CERTIK

Audit Report

Produced by CertiK

for HYCON

Nov 01, 2019

CertiK Audit Report For HYCON

Request Date: 2019-10-26
Revision Date: 2019-11-01
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 6.2B in assets.

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

Executive Summary

This report has been prepared for HYCON to discover issues and vulnerabilities in the source code of their smart contracts. A comprehensive examination has been performed, utilizing CertiK's Formal Verification Platform, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

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

Vulnerability Classification

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

Critical

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

Medium

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

Low

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

Testing Summary

PASS

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

Oct 29, 2019

Token Name: Hycon
The Contract Address: 0x07c9eefd90593811442656 00d3cbf4974312861b
Link Address: https://etherscan.io/token/0x07c9eefd9059381144265600d3cbf4974312861b

Type of Issues

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

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by 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.	1	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.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File ERC20.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File Ownable.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File ERC20Mintable.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File SafeMath.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File ERC20Detailed.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File Pausable.sol

```
pragma solidity ^0.5.0;
```

ⓘ Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

Line 1 in File Migrations.sol

```
  pragma solidity ^0.4.17;
```

❗ Version to compile has the following bug:
0.4.17: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ExpExponentCleanup, EventStructWrongData,

NestedArrayFunctionCallDecoder, ZeroFunctionSelector
0.4.18: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder
0.4.19: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder
0.4.20: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder
0.4.21: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, NestedArrayFunctionCallDecoder
0.4.22: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData, OneOfTwoConstructorsSkipped
0.4.23: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData
0.4.24: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x, ExpExponentCleanup, EventStructWrongData
0.4.25: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2, UninitializedFunctionPointerInConstructor_0.4.x, IncorrectEventSignatureInLibraries_0.4.x, ABIEncoderV2PackedStorage_0.4.x 0.4.26: SignedArrayStorageCopy, ABIEncoderV2StorageArrayWithMultiSlotElement, DynamicConstructorArgumentsClippedABIV2

INSECURE_COMPILER_VERSION

Line 1 in File ERC20Burnable.sol

```
pragma solidity ^0.5.0;
```

Only these compiler versions are safe to compile your code: 0.5.10

INSECURE_COMPILER_VERSION

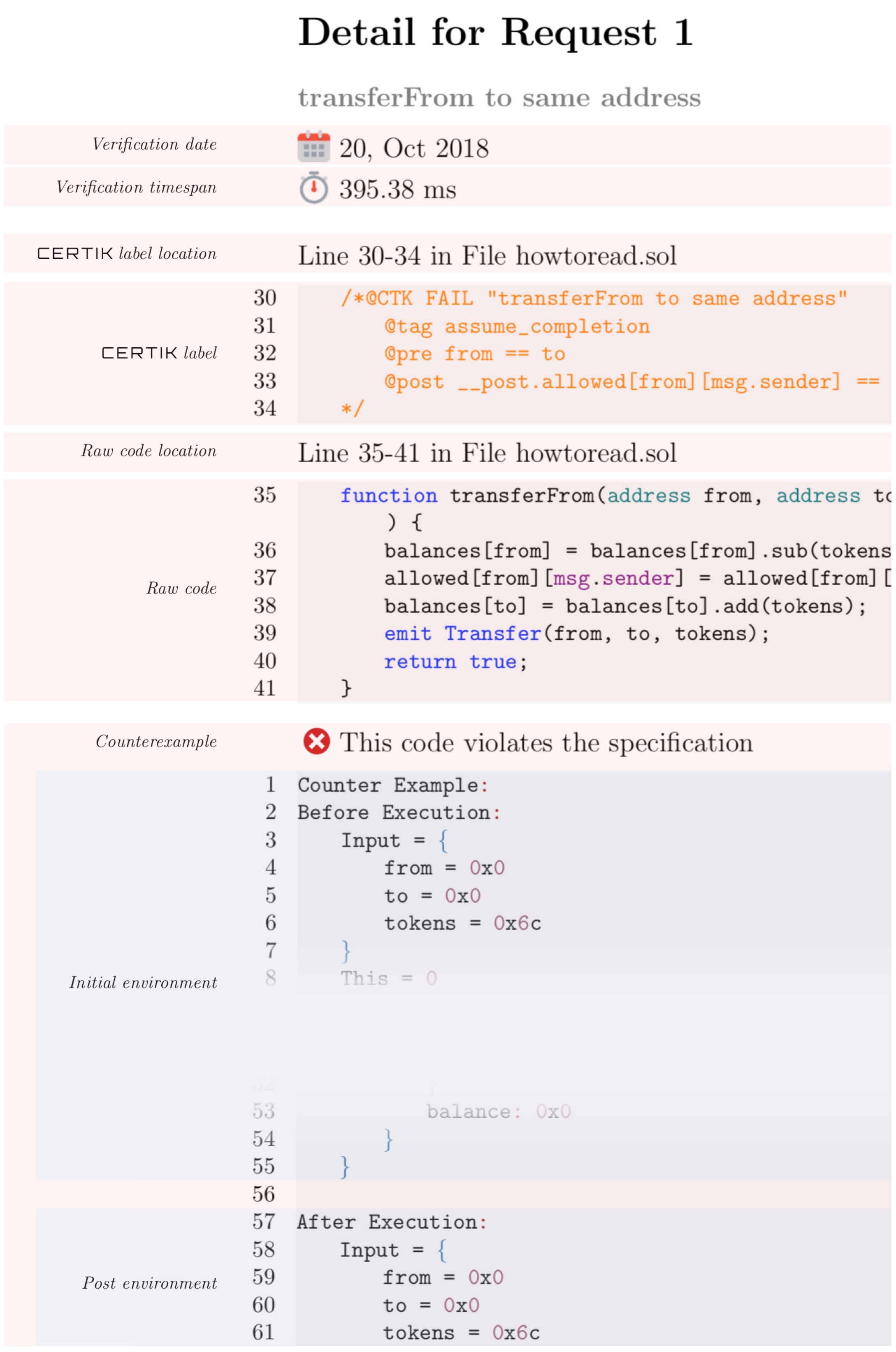
Line 1 in File ERC20Pausable.sol

```
pragma solidity ^0.5.0;
```

Only these compiler versions are safe to compile your code: 0.5.10

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*

```
/*@CTK FAIL "transferFrom to same address"
    @tag assume_completion
    @pre from == to
    @post __post.allowed[from][msg.sender] ==
*/
```

Raw code location Line 35-41 in File howtoread.sol

Raw code

```
function transferFrom(address from, address tc
    ) {
    balances[from] = balances[from].sub(tokens
    allowed[from][msg.sender] = allowed[from][
    balances[to] = balances[to].add(tokens);
    emit Transfer(from, to, tokens);
    return true;
}
```

Counterexample This code violates the specification

Initial environment

```
Counter Example:
Before Execution:
    Input = {
        from = 0x0
        to = 0x0
        tokens = 0x6c
    }
    This = 0

            balance: 0x0
        }
    }

```

Post environment

```
After Execution:
    Input = {
        from = 0x0
        to = 0x0
        tokens = 0x6c
```

Formal Verification Request 1

If method completes, integer overflow would not happen.

