CertiK Audit Report For Tepleton

Request Date: 2019-08-16
Revision Date: 2019-08-19
Platform Name: Ethereum

Contents

Disclaimer **1**

About CertiK **2**

Exective Summary **3**

Vulnerability Classification **3**

Testing Summary **4**
Audit Score 4
Type of Issues 4
Vulnerability Details 5

Manual Review Notes **6**

Static Analysis Results **7**

Formal Verification Results **8**
How to read 8

Source Code with CertiK Labels **31**

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 Tepleton(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/

Exective Summary

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

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

Vulnerability Classification

For every issue found, CertiK categorizes them into 3 buckets based on its risk level:

Critical

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

Medium

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

Low

The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerabilies, but no concern found yet.

Testing Summary

Type of Issues

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

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

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

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **TEPTokenFlat.sol**
  `644680fb82ad977313f636d07cb0368776b9dc8add1f957b9f2cf565ddb0d349`

Summary

CertiK was chosen by Tepleton to audit the design and implementation of its `TEPToken` smart contract. To ensure comprehensive protection,the source code has been analyzed by the proprietary CertiK formal verification engine and manually reviewed by our smart contract experts and engineers. That end-to-end process ensures proof of stability as well as a hands-on, engineering-focused process to close potential loopholes and recommend design changes in accordance with the best practices in the space.

Overall we found the smart contracts to follow good practices. With the final update of source code and delivery of the audit report, we conclude that the contract is structurally sound and not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend to seek multiple opinions, keep improving the codebase, and more test coverage and sandbox deployments before the mainnet release.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File TEPTokenFlat.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 }
```

Post environment

```
56
57 After Execution:
58     Input = {
59         from = 0x0
60         to = 0x0
61         tokens = 0x6c
```

Formal Verification Request 1

SafeMath add

19, Aug 2019
15.54 ms

Line 26-32 in File TEPTokenFlat.sol

```
26  /*@CTK "SafeMath add"
27    @post (a + b < a || a + b < b) == __reverted
28    @post !__reverted -> __return == a + b
29    @post !__reverted -> !__has_overflow
30    @post !(__has_buf_overflow)
31    @post !(__has_assertion_failure)
32   */
```

Line 33-38 in File TEPTokenFlat.sol

```
33  function add(uint256 a, uint256 b) internal pure returns (uint256) {
34      uint256 c = a + b;
35      require(c >= a, "SafeMath: addition overflow");
36
37      return c;
38  }
```

The code meets the specification.

Formal Verification Request 2

SafeMath sub

19, Aug 2019
11.29 ms

Line 65-71 in File TEPTokenFlat.sol

```
65  /*@CTK "SafeMath sub"
66    @post (a < b) == __reverted
67    @post !__reverted -> __return == a - b
68    @post !__reverted -> !__has_overflow
69    @post !(__has_buf_overflow)
70    @post !(__has_assertion_failure)
71   */
```

Line 72-77 in File TEPTokenFlat.sol

```
72  function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
        returns (uint256) {
73      require(b <= a, errorMessage);
74      uint256 c = a - b;
75
76      return c;
77  }
```

The code meets the specification.

Formal Verification Request 3

SafeMath mul

19, Aug 2019
289.07 ms

Line 88-94 in File TEPTokenFlat.sol

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

Line 95-107 in File TEPTokenFlat.sol

```
95  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
96      // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
97      // benefit is lost if 'b' is also tested.
98      // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
99      if (a == 0) {
100         return 0;
101     }
102
103     uint256 c = a * b;
104     require(c / a == b, "SafeMath: multiplication overflow");
105
106     return c;
107 }
```

The code meets the specification.

Formal Verification Request 4

SafeMath div

19, Aug 2019
12.32 ms

Line 135-141 in File TEPTokenFlat.sol

```
135 /*@CTK "SafeMath div"
136   @post b != 0 -> !__reverted
137   @post !__reverted -> __return == a / b
138   @post !__reverted -> !__has_overflow
139   @post !(__has_buf_overflow)
140   @post !(__has_assertion_failure)
141  */
```

Line 142-149 in File TEPTokenFlat.sol

