CertiK Audit Report For EZ Exchange

Request Date: 2019-07-03
Revision Date: 2019-10-02
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 EZ Exchange(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 EZ Exchange to discover issues and vulnerabilities in the source code of their ez365 smart contract. 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

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.	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

- **ez365.sol**
  `bfa02d3ccd1e25e8e12def46d37346ebd59c84ab692097ff3f6a19849d5eccfe`

Summary

CertiK was chosen by EZExchange to audit the design and implementation of its `EZ365` 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.

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 1 in File ez365.sol

```
1 pragma solidity ^0.5.11;
```

No compiler version found

TIMESTAMP_DEPENDENCY

Line 598 in File ez365.sol

```
598         require(block.timestamp >= _releaseTime);
```

”block.timestamp” can be influenced by minors to some degree

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

Ownable

02, Oct 2019
22.36 ms

Line 18-20 in File ez365.sol

```
18  /*@CTK Ownable
19    @post __post._owner == msg.sender
20   */
```

Line 21-24 in File ez365.sol

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

The code meets the specification.

Formal Verification Request 2

owner

02, Oct 2019
5.24 ms

Line 29-31 in File ez365.sol

```
29  /*@CTK owner
30    @post __return == _owner
31   */
```

Line 32-34 in File ez365.sol

```
32  function owner() public view returns (address) {
33      return _owner;
34  }
```

The code meets the specification.

Formal Verification Request 3

isOwner

02, Oct 2019
5.38 ms

Line 47-49 in File ez365.sol

```
47  /*@CTK isOwner
48    @post __return == (msg.sender == _owner)
49   */
```

Line 50-52 in File ez365.sol

```
50    function isOwner() public view returns (bool) {
51        return msg.sender == _owner;
52    }
```

The code meets the specification.

Formal Verification Request 4

renounceOwnership

02, Oct 2019
22.99 ms

Line 61-65 in File ez365.sol

```
61    /*@CTK renounceOwnership
62      @tag assume_completion
63      @post _owner == msg.sender
64      @post __post._owner == address(0)
65     */
```

Line 66-69 in File ez365.sol

```
66    function renounceOwnership() public onlyOwner {
67        emit OwnershipTransferred(_owner, address(0));
68        _owner = address(0);
69    }
```

The code meets the specification.

Formal Verification Request 5

transferOwnership

02, Oct 2019
54.94 ms

Line 75-78 in File ez365.sol

```
75    /*@CTK transferOwnership
76      @tag assume_completion
77      @post _owner == msg.sender
78     */
```

Line 79-81 in File ez365.sol

```
79    function transferOwnership(address newOwner) public onlyOwner {
80        _transferOwnership(newOwner);
81    }
```

The code meets the specification.

Formal Verification Request 6

_transferOwnership

02, Oct 2019
1.59 ms

Line 87-91 in File ez365.sol

```
87  /*@CTK _transferOwnership
88    @tag assume_completion
89    @post newOwner != address(0)
90    @post __post._owner == newOwner
91   */
```

Line 92-96 in File ez365.sol

```
92  function _transferOwnership(address newOwner) internal {
93      require(newOwner != address(0));
94      emit OwnershipTransferred(_owner, newOwner);
95      _owner = newOwner;
96  }
```

The code meets the specification.

Formal Verification Request 7

SafeMath mul

02, Oct 2019
338.47 ms

Line 106-112 in File ez365.sol

```
106 /*@CTK "SafeMath mul"
107   @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
108   @post !__reverted -> __return == a * b
109   @post !__reverted == !__has_overflow
110   @post !(__has_buf_overflow)
111   @post !(__has_assertion_failure)
112  */
```

Line 113-125 in File ez365.sol

```
113 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
114     // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
115     // benefit is lost if 'b' is also tested.
116     // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
117     if (a == 0) {
118         return 0;
119     }
120
121     uint256 c = a * b;
122     require(c / a == b);
123
124     return c;
125 }
```

The code meets the specification.

