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

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and HYCON(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

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

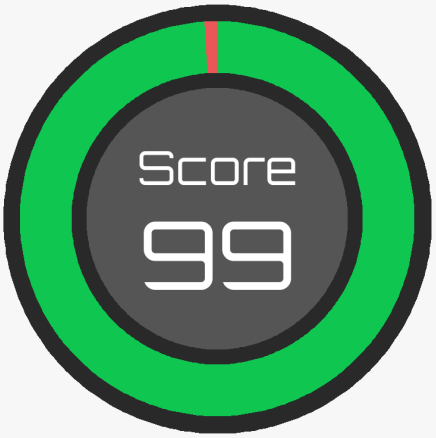
The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilies further down the line.

Testing Summary

PASS

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

Oct 30, 2019



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

```
1 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

```
1 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

```
1 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

```
1 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

```
1 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

```
1 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

```
1  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

```
1 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

```
1 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*

```
30  /*@CTK FAIL "transferFrom to same address"
31      @tag assume_completion
32      @pre from == to
33      @post __post.allowed[from][msg.sender] ==
34  */
```

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

Raw code

```
35  function transferFrom(address from, address to
        ) {
36      balances[from] = balances[from].sub(tokens
37      allowed[from][msg.sender] = allowed[from][
38      balances[to] = balances[to].add(tokens);
39      emit Transfer(from, to, tokens);
40      return true;
41  }
```

Counterexample: This code violates the specification

Initial environment

```
1  Counter Example:
2  Before Execution:
3      Input = {
4          from = 0x0
5          to = 0x0
6          tokens = 0x6c
7      }
8      This = 0
53             balance: 0x0
54         }
55     }
56
```

Post environment

```
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

If method completes, integer overflow would not happen.

30, Oct 2019
6.07 ms

Line 41 in File ERC20.sol

```
41    //@CTK NO_OVERFLOW
```

Line 47-49 in File ERC20.sol

```
47    function totalSupply() public view returns (uint256) {
48        return _totalSupply;
49    }
```

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

```
42    //@CTK NO_BUF_OVERFLOW
```

Line 47-49 in File ERC20.sol

```
47    function totalSupply() public view returns (uint256) {
48        return _totalSupply;
49    }
```

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

```
43    //@CTK NO_ASF
```

Line 47-49 in File ERC20.sol

```
47    function totalSupply() public view returns (uint256) {
48        return _totalSupply;
49    }
```

The code meets the specification.

Formal Verification Request 4

totalSupply

30, Oct 2019
0.51 ms

Line 44-46 in File ERC20.sol

```
44    /*@CTK totalSupply
45      @post __return == _totalSupply
46     */
```

Line 47-49 in File ERC20.sol

```
47    function totalSupply() public view returns (uint256) {
48        return _totalSupply;
49    }
```

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

```
54    //@CTK NO_OVERFLOW
```

Line 60-62 in File ERC20.sol

```
60    function balanceOf(address account) public view returns (uint256) {
61        return _balances[account];
62    }
```

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

```
55    //@CTK NO_BUF_OVERFLOW
```

Line 60-62 in File ERC20.sol

```
60    function balanceOf(address account) public view returns (uint256) {
61        return _balances[account];
62    }
```

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

```
56    //@CTK NO_ASF
```

Line 60-62 in File ERC20.sol

```
60    function balanceOf(address account) public view returns (uint256) {
61        return _balances[account];
62    }
```

The code meets the specification.

Formal Verification Request 8

balanceOf

30, Oct 2019
0.43 ms

Line 57-59 in File ERC20.sol

```
57    /*@CTK balanceOf
58      @post __return == _balances[account]
59     */
```

Line 60-62 in File ERC20.sol

```
60    function balanceOf(address account) public view returns (uint256) {
61        return _balances[account];
62    }
```

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

```
72    //@CTK NO_OVERFLOW
```

Line 82-85 in File ERC20.sol

```
82    function transfer(address recipient, uint256 amount) public returns (bool) {
83        _transfer(msg.sender, recipient, amount);
84        return true;
85    }
```

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

```
73    //@CTK NO_BUF_OVERFLOW
```

Line 82-85 in File ERC20.sol

```
82    function transfer(address recipient, uint256 amount) public returns (bool) {
83        _transfer(msg.sender, recipient, amount);
84        return true;
85    }
```

The code meets the specification.

Formal Verification Request 11

transfer

30, Oct 2019
36.5 ms

Line 74-81 in File ERC20.sol

```
74    /*@CTK transfer
75      @tag assume_completion
76      @pre msg.sender != recipient
77      @post msg.sender != address(0)
78      @post recipient != address(0)
79      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
80      @post __post._balances[recipient] == _balances[recipient] + amount
81     */
```

Line 82-85 in File ERC20.sol

```
82    function transfer(address recipient, uint256 amount) public returns (bool) {
83        _transfer(msg.sender, recipient, amount);
84        return true;
85    }
```

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

```
90    //@CTK NO_OVERFLOW
```

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 13

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

30, Oct 2019
0.47 ms

Line 91 in File ERC20.sol

```
91    //@CTK NO_BUF_OVERFLOW
```

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 14

Method will not encounter an assertion failure.

30, Oct 2019
0.45 ms

Line 92 in File ERC20.sol

```
92    //@CTK NO_ASF
```

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 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

```
109     //@CTK NO_ASF
```

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 19

approve

30, Oct 2019
2.76 ms

Line 110-115 in File ERC20.sol

```
110     /*@CTK approve
111       @tag assume_completion
112       @post msg.sender != address(0)
113       @post spender != address(0)
114       @post __post._allowances[msg.sender][spender] == value
115      */
```

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 20

If method completes, integer overflow would not happen.

30, Oct 2019
108.34 ms

Line 133 in File ERC20.sol

```
133     //@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));
```

```
147         return true;
148     }
```

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

```
162     //@CTK NO_OVERFLOW
```

Line 170-173 in File ERC20.sol

```
170     function increaseAllowance(address spender, uint256 addedValue) public returns (
            bool) {
171         _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
                ;
172         return true;
173     }
```

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

```
163     //@CTK NO_BUF_OVERFLOW
```

Line 170-173 in File ERC20.sol

```
170     function increaseAllowance(address spender, uint256 addedValue) public returns (
            bool) {
171         _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
                ;
172         return true;
173     }
```

The code meets the specification.

Formal Verification Request 25

increaseAllowance

30, Oct 2019
4.74 ms

Line 164-169 in File ERC20.sol

```
164     /*@CTK increaseAllowance
165       @tag assume_completion
166       @post msg.sender != address(0)
167       @post spender != address(0)
168       @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
              ] + addedValue
169      */
```

Line 170-173 in File ERC20.sol

```
170     function increaseAllowance(address spender, uint256 addedValue) public returns (
            bool) {
171         _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
                ;
172         return true;
173     }
```

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

```
189     //@CTK NO_OVERFLOW
```

Line 197-200 in File ERC20.sol

```
197     function decreaseAllowance(address spender, uint256 subtractedValue) public
            returns (bool) {
198         _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
                subtractedValue));
199         return true;
200     }
```

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

```
190     //@CTK NO_BUF_OVERFLOW
```

Line 197-200 in File ERC20.sol

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

```
199         return true;
200     }
```

The code meets the specification.