```
142 function div(uint256 a, uint256 b, string memory errorMessage) internal pure
        returns (uint256) {
143     // Solidity only automatically asserts when dividing by 0
144     require(b > 0, errorMessage);
145     uint256 c = a / b;
```

```
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }
```

The code meets the specification.

Formal Verification Request 5

SafeMath mod

19, Aug 2019
10.76 ms

Line 177-183 in File TEPTokenFlat.sol

```
    /*@CTK "SafeMath mod"
      @post b != 0 -> !__reverted
      @post !__reverted -> __return == a % b
      @post !__reverted -> !__has_overflow
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
     */
```

Line 184-187 in File TEPTokenFlat.sol

```
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
        returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
```

The code meets the specification.

Formal Verification Request 6

If method completes, integer overflow would not happen.

19, Aug 2019
4.47 ms

Line 301 in File TEPTokenFlat.sol

```
    //@CTK NO_OVERFLOW
```

Line 307-309 in File TEPTokenFlat.sol

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

The code meets the specification.

Formal Verification Request 7

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

19, Aug 2019
0.29 ms

Line 302 in File TEPTokenFlat.sol

```
302    //@CTK NO_BUF_OVERFLOW
```

Line 307-309 in File TEPTokenFlat.sol

```
307    function totalSupply() public view returns (uint256) {
308        return _totalSupply;
309    }
```

The code meets the specification.

Formal Verification Request 8

Method will not encounter an assertion failure.

19, Aug 2019
0.3 ms

Line 303 in File TEPTokenFlat.sol

```
303    //@CTK NO_ASF
```

Line 307-309 in File TEPTokenFlat.sol

```
307    function totalSupply() public view returns (uint256) {
308        return _totalSupply;
309    }
```

The code meets the specification.

Formal Verification Request 9

totalSupply correctness

19, Aug 2019
0.31 ms

Line 304-306 in File TEPTokenFlat.sol

```
304    /*@CTK "totalSupply correctness"
305      @post __return == _totalSupply
306     */
```

Line 307-309 in File TEPTokenFlat.sol

```
307    function totalSupply() public view returns (uint256) {
308        return _totalSupply;
309    }
```

The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

19, Aug 2019
4.39 ms

Line 314 in File TEPTokenFlat.sol

```
314    //@CTK NO_OVERFLOW
```

Line 320-322 in File TEPTokenFlat.sol

```
320    function balanceOf(address account) public view returns (uint256) {
321        return _balances[account];
322    }
```

The code meets the specification.

Formal Verification Request 11

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

19, Aug 2019
0.31 ms

Line 315 in File TEPTokenFlat.sol

```
315    //@CTK NO_BUF_OVERFLOW
```

Line 320-322 in File TEPTokenFlat.sol

```
320    function balanceOf(address account) public view returns (uint256) {
321        return _balances[account];
322    }
```

The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

19, Aug 2019
0.32 ms

Line 316 in File TEPTokenFlat.sol

```
316    //@CTK NO_ASF
```

Line 320-322 in File TEPTokenFlat.sol

```
320    function balanceOf(address account) public view returns (uint256) {
321        return _balances[account];
322    }
```

The code meets the specification.

Formal Verification Request 13

balanceOf correctness

19, Aug 2019
0.31 ms

Line 317-319 in File TEPTokenFlat.sol

```
317 /*@CTK "balanceOf correctness"
318   @post __return == _balances[account]
319  */
```

Line 320-322 in File TEPTokenFlat.sol

```
320 function balanceOf(address account) public view returns (uint256) {
321     return _balances[account];
322 }
```

The code meets the specification.

Formal Verification Request 14

If method completes, integer overflow would not happen.

19, Aug 2019
169.15 ms

Line 332 in File TEPTokenFlat.sol

```
332 //@CTK NO_OVERFLOW
```

Line 343-346 in File TEPTokenFlat.sol

```
343 function transfer(address recipient, uint256 amount) public returns (bool) {
344     _transfer(msg.sender, recipient, amount);
345     return true;
346 }
```

The code meets the specification.

Formal Verification Request 15

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

19, Aug 2019
16.52 ms

Line 333 in File TEPTokenFlat.sol

