "ERC20Detailed symbol correctness"
50   @post __return == _symbol
51  */
```

Line 52-54 in File ERC20Detailed.sol

```
52 function symbol() public view returns(string) {
53   return _symbol;
54 }
```

The code meets the specification

Formal Verification Request 69

If method completes, integer overflow would not happen.

11, Feb 2019
5.88 ms

Line 56 in File ERC20Detailed.sol

```
56  //@CTK NO_OVERFLOW
```

Line 65-67 in File ERC20Detailed.sol

```
65  function decimals() public view returns(uint8) {
66    return _decimals;
67  }
```

The code meets the specification

Formal Verification Request 70

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

11, Feb 2019
0.47 ms

Line 57 in File ERC20Detailed.sol

```
57  //@CTK NO_BUF_OVERFLOW
```

Line 65-67 in File ERC20Detailed.sol

```
65  function decimals() public view returns(uint8) {
66    return _decimals;
67  }
```

The code meets the specification

Formal Verification Request 71

Method will not encounter an assertion failure.

11, Feb 2019
0.44 ms

Line 58 in File ERC20Detailed.sol

```
58  //@CTK NO_ASF
```

Line 65-67 in File ERC20Detailed.sol

```
65  function decimals() public view returns(uint8) {
66    return _decimals;
67  }
```

The code meets the specification

Formal Verification Request 72

ERC20Detailed decimals correctness

11, Feb 2019
0.61 ms

Line 59-61 in File ERC20Detailed.sol

```
59 /*@CTK "ERC20Detailed decimals correctness"
60   @post __return == _decimals
61  */
```

Line 65-67 in File ERC20Detailed.sol

```
65 function decimals() public view returns(uint8) {
66   return _decimals;
67 }
```

The code meets the specification