30, Oct 2019
6.07 ms

Line 41 in File ERC20.sol

```
//@CTK NO_OVERFLOW
```

Line 47-49 in File ERC20.sol

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

✓ The code meets the specification.

Formal Verification Request 2

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

30, Oct 2019
0.47 ms

Line 42 in File ERC20.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 47-49 in File ERC20.sol

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

✓ The code meets the specification.

Formal Verification Request 3

Method will not encounter an assertion failure.

30, Oct 2019
0.45 ms

Line 43 in File ERC20.sol

```
//@CTK NO_ASF
```

Line 47-49 in File ERC20.sol

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

✓ The code meets the specification.

Formal Verification Request 4

totalSupply

📅 30, Oct 2019
⏱ 0.51 ms

Line 44-46 in File ERC20.sol

```
    /*@CTK totalSupply
      @post __return == _totalSupply
     */
```

Line 47-49 in File ERC20.sol

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

✓ The code meets the specification.

Formal Verification Request 5

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 5.82 ms

Line 54 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 60-62 in File ERC20.sol

```
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
```

✓ The code meets the specification.

Formal Verification Request 6

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

📅 30, Oct 2019
⏱ 0.45 ms

Line 55 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 60-62 in File ERC20.sol

```
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
```

✓ The code meets the specification.

Formal Verification Request 7

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.46 ms

Line 56 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 60-62 in File ERC20.sol

```
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
```

✓ The code meets the specification.

Formal Verification Request 8

balanceOf

📅 30, Oct 2019
⏱ 0.43 ms

Line 57-59 in File ERC20.sol

```
    /*@CTK balanceOf
      @post __return == _balances[account]
     */
```

Line 60-62 in File ERC20.sol

```
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }
```

✓ The code meets the specification.

Formal Verification Request 9

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 127.28 ms

Line 72 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 82-85 in File ERC20.sol

```
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 10

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

30, Oct 2019
4.01 ms

Line 73 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 82-85 in File ERC20.sol

```
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 11

transfer

30, Oct 2019
36.5 ms

Line 74-81 in File ERC20.sol

```
    /*@CTK transfer
      @tag assume_completion
      @pre msg.sender != recipient
      @post msg.sender != address(0)
      @post recipient != address(0)
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
      @post __post._balances[recipient] == _balances[recipient] + amount
     */
```

Line 82-85 in File ERC20.sol

```
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 12

If method completes, integer overflow would not happen.

30, Oct 2019
6.17 ms

Line 90 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 96-98 in File ERC20.sol

```
function allowance(address owner, address spender) public view returns (uint256) {
    return _allowances[owner][spender];
}
```

✓ The code meets the specification.

Formal Verification Request 13

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

30, Oct 2019
0.47 ms

Line 91 in File ERC20.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 96-98 in File ERC20.sol

```
function allowance(address owner, address spender) public view returns (uint256) {
    return _allowances[owner][spender];
}
```

✓ The code meets the specification.

Formal Verification Request 14

Method will not encounter an assertion failure.

30, Oct 2019
0.45 ms

Line 92 in File ERC20.sol

```
//@CTK NO_ASF
```

Line 96-98 in File ERC20.sol

```
function allowance(address owner, address spender) public view returns (uint256) {
    return _allowances[owner][spender];
}
```

✓ The code meets the specification.

Formal Verification Request 15

allowance

30, Oct 2019
0.49 ms

Line 93-95 in File ERC20.sol

```
93      /*@CTK allowance
94        @post __return == _allowances[owner][spender]
95       */
```

Line 96-98 in File ERC20.sol

```
96      function allowance(address owner, address spender) public view returns (uint256) {
97          return _allowances[owner][spender];
98      }
```

✅ The code meets the specification.

Formal Verification Request 16

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 76.84 ms

Line 107 in File ERC20.sol

```
107     //@CTK NO_OVERFLOW
```

Line 116-119 in File ERC20.sol

```
116     function approve(address spender, uint256 value) public returns (bool) {
117         _approve(msg.sender, spender, value);
118         return true;
119     }
```

✅ The code meets the specification.

Formal Verification Request 17

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

📅 30, Oct 2019
⏱ 0.86 ms

Line 108 in File ERC20.sol

```
108     //@CTK NO_BUF_OVERFLOW
```

Line 116-119 in File ERC20.sol

```
116     function approve(address spender, uint256 value) public returns (bool) {
117         _approve(msg.sender, spender, value);
118         return true;
119     }
```

✅ The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.86 ms

Line 109 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 116-119 in File ERC20.sol

```
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 19

approve

📅 30, Oct 2019
⏱ 2.76 ms

Line 110-115 in File ERC20.sol

```
    /*@CTK approve
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == value
     */
```

Line 116-119 in File ERC20.sol

```
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 20

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 108.34 ms

Line 133 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 144-148 in File ERC20.sol

```
144     function transferFrom(address sender, address recipient, uint256 amount) public
            returns (bool) {
145         _transfer(sender, recipient, amount);
146         _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));
147         return true;
148     }
```

✓ The code meets the specification.

Formal Verification Request 21

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

30, Oct 2019
5.49 ms

Line 134 in File ERC20.sol

```
134     //@CTK NO_BUF_OVERFLOW
```

Line 144-148 in File ERC20.sol

```
144     function transferFrom(address sender, address recipient, uint256 amount) public
            returns (bool) {
145         _transfer(sender, recipient, amount);
146         _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));
147         return true;
148     }
```

✓ The code meets the specification.

Formal Verification Request 22

transferFrom

30, Oct 2019
91.74 ms

Line 135-143 in File ERC20.sol

```
135     /*@CTK transferFrom
136       @tag assume_completion
137       @pre sender != recipient
138       @post sender != address(0)
139       @post recipient != address(0)
140       @post __post._balances[sender] == _balances[sender] - amount
141       @post __post._balances[recipient] == _balances[recipient] + amount
142       @post __post._allowances[sender][msg.sender] == _allowances[sender][msg.sender] -
              amount
143      */
```

Line 144-148 in File ERC20.sol

```
144     function transferFrom(address sender, address recipient, uint256 amount) public
            returns (bool) {
145         _transfer(sender, recipient, amount);
146         _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));
```

```
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 23

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 53.06 ms

Line 162 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 170-173 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
            ;
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 24