```
333 //@CTK NO_BUF_OVERFLOW
```

Line 343-346 in File TEPTokenFlat.sol

```
343 function transfer(address recipient, uint256 amount) public returns (bool) {
344     _transfer(msg.sender, recipient, amount);
345     return true;
346 }
```

The code meets the specification.

Formal Verification Request 16

Method will not encounter an assertion failure.

19, Aug 2019
17.61 ms

Line 334 in File TEPTokenFlat.sol

```
334    //@CTK NO_ASF
```

Line 343-346 in File TEPTokenFlat.sol

```
343    function transfer(address recipient, uint256 amount) public returns (bool) {
344        _transfer(msg.sender, recipient, amount);
345        return true;
346    }
```

The code meets the specification.

Formal Verification Request 17

transfer correctness

19, Aug 2019
165.46 ms

Line 335-342 in File TEPTokenFlat.sol

```
335    /*@CTK "transfer correctness"
336      @tag assume_completion
337      @post recipient != 0x0
338      @post amount <= _balances[msg.sender]
339      @post recipient != msg.sender -> __post._balances[msg.sender] == _balances[msg.
             sender] - amount
340      @post recipient != msg.sender -> __post._balances[recipient] == _balances[
             recipient] + amount
341      @post recipient == msg.sender -> __post._balances[msg.sender] == _balances[msg.
             sender]
342     */
```

Line 343-346 in File TEPTokenFlat.sol

```
343    function transfer(address recipient, uint256 amount) public returns (bool) {
344        _transfer(msg.sender, recipient, amount);
345        return true;
346    }
```

The code meets the specification.

Formal Verification Request 18

If method completes, integer overflow would not happen.

19, Aug 2019
5.07 ms

Line 351 in File TEPTokenFlat.sol

```
351     //@CTK NO_OVERFLOW
```

Line 357-359 in File TEPTokenFlat.sol

```
357     function allowance(address owner, address spender) public view returns (uint256) {
358         return _allowances[owner][spender];
359     }
```

The code meets the specification.

Formal Verification Request 19

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

19, Aug 2019
0.3 ms

Line 352 in File TEPTokenFlat.sol

```
352     //@CTK NO_BUF_OVERFLOW
```

Line 357-359 in File TEPTokenFlat.sol

```
357     function allowance(address owner, address spender) public view returns (uint256) {
358         return _allowances[owner][spender];
359     }
```

The code meets the specification.

Formal Verification Request 20

Method will not encounter an assertion failure.

19, Aug 2019
0.29 ms

Line 353 in File TEPTokenFlat.sol

```
353     //@CTK NO_ASF
```

Line 357-359 in File TEPTokenFlat.sol

```
357     function allowance(address owner, address spender) public view returns (uint256) {
358         return _allowances[owner][spender];
359     }
```

The code meets the specification.

Formal Verification Request 21

allowance correctness

19, Aug 2019
0.29 ms

Line 354-356 in File TEPTokenFlat.sol

```
354     /*@CTK "allowance correctness"
355       @post __return == _allowances[owner][spender]
356      */
```

Line 357-359 in File TEPTokenFlat.sol

```
357     function allowance(address owner, address spender) public view returns (uint256) {
358         return _allowances[owner][spender];
359     }
```

The code meets the specification.

Formal Verification Request 22

If method completes, integer overflow would not happen.

19, Aug 2019
50.37 ms

Line 368 in File TEPTokenFlat.sol

```
368     //@CTK NO_OVERFLOW
```

Line 376-379 in File TEPTokenFlat.sol

```
376     function approve(address spender, uint256 value) public returns (bool) {
377         _approve(msg.sender, spender, value);
378         return true;
379     }
```

The code meets the specification.

Formal Verification Request 23

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

19, Aug 2019
0.51 ms

Line 369 in File TEPTokenFlat.sol

```
369     //@CTK NO_BUF_OVERFLOW
```

Line 376-379 in File TEPTokenFlat.sol

```
376     function approve(address spender, uint256 value) public returns (bool) {
377         _approve(msg.sender, spender, value);
378         return true;
379     }
```

The code meets the specification.

Formal Verification Request 24

Method will not encounter an assertion failure.

19, Aug 2019
0.5 ms

Line 370 in File TEPTokenFlat.sol