Formal Verification Request 8

SafeMath div

02, Oct 2019
12.53 ms

Line 130-136 in File ez365.sol

```
130   /*@CTK "SafeMath div"
131     @post b != 0 -> !__reverted
132     @post !__reverted -> __return == a / b
133     @post !__reverted -> !__has_overflow
134     @post !(__has_buf_overflow)
135     @post !(__has_assertion_failure)
136    */
```

Line 137-144 in File ez365.sol

```
137   function div(uint256 a, uint256 b) internal pure returns (uint256) {
138       // Solidity only automatically asserts when dividing by 0
139       require(b > 0);
140       uint256 c = a / b;
141       // assert(a == b * c + a % b); // There is no case in which this doesn't hold
142
143       return c;
144   }
```

The code meets the specification.

Formal Verification Request 9

SafeMath sub

02, Oct 2019
10.99 ms

Line 149-155 in File ez365.sol

```
149   /*@CTK "SafeMath sub"
150     @post (a < b) == __reverted
151     @post !__reverted -> __return == a - b
152     @post !__reverted -> !__has_overflow
153     @post !(__has_buf_overflow)
154     @post !(__has_assertion_failure)
155    */
```

Line 156-161 in File ez365.sol

```
156   function sub(uint256 a, uint256 b) internal pure returns (uint256) {
157       require(b <= a);
158       uint256 c = a - b;
159
160       return c;
161   }
```

The code meets the specification.

Formal Verification Request 10

SafeMath add

02, Oct 2019
12.95 ms

Line 166-172 in File ez365.sol

```
166 /*@CTK "SafeMath add"
167   @post (a + b < a || a + b < b) == __reverted
168   @post !__reverted -> __return == a + b
169   @post !__reverted -> !__has_overflow
170   @post !(__has_buf_overflow)
171   @post !(__has_assertion_failure)
172  */
```

Line 173-178 in File ez365.sol

```
173 function add(uint256 a, uint256 b) internal pure returns (uint256) {
174     uint256 c = a + b;
175     require(c >= a);
176
177     return c;
178 }
```

The code meets the specification.

Formal Verification Request 11

SafeMath mod

02, Oct 2019
10.59 ms

Line 184-190 in File ez365.sol

```
184 /*@CTK "SafeMath mod"
185   @post b != 0 -> !__reverted
186   @post !__reverted -> __return == a % b
187   @post !__reverted -> !__has_overflow
188   @post !(__has_buf_overflow)
189   @post !(__has_assertion_failure)
190  */
```

Line 191-194 in File ez365.sol

```
191 function mod(uint256 a, uint256 b) internal pure returns (uint256) {
192     require(b != 0);
193     return a % b;
194 }
```

The code meets the specification.

Formal Verification Request 12

If method completes, integer overflow would not happen.

02, Oct 2019
4.6 ms

Line 242 in File ez365.sol

```
242    //@CTK NO_OVERFLOW
```

Line 248-250 in File ez365.sol

```
248    function totalSupply() public view returns (uint256) {
249        return _totalSupply;
250    }
```

The code meets the specification.

Formal Verification Request 13

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

02, Oct 2019
0.32 ms

Line 243 in File ez365.sol

```
243    //@CTK NO_BUF_OVERFLOW
```

Line 248-250 in File ez365.sol

```
248    function totalSupply() public view returns (uint256) {
249        return _totalSupply;
250    }
```

The code meets the specification.

Formal Verification Request 14

Method will not encounter an assertion failure.

02, Oct 2019
0.32 ms

Line 244 in File ez365.sol

```
244    //@CTK NO_ASF
```

Line 248-250 in File ez365.sol

```
248    function totalSupply() public view returns (uint256) {
249        return _totalSupply;
250    }
```

The code meets the specification.