Formal Verification Request 28

decreaseAllowance

30, Oct 2019
2.96 ms

Line 191-196 in File ERC20.sol

```
191     /*@CTK decreaseAllowance
192       @tag assume_completion
193       @post msg.sender != address(0)
194       @post spender != address(0)
195       @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
              ] - subtractedValue
196      */
```

Line 197-200 in File ERC20.sol

```
197     function decreaseAllowance(address spender, uint256 subtractedValue) public
            returns (bool) {
198         _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
                subtractedValue));
199         return true;
200     }
```

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

```
216     //@CTK NO_OVERFLOW
```

Line 226-233 in File ERC20.sol

```
226     function _transfer(address sender, address recipient, uint256 amount) internal {
227         require(sender != address(0), "ERC20: transfer from the zero address");
228         require(recipient != address(0), "ERC20: transfer to the zero address");
229 
230         _balances[sender] = _balances[sender].sub(amount);
231         _balances[recipient] = _balances[recipient].add(amount);
232         emit Transfer(sender, recipient, amount);
233     }
```

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

```
217     //@CTK NO_BUF_OVERFLOW
```

Line 226-233 in File ERC20.sol

```
226     function _transfer(address sender, address recipient, uint256 amount) internal {
227         require(sender != address(0), "ERC20: transfer from the zero address");
228         require(recipient != address(0), "ERC20: transfer to the zero address");
229 
230         _balances[sender] = _balances[sender].sub(amount);
231         _balances[recipient] = _balances[recipient].add(amount);
232         emit Transfer(sender, recipient, amount);
233     }
```

The code meets the specification.

Formal Verification Request 31

_transfer

30, Oct 2019
43.52 ms

Line 218-225 in File ERC20.sol

```
218     /*@CTK _transfer
219       @tag assume_completion
220       @pre sender != recipient
221       @post sender != address(0)
222       @post recipient != address(0)
223       @post __post._balances[sender] == _balances[sender] - amount
224       @post __post._balances[recipient] == _balances[recipient] + amount
225      */
```

Line 226-233 in File ERC20.sol

```
226     function _transfer(address sender, address recipient, uint256 amount) internal {
227         require(sender != address(0), "ERC20: transfer from the zero address");
228         require(recipient != address(0), "ERC20: transfer to the zero address");
229 
230         _balances[sender] = _balances[sender].sub(amount);
231         _balances[recipient] = _balances[recipient].add(amount);
232         emit Transfer(sender, recipient, amount);
233     }
```

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

```
244     //@CTK NO_OVERFLOW
```

Line 252-258 in File ERC20.sol

```
252     function _mint(address account, uint256 amount) internal {
253         require(account != address(0), "ERC20: mint to the zero address");
254 
255         _totalSupply = _totalSupply.add(amount);
256         _balances[account] = _balances[account].add(amount);
257         emit Transfer(address(0), account, amount);
258     }
```

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

```
245     //@CTK NO_BUF_OVERFLOW
```

Line 252-258 in File ERC20.sol

```
252     function _mint(address account, uint256 amount) internal {
253         require(account != address(0), "ERC20: mint to the zero address");
254 
255         _totalSupply = _totalSupply.add(amount);
256         _balances[account] = _balances[account].add(amount);
257         emit Transfer(address(0), account, amount);
258     }
```

The code meets the specification.

Formal Verification Request 34

_mint

30, Oct 2019
12.45 ms

Line 246-251 in File ERC20.sol

```
246     /*@CTK _mint
247       @tag assume_completion
248       @post account != address(0)
249       @post __post._totalSupply == _totalSupply + amount
250       @post __post._balances[account] == _balances[account] + amount
251      */
```

Line 252-258 in File ERC20.sol

```
252     function _mint(address account, uint256 amount) internal {
253         require(account != address(0), "ERC20: mint to the zero address");
254 
255         _totalSupply = _totalSupply.add(amount);
256         _balances[account] = _balances[account].add(amount);
257         emit Transfer(address(0), account, amount);
258     }
```

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

```
271     //@CTK NO_OVERFLOW
```

Line 279-285 in File ERC20.sol

```
279     function _burn(address account, uint256 value) internal {
280         require(account != address(0), "ERC20: burn from the zero address");
281 
282         _totalSupply = _totalSupply.sub(value);
283         _balances[account] = _balances[account].sub(value);
284         emit Transfer(account, address(0), value);
285     }
```

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

```
272     //@CTK NO_BUF_OVERFLOW
```

Line 279-285 in File ERC20.sol

```
279     function _burn(address account, uint256 value) internal {
280         require(account != address(0), "ERC20: burn from the zero address");
281 
282         _totalSupply = _totalSupply.sub(value);
283         _balances[account] = _balances[account].sub(value);
284         emit Transfer(account, address(0), value);
285     }
```

The code meets the specification.

Formal Verification Request 37

_burn

30, Oct 2019
16.6 ms

Line 273-278 in File ERC20.sol

```
273     /*@CTK _burn
274       @tag assume_completion
275       @post account != address(0)
276       @post __post._totalSupply == _totalSupply - value
277       @post __post._balances[account] == _balances[account] - value
278      */
```

Line 279-285 in File ERC20.sol

```
279     function _burn(address account, uint256 value) internal {
280         require(account != address(0), "ERC20: burn from the zero address");
281 
282         _totalSupply = _totalSupply.sub(value);
283         _balances[account] = _balances[account].sub(value);
284         emit Transfer(account, address(0), value);
285     }
```

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

```
300     //@CTK NO_OVERFLOW
```

Line 309-315 in File ERC20.sol

```
309     function _approve(address owner, address spender, uint256 value) internal {
310         require(owner != address(0), "ERC20: approve from the zero address");
311         require(spender != address(0), "ERC20: approve to the zero address");
312 
313         _allowances[owner][spender] = value;
314         emit Approval(owner, spender, value);
315     }
```

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

```
301     //@CTK NO_BUF_OVERFLOW
```

Line 309-315 in File ERC20.sol

```
309     function _approve(address owner, address spender, uint256 value) internal {
310         require(owner != address(0), "ERC20: approve from the zero address");
311         require(spender != address(0), "ERC20: approve to the zero address");
312 
313         _allowances[owner][spender] = value;
314         emit Approval(owner, spender, value);
315     }
```

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

```
302     //@CTK NO_ASF
```

Line 309-315 in File ERC20.sol

```
309     function _approve(address owner, address spender, uint256 value) internal {
310         require(owner != address(0), "ERC20: approve from the zero address");
311         require(spender != address(0), "ERC20: approve to the zero address");
312 
313         _allowances[owner][spender] = value;
314         emit Approval(owner, spender, value);
315     }
```

The code meets the specification.

Formal Verification Request 41

_approve

30, Oct 2019
2.32 ms

Line 303-308 in File ERC20.sol

```
303     /*@CTK _approve
304       @tag assume_completion
305       @post owner != address(0)
306       @post spender != address(0)
307       @post __post._allowances[owner][spender] == value
308      */
```

Line 309-315 in File ERC20.sol

```
309     function _approve(address owner, address spender, uint256 value) internal {
310         require(owner != address(0), "ERC20: approve from the zero address");
311         require(spender != address(0), "ERC20: approve to the zero address");
312 
313         _allowances[owner][spender] = value;
314         emit Approval(owner, spender, value);
315     }
```

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

```
323     //@CTK NO_OVERFLOW
```

Line 333-336 in File ERC20.sol

```
333     function _burnFrom(address account, uint256 amount) internal {
334         _burn(account, amount);
335         _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
336     }
```

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

```
324     //@CTK NO_BUF_OVERFLOW
```

Line 333-336 in File ERC20.sol

```
333     function _burnFrom(address account, uint256 amount) internal {
334         _burn(account, amount);
335         _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
336     }
```

The code meets the specification.