```
370     //@CTK NO_ASF
```

Line 376-379 in File TEPTokenFlat.sol

```
376     function approve(address spender, uint256 value) public returns (bool) {
377         _approve(msg.sender, spender, value);
378         return true;
379     }
```

The code meets the specification.

Formal Verification Request 25

approve correctness

19, Aug 2019
2.57 ms

Line 371-375 in File TEPTokenFlat.sol

```
371     /*@CTK "approve correctness"
372       @tag assume_completion
373       @post spender != 0x0
374       @post __post._allowances[msg.sender][spender] == value
375      */
```

Line 376-379 in File TEPTokenFlat.sol

```
376     function approve(address spender, uint256 value) public returns (bool) {
377         _approve(msg.sender, spender, value);
378         return true;
379     }
```

The code meets the specification.

Formal Verification Request 26

If method completes, integer overflow would not happen.

19, Aug 2019
134.77 ms

Line 393 in File TEPTokenFlat.sol

```
393     //@CTK NO_OVERFLOW
```

Line 405-409 in File TEPTokenFlat.sol

```
405    function transferFrom(address sender, address recipient, uint256 amount) public
           returns (bool) {
406        _transfer(sender, recipient, amount);
407        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20
               : transfer amount exceeds allowance"));
408        return true;
409    }
```

The code meets the specification.

Formal Verification Request 27

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

19, Aug 2019
22.94 ms

Line 394 in File TEPTokenFlat.sol

```
394    //@CTK NO_BUF_OVERFLOW
```

Line 405-409 in File TEPTokenFlat.sol

```
405    function transferFrom(address sender, address recipient, uint256 amount) public
           returns (bool) {
406        _transfer(sender, recipient, amount);
407        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20
               : transfer amount exceeds allowance"));
408        return true;
409    }
```

The code meets the specification.

Formal Verification Request 28

Method will not encounter an assertion failure.

19, Aug 2019
23.3 ms

Line 395 in File TEPTokenFlat.sol

```
395    //@CTK NO_ASF
```

Line 405-409 in File TEPTokenFlat.sol

```
405    function transferFrom(address sender, address recipient, uint256 amount) public
           returns (bool) {
406        _transfer(sender, recipient, amount);
407        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20
               : transfer amount exceeds allowance"));
408        return true;
409    }
```

The code meets the specification.

Formal Verification Request 29

transferFrom correctness

19, Aug 2019
473.38 ms

Line 396-404 in File TEPTokenFlat.sol

```
396   /*@CTK "transferFrom correctness"
397     @tag assume_completion
398     @post recipient != 0x0
399     @post amount <= _balances[sender] && amount <= _allowances[sender][msg.sender]
400     @post recipient != sender -> __post._balances[sender] == _balances[sender] -
          amount
401     @post recipient != sender -> __post._balances[recipient] == _balances[recipient]
           + amount
402     @post recipient == sender -> __post._balances[sender] == _balances[sender]
403     @post __post._allowances[sender][msg.sender] == _allowances[sender][msg.sender]
          - amount
404    */
```

Line 405-409 in File TEPTokenFlat.sol

```
405   function transferFrom(address sender, address recipient, uint256 amount) public
          returns (bool) {
406       _transfer(sender, recipient, amount);
407       _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20
              : transfer amount exceeds allowance"));
408       return true;
409   }
```

The code meets the specification.

Formal Verification Request 30

If method completes, integer overflow would not happen.

19, Aug 2019
54.47 ms

Line 423 in File TEPTokenFlat.sol

```
423   //@CTK NO_OVERFLOW
```

Line 431-434 in File TEPTokenFlat.sol

```
431   function increaseAllowance(address spender, uint256 addedValue) public returns (
          bool) {
432       _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
              ;
433       return true;
434   }
```

The code meets the specification.

Formal Verification Request 31

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

19, Aug 2019
1.07 ms

Line 424 in File TEPTokenFlat.sol