Formal Verification Request 15

totalSupply correctness

02, Oct 2019
0.33 ms

Line 245-247 in File ez365.sol

```
245 /*@CTK "totalSupply correctness"
246   @post __return == _totalSupply
247  */
```

Line 248-250 in File ez365.sol

```
248 function totalSupply() public view returns (uint256) {
249     return _totalSupply;
250 }
```

The code meets the specification.

Formal Verification Request 16

If method completes, integer overflow would not happen.

02, Oct 2019
4.99 ms

Line 257 in File ez365.sol

```
257 //@CTK NO_OVERFLOW
```

Line 263-265 in File ez365.sol

```
263 function balanceOf(address owner) public view returns (uint256) {
264     return _balances[owner];
265 }
```

The code meets the specification.

Formal Verification Request 17

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

02, Oct 2019
0.47 ms

Line 258 in File ez365.sol

```
258 //@CTK NO_BUF_OVERFLOW
```

Line 263-265 in File ez365.sol

```
263 function balanceOf(address owner) public view returns (uint256) {
264     return _balances[owner];
265 }
```

The code meets the specification.

Formal Verification Request 18

Method will not encounter an assertion failure.

02, Oct 2019
0.4 ms

Line 259 in File ez365.sol

```
259    //@CTK NO_ASF
```

Line 263-265 in File ez365.sol

```
263    function balanceOf(address owner) public view returns (uint256) {
264        return _balances[owner];
265    }
```

The code meets the specification.

Formal Verification Request 19

balanceOf correctness

02, Oct 2019
0.37 ms

Line 260-262 in File ez365.sol

```
260    /*@CTK "balanceOf correctness"
261      @post __return == _balances[owner]
262     */
```

Line 263-265 in File ez365.sol

```
263    function balanceOf(address owner) public view returns (uint256) {
264        return _balances[owner];
265    }
```

The code meets the specification.

Formal Verification Request 20

If method completes, integer overflow would not happen.

02, Oct 2019
5.56 ms

Line 273 in File ez365.sol

```
273    //@CTK NO_OVERFLOW
```

Line 279-281 in File ez365.sol

```
279    function allowance(address owner, address spender) public view returns (uint256) {
280        return _allowed[owner][spender];
281    }
```

The code meets the specification.

Formal Verification Request 21

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

02, Oct 2019
0.33 ms

Line 274 in File ez365.sol

```
274    //@CTK NO_BUF_OVERFLOW
```

Line 279-281 in File ez365.sol

```
279    function allowance(address owner, address spender) public view returns (uint256) {
280        return _allowed[owner][spender];
281    }
```

The code meets the specification.

Formal Verification Request 22

Method will not encounter an assertion failure.

02, Oct 2019
0.37 ms

Line 275 in File ez365.sol

```
275    //@CTK NO_ASF
```

Line 279-281 in File ez365.sol

```
279    function allowance(address owner, address spender) public view returns (uint256) {
280        return _allowed[owner][spender];
281    }
```

The code meets the specification.

Formal Verification Request 23

allowance correctness

02, Oct 2019
0.34 ms

Line 276-278 in File ez365.sol

```
276    /*@CTK "allowance correctness"
277      @post __return == _allowed[owner][spender]
278     */
```

Line 279-281 in File ez365.sol

```
279    function allowance(address owner, address spender) public view returns (uint256) {
280        return _allowed[owner][spender];
281    }
```

The code meets the specification.

Formal Verification Request 24

If method completes, integer overflow would not happen.

02, Oct 2019
148.52 ms

Line 288 in File ez365.sol

```
288    //@CTK NO_OVERFLOW
```

Line 300-303 in File ez365.sol

```
300    function transfer(address to, uint256 value) public returns (bool) {
301        _transfer(msg.sender, to, value);
302        return true;
303    }
```

The code meets the specification.