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

📅 30, Oct 2019
⏱ 1.09 ms

Line 163 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 170-173 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
            ;
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 25

increaseAllowance

📅 30, Oct 2019
⏱ 4.74 ms

Line 164-169 in File ERC20.sol

```
    /*@CTK increaseAllowance
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender]
          + addedValue
     */
```

Line 170-173 in File ERC20.sol

```
    function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
            ;
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 26

If method completes, integer overflow would not happen.

30, Oct 2019
70.91 ms

Line 189 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 197-200 in File ERC20.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
            subtractedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 27

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

30, Oct 2019
0.99 ms

Line 190 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 197-200 in File ERC20.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
            subtractedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 28

decreaseAllowance

📅 30, Oct 2019
⏱ 2.96 ms

Line 191-196 in File ERC20.sol

```
    /*@CTK decreaseAllowance
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender]
           - subtractedValue
     */
```

Line 197-200 in File ERC20.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
            subtractedValue));
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 29

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 4.0 ms

Line 216 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 226-233 in File ERC20.sol

```
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 30

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

30, Oct 2019
5.03 ms

Line 217 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 226-233 in File ERC20.sol

```
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 31

__transfer

30, Oct 2019
43.52 ms

Line 218-225 in File ERC20.sol

```
    /*@CTK _transfer
      @tag assume_completion
      @pre sender != recipient
      @post sender != address(0)
      @post recipient != address(0)
      @post __post._balances[sender] == _balances[sender] - amount
      @post __post._balances[recipient] == _balances[recipient] + amount
     */
```

Line 226-233 in File ERC20.sol

```
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 32

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 45.41 ms

Line 244 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 252-258 in File ERC20.sol

```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

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

✓ The code meets the specification.

Formal Verification Request 33

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

📅 30, Oct 2019
⏱ 3.5 ms

Line 245 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 252-258 in File ERC20.sol

```
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

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

✓ The code meets the specification.

Formal Verification Request 34

_mint

📅 30, Oct 2019
⏱ 12.45 ms

Line 246-251 in File ERC20.sol

```
/*@CTK _mint
  @tag assume_completion
  @post account != address(0)
  @post __post._totalSupply == _totalSupply + amount
  @post __post._balances[account] == _balances[account] + amount
 */
```

Line 252-258 in File ERC20.sol

```
function _mint(address account, uint256 amount) internal {
    require(account != address(0), "ERC20: mint to the zero address");

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

✓ The code meets the specification.

Formal Verification Request 35

If method completes, integer overflow would not happen.

30, Oct 2019
57.8 ms

Line 271 in File ERC20.sol

```
//@CTK NO_OVERFLOW
```

Line 279-285 in File ERC20.sol

```
function _burn(address account, uint256 value) internal {
    require(account != address(0), "ERC20: burn from the zero address");

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

✓ The code meets the specification.

Formal Verification Request 36

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

30, Oct 2019
3.35 ms

Line 272 in File ERC20.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 279-285 in File ERC20.sol

```
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

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

✓ The code meets the specification.

Formal Verification Request 37

__burn

30, Oct 2019
16.6 ms

Line 273-278 in File ERC20.sol

```
    /*@CTK _burn
      @tag assume_completion
      @post account != address(0)
      @post __post._totalSupply == _totalSupply - value
      @post __post._balances[account] == _balances[account] - value
     */
```

Line 279-285 in File ERC20.sol

```
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

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

✓ The code meets the specification.

Formal Verification Request 38

If method completes, integer overflow would not happen.

30, Oct 2019
0.77 ms

Line 300 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 309-315 in File ERC20.sol

```
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 39

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

30, Oct 2019
0.72 ms

Line 301 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 309-315 in File ERC20.sol

```
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 40

Method will not encounter an assertion failure.

30, Oct 2019
0.7 ms

Line 302 in File ERC20.sol

```
    //@CTK NO_ASF
```

Line 309-315 in File ERC20.sol

```
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 41

_approve

30, Oct 2019
2.32 ms

Line 303-308 in File ERC20.sol

```
    /*@CTK _approve
      @tag assume_completion
      @post owner != address(0)
      @post spender != address(0)
      @post __post._allowances[owner][spender] == value
     */
```

Line 309-315 in File ERC20.sol

```
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 42

If method completes, integer overflow would not happen.

30, Oct 2019
96.76 ms

Line 323 in File ERC20.sol

```
    //@CTK NO_OVERFLOW
```

Line 333-336 in File ERC20.sol

```
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }
```

✓ The code meets the specification.

Formal Verification Request 43

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

30, Oct 2019
5.49 ms

Line 324 in File ERC20.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 333-336 in File ERC20.sol

```
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }
```

✓ The code meets the specification.

Formal Verification Request 44

__burn

📅 30, Oct 2019
⏱ 64.18 ms

Line 325-332 in File ERC20.sol

```
/*@CTK _burn
  @tag assume_completion
  @post account != address(0)
  @post msg.sender != address(0)
  @post __post._totalSupply == _totalSupply - amount
  @post __post._balances[account] == _balances[account] - amount
  @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender]
       - amount
 */
```

Line 333-336 in File ERC20.sol

```
function _burnFrom(address account, uint256 amount) internal {
    _burn(account, amount);
    _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
}
```

✅ The code meets the specification.

Formal Verification Request 45

Ownable

📅 30, Oct 2019
⏱ 7.02 ms

Line 20-22 in File Ownable.sol

```
/*@CTK Ownable
  @post __post._owner == msg.sender
 */
```

Line 23-26 in File Ownable.sol

```
constructor () internal {
    _owner = msg.sender;
    emit OwnershipTransferred(address(0), _owner);
}
```

✅ The code meets the specification.

Formal Verification Request 46

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 11.05 ms

Line 31 in File Ownable.sol

```
    //@CTK NO_OVERFLOW
```

Line 37-39 in File Ownable.sol

```
    function owner() public view returns (address) {
        return _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 47

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

📅 30, Oct 2019
⏱ 0.52 ms

Line 32 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 37-39 in File Ownable.sol

```
    function owner() public view returns (address) {
        return _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 48

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.52 ms

Line 33 in File Ownable.sol

```
    //@CTK NO_ASF
```

Line 37-39 in File Ownable.sol

```
    function owner() public view returns (address) {
        return _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 49

owner

📅 30, Oct 2019
⏱ 0.46 ms

Line 34-36 in File Ownable.sol

```
    /*@CTK owner
      @post __return == _owner
     */
```

Line 37-39 in File Ownable.sol

```
    function owner() public view returns (address) {
        return _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 50

If method completes, integer overflow would not happen.