```
424    //@CTK NO_BUF_OVERFLOW
```

Line 431-434 in File TEPTokenFlat.sol

```
431    function increaseAllowance(address spender, uint256 addedValue) public returns (
           bool) {
432        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
               ;
433        return true;
434    }
```

The code meets the specification.

Formal Verification Request 32

Method will not encounter an assertion failure.

19, Aug 2019
0.96 ms

Line 425 in File TEPTokenFlat.sol

```
425    //@CTK NO_ASF
```

Line 431-434 in File TEPTokenFlat.sol

```
431    function increaseAllowance(address spender, uint256 addedValue) public returns (
           bool) {
432        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
               ;
433        return true;
434    }
```

The code meets the specification.

Formal Verification Request 33

increaseAllowance correctness

19, Aug 2019
3.94 ms

Line 426-430 in File TEPTokenFlat.sol

```
426    /*@CTK "increaseAllowance correctness"
427      @tag assume_completion
428      @post spender != 0x0
429      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
             ] + addedValue
430     */
```

Line 431-434 in File TEPTokenFlat.sol

```
431    function increaseAllowance(address spender, uint256 addedValue) public returns (
           bool) {
432        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue))
               ;
433        return true;
434    }
```

The code meets the specification.

Formal Verification Request 34

If method completes, integer overflow would not happen.

19, Aug 2019
54.19 ms

Line 450 in File TEPTokenFlat.sol

```
450    //@CTK NO_OVERFLOW
```

Line 458-461 in File TEPTokenFlat.sol

```
458    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
459        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
               subtractedValue, "ERC20: decreased allowance below zero"));
460        return true;
461    }
```

The code meets the specification.

Formal Verification Request 35

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

19, Aug 2019
0.77 ms

Line 451 in File TEPTokenFlat.sol

```
451    //@CTK NO_BUF_OVERFLOW
```

Line 458-461 in File TEPTokenFlat.sol

```
458    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
459        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
               subtractedValue, "ERC20: decreased allowance below zero"));
460        return true;
461    }
```

The code meets the specification.

Formal Verification Request 36

Method will not encounter an assertion failure.

19, Aug 2019
0.81 ms

Line 452 in File TEPTokenFlat.sol

```
452    //@CTK NO_ASF
```

Line 458-461 in File TEPTokenFlat.sol

```
458    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
459        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
               subtractedValue, "ERC20: decreased allowance below zero"));
460        return true;
461    }
```

The code meets the specification.

Formal Verification Request 37

decreaseAllowance correctness

19, Aug 2019
3.13 ms

Line 453-457 in File TEPTokenFlat.sol

```
453    /*@CTK "decreaseAllowance correctness"
454      @tag assume_completion
455      @post spender != 0x0
456      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
             ] - subtractedValue
457     */
```

Line 458-461 in File TEPTokenFlat.sol

```
458    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
459        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
               subtractedValue, "ERC20: decreased allowance below zero"));
460        return true;
461    }
```

The code meets the specification.

Formal Verification Request 38

If method completes, integer overflow would not happen.

19, Aug 2019
69.08 ms

Line 495 in File TEPTokenFlat.sol

```
495     //@CTK NO_OVERFLOW
```

Line 504-510 in File TEPTokenFlat.sol

```
504     function _mint(address account, uint256 amount) internal {
505         require(account != address(0), "ERC20: mint to the zero address");
506
507         _totalSupply = _totalSupply.add(amount);
508         _balances[account] = _balances[account].add(amount);
509         emit Transfer(address(0), account, amount);
510     }
```

The code meets the specification.

Formal Verification Request 39

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

19, Aug 2019
8.19 ms

Line 496 in File TEPTokenFlat.sol

```
496     //@CTK NO_BUF_OVERFLOW
```

Line 504-510 in File TEPTokenFlat.sol

```
504     function _mint(address account, uint256 amount) internal {
505         require(account != address(0), "ERC20: mint to the zero address");
506
507         _totalSupply = _totalSupply.add(amount);
508         _balances[account] = _balances[account].add(amount);
509         emit Transfer(address(0), account, amount);
510     }
```

The code meets the specification.

Formal Verification Request 40

Method will not encounter an assertion failure.

19, Aug 2019
8.98 ms

Line 497 in File TEPTokenFlat.sol