Formal Verification Request 25

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

02, Oct 2019
8.73 ms

Line 289 in File ez365.sol

```
289    //@CTK NO_BUF_OVERFLOW
```

Line 300-303 in File ez365.sol

```
300    function transfer(address to, uint256 value) public returns (bool) {
301        _transfer(msg.sender, to, value);
302        return true;
303    }
```

The code meets the specification.

Formal Verification Request 26

Method will not encounter an assertion failure.

02, Oct 2019
5.26 ms

Line 290 in File ez365.sol

```
290    //@CTK NO_ASF
```

Line 300-303 in File ez365.sol

```
300    function transfer(address to, uint256 value) public returns (bool) {
301        _transfer(msg.sender, to, value);
302        return true;
303    }
```

The code meets the specification.

Formal Verification Request 27

transfer correctness

02, Oct 2019
98.48 ms

Line 291-299 in File ez365.sol

```
291   /*@CTK "transfer correctness"
292     @tag assume_completion
293     @post to != 0x0
294     @post value <= _balances[msg.sender]
295     @post to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
          - value
296     @post to != msg.sender -> __post._balances[to] == _balances[to] + value
297     @post to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
298     @post __return == true
299    */
```

Line 300-303 in File ez365.sol

```
300   function transfer(address to, uint256 value) public returns (bool) {
301       _transfer(msg.sender, to, value);
302       return true;
303   }
```

The code meets the specification.

Formal Verification Request 28

If method completes, integer overflow would not happen.

02, Oct 2019
45.4 ms

Line 314 in File ez365.sol

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

Line 322-325 in File ez365.sol

```
322   function approve(address spender, uint256 value) public returns (bool) {
323       _approve(msg.sender, spender, value);
324       return true;
325   }
```

The code meets the specification.

Formal Verification Request 29

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

02, Oct 2019
0.56 ms

Line 315 in File ez365.sol

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

Line 322-325 in File ez365.sol

```
322     function approve(address spender, uint256 value) public returns (bool) {
323         _approve(msg.sender, spender, value);
324         return true;
325     }
```

The code meets the specification.

Formal Verification Request 30

Method will not encounter an assertion failure.

02, Oct 2019
0.51 ms

Line 316 in File ez365.sol

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

Line 322-325 in File ez365.sol

```
322     function approve(address spender, uint256 value) public returns (bool) {
323         _approve(msg.sender, spender, value);
324         return true;
325     }
```

The code meets the specification.

Formal Verification Request 31

approve correctness

02, Oct 2019
3.3 ms

Line 317-321 in File ez365.sol

```
317     /*@CTK "approve correctness"
318       @pre msg.sender != 0x0
319       @post spender == 0x0 -> __reverted
320       @post spender != 0x0 -> __post._allowed[msg.sender][spender] == value
321      */
```

Line 322-325 in File ez365.sol

```
322     function approve(address spender, uint256 value) public returns (bool) {
323         _approve(msg.sender, spender, value);
324         return true;
325     }
```

The code meets the specification.

Formal Verification Request 32

If method completes, integer overflow would not happen.

02, Oct 2019
125.82 ms

Line 335 in File ez365.sol

```
335    //@CTK NO_OVERFLOW
```

Line 348-352 in File ez365.sol

```
348    function transferFrom(address from, address to, uint256 value) public returns (
           bool) {
349        _transfer(from, to, value);
350        _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
351        return true;
352    }
```

The code meets the specification.

Formal Verification Request 33

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

02, Oct 2019
12.76 ms

Line 336 in File ez365.sol

```
336    //@CTK NO_BUF_OVERFLOW
```

Line 348-352 in File ez365.sol

```
348    function transferFrom(address from, address to, uint256 value) public returns (
           bool) {
349        _transfer(from, to, value);
350        _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
351        return true;
352    }
```

The code meets the specification.

Formal Verification Request 34

Method will not encounter an assertion failure.