Formal Verification Request 44

_burn

30, Oct 2019
64.18 ms

Line 325-332 in File ERC20.sol

```
325     /*@CTK _burn
326       @tag assume_completion
327       @post account != address(0)
328       @post msg.sender != address(0)
329       @post __post._totalSupply == _totalSupply - amount
330       @post __post._balances[account] == _balances[account] - amount
331       @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender
              ] - amount
332      */
```

Line 333-336 in File ERC20.sol

```
333     function _burnFrom(address account, uint256 amount) internal {
334         _burn(account, amount);
335         _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
336     }
```

The code meets the specification.

Formal Verification Request 45

Ownable

30, Oct 2019
7.02 ms

Line 20-22 in File Ownable.sol

```
20      /*@CTK Ownable
21        @post __post._owner == msg.sender
22       */
```

Line 23-26 in File Ownable.sol

```
23      constructor () internal {
24          _owner = msg.sender;
25          emit OwnershipTransferred(address(0), _owner);
26      }
```

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

```
31    //@CTK NO_OVERFLOW
```

Line 37-39 in File Ownable.sol

```
37    function owner() public view returns (address) {
38        return _owner;
39    }
```

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

```
32    //@CTK NO_BUF_OVERFLOW
```

Line 37-39 in File Ownable.sol

```
37    function owner() public view returns (address) {
38        return _owner;
39    }
```

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

```
33    //@CTK NO_ASF
```

Line 37-39 in File Ownable.sol

```
37    function owner() public view returns (address) {
38        return _owner;
39    }
```

The code meets the specification.

Formal Verification Request 49

owner

30, Oct 2019
0.46 ms

Line 34-36 in File Ownable.sol

```
34    /*@CTK owner
35      @post __return == _owner
36     */
```

Line 37-39 in File Ownable.sol

```
37    function owner() public view returns (address) {
38        return _owner;
39    }
```

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

```
52    //@CTK NO_OVERFLOW
```

Line 58-60 in File Ownable.sol

```
58    function isOwner() public view returns (bool) {
59        return msg.sender == _owner;
60    }
```

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

```
53    //@CTK NO_BUF_OVERFLOW
```

Line 58-60 in File Ownable.sol

```
58    function isOwner() public view returns (bool) {
59        return msg.sender == _owner;
60    }
```

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

```
54    //@CTK NO_ASF
```

Line 58-60 in File Ownable.sol

```
58    function isOwner() public view returns (bool) {
59        return msg.sender == _owner;
60    }
```

The code meets the specification.

Formal Verification Request 53

isOwner

30, Oct 2019
0.81 ms

Line 55-57 in File Ownable.sol

```
55    /*@CTK isOwner
56      @post __return == (msg.sender == _owner)
57     */
```

Line 58-60 in File Ownable.sol

```
58    function isOwner() public view returns (bool) {
59        return msg.sender == _owner;
60    }
```

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

```
69    //@CTK NO_OVERFLOW
```

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 55

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

30, Oct 2019
0.69 ms

Line 70 in File Ownable.sol

```
70    //@CTK NO_BUF_OVERFLOW
```

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 56

Method will not encounter an assertion failure.

30, Oct 2019
0.66 ms

Line 71 in File Ownable.sol

```
71    //@CTK NO_ASF
```

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 57

renounceOwnership

30, Oct 2019
1.13 ms

Line 72-76 in File Ownable.sol

```
72    /*@CTK renounceOwnership
73      @tag assume_completion
74      @post msg.sender == _owner
75      @post __post._owner == address(0)
76     */
```

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

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

The code meets the specification.

Formal Verification Request 61

transferOwnership

30, Oct 2019
5.0 ms

Line 89-94 in File Ownable.sol

```
89    /*@CTK transferOwnership
90      @tag assume_completion
91      @post msg.sender == _owner
92      @post newOwner != address(0)
93      @post __post._owner == newOwner
94     */
```

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 62

If method completes, integer overflow would not happen.

30, Oct 2019
0.89 ms

Line 102 in File Ownable.sol

```
102   //@CTK NO_OVERFLOW
```

Line 110-114 in File Ownable.sol

```
110   function _transferOwnership(address newOwner) internal {
111       require(newOwner != address(0), "Ownable: new owner is the zero address");
112       emit OwnershipTransferred(_owner, newOwner);
113       _owner = newOwner;
114   }
```

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

```
103     //@CTK NO_BUF_OVERFLOW
```

Line 110-114 in File Ownable.sol

```
110     function _transferOwnership(address newOwner) internal {
111         require(newOwner != address(0), "Ownable: new owner is the zero address");
112         emit OwnershipTransferred(_owner, newOwner);
113         _owner = newOwner;
114     }
```

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

```
104     //@CTK NO_ASF
```

Line 110-114 in File Ownable.sol

```
110     function _transferOwnership(address newOwner) internal {
111         require(newOwner != address(0), "Ownable: new owner is the zero address");
112         emit OwnershipTransferred(_owner, newOwner);
113         _owner = newOwner;
114     }
```

The code meets the specification.

Formal Verification Request 65

_transferOwnership

30, Oct 2019
1.57 ms

Line 105-109 in File Ownable.sol

```
105     /*@CTK _transferOwnership
106       @tag assume_completion
107       @post newOwner != address(0)
108       @post __post._owner == newOwner
109      */
```

Line 110-114 in File Ownable.sol

```
110     function _transferOwnership(address newOwner) internal {
111         require(newOwner != address(0), "Ownable: new owner is the zero address");
112         emit OwnershipTransferred(_owner, newOwner);
113         _owner = newOwner;
114     }
```

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

```
20     //@CTK NO_OVERFLOW
```

Line 22-25 in File ERC20Mintable.sol

```
22     function mint(address account, uint256 amount) public onlyMinter returns (bool) {
23         _mint(account, amount);
24         return true;
25     }
```

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

```
21     //@CTK NO_BUF_OVERFLOW
```

Line 22-25 in File ERC20Mintable.sol

```
22     function mint(address account, uint256 amount) public onlyMinter returns (bool) {
23         _mint(account, amount);
24         return true;
25     }
```

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

```
26    //@CTK NO_OVERFLOW
```

Line 38-43 in File SafeMath.sol

```
38    function add(uint256 a, uint256 b) internal pure returns (uint256) {
39        uint256 c = a + b;
40        require(c >= a, "SafeMath: addition overflow");
41
42        return c;
43    }
```

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

```
27    //@CTK NO_BUF_OVERFLOW
```

Line 38-43 in File SafeMath.sol

```
38    function add(uint256 a, uint256 b) internal pure returns (uint256) {
39        uint256 c = a + b;
40        require(c >= a, "SafeMath: addition overflow");
41
42        return c;
43    }
```

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

```
28    //@CTK NO_ASF
```

Line 38-43 in File SafeMath.sol

```
38    function add(uint256 a, uint256 b) internal pure returns (uint256) {
39        uint256 c = a + b;
40        require(c >= a, "SafeMath: addition overflow");
41
42        return c;
43    }
```

The code meets the specification.