30, Oct 2019
9.03 ms

Line 52 in File Ownable.sol

```
    //@CTK NO_OVERFLOW
```

Line 58-60 in File Ownable.sol

```
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 51

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

30, Oct 2019
1.45 ms

Line 53 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 58-60 in File Ownable.sol

```
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }
```

✓ The code meets the specification.

Formal Verification Request 52

Method will not encounter an assertion failure.

30, Oct 2019
0.8 ms

Line 54 in File Ownable.sol

```
//@CTK NO_ASF
```

Line 58-60 in File Ownable.sol

```
function isOwner() public view returns (bool) {
    return msg.sender == _owner;
}
```

✓ The code meets the specification.

Formal Verification Request 53

isOwner

📅 30, Oct 2019
⏱ 0.81 ms

Line 55-57 in File Ownable.sol

```
/*@CTK isOwner
  @post __return == (msg.sender == _owner)
 */
```

Line 58-60 in File Ownable.sol

```
function isOwner() public view returns (bool) {
    return msg.sender == _owner;
}
```

✓ The code meets the specification.

Formal Verification Request 54

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 40.43 ms

Line 69 in File Ownable.sol

```
//@CTK NO_OVERFLOW
```

Line 77-80 in File Ownable.sol

```
function renounceOwnership() public onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}
```

✓ The code meets the specification.

Formal Verification Request 55

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

📅 30, Oct 2019
⏱ 0.69 ms

Line 70 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 77-80 in File Ownable.sol

```
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

✅ The code meets the specification.

Formal Verification Request 56

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.66 ms

Line 71 in File Ownable.sol

```
    //@CTK NO_ASF
```

Line 77-80 in File Ownable.sol

```
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }
```

✅ The code meets the specification.

Formal Verification Request 57

renounceOwnership

📅 30, Oct 2019
⏱ 1.13 ms

Line 72-76 in File Ownable.sol

```
    /*@CTK renounceOwnership
      @tag assume_completion
      @post msg.sender == _owner
      @post __post._owner == address(0)
     */
```

Line 77-80 in File Ownable.sol

```
77     function renounceOwnership() public onlyOwner {
78         emit OwnershipTransferred(_owner, address(0));
79         _owner = address(0);
80     }
```

✓ The code meets the specification.

Formal Verification Request 58

If method completes, integer overflow would not happen.

30, Oct 2019
102.29 ms

Line 86 in File Ownable.sol

```
86     //@CTK NO_OVERFLOW
```

Line 95-97 in File Ownable.sol

```
95     function transferOwnership(address newOwner) public onlyOwner {
96         _transferOwnership(newOwner);
97     }
```

✓ The code meets the specification.

Formal Verification Request 59

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

30, Oct 2019
2.27 ms

Line 87 in File Ownable.sol

```
87     //@CTK NO_BUF_OVERFLOW
```

Line 95-97 in File Ownable.sol

```
95     function transferOwnership(address newOwner) public onlyOwner {
96         _transferOwnership(newOwner);
97     }
```

✓ The code meets the specification.

Formal Verification Request 60

Method will not encounter an assertion failure.

30, Oct 2019
3.84 ms

Line 88 in File Ownable.sol

```
88     //@CTK NO_ASF
```

Line 95-97 in File Ownable.sol

```
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
```

✓ The code meets the specification.

Formal Verification Request 61

transferOwnership

30, Oct 2019
5.0 ms

Line 89-94 in File Ownable.sol

```
    /*@CTK transferOwnership
      @tag assume_completion
      @post msg.sender == _owner
      @post newOwner != address(0)
      @post __post._owner == newOwner
     */
```

Line 95-97 in File Ownable.sol

```
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }
```

✓ The code meets the specification.

Formal Verification Request 62

If method completes, integer overflow would not happen.

30, Oct 2019
0.89 ms

Line 102 in File Ownable.sol

```
    //@CTK NO_OVERFLOW
```

Line 110-114 in File Ownable.sol

```
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 63

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

30, Oct 2019
0.75 ms

Line 103 in File Ownable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 110-114 in File Ownable.sol

```
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 64

Method will not encounter an assertion failure.

30, Oct 2019
0.64 ms

Line 104 in File Ownable.sol

```
    //@CTK NO_ASF
```

Line 110-114 in File Ownable.sol

```
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 65

_transferOwnership

30, Oct 2019
1.57 ms

Line 105-109 in File Ownable.sol

```
    /*@CTK _transferOwnership
      @tag assume_completion
      @post newOwner != address(0)
      @post __post._owner == newOwner
     */
```

Line 110-114 in File Ownable.sol

```
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

✓ The code meets the specification.

Formal Verification Request 66

If method completes, integer overflow would not happen.

30, Oct 2019
220.03 ms

Line 20 in File ERC20Mintable.sol

```
    //@CTK NO_OVERFLOW
```

Line 22-25 in File ERC20Mintable.sol

```
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 67

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

30, Oct 2019
4.59 ms

Line 21 in File ERC20Mintable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 22-25 in File ERC20Mintable.sol

```
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
```

✓ The code meets the specification.

Formal Verification Request 68

If method completes, integer overflow would not happen.

30, Oct 2019
27.96 ms

Line 26 in File SafeMath.sol

```
    //@CTK NO_OVERFLOW
```

Line 38-43 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 69

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

30, Oct 2019
0.76 ms

Line 27 in File SafeMath.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 38-43 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 70

Method will not encounter an assertion failure.

30, Oct 2019
1.04 ms

Line 28 in File SafeMath.sol

```
    //@CTK NO_ASF
```

Line 38-43 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 71

SafeMath_add

📅 30, Oct 2019
⏱ 2.55 ms

Line 29-37 in File SafeMath.sol

```
    /*@CTK "SafeMath_add"
      @post (__reverted) == (__has_overflow)
      @post (!(__reverted)) -> ((__return) == ((a) + (b)))
      @post (msg) == (msg__post)
      @post ((((a) + (b)) < (a)) || (((a) + (b)) < (b))) == __reverted
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
```

Line 38-43 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 72

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 30.08 ms

Line 54 in File SafeMath.sol

```
    //@CTK NO_OVERFLOW
```

Line 66-71 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 73

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

📅 30, Oct 2019
⏱ 0.87 ms

Line 55 in File SafeMath.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 66-71 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 74

Method will not encounter an assertion failure.

30, Oct 2019
0.65 ms

Line 56 in File SafeMath.sol

```
    //@CTK NO_ASF
```

Line 66-71 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 75

SafeMath_sub

30, Oct 2019
1.45 ms

Line 57-65 in File SafeMath.sol

```
    /*@CTK "SafeMath_sub"
      @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
      @post (!(__reverted)) -> ((__return) == ((a) - (b)))
      @post (msg) == (msg__post)
      @post ((a) < (b)) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
```

Line 66-71 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 76

If method completes, integer overflow would not happen.

30, Oct 2019
50.79 ms

Line 82 in File SafeMath.sol