```
497     //@CTK NO_ASF
```

Line 504-510 in File TEPTokenFlat.sol

```
504     function _mint(address account, uint256 amount) internal {
505         require(account != address(0), "ERC20: mint to the zero address");
506
507         _totalSupply = _totalSupply.add(amount);
508         _balances[account] = _balances[account].add(amount);
509         emit Transfer(address(0), account, amount);
510     }
```

The code meets the specification.

Formal Verification Request 41

_mint correctness

19, Aug 2019
31.46 ms

Line 498-503 in File TEPTokenFlat.sol

```
498 /*@CTK "_mint correctness"
499   @tag assume_completion
500   @post account != 0x0
501   @post __post._balances[account] == _balances[account] + amount
502   @post __post._totalSupply == _totalSupply + amount
503  */
```

Line 504-510 in File TEPTokenFlat.sol

```
504 function _mint(address account, uint256 amount) internal {
505     require(account != address(0), "ERC20: mint to the zero address");
506
507     _totalSupply = _totalSupply.add(amount);
508     _balances[account] = _balances[account].add(amount);
509     emit Transfer(address(0), account, amount);
510 }
```

The code meets the specification.

Formal Verification Request 42

If method completes, integer overflow would not happen.

19, Aug 2019
62.98 ms

Line 523 in File TEPTokenFlat.sol

```
523 //@CTK NO_OVERFLOW
```

Line 533-539 in File TEPTokenFlat.sol

```
533 function _burn(address account, uint256 value) internal {
534     require(account != address(0), "ERC20: burn from the zero address");
535
536     _balances[account] = _balances[account].sub(value, "ERC20: burn amount exceeds
            balance");
537     _totalSupply = _totalSupply.sub(value);
538     emit Transfer(account, address(0), value);
539 }
```

The code meets the specification.

Formal Verification Request 43

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

19, Aug 2019
9.2 ms

Line 524 in File TEPTokenFlat.sol

```
524    //@CTK NO_BUF_OVERFLOW
```

Line 533-539 in File TEPTokenFlat.sol

```
533    function _burn(address account, uint256 value) internal {
534        require(account != address(0), "ERC20: burn from the zero address");
535
536        _balances[account] = _balances[account].sub(value, "ERC20: burn amount exceeds
               balance");
537        _totalSupply = _totalSupply.sub(value);
538        emit Transfer(account, address(0), value);
539    }
```

✅ The code meets the specification.

Formal Verification Request 44

Method will not encounter an assertion failure.

📅 19, Aug 2019
⏱ 8.08 ms

Line 525 in File TEPTokenFlat.sol

```
525    //@CTK NO_ASF
```

Line 533-539 in File TEPTokenFlat.sol

```
533    function _burn(address account, uint256 value) internal {
534        require(account != address(0), "ERC20: burn from the zero address");
535
536        _balances[account] = _balances[account].sub(value, "ERC20: burn amount exceeds
               balance");
537        _totalSupply = _totalSupply.sub(value);
538        emit Transfer(account, address(0), value);
539    }
```

✅ The code meets the specification.

Formal Verification Request 45

_burn correctness

📅 19, Aug 2019
⏱ 49.37 ms

Line 526-532 in File TEPTokenFlat.sol

```
526    /*@CTK "_burn correctness"
527      @tag assume_completion
528      @post account != 0x0
529      @post value <= _balances[account]
530      @post __post._balances[account] == _balances[account] - value
531      @post __post._totalSupply == _totalSupply - value
532     */
```

Line 533-539 in File TEPTokenFlat.sol

```
533 function _burn(address account, uint256 value) internal {
534     require(account != address(0), "ERC20: burn from the zero address");
535
536     _balances[account] = _balances[account].sub(value, "ERC20: burn amount exceeds
            balance");
537     _totalSupply = _totalSupply.sub(value);
538     emit Transfer(account, address(0), value);
539 }
```

The code meets the specification.

Formal Verification Request 46

If method completes, integer overflow would not happen.

19, Aug 2019
120.32 ms

Line 568 in File TEPTokenFlat.sol