Formal Verification Request 71

SafeMath_add

30, Oct 2019
2.55 ms

Line 29-37 in File SafeMath.sol

```
29    /*@CTK "SafeMath_add"
30      @post (__reverted) == (__has_overflow)
31      @post (!(__reverted)) -> ((__return) == ((a) + (b)))
32      @post (msg) == (msg__post)
33      @post ((((a) + (b)) < (a)) || (((a) + (b)) < (b))) == __reverted
34      @post (__addr_map) == (__addr_map__post)
35      @post !(__has_buf_overflow)
36      @tag spec
37    */
```

Line 38-43 in File SafeMath.sol

```
38    function add(uint256 a, uint256 b) internal pure returns (uint256) {
39        uint256 c = a + b;
40        require(c >= a, "SafeMath: addition overflow");
41
42        return c;
43    }
```

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

```
54    //@CTK NO_OVERFLOW
```

Line 66-71 in File SafeMath.sol

```
66    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
67        require(b <= a, "SafeMath: subtraction overflow");
68        uint256 c = a - b;
69
70        return c;
71    }
```

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

```
55    //@CTK NO_BUF_OVERFLOW
```

Line 66-71 in File SafeMath.sol

```
66    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
67        require(b <= a, "SafeMath: subtraction overflow");
68        uint256 c = a - b;
69
70        return c;
71    }
```

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

```
56    //@CTK NO_ASF
```

Line 66-71 in File SafeMath.sol

```
66    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
67        require(b <= a, "SafeMath: subtraction overflow");
68        uint256 c = a - b;
69
70        return c;
71    }
```

The code meets the specification.

Formal Verification Request 75

SafeMath_sub

30, Oct 2019
1.45 ms

Line 57-65 in File SafeMath.sol

```
57    /*@CTK "SafeMath_sub"
58      @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
59      @post (!(__reverted)) -> ((__return) == ((a) - (b)))
60      @post (msg) == (msg__post)
61      @post ((a) < (b)) == (__reverted)
62      @post (__addr_map) == (__addr_map__post)
63      @post !(__has_buf_overflow)
64      @tag spec
65    */
```

Line 66-71 in File SafeMath.sol

```
66      function sub(uint256 a, uint256 b) internal pure returns (uint256) {
67          require(b <= a, "SafeMath: subtraction overflow");
68          uint256 c = a - b;
69
70          return c;
71      }
```

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

```
82      //@CTK NO_OVERFLOW
```

Line 94-106 in File SafeMath.sol

```
94      function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95          // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96          // benefit is lost if 'b' is also tested.
97          // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
98          if (a == 0) {
99              return 0;
100         }
101
102         uint256 c = a * b;
103         require(c / a == b, "SafeMath: multiplication overflow");
104
105         return c;
106     }
```

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

```
83      //@CTK NO_BUF_OVERFLOW
```

Line 94-106 in File SafeMath.sol

```
94      function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95          // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96          // benefit is lost if 'b' is also tested.
97          // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
98          if (a == 0) {
99              return 0;
```

```
100         }
101
102         uint256 c = a * b;
103         require(c / a == b, "SafeMath: multiplication overflow");
104
105         return c;
106     }
```

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

```
84      //@CTK NO_ASF
```

Line 94-106 in File SafeMath.sol

```
94      function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95          // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96          // benefit is lost if 'b' is also tested.
97          // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
98          if (a == 0) {
99              return 0;
100         }
101
102         uint256 c = a * b;
103         require(c / a == b, "SafeMath: multiplication overflow");
104
105         return c;
106     }
```

The code meets the specification.

Formal Verification Request 79

SafeMath_mul

30, Oct 2019
208.82 ms

Line 85-93 in File SafeMath.sol

```
85      /*@CTK "SafeMath_mul"
86        @post (__reverted) == (__has_overflow)
87        @post (!(__reverted)) -> ((__return) == ((a) * (b)))
88        @post (msg) == (msg__post)
89        @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
90        @post (__addr_map) == (__addr_map__post)
91        @post !(__has_buf_overflow)
92        @tag spec
93      */
```

Line 94-106 in File SafeMath.sol

```
94    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96        // benefit is lost if 'b' is also tested.
97        // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
98        if (a == 0) {
99            return 0;
100       }
101
102       uint256 c = a * b;
103       require(c / a == b, "SafeMath: multiplication overflow");
104
105       return c;
106   }
```

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

```
119   //@CTK NO_OVERFLOW
```

Line 134-141 in File SafeMath.sol

```
134   function div(uint256 a, uint256 b) internal pure returns (uint256) {
135       // Solidity only automatically asserts when dividing by 0
136       require(b > 0, "SafeMath: division by zero");
137       uint256 c = a / b;
138       // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
140       return c;
141   }
```

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

```
120   //@CTK NO_BUF_OVERFLOW
```

Line 134-141 in File SafeMath.sol

```
134     function div(uint256 a, uint256 b) internal pure returns (uint256) {
135         // Solidity only automatically asserts when dividing by 0
136         require(b > 0, "SafeMath: division by zero");
137         uint256 c = a / b;
138         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
140         return c;
141     }
```

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

```
121     //@CTK NO_ASF
```

Line 134-141 in File SafeMath.sol

```
134     function div(uint256 a, uint256 b) internal pure returns (uint256) {
135         // Solidity only automatically asserts when dividing by 0
136         require(b > 0, "SafeMath: division by zero");
137         uint256 c = a / b;
138         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
140         return c;
141     }
```

The code meets the specification.

Formal Verification Request 83

SafeMath_div

30, Oct 2019
1.73 ms

Line 122-130 in File SafeMath.sol

```
122     /*@CTK "SafeMath_div"
123       @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
124       @post (!(__reverted)) -> ((__return) == ((a) / (b)))
125       @post (msg) == (msg__post)
126       @post ((b) == (0)) == (__reverted)
127       @post (__addr_map) == (__addr_map__post)
128       @post !(__has_buf_overflow)
129       @tag spec
130     */
```

Line 134-141 in File SafeMath.sol

```
134     function div(uint256 a, uint256 b) internal pure returns (uint256) {
135         // Solidity only automatically asserts when dividing by 0
136         require(b > 0, "SafeMath: division by zero");
137         uint256 c = a / b;
138         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
140         return c;
141     }
```

The code meets the specification.

Formal Verification Request 84

SafeMath_div

30, Oct 2019
1.73 ms

Line 131-133 in File SafeMath.sol

```
131     /*@CTK "SafeMath_div"
132       @post ((__reverted) == (false)) -> ((__return) == ((a) / (b)))
133     */
```

Line 134-141 in File SafeMath.sol

```
134     function div(uint256 a, uint256 b) internal pure returns (uint256) {
135         // Solidity only automatically asserts when dividing by 0
136         require(b > 0, "SafeMath: division by zero");
137         uint256 c = a / b;
138         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
140         return c;
141     }
```

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

```
154     //@CTK NO_OVERFLOW
```

Line 157-160 in File SafeMath.sol

```
157     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
158         require(b != 0, "SafeMath: modulo by zero");
159         return a % b;
160     }
```

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

```
155     //@CTK NO_BUF_OVERFLOW
```

Line 157-160 in File SafeMath.sol

```
157     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
158         require(b != 0, "SafeMath: modulo by zero");
159         return a % b;
160     }
```

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

```
156     //@CTK NO_ASF
```

Line 157-160 in File SafeMath.sol

```
157     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
158         require(b != 0, "SafeMath: modulo by zero");
159         return a % b;
160     }
```

The code meets the specification.

Formal Verification Request 88

ERC20Detailed

30, Oct 2019
17.43 ms

Line 18-22 in File ERC20Detailed.sol

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

Line 23-27 in File ERC20Detailed.sol

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

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

```
32      //@CTK NO_OVERFLOW
```

Line 38-40 in File ERC20Detailed.sol

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

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

```
33      //@CTK NO_BUF_OVERFLOW
```

Line 38-40 in File ERC20Detailed.sol

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

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

```
34      //@CTK NO_ASF
```

Line 38-40 in File ERC20Detailed.sol

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

The code meets the specification.