02, Oct 2019
13.27 ms

Line 337 in File ez365.sol

```
337    //@CTK NO_ASF
```

Line 348-352 in File ez365.sol

```
348     function transferFrom(address from, address to, uint256 value) public returns (
            bool) {
349         _transfer(from, to, value);
350         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
351         return true;
352     }
```

The code meets the specification.

Formal Verification Request 35

transferFrom correctness

02, Oct 2019
200.43 ms

Line 338-347 in File ez365.sol

```
338     /*@CTK "transferFrom correctness"
339       @tag assume_completion
340       @post to != 0x0
341       @post value <= _balances[from] && value <= _allowed[from][msg.sender]
342       @post to != from -> __post._balances[from] == _balances[from] - value
343       @post to != from -> __post._balances[to] == _balances[to] + value
344       @post to == from -> __post._balances[from] == _balances[from]
345       @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
346       @post __return == true
347      */
```

Line 348-352 in File ez365.sol

```
348     function transferFrom(address from, address to, uint256 value) public returns (
            bool) {
349         _transfer(from, to, value);
350         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
351         return true;
352     }
```

The code meets the specification.

Formal Verification Request 36

If method completes, integer overflow would not happen.

02, Oct 2019
60.95 ms

Line 364 in File ez365.sol

```
364     //@CTK NO_OVERFLOW
```

Line 373-376 in File ez365.sol

```
373     function increaseAllowance(address spender, uint256 addedValue) public returns (
            bool) {
374         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
375         return true;
376     }
```

The code meets the specification.

Formal Verification Request 37

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

02, Oct 2019
0.89 ms

Line 365 in File ez365.sol

```
365    //@CTK NO_BUF_OVERFLOW
```

Line 373-376 in File ez365.sol

```
373    function increaseAllowance(address spender, uint256 addedValue) public returns (
           bool) {
374        _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
375        return true;
376    }
```

The code meets the specification.

Formal Verification Request 38

Method will not encounter an assertion failure.

02, Oct 2019
0.82 ms

Line 366 in File ez365.sol

```
366    //@CTK NO_ASF
```

Line 373-376 in File ez365.sol

```
373    function increaseAllowance(address spender, uint256 addedValue) public returns (
           bool) {
374        _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
375        return true;
376    }
```

The code meets the specification.

Formal Verification Request 39

increaseAllowance correctness

02, Oct 2019
4.42 ms

Line 367-372 in File ez365.sol

```
367 /*@CTK "increaseAllowance correctness"
368   @tag assume_completion
369   @post spender != 0x0
370   @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] +
        addedValue
371   @post __return == true
372 */
```

Line 373-376 in File ez365.sol

```
373 function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
374     _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
375     return true;
376 }
```

The code meets the specification.

Formal Verification Request 40

If method completes, integer overflow would not happen.

02, Oct 2019
52.4 ms

Line 388 in File ez365.sol

```
388 //@CTK NO_OVERFLOW
```

Line 397-400 in File ez365.sol

```
397 function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
398     _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
            ));
399     return true;
400 }
```

The code meets the specification.

Formal Verification Request 41

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

02, Oct 2019
0.85 ms

Line 389 in File ez365.sol

```
389 //@CTK NO_BUF_OVERFLOW
```

Line 397-400 in File ez365.sol

```
397 function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
398     _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
            ));
399     return true;
400 }
```

The code meets the specification.

Formal Verification Request 42

Method will not encounter an assertion failure.

02, Oct 2019
0.77 ms

Line 390 in File ez365.sol

```
390    //@CTK NO_ASF
```

Line 397-400 in File ez365.sol

```
397    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
398        _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
               ));
399        return true;
400    }
```

The code meets the specification.

Formal Verification Request 43

decreaseAllowance correctness

02, Oct 2019
3.48 ms

Line 391-396 in File ez365.sol

```
391    /*@CTK "decreaseAllowance correctness"
392      @tag assume_completion
393      @post spender != 0x0
394      @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] -
             subtractedValue
395      @post __return == true
396     */
```

Line 397-400 in File ez365.sol

```
397    function decreaseAllowance(address spender, uint256 subtractedValue) public
           returns (bool) {
398        _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
               ));
399        return true;
400    }
```

The code meets the specification.