```
568 //@CTK NO_OVERFLOW
```

Line 579-582 in File TEPTokenFlat.sol

```
579 function _burnFrom(address account, uint256 amount) internal {
580     _burn(account, amount);
581     _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "
            ERC20: burn amount exceeds allowance"));
582 }
```

The code meets the specification.

Formal Verification Request 47

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

19, Aug 2019
13.11 ms

Line 569 in File TEPTokenFlat.sol

```
569 //@CTK NO_BUF_OVERFLOW
```

Line 579-582 in File TEPTokenFlat.sol

```
579 function _burnFrom(address account, uint256 amount) internal {
580     _burn(account, amount);
581     _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "
            ERC20: burn amount exceeds allowance"));
582 }
```

The code meets the specification.

Formal Verification Request 48

Method will not encounter an assertion failure.

19, Aug 2019
12.85 ms

Line 570 in File TEPTokenFlat.sol

```
570    //@CTK NO_ASF
```

Line 579-582 in File TEPTokenFlat.sol

```
579    function _burnFrom(address account, uint256 amount) internal {
580        _burn(account, amount);
581        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "
               ERC20: burn amount exceeds allowance"));
582    }
```

The code meets the specification.

Formal Verification Request 49

_burnFrom correctness

19, Aug 2019
147.44 ms

Line 571-578 in File TEPTokenFlat.sol

```
571    /*@CTK "_burnFrom correctness"
572      @tag assume_completion
573      @post account != 0x0
574      @post amount <= _balances[account] && amount <= _allowances[account][msg.sender]
575      @post __post._balances[account] == _balances[account] - amount
576      @post __post._totalSupply == _totalSupply - amount
577      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender
             ] - amount
578     */
```

Line 579-582 in File TEPTokenFlat.sol

```
579    function _burnFrom(address account, uint256 amount) internal {
580        _burn(account, amount);
581        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "
               ERC20: burn amount exceeds allowance"));
582    }
```

The code meets the specification.

Formal Verification Request 50

ERC20Detailed

19, Aug 2019
8.11 ms

Line 598-602 in File TEPTokenFlat.sol

```
598 /*@CTK ERC20Detailed
599   @post __post._name == name
600   @post __post._symbol == symbol
601   @post __post._decimals == decimals
602  */
```

Line 603-607 in File TEPTokenFlat.sol

```
603 constructor (string memory name, string memory symbol, uint8 decimals) public {
604     _name = name;
605     _symbol = symbol;
606     _decimals = decimals;
607 }
```

The code meets the specification.

Formal Verification Request 51

name

19, Aug 2019
4.63 ms

Line 612-614 in File TEPTokenFlat.sol

```
612 /*@CTK name
613   @post __return == _name
614  */
```

Line 615-617 in File TEPTokenFlat.sol

```
615 function name() public view returns (string memory) {
616     return _name;
617 }
```

The code meets the specification.

Formal Verification Request 52

symbol

19, Aug 2019
4.78 ms

Line 623-625 in File TEPTokenFlat.sol

```
623 /*@CTK symbol
624   @post __return == _symbol
625  */
```

Line 626-628 in File TEPTokenFlat.sol

```
626 function symbol() public view returns (string memory) {
627     return _symbol;
628 }
```

The code meets the specification.

Formal Verification Request 53

decimals

19, Aug 2019
4.12 ms

Line 642-644 in File TEPTokenFlat.sol

```
642     /*@CTK decimals
643       @post __return == _decimals
644      */
```

Line 645-647 in File TEPTokenFlat.sol

```
645     function decimals() public view returns (uint8) {
646         return _decimals;
647     }
```

The code meets the specification.

Source Code with CertiK Labels

File TEPTokenFlat.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 "SafeMath add"
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "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.
     */
    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 subtraction of two unsigned integers, reverting with custom
         message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    /*@CTK "SafeMath sub"
      @post (a < b) == __reverted
      @post !__reverted -> __return == a - b
      @post !__reverted -> !__has_overflow
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
        returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    /*@CTK "SafeMath mul"
      @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
      @post !__reverted -> __return == a * b
      @post !__reverted == !__has_overflow
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
```

```
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

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

        return c;
    }

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