Formal Verification Request 92

name

30, Oct 2019
1.04 ms

Line 35-37 in File ERC20Detailed.sol

```
35    /*@CTK name
36      @post __return == __post._name
37     */
```

Line 38-40 in File ERC20Detailed.sol

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

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

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

Line 52-54 in File ERC20Detailed.sol

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

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

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

Line 52-54 in File ERC20Detailed.sol

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

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

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

Line 52-54 in File ERC20Detailed.sol

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

The code meets the specification.

Formal Verification Request 96

symbol

30, Oct 2019
0.8 ms

Line 49-51 in File ERC20Detailed.sol

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

Line 52-54 in File ERC20Detailed.sol

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

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

```
68    //@CTK NO_OVERFLOW
```

Line 74-76 in File ERC20Detailed.sol

```
74    function decimals() public view returns (uint8) {
75        return _decimals;
76    }
```

✅ 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

```
69    //@CTK NO_BUF_OVERFLOW
```

Line 74-76 in File ERC20Detailed.sol

```
74    function decimals() public view returns (uint8) {
75        return _decimals;
76    }
```

✅ 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

```
70    //@CTK NO_ASF
```

Line 74-76 in File ERC20Detailed.sol

```
74    function decimals() public view returns (uint8) {
75        return _decimals;
76    }
```

✅ The code meets the specification.

Formal Verification Request 100

decimals

📅 30, Oct 2019
⏱ 0.57 ms

Line 71-73 in File ERC20Detailed.sol

```
71    /*@CTK decimals
72      @post __return == _decimals
73     */
```

Line 74-76 in File ERC20Detailed.sol

```
74    function decimals() public view returns (uint8) {
75        return _decimals;
76    }
```

The code meets the specification.

Formal Verification Request 101

Pausable

30, Oct 2019
7.44 ms

Line 31-33 in File Pausable.sol

```
31    /*@CTK Pausable
32      @post !__post._paused
33     */
```

Line 34-36 in File Pausable.sol

```
34    constructor () internal {
35        _paused = false;
36    }
```

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

```
41    //@CTK NO_OVERFLOW
```

Line 47-49 in File Pausable.sol

```
47    function paused() public view returns (bool) {
48        return _paused;
49    }
```

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

```
42    //@CTK NO_BUF_OVERFLOW
```

Line 47-49 in File Pausable.sol

```
47    function paused() public view returns (bool) {
48        return _paused;
49    }
```

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

```
43    //@CTK NO_ASF
```

Line 47-49 in File Pausable.sol

```
47    function paused() public view returns (bool) {
48        return _paused;
49    }
```

The code meets the specification.

Formal Verification Request 105

paused

30, Oct 2019
0.55 ms

Line 44-46 in File Pausable.sol

```
44    /*@CTK paused
45      @post __return == _paused
46     */
```

Line 47-49 in File Pausable.sol

```
47    function paused() public view returns (bool) {
48        return _paused;
49    }
```

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

```
70    //@CTK NO_OVERFLOW
```

Line 78-81 in File Pausable.sol

```
78    function pause() public onlyPauser whenNotPaused {
79        _paused = true;
80        emit Paused(msg.sender);
81    }
```

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

```
71    //@CTK NO_BUF_OVERFLOW
```

Line 78-81 in File Pausable.sol

```
78    function pause() public onlyPauser whenNotPaused {
79        _paused = true;
80        emit Paused(msg.sender);
81    }
```

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

```
72    //@CTK NO_ASF
```

Line 78-81 in File Pausable.sol

```
78    function pause() public onlyPauser whenNotPaused {
79        _paused = true;
80        emit Paused(msg.sender);
81    }
```

The code meets the specification.

Formal Verification Request 109

pause

30, Oct 2019
4.42 ms

Line 73-77 in File Pausable.sol

```
73    /*@CTK pause
74      @tag assume_completion
75      @post !_paused
76      @post __post._paused
77     */
```

Line 78-81 in File Pausable.sol

```
78    function pause() public onlyPauser whenNotPaused {
79        _paused = true;
80        emit Paused(msg.sender);
81    }
```

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

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

Line 94-97 in File Pausable.sol

```
94    function unpause() public onlyPauser whenPaused {
95        _paused = false;
96        emit Unpaused(msg.sender);
97    }
```

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

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

Line 94-97 in File Pausable.sol

```
94      function unpause() public onlyPauser whenPaused {
95          _paused = false;
96          emit Unpaused(msg.sender);
97      }
```

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

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

Line 94-97 in File Pausable.sol

```
94      function unpause() public onlyPauser whenPaused {
95          _paused = false;
96          emit Unpaused(msg.sender);
97      }
```

The code meets the specification.

Formal Verification Request 113

unpause

30, Oct 2019
3.44 ms

Line 89-93 in File Pausable.sol

```
89      /*@CTK unpause
90        @tag assume_completion
91        @post _paused
92        @post !__post._paused
93       */
```

Line 94-97 in File Pausable.sol

```
94      function unpause() public onlyPauser whenPaused {
95          _paused = false;
96          emit Unpaused(msg.sender);
97      }
```

The code meets the specification.

Formal Verification Request 114

Migrations

30, Oct 2019
6.64 ms

Line 11-13 in File Migrations.sol

```
11  /*@CTK Migrations
12    @post __post.owner == msg.sender
13   */
```

Line 14-16 in File Migrations.sol

```
14  function Migrations() public {
15    owner = msg.sender;
16  }
```

The code meets the specification.

Formal Verification Request 115

setCompleted

30, Oct 2019
15.23 ms

Line 18-21 in File Migrations.sol

```
18  /*@CTK setCompleted
19    @pre msg.sender == owner
20    @post __post.last_completed_migration == completed
21   */
```

Line 22-24 in File Migrations.sol

```
22  function setCompleted(uint completed) public restricted {
23    last_completed_migration = completed;
24  }
```

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

```
16  //@CTK NO_OVERFLOW
```

Line 24-26 in File ERC20Burnable.sol

```
24  function burn(uint256 amount) public {
25      _burn(msg.sender, amount);
26  }
```

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

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

Line 24-26 in File ERC20Burnable.sol

```
24    function burn(uint256 amount) public {
25        _burn(msg.sender, amount);
26    }
```

The code meets the specification.

Formal Verification Request 118

burn

30, Oct 2019
17.9 ms

Line 18-23 in File ERC20Burnable.sol

```
18    /*@CTK burn
19      @tag assume_completion
20      @post msg.sender != address(0)
21      @post __post._totalSupply == _totalSupply - amount
22      @post __post._balances[msg.sender] == _balances[msg.sender] - amount
23     */
```

Line 24-26 in File ERC20Burnable.sol

```
24    function burn(uint256 amount) public {
25        _burn(msg.sender, amount);
26    }
```

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

```
31    //@CTK NO_OVERFLOW
```

Line 41-43 in File ERC20Burnable.sol

```
41    function burnFrom(address account, uint256 amount) public {
42        _burnFrom(account, amount);
43    }
```

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

```
32    //@CTK NO_BUF_OVERFLOW
```

Line 41-43 in File ERC20Burnable.sol

```
41    function burnFrom(address account, uint256 amount) public {
42        _burnFrom(account, amount);
43    }
```

The code meets the specification.