Formal Verification Request 44

If method completes, integer overflow would not happen.

02, Oct 2019
68.92 ms

Line 423 in File ez365.sol

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

Line 432-438 in File ez365.sol

```
432    function _mint(address account, uint256 value) internal {
433        require(account != address(0));
434
435        _totalSupply = _totalSupply.add(value);
436        _balances[account] = _balances[account].add(value);
437        emit Transfer(address(0), account, value);
438    }
```

The code meets the specification.

Formal Verification Request 45

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

02, Oct 2019
8.93 ms

Line 424 in File ez365.sol

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

Line 432-438 in File ez365.sol

```
432    function _mint(address account, uint256 value) internal {
433        require(account != address(0));
434
435        _totalSupply = _totalSupply.add(value);
436        _balances[account] = _balances[account].add(value);
437        emit Transfer(address(0), account, value);
438    }
```

The code meets the specification.

Formal Verification Request 46

Method will not encounter an assertion failure.

02, Oct 2019
8.81 ms

Line 425 in File ez365.sol

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

Line 432-438 in File ez365.sol

```
432     function _mint(address account, uint256 value) internal {
433         require(account != address(0));
434
435         _totalSupply = _totalSupply.add(value);
436         _balances[account] = _balances[account].add(value);
437         emit Transfer(address(0), account, value);
438     }
```

The code meets the specification.

Formal Verification Request 47

_mint correctness

02, Oct 2019
40.44 ms

Line 426-431 in File ez365.sol

```
426     /*@CTK "_mint correctness"
427       @tag assume_completion
428       @post account != 0x0
429       @post __post._balances[account] == _balances[account] + value
430       @post __post._totalSupply == _totalSupply + value
431      */
```

Line 432-438 in File ez365.sol

```
432     function _mint(address account, uint256 value) internal {
433         require(account != address(0));
434
435         _totalSupply = _totalSupply.add(value);
436         _balances[account] = _balances[account].add(value);
437         emit Transfer(address(0), account, value);
438     }
```

The code meets the specification.

Formal Verification Request 48

If method completes, integer overflow would not happen.

02, Oct 2019
5.44 ms

Line 498 in File ez365.sol

```
498     //@CTK NO_OVERFLOW
```

Line 504-506 in File ez365.sol

```
504     function name() public pure returns (string memory) {
505         return _name;
506     }
```

The code meets the specification.

Formal Verification Request 49

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

02, Oct 2019
0.34 ms

Line 499 in File ez365.sol

```
499    //@CTK NO_BUF_OVERFLOW
```

Line 504-506 in File ez365.sol

```
504    function name() public pure returns (string memory) {
505        return _name;
506    }
```

The code meets the specification.

Formal Verification Request 50

Method will not encounter an assertion failure.

02, Oct 2019
0.33 ms

Line 500 in File ez365.sol

```
500    //@CTK NO_ASF
```

Line 504-506 in File ez365.sol

```
504    function name() public pure returns (string memory) {
505        return _name;
506    }
```

The code meets the specification.

Formal Verification Request 51

ERC20Detailed name correctness

02, Oct 2019
0.36 ms

Line 501-503 in File ez365.sol

```
501    /*@CTK "ERC20Detailed name correctness"
502      @post __return == _name
503     */
```

Line 504-506 in File ez365.sol

```
504    function name() public pure returns (string memory) {
505        return _name;
506    }
```

The code meets the specification.

Formal Verification Request 52

If method completes, integer overflow would not happen.

02, Oct 2019
5.03 ms

Line 511 in File ez365.sol

```
511    //@CTK NO_OVERFLOW
```

Line 517-519 in File ez365.sol

```
517    function symbol() public pure returns (string memory) {
518        return _symbol;
519    }
```

The code meets the specification.