```
    //@CTK NO_OVERFLOW
```

Line 94-106 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 77

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

30, Oct 2019
0.74 ms

Line 83 in File SafeMath.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 94-106 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 78

Method will not encounter an assertion failure.

30, Oct 2019
0.75 ms

Line 84 in File SafeMath.sol

```
    //@CTK NO_ASF
```

Line 94-106 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 79

SafeMath_mul

30, Oct 2019
208.82 ms

Line 85-93 in File SafeMath.sol

```
    /*@CTK "SafeMath_mul"
      @post (__reverted) == (__has_overflow)
      @post (!(__reverted)) -> ((__return) == ((a) * (b)))
      @post (msg) == (msg__post)
      @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
```

Line 94-106 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 80

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 21.82 ms

Line 119 in File SafeMath.sol

```
//@CTK NO_OVERFLOW
```

Line 134-141 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 81

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

📅 30, Oct 2019
⏱ 0.69 ms

Line 120 in File SafeMath.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 134-141 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 82

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.99 ms

Line 121 in File SafeMath.sol

```
    //@CTK NO_ASF
```

Line 134-141 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 83

SafeMath_div

📅 30, Oct 2019
⏱ 1.73 ms

Line 122-130 in File SafeMath.sol

```
    /*@CTK "SafeMath_div"
      @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
      @post (!(__reverted)) -> ((__return) == ((a) / (b)))
      @post (msg) == (msg__post)
      @post ((b) == (0)) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
```

Line 134-141 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 84

SafeMath_div

30, Oct 2019
1.73 ms

Line 131-133 in File SafeMath.sol

```
/*@CTK "SafeMath_div"
  @post ((__reverted) == (false)) -> ((__return) == ((a) / (b)))
*/
```

Line 134-141 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 85

If method completes, integer overflow would not happen.

30, Oct 2019
20.59 ms

Line 154 in File SafeMath.sol

```
//@CTK NO_OVERFLOW
```

Line 157-160 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 86

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

📅 30, Oct 2019
⏱ 0.7 ms

Line 155 in File SafeMath.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 157-160 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 87

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.63 ms

Line 156 in File SafeMath.sol

```
    //@CTK NO_ASF
```

Line 157-160 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 88

ERC20Detailed

📅 30, Oct 2019
⏱ 17.43 ms

Line 18-22 in File ERC20Detailed.sol

```
    /*@CTK ERC20Detailed
      @post __post._name == name
      @post __post._symbol == symbol
      @post __post._decimals == decimals
     */
```

Line 23-27 in File ERC20Detailed.sol

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

✓ The code meets the specification.

Formal Verification Request 89

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 9.81 ms

Line 32 in File ERC20Detailed.sol

```
    //@CTK NO_OVERFLOW
```

Line 38-40 in File ERC20Detailed.sol

```
    function name() public view returns (string memory) {
        return _name;
    }
```

✓ The code meets the specification.

Formal Verification Request 90

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

📅 30, Oct 2019
⏱ 0.52 ms

Line 33 in File ERC20Detailed.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 38-40 in File ERC20Detailed.sol

```
    function name() public view returns (string memory) {
        return _name;
    }
```

✓ The code meets the specification.

Formal Verification Request 91

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 0.49 ms

Line 34 in File ERC20Detailed.sol

```
    //@CTK NO_ASF
```

Line 38-40 in File ERC20Detailed.sol

```
function name() public view returns (string memory) {
    return _name;
}
```

✓ The code meets the specification.

Formal Verification Request 92

name

30, Oct 2019
1.04 ms

Line 35-37 in File ERC20Detailed.sol

```
/*@CTK name
  @post __return == __post._name
 */
```

Line 38-40 in File ERC20Detailed.sol

```
function name() public view returns (string memory) {
    return _name;
}
```

✓ The code meets the specification.

Formal Verification Request 93

If method completes, integer overflow would not happen.

30, Oct 2019
9.75 ms

Line 46 in File ERC20Detailed.sol

```
//@CTK NO_OVERFLOW
```

Line 52-54 in File ERC20Detailed.sol

```
function symbol() public view returns (string memory) {
    return _symbol;
}
```

✓ The code meets the specification.

Formal Verification Request 94

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

30, Oct 2019
1.06 ms

Line 47 in File ERC20Detailed.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 52-54 in File ERC20Detailed.sol

```
function symbol() public view returns (string memory) {
    return _symbol;
}
```

✓ The code meets the specification.

Formal Verification Request 95

Method will not encounter an assertion failure.

30, Oct 2019
0.89 ms

Line 48 in File ERC20Detailed.sol

```
//@CTK NO_ASF
```

Line 52-54 in File ERC20Detailed.sol

```
function symbol() public view returns (string memory) {
    return _symbol;
}
```

✓ The code meets the specification.

Formal Verification Request 96

symbol

30, Oct 2019
0.8 ms

Line 49-51 in File ERC20Detailed.sol

```
/*@CTK symbol
  @post __return == _symbol
 */
```

Line 52-54 in File ERC20Detailed.sol

```
function symbol() public view returns (string memory) {
    return _symbol;
}
```

✓ The code meets the specification.

Formal Verification Request 97

If method completes, integer overflow would not happen.

30, Oct 2019
8.98 ms

Line 68 in File ERC20Detailed.sol

```
    //@CTK NO_OVERFLOW
```

Line 74-76 in File ERC20Detailed.sol

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

✓ The code meets the specification.

Formal Verification Request 98

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

30, Oct 2019
0.56 ms

Line 69 in File ERC20Detailed.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 74-76 in File ERC20Detailed.sol

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

✓ The code meets the specification.

Formal Verification Request 99

Method will not encounter an assertion failure.

30, Oct 2019
0.56 ms

Line 70 in File ERC20Detailed.sol

```
    //@CTK NO_ASF
```

Line 74-76 in File ERC20Detailed.sol

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

✓ The code meets the specification.

Formal Verification Request 100

decimals

30, Oct 2019
0.57 ms

Line 71-73 in File ERC20Detailed.sol

```
    /*@CTK decimals
      @post __return == _decimals
     */
```

Line 74-76 in File ERC20Detailed.sol

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

✓ The code meets the specification.

Formal Verification Request 101

Pausable

30, Oct 2019
7.44 ms

Line 31-33 in File Pausable.sol

```
    /*@CTK Pausable
      @post !__post._paused
     */
```

Line 34-36 in File Pausable.sol

```
    constructor () internal {
        _paused = false;
    }
```

✓ The code meets the specification.

Formal Verification Request 102

If method completes, integer overflow would not happen.

30, Oct 2019
12.16 ms

Line 41 in File Pausable.sol

```
    //@CTK NO_OVERFLOW
```

Line 47-49 in File Pausable.sol

```
    function paused() public view returns (bool) {
        return _paused;
    }
```

✓ The code meets the specification.

Formal Verification Request 103

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

30, Oct 2019
0.72 ms

Line 42 in File Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 47-49 in File Pausable.sol