```

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

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
```

```
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets ‘amount‘ as the allowance of ‘spender‘ over the caller’s tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender’s allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves ‘amount‘ tokens from ‘sender‘ to ‘recipient‘ using the
     * allowance mechanism. ‘amount‘ is then deducted from the caller’s
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when ‘value‘ tokens are moved from one account (‘from‘) to
     * another (‘to‘).
     *
     * Note that ‘value‘ may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a ‘spender‘ for an ‘owner‘ is set by
     * a call to {approve}. ‘value‘ is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20Mintable}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
```

```
 * of returning ‘false‘ on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn’t required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

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

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

    /**
     * @dev See {IERC20-transfer}.
     *
     * Requirements:
     *
     * - ‘recipient‘ cannot be the zero address.
     * - the caller must have a balance of at least ‘amount‘.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
```

```
    /*@CTK "transfer correctness"
      @tag assume_completion
      @post recipient != 0x0
      @post amount <= _balances[msg.sender]
      @post recipient != msg.sender -> __post._balances[msg.sender] == _balances[msg.
          sender] - amount
      @post recipient != msg.sender -> __post._balances[recipient] == _balances[
          recipient] + amount
      @post recipient == msg.sender -> __post._balances[msg.sender] == _balances[msg.
          sender]
     */
    function transfer(address recipient, uint256 amount) public returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

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

    /**
     * @dev See {IERC20-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "approve correctness"
      @tag assume_completion
      @post spender != 0x0
      @post __post._allowances[msg.sender][spender] == value
     */
    function approve(address spender, uint256 value) public returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `value`.
```

```
     * - the caller must have allowance for `sender`'s tokens of at least
     * `amount`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "transferFrom correctness"
      @tag assume_completion
      @post recipient != 0x0
      @post amount <= _balances[sender] && amount <= _allowances[sender][msg.sender]
      @post recipient != sender -> __post._balances[sender] == _balances[sender] - amount
      @post recipient != sender -> __post._balances[recipient] == _balances[recipient] + amount
      @post recipient == sender -> __post._balances[sender] == _balances[sender]
      @post __post._allowances[sender][msg.sender] == _allowances[sender][msg.sender] - amount
     */
    function transferFrom(address sender, address recipient, uint256 amount) public returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "increaseAllowance correctness"
      @tag assume_completion
      @post spender != 0x0
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender] + addedValue
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
```

```
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "decreaseAllowance correctness"
      @tag assume_completion
      @post spender != 0x0
      @post __post._allowances[msg.sender][spender] == _allowances[msg.sender][spender
          ] - subtractedValue
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(
            subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    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, "ERC20: transfer amount
            exceeds balance");
        _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 NO_ASF
    /*@CTK "_mint correctness"
      @tag assume_completion
      @post account != 0x0
      @post __post._balances[account] == _balances[account] + amount
      @post __post._totalSupply == _totalSupply + 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 NO_ASF
    /*@CTK "_burn correctness"
      @tag assume_completion
      @post account != 0x0
      @post value <= _balances[account]
      @post __post._balances[account] == _balances[account] - value
      @post __post._totalSupply == _totalSupply - value
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0), "ERC20: burn from the zero address");

        _balances[account] = _balances[account].sub(value, "ERC20: burn amount exceeds
            balance");
        _totalSupply = _totalSupply.sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
```

```
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

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

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "_burnFrom correctness"
      @tag assume_completion
      @post account != 0x0
      @post amount <= _balances[account] && amount <= _allowances[account][msg.sender]
      @post __post._balances[account] == _balances[account] - amount
      @post __post._totalSupply == _totalSupply - amount
      @post __post._allowances[account][msg.sender] == _allowances[account][msg.sender
          ] - amount
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, msg.sender, _allowances[account][msg.sender].sub(amount, "
            ERC20: burn amount exceeds allowance"));
    }
}

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

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

```
    }

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

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

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

contract TEPToken is ERC20, ERC20Detailed {

    /**
     * @dev Constructor that gives msg.sender all of existing tokens.
     */
    constructor () public ERC20Detailed("Tepleton", "TEP", 8) {
        _mint(msg.sender, 1000000000 * (10 ** uint256(decimals())));
    }
}
```