Formal Verification Request 53

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

02, Oct 2019
0.33 ms

Line 512 in File ez365.sol

```
512    //@CTK NO_BUF_OVERFLOW
```

Line 517-519 in File ez365.sol

```
517    function symbol() public pure returns (string memory) {
518        return _symbol;
519    }
```

The code meets the specification.

Formal Verification Request 54

Method will not encounter an assertion failure.

02, Oct 2019
0.34 ms

Line 513 in File ez365.sol

```
513    //@CTK NO_ASF
```

Line 517-519 in File ez365.sol

```
517    function symbol() public pure returns (string memory) {
518        return _symbol;
519    }
```

The code meets the specification.

Formal Verification Request 55

ERC20Detailed symbol correctness

02, Oct 2019
0.41 ms

Line 514-516 in File ez365.sol

```
514 /*@CTK "ERC20Detailed symbol correctness"
515   @post __return == _symbol
516  */
```

Line 517-519 in File ez365.sol

```
517 function symbol() public pure returns (string memory) {
518     return _symbol;
519 }
```

The code meets the specification.

Formal Verification Request 56

If method completes, integer overflow would not happen.

02, Oct 2019
5.33 ms

Line 524 in File ez365.sol

```
524 //@CTK NO_OVERFLOW
```

Line 530-532 in File ez365.sol

```
530 function decimals() public pure returns (uint256) {
531     return _decimals;
532 }
```

The code meets the specification.

Formal Verification Request 57

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

02, Oct 2019
0.35 ms

Line 525 in File ez365.sol

```
525 //@CTK NO_BUF_OVERFLOW
```

Line 530-532 in File ez365.sol

```
530 function decimals() public pure returns (uint256) {
531     return _decimals;
532 }
```

The code meets the specification.

Formal Verification Request 58

Method will not encounter an assertion failure.

02, Oct 2019
0.31 ms

Line 526 in File ez365.sol

```
526    //@CTK NO_ASF
```

Line 530-532 in File ez365.sol

```
530    function decimals() public pure returns (uint256) {
531        return _decimals;
532    }
```

The code meets the specification.

Formal Verification Request 59

ERC20Detailed decimals correctness

02, Oct 2019
0.35 ms

Line 527-529 in File ez365.sol

```
527    /*@CTK "ERC20Detailed decimals correctness"
528      @post __return == _decimals
529     */
```

Line 530-532 in File ez365.sol

```
530    function decimals() public pure returns (uint256) {
531        return _decimals;
532    }
```

The code meets the specification.

Formal Verification Request 60

If method completes, integer overflow would not happen.

02, Oct 2019
120.3 ms

Line 546 in File ez365.sol

```
546    //@CTK NO_OVERFLOW
```

Line 556-558 in File ez365.sol

```
556    function burn(uint256 value) public {
557        _burn(msg.sender, value);
558    }
```

The code meets the specification.

Formal Verification Request 61

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

02, Oct 2019
13.88 ms

Line 547 in File ez365.sol

```
547    //@CTK NO_BUF_OVERFLOW
```

Line 556-558 in File ez365.sol

```
556    function burn(uint256 value) public {
557        _burn(msg.sender, value);
558    }
```

The code meets the specification.

Formal Verification Request 62

Method will not encounter an assertion failure.

02, Oct 2019
15.34 ms

Line 548 in File ez365.sol

```
548    //@CTK NO_ASF
```

Line 556-558 in File ez365.sol

```
556    function burn(uint256 value) public {
557        _burn(msg.sender, value);
558    }
```

The code meets the specification.

Formal Verification Request 63

burn correctness

02, Oct 2019
60.81 ms

Line 549-555 in File ez365.sol

```
549    /*@CTK "burn correctness"
550      @tag assume_completion
551      @post msg.sender != 0x0
552      @post value <= _balances[msg.sender]
553      @post __post._balances[msg.sender] == _balances[msg.sender] - value
554      @post __post._totalSupply == _totalSupply - value
555     */
```

Line 556-558 in File ez365.sol

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

The code meets the specification.