```
    function paused() public view returns (bool) {
        return _paused;
    }
```

✓ The code meets the specification.

Formal Verification Request 104

Method will not encounter an assertion failure.

30, Oct 2019
0.54 ms

Line 43 in File Pausable.sol

```
    //@CTK NO_ASF
```

Line 47-49 in File Pausable.sol

```
    function paused() public view returns (bool) {
        return _paused;
    }
```

✓ The code meets the specification.

Formal Verification Request 105

paused

30, Oct 2019
0.55 ms

Line 44-46 in File Pausable.sol

```
    /*@CTK paused
      @post __return == _paused
     */
```

Line 47-49 in File Pausable.sol

```
    function paused() public view returns (bool) {
        return _paused;
    }
```

✓ The code meets the specification.

Formal Verification Request 106

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 171.69 ms

Line 70 in File Pausable.sol

```
    //@CTK NO_OVERFLOW
```

Line 78-81 in File Pausable.sol

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 107

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

📅 30, Oct 2019
⏱ 1.37 ms

Line 71 in File Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 78-81 in File Pausable.sol

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 108

Method will not encounter an assertion failure.

📅 30, Oct 2019
⏱ 1.85 ms

Line 72 in File Pausable.sol

```
    //@CTK NO_ASF
```

Line 78-81 in File Pausable.sol

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 109

pause

📅 30, Oct 2019
⏱ 4.42 ms

Line 73-77 in File Pausable.sol

```
    /*@CTK pause
      @tag assume_completion
      @post !_paused
      @post __post._paused
     */
```

Line 78-81 in File Pausable.sol

```
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }
```

✅ The code meets the specification.

Formal Verification Request 110

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 85.44 ms

Line 86 in File Pausable.sol

```
    //@CTK NO_OVERFLOW
```

Line 94-97 in File Pausable.sol

```
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
```

✅ The code meets the specification.

Formal Verification Request 111

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

📅 30, Oct 2019
⏱ 1.26 ms

Line 87 in File Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 94-97 in File Pausable.sol

```
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 112

Method will not encounter an assertion failure.

30, Oct 2019
1.15 ms

Line 88 in File Pausable.sol

```
    //@CTK NO_ASF
```

Line 94-97 in File Pausable.sol

```
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 113

unpause

30, Oct 2019
3.44 ms

Line 89-93 in File Pausable.sol

```
    /*@CTK unpause
      @tag assume_completion
      @post _paused
      @post !__post._paused
     */
```

Line 94-97 in File Pausable.sol

```
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
```

✓ The code meets the specification.

Formal Verification Request 114

Migrations

📅 30, Oct 2019
⏱ 6.64 ms

Line 11-13 in File Migrations.sol

```
/*@CTK Migrations
  @post __post.owner == msg.sender
 */
```

Line 14-16 in File Migrations.sol

```
function Migrations() public {
  owner = msg.sender;
}
```

✓ The code meets the specification.

Formal Verification Request 115

setCompleted

📅 30, Oct 2019
⏱ 15.23 ms

Line 18-21 in File Migrations.sol

```
/*@CTK setCompleted
  @pre msg.sender == owner
  @post __post.last_completed_migration == completed
 */
```

Line 22-24 in File Migrations.sol

```
function setCompleted(uint completed) public restricted {
  last_completed_migration = completed;
}
```

✓ The code meets the specification.

Formal Verification Request 116

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 93.77 ms

Line 16 in File ERC20Burnable.sol

```
//@CTK NO_OVERFLOW
```

Line 24-26 in File ERC20Burnable.sol

```
function burn(uint256 amount) public {
    _burn(msg.sender, amount);
}
```

✓ The code meets the specification.

Formal Verification Request 117

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

30, Oct 2019
4.76 ms

Line 17 in File ERC20Burnable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 24-26 in File ERC20Burnable.sol

```
    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 118

burn

30, Oct 2019
17.9 ms

Line 18-23 in File ERC20Burnable.sol

```
    /*@CTK burn
      @tag assume_completion
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply - amount
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
     */
```

Line 24-26 in File ERC20Burnable.sol

```
    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 119

If method completes, integer overflow would not happen.

30, Oct 2019
243.13 ms

Line 31 in File ERC20Burnable.sol

```
    //@CTK NO_OVERFLOW
```

Line 41-43 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 120

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

📅 30, Oct 2019
⏱ 6.09 ms

Line 32 in File ERC20Burnable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 41-43 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 121

burnFrom

📅 30, Oct 2019
⏱ 49.99 ms

Line 33-40 in File ERC20Burnable.sol

```
    /*@CTK burnFrom
      @tag assume_completion
      @post account != address(0)
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply - amount
      @post __post._balances[account] == _balances[account] - amount
      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender]
           - amount
     */
```

Line 41-43 in File ERC20Burnable.sol

```
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
```

✓ The code meets the specification.

Formal Verification Request 122

If method completes, integer overflow would not happen.

30, Oct 2019
201.66 ms

Line 15 in File ERC20Pausable.sol

```
//@CTK NO_OVERFLOW
```

Line 17-19 in File ERC20Pausable.sol

```
function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
    return super.transfer(to, value);
}
```

✓ The code meets the specification.

Formal Verification Request 123

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

30, Oct 2019
5.16 ms

Line 16 in File ERC20Pausable.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 17-19 in File ERC20Pausable.sol

```
function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
    return super.transfer(to, value);
}
```

✓ The code meets the specification.

Formal Verification Request 124

If method completes, integer overflow would not happen.

30, Oct 2019
271.61 ms

Line 21 in File ERC20Pausable.sol

```
//@CTK NO_OVERFLOW
```

Line 23-25 in File ERC20Pausable.sol

```
function transferFrom(address from, address to, uint256 value) public
    whenNotPaused returns (bool) {
    return super.transferFrom(from, to, value);
}
```

✓ The code meets the specification.

Formal Verification Request 125

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

📅 30, Oct 2019
⏱ 10.66 ms

Line 22 in File ERC20Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 23-25 in File ERC20Pausable.sol

```
    function transferFrom(address from, address to, uint256 value) public
        whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 126

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 117.05 ms

Line 27 in File ERC20Pausable.sol

```
    //@CTK NO_OVERFLOW
```

Line 30-32 in File ERC20Pausable.sol

```
    function approve(address spender, uint256 value) public whenNotPaused returns (
        bool) {
        return super.approve(spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 127

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

📅 30, Oct 2019
⏱ 1.22 ms

Line 28 in File ERC20Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 30-32 in File ERC20Pausable.sol