Formal Verification Request 121

burnFrom

30, Oct 2019
49.99 ms

Line 33-40 in File ERC20Burnable.sol

```
33    /*@CTK burnFrom
34      @tag assume_completion
35      @post account != address(0)
36      @post msg.sender != address(0)
37      @post __post._totalSupply == _totalSupply - amount
38      @post __post._balances[account] == _balances[account] - amount
39      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender
            ] - amount
40     */
```

Line 41-43 in File ERC20Burnable.sol

```
41    function burnFrom(address account, uint256 amount) public {
42        _burnFrom(account, amount);
43    }
```

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

```
15      //@CTK NO_OVERFLOW
```

Line 17-19 in File ERC20Pausable.sol

```
17      function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
18          return super.transfer(to, value);
19      }
```

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

```
16      //@CTK NO_BUF_OVERFLOW
```

Line 17-19 in File ERC20Pausable.sol

```
17      function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
18          return super.transfer(to, value);
19      }
```

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

```
21      //@CTK NO_OVERFLOW
```

Line 23-25 in File ERC20Pausable.sol

```
23      function transferFrom(address from, address to, uint256 value) public
            whenNotPaused returns (bool) {
24          return super.transferFrom(from, to, value);
25      }
```

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

```
22    //@CTK NO_BUF_OVERFLOW
```

Line 23-25 in File ERC20Pausable.sol

```
23    function transferFrom(address from, address to, uint256 value) public
          whenNotPaused returns (bool) {
24        return super.transferFrom(from, to, value);
25    }
```

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

```
27    //@CTK NO_OVERFLOW
```

Line 30-32 in File ERC20Pausable.sol

```
30    function approve(address spender, uint256 value) public whenNotPaused returns (
          bool) {
31        return super.approve(spender, value);
32    }
```

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

```
28    //@CTK NO_BUF_OVERFLOW
```

Line 30-32 in File ERC20Pausable.sol

```
30    function approve(address spender, uint256 value) public whenNotPaused returns (
          bool) {
31        return super.approve(spender, value);
32    }
```

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

```
29    //@CTK NO_ASF
```

Line 30-32 in File ERC20Pausable.sol

```
30    function approve(address spender, uint256 value) public whenNotPaused returns (
          bool) {
31        return super.approve(spender, value);
32    }
```

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

```
34    //@CTK NO_OVERFLOW
```

Line 36-38 in File ERC20Pausable.sol

```
36    function increaseAllowance(address spender, uint256 addedValue) public
          whenNotPaused returns (bool) {
37        return super.increaseAllowance(spender, addedValue);
38    }
```

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

```
35    //@CTK NO_BUF_OVERFLOW
```

Line 36-38 in File ERC20Pausable.sol

```
36    function increaseAllowance(address spender, uint256 addedValue) public
          whenNotPaused returns (bool) {
37        return super.increaseAllowance(spender, addedValue);
38    }
```

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

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

Line 42-44 in File ERC20Pausable.sol

```
42    function decreaseAllowance(address spender, uint256 subtractedValue) public
          whenNotPaused returns (bool) {
43        return super.decreaseAllowance(spender, subtractedValue);
44    }
```

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

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

Line 42-44 in File ERC20Pausable.sol

```
42    function decreaseAllowance(address spender, uint256 subtractedValue) public
          whenNotPaused returns (bool) {
43        return super.decreaseAllowance(spender, subtractedValue);
44    }
```

The code meets the specification.

Source Code with CertiK Labels

File ERC20.sol

```
1  pragma solidity ^0.5.0;
2
3  import "./IERC20.sol";
4  import "./SafeMath.sol";
5
6  /**
7   * @dev Implementation of the `IERC20` interface.
8   *
9   * This implementation is agnostic to the way tokens are created. This means
10  * that a supply mechanism has to be added in a derived contract using `_mint`.
11  * For a generic mechanism see `ERC20Mintable`.
12  *
13  * *For a detailed writeup see our guide [How to implement supply
14  * mechanisms](https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-
        mechanisms/226).*
15  *
16  * We have followed general OpenZeppelin guidelines: functions revert instead
17  * of returning `false` on failure. This behavior is nonetheless conventional
18  * and does not conflict with the expectations of ERC20 applications.
19  *
20  * Additionally, an `Approval` event is emitted on calls to `transferFrom`.
21  * This allows applications to reconstruct the allowance for all accounts just
22  * by listening to said events. Other implementations of the EIP may not emit
23  * these events, as it isn't required by the specification.
24  *
25  * Finally, the non-standard `decreaseAllowance` and `increaseAllowance`
26  * functions have been added to mitigate the well-known issues around setting
27  * allowances. See `IERC20.approve`.
28  */
29 contract ERC20 is IERC20 {
30     using SafeMath for uint256;
31
32     mapping (address => uint256) private _balances;
33
34     mapping (address => mapping (address => uint256)) private _allowances;
35
36     uint256 private _totalSupply;
37
38     /**
39      * @dev See `IERC20.totalSupply`.
40      */
41     //@CTK NO_OVERFLOW
42     //@CTK NO_BUF_OVERFLOW
43     //@CTK NO_ASF
44     /*@CTK totalSupply
45       @post __return == _totalSupply
46      */
47     function totalSupply() public view returns (uint256) {
48         return _totalSupply;
49     }
50
51     /**
52      * @dev See `IERC20.balanceOf`.
53      */
```

```
54      //@CTK NO_OVERFLOW
55      //@CTK NO_BUF_OVERFLOW
56      //@CTK NO_ASF
57      /*@CTK balanceOf
58        @post __return == _balances[account]
59       */
60      function balanceOf(address account) public view returns (uint256) {
61          return _balances[account];
62      }
63
64      /**
65       * @dev See `IERC20.transfer`.
66       *
67       * Requirements:
68       *
69       * - `recipient` cannot be the zero address.
70       * - the caller must have a balance of at least `amount`.
71       */
72      //@CTK NO_OVERFLOW
73      //@CTK NO_BUF_OVERFLOW
74      /*@CTK transfer
75        @tag assume_completion
76        @pre msg.sender != recipient
77        @post msg.sender != address(0)
78        @post recipient != address(0)
79        @post __post._balances[msg.sender] == _balances[msg.sender] - amount
80        @post __post._balances[recipient] == _balances[recipient] + amount
81       */
82      function transfer(address recipient, uint256 amount) public returns (bool) {
83          _transfer(msg.sender, recipient, amount);
84          return true;
85      }
86
87      /**
88       * @dev See `IERC20.allowance`.
89       */
90      //@CTK NO_OVERFLOW
91      //@CTK NO_BUF_OVERFLOW
92      //@CTK NO_ASF
93      /*@CTK allowance
94        @post __return == _allowances[owner][spender]
95       */
96      function allowance(address owner, address spender) public view returns (uint256) {
97          return _allowances[owner][spender];
98      }
99
100     /**
101      * @dev See `IERC20.approve`.
102      *
103      * Requirements:
104      *
105      * - `spender` cannot be the zero address.
106      */
107     //@CTK NO_OVERFLOW
108     //@CTK NO_BUF_OVERFLOW
109     //@CTK NO_ASF
110     /*@CTK approve
111       @tag assume_completion
```

```
112       @post msg.sender != address(0)
113       @post spender != address(0)
114       @post __post._allowances[msg.sender][spender] == value
115     */
116     function approve(address spender, uint256 value) public returns (bool) {
117         _approve(msg.sender, spender, value);
118         return true;
119     }
120
121     /**
122      * @dev See `IERC20.transferFrom`.
123      *
124      * Emits an `Approval` event indicating the updated allowance. This is not
125      * required by the EIP. See the note at the beginning of `ERC20`;
126      *
127      * Requirements:
128      * - `sender` and `recipient` cannot be the zero address.
129      * - `sender` must have a balance of at least `value`.
130      * - the caller must have allowance for `sender`'s tokens of at least
131      * `amount`.
132      */
133     //@CTK NO_OVERFLOW
134     //@CTK NO_BUF_OVERFLOW
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     */
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     }
149
150     /**
151      * @dev Atomically increases the allowance granted to `spender` by the caller.
152      *
153      * This is an alternative to `approve` that can be used as a mitigation for
154      * problems described in `IERC20.approve`.
155      *
156      * Emits an `Approval` event indicating the updated allowance.
157      *
158      * Requirements:
159      *
160      * - `spender` cannot be the zero address.
161      */
162     //@CTK NO_OVERFLOW
163     //@CTK NO_BUF_OVERFLOW
164     /*@CTK increaseAllowance
165       @tag assume_completion
166       @post msg.sender != address(0)
167       @post spender != address(0)
```

```
168      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
             ] + addedValue
169      */
170     function increaseAllowance(address spender, uint256 addedValue) public returns (
             bool) {
171         _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
                 ;
172         return true;
173     }
174
175     /**
176      * @dev Atomically decreases the allowance granted to `spender` by the caller.
177      *
178      * This is an alternative to `approve` that can be used as a mitigation for
179      * problems described in `IERC20.approve`.
180      *
181      * Emits an `Approval` event indicating the updated allowance.
182      *
183      * Requirements:
184      *
185      * - `spender` cannot be the zero address.
186      * - `spender` must have allowance for the caller of at least
187      * `subtractedValue`.
188      */
189     //@CTK NO_OVERFLOW
190     //@CTK NO_BUF_OVERFLOW
191     /*@CTK decreaseAllowance
192       @tag assume_completion
193       @post msg.sender != address(0)
194       @post spender != address(0)
195       @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
             ] - subtractedValue
196      */
197     function decreaseAllowance(address spender, uint256 subtractedValue) public
             returns (bool) {
198         _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
                 subtractedValue));
199         return true;
200     }
201
202     /**
203      * @dev Moves tokens `amount` from `sender` to `recipient`.
204      *
205      * This is internal function is equivalent to `transfer`, and can be used to
206      * e.g. implement automatic token fees, slashing mechanisms, etc.
207      *
208      * Emits a `Transfer` event.
209      *
210      * Requirements:
211      *
212      * - `sender` cannot be the zero address.
213      * - `recipient` cannot be the zero address.
214      * - `sender` must have a balance of at least `amount`.
215      */
216     //@CTK NO_OVERFLOW
217     //@CTK NO_BUF_OVERFLOW
218     /*@CTK _transfer
219       @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
```

```
278      */
279     function _burn(address account, uint256 value) internal {
280         require(account != address(0), "ERC20: burn from the zero address");
281 
282         _totalSupply = _totalSupply.sub(value);
283         _balances[account] = _balances[account].sub(value);
284         emit Transfer(account, address(0), value);
285     }
286 
287     /**
288      * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
289      *
290      * This is internal function is equivalent to `approve`, and can be used to
291      * e.g. set automatic allowances for certain subsystems, etc.
292      *
293      * Emits an `Approval` event.
294      *
295      * Requirements:
296      *
297      * - `owner` cannot be the zero address.
298      * - `spender` cannot be the zero address.
299      */
300     //@CTK NO_OVERFLOW
301     //@CTK NO_BUF_OVERFLOW
302     //@CTK NO_ASF
303     /*@CTK _approve
304       @tag assume_completion
305       @post owner != address(0)
306       @post spender != address(0)
307       @post __post._allowances[owner][spender] == value
308      */
309     function _approve(address owner, address spender, uint256 value) internal {
310         require(owner != address(0), "ERC20: approve from the zero address");
311         require(spender != address(0), "ERC20: approve to the zero address");
312 
313         _allowances[owner][spender] = value;
314         emit Approval(owner, spender, value);
315     }
316 
317     /**
318      * @dev Destoys `amount` tokens from `account`.`amount` is then deducted
319      * from the caller's allowance.
320      *
321      * See `_burn` and `_approve`.
322      */
323     //@CTK NO_OVERFLOW
324     //@CTK NO_BUF_OVERFLOW
325     /*@CTK _burn
326       @tag assume_completion
327       @post account != address(0)
328       @post msg.sender != address(0)
329       @post __post._totalSupply == _totalSupply - amount
330       @post __post._balances[account] == _balances[account] - amount
331       @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender
              ] - amount
332      */
333     function _burnFrom(address account, uint256 amount) internal {
334         _burn(account, amount);
```

```
335         _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount));
336     }
337 }
```