```
    function approve(address spender, uint256 value) public whenNotPaused returns (
        bool) {
        return super.approve(spender, value);
    }
```

✓ The code meets the specification.

Formal Verification Request 128

Method will not encounter an assertion failure.

30, Oct 2019
1.12 ms

Line 29 in File ERC20Pausable.sol

```
//@CTK NO_ASF
```

Line 30-32 in File ERC20Pausable.sol

```
function approve(address spender, uint256 value) public whenNotPaused returns (
    bool) {
    return super.approve(spender, value);
}
```

✓ The code meets the specification.

Formal Verification Request 129

If method completes, integer overflow would not happen.

30, Oct 2019
164.77 ms

Line 34 in File ERC20Pausable.sol

```
//@CTK NO_OVERFLOW
```

Line 36-38 in File ERC20Pausable.sol

```
function increaseAllowance(address spender, uint256 addedValue) public
    whenNotPaused returns (bool) {
    return super.increaseAllowance(spender, addedValue);
}
```

✓ The code meets the specification.

Formal Verification Request 130

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

30, Oct 2019
2.31 ms

Line 35 in File ERC20Pausable.sol

```
//@CTK NO_BUF_OVERFLOW
```

Line 36-38 in File ERC20Pausable.sol

```
function increaseAllowance(address spender, uint256 addedValue) public
    whenNotPaused returns (bool) {
    return super.increaseAllowance(spender, addedValue);
}
```

✓ The code meets the specification.

Formal Verification Request 131

If method completes, integer overflow would not happen.

📅 30, Oct 2019
⏱ 270.79 ms

Line 40 in File ERC20Pausable.sol

```
    //@CTK NO_OVERFLOW
```

Line 42-44 in File ERC20Pausable.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
```

✓ The code meets the specification.

Formal Verification Request 132

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

📅 30, Oct 2019
⏱ 2.93 ms

Line 41 in File ERC20Pausable.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 42-44 in File ERC20Pausable.sol

```
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
```

✓ The code meets the specification.

Source Code with CertiK Labels

File ERC20.sol

```
pragma solidity ^0.5.0;

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

/**
 * @dev Implementation of the `IERC20` interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using `_mint`.
 * For a generic mechanism see `ERC20Mintable`.
 *
 * *For a detailed writeup see our guide [How to implement supply
 * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
    mechanisms/226).*
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an `Approval` event is emitted on calls to `transferFrom`.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard `decreaseAllowance` and `increaseAllowance`
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See `IERC20.approve`.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

    /**
     * @dev See `IERC20.totalSupply`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK totalSupply
      @post __return == _totalSupply
     */
    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    /**
     * @dev See `IERC20.balanceOf`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK balanceOf
      @post __return == _balances[account]
     */
    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    /**
     * @dev See `IERC20.transfer`.
     *
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK transfer
      @tag assume_completion
      @pre msg.sender != recipient
      @post msg.sender != address(0)
      @post recipient != address(0)
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
      @post __post._balances[recipient] == _balances[recipient] + amount
     */
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    /**
     * @dev See `IERC20.allowance`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK allowance
      @post __return == _allowances[owner][spender]
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See `IERC20.approve`.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK approve
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == value
     */
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    /**
     * @dev See `IERC20.transferFrom`.
     *
     * Emits an `Approval` event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of `ERC20`;
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `value`.
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK transferFrom
      @tag assume_completion
      @pre sender != recipient
      @post sender != address(0)
      @post recipient != address(0)
      @post __post._balances[sender] == _balances[sender] - amount
      @post __post._balances[recipient] == _balances[recipient] + amount
      @post __post._allowances[sender][msg.sender] == _allowances[sender][msg.sender] -
          amount
     */
    function transferFrom(address sender, address recipient, uint256 amount) public
        returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to `approve` that can be used as a mitigation for
     * problems described in `IERC20.approve`.
     *
     * Emits an `Approval` event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK increaseAllowance
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender]
          + addedValue
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
            ;
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to `approve` that can be used as a mitigation for
     * problems described in `IERC20.approve`.
     *
     * Emits an `Approval` event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK decreaseAllowance
      @tag assume_completion
      @post msg.sender != address(0)
      @post spender != address(0)
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender]
          - subtractedValue
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
            subtractedValue));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to `transfer`, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a `Transfer` event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK _transfer
      @tag assume_completion
      @pre sender != recipient
      @post sender != address(0)
      @post recipient != address(0)
      @post __post._balances[sender] == _balances[sender] - amount
      @post __post._balances[recipient] == _balances[recipient] + amount
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _balances[sender] = _balances[sender].sub(amount);
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a `Transfer` event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK _mint
      @tag assume_completion
      @post account != address(0)
      @post __post._totalSupply == _totalSupply + amount
      @post __post._balances[account] == _balances[account] + amount
     */
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "ERC20: mint to the zero address");

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

     /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a `Transfer` event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK _burn
      @tag assume_completion
      @post account != address(0)
      @post __post._totalSupply == _totalSupply - value
      @post __post._balances[account] == _balances[account] - value
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

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

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an `Approval` event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _approve
      @tag assume_completion
      @post owner != address(0)
      @post spender != address(0)
      @post __post._allowances[owner][spender] == value
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    /**
     * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See `_burn` and `_approve`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK _burn
      @tag assume_completion
      @post account != address(0)
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply - amount
      @post __post._balances[account] == _balances[account] - amount
      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender]
          - amount
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
    }
}
```

File Ownable.sol

```
pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner
        );

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    /*@CTK Ownable
      @post __post._owner == msg.sender
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK owner
      @post __return == _owner
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK isOwner
      @post __return == (msg.sender == _owner)
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * > Note: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK renounceOwnership
      @tag assume_completion
      @post msg.sender == _owner
      @post __post._owner == address(0)
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK transferOwnership
      @tag assume_completion
      @post msg.sender == _owner
      @post newOwner != address(0)
      @post __post._owner == newOwner
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK _transferOwnership
      @tag assume_completion
      @post newOwner != address(0)
      @post __post._owner == newOwner
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

File ERC20Mintable.sol

```
pragma solidity ^0.5.0;

import "./ERC20.sol";
import "./MinterRole.sol";

/**
 * @dev Extension of `ERC20` that adds a set of accounts with the `MinterRole`,
 * which have permission to mint (create) new tokens as they see fit.
 *
 * At construction, the deployer of the contract is the only minter.
 */
contract ERC20Mintable is ERC20, MinterRole {
    /**
     * @dev See `ERC20._mint`.
     *
     * Requirements:
     *
     * - the caller must have the `MinterRole`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function mint(address account, uint256 amount) public onlyMinter returns (bool) {
        _mint(account, amount);
        return true;
    }
}
```

File SafeMath.sol

```
pragma solidity ^0.5.0;