File Ownable.sol

```
1  pragma solidity ^0.5.0;
2
3  /**
4   * @dev Contract module which provides a basic access control mechanism, where
5   * there is an account (an owner) that can be granted exclusive access to
6   * specific functions.
7   *
8   * This module is used through inheritance. It will make available the modifier
9   * `onlyOwner`, which can be applied to your functions to restrict their use to
10  * the owner.
11  */
12 contract Ownable {
13     address private _owner;
14
15     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner
        );
16
17     /**
18      * @dev Initializes the contract setting the deployer as the initial owner.
19      */
20     /*@CTK Ownable
21       @post __post._owner == msg.sender
22      */
23     constructor () internal {
24         _owner = msg.sender;
25         emit OwnershipTransferred(address(0), _owner);
26     }
27
28     /**
29      * @dev Returns the address of the current owner.
30      */
31     //@CTK NO_OVERFLOW
32     //@CTK NO_BUF_OVERFLOW
33     //@CTK NO_ASF
34     /*@CTK owner
35       @post __return == _owner
36      */
37     function owner() public view returns (address) {
38         return _owner;
39     }
40
41     /**
42      * @dev Throws if called by any account other than the owner.
43      */
44     modifier onlyOwner() {
45         require(isOwner(), "Ownable: caller is not the owner");
46         _;
47     }
48
49     /**
50      * @dev Returns true if the caller is the current owner.
51      */
52     //@CTK NO_OVERFLOW
```

```
53      //@CTK NO_BUF_OVERFLOW
54      //@CTK NO_ASF
55      /*@CTK isOwner
56        @post __return == (msg.sender == _owner)
57       */
58      function isOwner() public view returns (bool) {
59          return msg.sender == _owner;
60      }
61
62      /**
63       * @dev Leaves the contract without owner. It will not be possible to call
64       * `onlyOwner` functions anymore. Can only be called by the current owner.
65       *
66       * > Note: Renouncing ownership will leave the contract without an owner,
67       * thereby removing any functionality that is only available to the owner.
68       */
69      //@CTK NO_OVERFLOW
70      //@CTK NO_BUF_OVERFLOW
71      //@CTK NO_ASF
72      /*@CTK renounceOwnership
73        @tag assume_completion
74        @post msg.sender == _owner
75        @post __post._owner == address(0)
76       */
77      function renounceOwnership() public onlyOwner {
78          emit OwnershipTransferred(_owner, address(0));
79          _owner = address(0);
80      }
81
82      /**
83       * @dev Transfers ownership of the contract to a new account (`newOwner`).
84       * Can only be called by the current owner.
85       */
86      //@CTK NO_OVERFLOW
87      //@CTK NO_BUF_OVERFLOW
88      //@CTK NO_ASF
89      /*@CTK transferOwnership
90        @tag assume_completion
91        @post msg.sender == _owner
92        @post newOwner != address(0)
93        @post __post._owner == newOwner
94       */
95      function transferOwnership(address newOwner) public onlyOwner {
96          _transferOwnership(newOwner);
97      }
98
99      /**
100      * @dev Transfers ownership of the contract to a new account (`newOwner`).
101      */
102     //@CTK NO_OVERFLOW
103     //@CTK NO_BUF_OVERFLOW
104     //@CTK NO_ASF
105     /*@CTK _transferOwnership
106       @tag assume_completion
107       @post newOwner != address(0)
108       @post __post._owner == newOwner
109      */
110     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.
     */
```

```
82      //@CTK NO_OVERFLOW
83      //@CTK NO_BUF_OVERFLOW
84      //@CTK NO_ASF
85      /*@CTK "SafeMath_mul"
86        @post (__reverted) == (__has_overflow)
87        @post (!(__reverted)) -> ((__return) == ((a) * (b)))
88        @post (msg) == (msg__post)
89        @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
90        @post (__addr_map) == (__addr_map__post)
91        @post !(__has_buf_overflow)
92        @tag spec
93      */
94      function mul(uint256 a, uint256 b) internal pure returns (uint256) {
95          // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
96          // benefit is lost if 'b' is also tested.
97          // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
98          if (a == 0) {
99              return 0;
100         }
101
102         uint256 c = a * b;
103         require(c / a == b, "SafeMath: multiplication overflow");
104
105         return c;
106     }
107
108     /**
109      * @dev Returns the integer division of two unsigned integers. Reverts on
110      * division by zero. The result is rounded towards zero.
111      *
112      * Counterpart to Solidity's `/` operator. Note: this function uses a
113      * `revert` opcode (which leaves remaining gas untouched) while Solidity
114      * uses an invalid opcode to revert (consuming all remaining gas).
115      *
116      * Requirements:
117      * - The divisor cannot be zero.
118      */
119     //@CTK NO_OVERFLOW
120     //@CTK NO_BUF_OVERFLOW
121     //@CTK NO_ASF
122     /*@CTK "SafeMath_div"
123       @post ((__has_overflow) == (true)) -> ((__reverted) == (true))
124       @post (!(__reverted)) -> ((__return) == ((a) / (b)))
125       @post (msg) == (msg__post)
126       @post ((b) == (0)) == (__reverted)
127       @post (__addr_map) == (__addr_map__post)
128       @post !(__has_buf_overflow)
129       @tag spec
130     */
131     /*@CTK "SafeMath_div"
132       @post ((__reverted) == (false)) -> ((__return) == ((a) / (b)))
133     */
134     function div(uint256 a, uint256 b) internal pure returns (uint256) {
135         // Solidity only automatically asserts when dividing by 0
136         require(b > 0, "SafeMath: division by zero");
137         uint256 c = a / b;
138         // assert(a == b * c + a % b); // There is no case in which this doesn't hold
139
```

```
140         return c;
141     }
142
143     /**
144      * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
             modulo),
145      * Reverts when dividing by zero.
146      *
147      * Counterpart to Solidity's `%` operator. This function uses a `revert`
148      * opcode (which leaves remaining gas untouched) while Solidity uses an
149      * invalid opcode to revert (consuming all remaining gas).
150      *
151      * Requirements:
152      * - The divisor cannot be zero.
153      */
154     //@CTK NO_OVERFLOW
155     //@CTK NO_BUF_OVERFLOW
156     //@CTK NO_ASF
157     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
158         require(b != 0, "SafeMath: modulo by zero");
159         return a % b;
160     }
161 }
```