/**
 * @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 NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SafeMath_add"
      @post (__reverted) == (__has_overflow)
      @post (!(__reverted)) -> ((__return) == ((a) + (b)))
      @post (msg) == (msg__post)
      @post ((((a) + (b)) < (a)) || (((a) + (b)) < (b))) == __reverted
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
    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 NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SafeMath_sub"
      @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
      @post (!(__reverted)) -> ((__return) == ((a) - (b)))
      @post (msg) == (msg__post)
      @post ((a) < (b)) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
    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 NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SafeMath_mul"
      @post (__reverted) == (__has_overflow)
      @post (!(__reverted)) -> ((__return) == ((a) * (b)))
      @post (msg) == (msg__post)
      @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
    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 NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "SafeMath_div"
      @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
      @post (!(__reverted)) -> ((__return) == ((a) / (b)))
      @post (msg) == (msg__post)
      @post ((b) == (0)) == (__reverted)
      @post (__addr_map) == (__addr_map__post)
      @post !(__has_buf_overflow)
      @tag spec
    */
    /*@CTK "SafeMath_div"
      @post ((__reverted) == (false)) -> ((__return) == ((a) / (b)))
    */
    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 NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }
}
```

File ERC20Detailed.sol

```
pragma solidity ^0.5.0;

import "./IERC20.sol";

/**
 * @dev Optional functions from the ERC20 standard.
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    /**
     * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
     * these values are immutable: they can only be set once during
     * construction.
     */
    /*@CTK ERC20Detailed
      @post __post._name == name
      @post __post._symbol == symbol
      @post __post._decimals == decimals
     */
    constructor (string memory name, string memory symbol, uint8 decimals) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
    }

    /**
     * @dev Returns the name of the token.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK name
      @post __return == __post._name
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @dev Returns the symbol of the token, usually a shorter version of the
     * name.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK symbol
      @post __return == _symbol
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @dev Returns the number of decimals used to get its user representation.
     * For example, if `decimals` equals `2`, a balance of `505` tokens should
     * be displayed to a user as `5,05` (`505 / 10 ** 2`).
     *
     * Tokens usually opt for a value of 18, imitating the relationship between
     * Ether and Wei.
     *
     * > Note that this information is only used for _display_ purposes: it in
     * no way affects any of the arithmetic of the contract, including
     * `IERC20.balanceOf` and `IERC20.transfer`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK decimals
      @post __return == _decimals
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}
```

File Pausable.sol

```
pragma solidity ^0.5.0;

import "./PauserRole.sol";

/**
 * @dev Contract module which allows children to implement an emergency stop
 * mechanism that can be triggered by an authorized account.
 *
 * This module is used through inheritance. It will make available the
 * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
 * the functions of your contract. Note that they will not be pausable by
 * simply including this module, only once the modifiers are put in place.
 */
contract Pausable is PauserRole {
    /**
     * @dev Emitted when the pause is triggered by a pauser (`account`).
     */
    event Paused(address account);

    /**
     * @dev Emitted when the pause is lifted by a pauser (`account`).
     */
    event Unpaused(address account);

    bool private _paused;

    /**
     * @dev Initializes the contract in unpaused state. Assigns the Pauser role
     * to the deployer.
     */
    /*@CTK Pausable
      @post !__post._paused
     */
    constructor () internal {
        _paused = false;
    }

    /**
     * @dev Returns true if the contract is paused, and false otherwise.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK paused
      @post __return == _paused
     */
    function paused() public view returns (bool) {
        return _paused;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is not paused.
     */
    modifier whenNotPaused() {
        require(!_paused, "Pausable: paused");
        _;
    }

    /**
     * @dev Modifier to make a function callable only when the contract is paused.
     */
    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    /**
     * @dev Called by a pauser to pause, triggers stopped state.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK pause
      @tag assume_completion
      @post !_paused
      @post __post._paused
     */
    function pause() public onlyPauser whenNotPaused {
        _paused = true;
        emit Paused(msg.sender);
    }

    /**
     * @dev Called by a pauser to unpause, returns to normal state.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK unpause
      @tag assume_completion
      @post _paused
      @post !__post._paused
     */
    function unpause() public onlyPauser whenPaused {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}
```

File Migrations.sol

```
 pragma solidity ^0.4.17;

 contract Migrations {
   address public owner;
   uint public last_completed_migration;

   modifier restricted() {
     if (msg.sender == owner) _;
   }

   /*@CTK Migrations
     @post __post.owner == msg.sender
    */
   function Migrations() public {
     owner = msg.sender;
   }

   /*@CTK setCompleted
     @pre msg.sender == owner
     @post __post.last_completed_migration == completed
    */
   function setCompleted(uint completed) public restricted {
     last_completed_migration = completed;
   }

   function upgrade(address new_address) public restricted {
     Migrations upgraded = Migrations(new_address);
     upgraded.setCompleted(last_completed_migration);
    }
  }
```

File ERC20Burnable.sol

```
pragma solidity ^0.5.0;

import "./ERC20.sol";

/**
 * @dev Extension of `ERC20` that allows token holders to destroy both their own
 * tokens and those that they have an allowance for, in a way that can be
 * recognized off-chain (via event analysis).
 */
contract ERC20Burnable is ERC20 {
    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See `ERC20._burn`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK burn
      @tag assume_completion
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply - amount
      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
     */
    function burn(uint256 amount) public {
        _burn(msg.sender, amount);
    }

    /**
     * @dev See `ERC20._burnFrom`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    /*@CTK burnFrom
      @tag assume_completion
      @post account != address(0)
      @post msg.sender != address(0)
      @post __post._totalSupply == _totalSupply - amount
      @post __post._balances[account] == _balances[account] - amount
      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender]
          - amount
     */
    function burnFrom(address account, uint256 amount) public {
        _burnFrom(account, amount);
    }
}
```

File ERC20Pausable.sol

```
pragma solidity ^0.5.0;

import "./ERC20.sol";
import "./Pausable.sol";

/**
 * @title Pausable token
 * @dev ERC20 with pausable transfers and allowances.
 *
 * Useful if you want to e.g. stop trades until the end of a crowdsale, or have
 * an emergency switch for freezing all token transfers in the event of a large
 * bug.
 */
contract ERC20Pausable is ERC20, Pausable {
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
        return super.transfer(to, value);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function transferFrom(address from, address to, uint256 value) public
        whenNotPaused returns (bool) {
        return super.transferFrom(from, to, value);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    function approve(address spender, uint256 value) public whenNotPaused returns (
        bool) {
        return super.approve(spender, value);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function increaseAllowance(address spender, uint256 addedValue) public
        whenNotPaused returns (bool) {
        return super.increaseAllowance(spender, addedValue);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        whenNotPaused returns (bool) {
        return super.decreaseAllowance(spender, subtractedValue);
    }
}
```

CERTIK

Building Fully Trustworthy Smart Contracts and Blockchain Ecosystems