File ERC20Detailed.sol

```
1  pragma solidity ^0.5.0;
2
3  import "./IERC20.sol";
4
5  /**
6   * @dev Optional functions from the ERC20 standard.
7   */
8  contract ERC20Detailed is IERC20 {
9      string private _name;
10     string private _symbol;
11     uint8 private _decimals;
12
13     /**
14      * @dev Sets the values for `name`, `symbol`, and `decimals`. All three of
15      * these values are immutable: they can only be set once during
16      * construction.
17      */
18     /*@CTK ERC20Detailed
19       @post __post._name == name
20       @post __post._symbol == symbol
21       @post __post._decimals == decimals
22      */
23     constructor (string memory name, string memory symbol, uint8 decimals) public {
24         _name = name;
25         _symbol = symbol;
26         _decimals = decimals;
27     }
28
29     /**
30      * @dev Returns the name of the token.
31      */
32     //@CTK NO_OVERFLOW
33     //@CTK NO_BUF_OVERFLOW
```

```
34      //@CTK NO_ASF
35      /*@CTK name
36        @post __return == __post._name
37       */
38      function name() public view returns (string memory) {
39          return _name;
40      }
41
42      /**
43       * @dev Returns the symbol of the token, usually a shorter version of the
44       * name.
45       */
46      //@CTK NO_OVERFLOW
47      //@CTK NO_BUF_OVERFLOW
48      //@CTK NO_ASF
49      /*@CTK symbol
50        @post __return == _symbol
51       */
52      function symbol() public view returns (string memory) {
53          return _symbol;
54      }
55
56      /**
57       * @dev Returns the number of decimals used to get its user representation.
58       * For example, if `decimals` equals `2`, a balance of `505` tokens should
59       * be displayed to a user as `5,05` (`505 / 10 ** 2`).
60       *
61       * Tokens usually opt for a value of 18, imitating the relationship between
62       * Ether and Wei.
63       *
64       * > Note that this information is only used for _display_ purposes: it in
65       * no way affects any of the arithmetic of the contract, including
66       * `IERC20.balanceOf` and `IERC20.transfer`.
67       */
68      //@CTK NO_OVERFLOW
69      //@CTK NO_BUF_OVERFLOW
70      //@CTK NO_ASF
71      /*@CTK decimals
72        @post __return == _decimals
73       */
74      function decimals() public view returns (uint8) {
75          return _decimals;
76      }
77  }
```

File Pausable.sol

```
1   pragma solidity ^0.5.0;
2
3   import "./PauserRole.sol";
4
5   /**
6    * @dev Contract module which allows children to implement an emergency stop
7    * mechanism that can be triggered by an authorized account.
8    *
9    * This module is used through inheritance. It will make available the
10   * modifiers `whenNotPaused` and `whenPaused`, which can be applied to
11   * the functions of your contract. Note that they will not be pausable by
12   * 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
```

```
71      //@CTK NO_BUF_OVERFLOW
72      //@CTK NO_ASF
73      /*@CTK pause
74        @tag assume_completion
75        @post !_paused
76        @post __post._paused
77       */
78      function pause() public onlyPauser whenNotPaused {
79          _paused = true;
80          emit Paused(msg.sender);
81      }
82
83      /**
84       * @dev Called by a pauser to unpause, returns to normal state.
85       */
86      //@CTK NO_OVERFLOW
87      //@CTK NO_BUF_OVERFLOW
88      //@CTK NO_ASF
89      /*@CTK unpause
90        @tag assume_completion
91        @post _paused
92        @post !__post._paused
93       */
94      function unpause() public onlyPauser whenPaused {
95          _paused = false;
96          emit Unpaused(msg.sender);
97      }
98  }
```

File Migrations.sol

```
1   pragma solidity ^0.4.17;
2
3   contract Migrations {
4     address public owner;
5     uint public last_completed_migration;
6
7     modifier restricted() {
8       if (msg.sender == owner) _;
9     }
10
11    /*@CTK Migrations
12      @post __post.owner == msg.sender
13     */
14    function Migrations() public {
15      owner = msg.sender;
16    }
17
18    /*@CTK setCompleted
19      @pre msg.sender == owner
20      @post __post.last_completed_migration == completed
21     */
22    function setCompleted(uint completed) public restricted {
23      last_completed_migration = completed;
24    }
25
26    function upgrade(address new_address) public restricted {
27      Migrations upgraded = Migrations(new_address);
28      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