Formal Verification Request 64

If method completes, integer overflow would not happen.

02, Oct 2019
258.85 ms

Line 567 in File ez365.sol

```
    //@CTK NO_OVERFLOW
```

Line 578-580 in File ez365.sol

```
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }
```

The code meets the specification.

Formal Verification Request 65

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

02, Oct 2019
26.91 ms

Line 568 in File ez365.sol

```
    //@CTK NO_BUF_OVERFLOW
```

Line 578-580 in File ez365.sol

```
    function burnFrom(address from, uint256 value) public {
        _burnFrom(from, value);
    }
```

The code meets the specification.

Formal Verification Request 66

Method will not encounter an assertion failure.

02, Oct 2019
28.56 ms

Line 569 in File ez365.sol

```
    //@CTK NO_ASF
```

Line 578-580 in File ez365.sol

```
578     function burnFrom(address from, uint256 value) public {
579         _burnFrom(from, value);
580     }
```

The code meets the specification.

Formal Verification Request 67

burnFrom correctness

02, Oct 2019
192.37 ms

Line 570-577 in File ez365.sol

```
570     /*@CTK "burnFrom correctness"
571       @tag assume_completion
572       @post from != 0x0
573       @post value <= _balances[from] && value <= _allowed[from][msg.sender]
574       @post __post._balances[from] == _balances[from] - value
575       @post __post._totalSupply == _totalSupply - value
576       @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
577      */
```

Line 578-580 in File ez365.sol

```
578     function burnFrom(address from, uint256 value) public {
579         _burnFrom(from, value);
580     }
```

The code meets the specification.

Formal Verification Request 68

If method completes, integer overflow would not happen.

02, Oct 2019
23.63 ms

Line 583 in File ez365.sol

```
583     //@CTK NO_OVERFLOW
```

Line 590-592 in File ez365.sol

```
590     function updateReleaseTokenTime(uint256 tokenTime) public onlyOwner {
591         _releaseTime = tokenTime;
592     }
```

The code meets the specification.

Formal Verification Request 69

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

02, Oct 2019
0.42 ms

Line 584 in File ez365.sol

```
584    //@CTK NO_BUF_OVERFLOW
```

Line 590-592 in File ez365.sol

```
590    function updateReleaseTokenTime(uint256 tokenTime) public onlyOwner {
591        _releaseTime = tokenTime;
592    }
```

The code meets the specification.

Formal Verification Request 70

Method will not encounter an assertion failure.

02, Oct 2019
0.41 ms

Line 585 in File ez365.sol

```
585    //@CTK NO_ASF
```

Line 590-592 in File ez365.sol

```
590    function updateReleaseTokenTime(uint256 tokenTime) public onlyOwner {
591        _releaseTime = tokenTime;
592    }
```

The code meets the specification.

Formal Verification Request 71

updateReleaseTokenTime correctness

02, Oct 2019
2.16 ms

Line 586-589 in File ez365.sol

```
586    /*@CTK "updateReleaseTokenTime correctness"
587      @post _owner != msg.sender -> __reverted
588      @post _owner == msg.sender -> __post._releaseTime == tokenTime
589     */
```

Line 590-592 in File ez365.sol

```
590    function updateReleaseTokenTime(uint256 tokenTime) public onlyOwner {
591        _releaseTime = tokenTime;
592    }
```

The code meets the specification.

Formal Verification Request 72

If method completes, integer overflow would not happen.

02, Oct 2019
598.56 ms

Line 613 in File ez365.sol

```
613    //@CTK NO_OVERFLOW
```

Line 626-628 in File ez365.sol

```
626    function transfer(address _to, uint256 _value) public isTokenReleased returns (
           bool) {
627        return super.transfer(_to,_value);
628    }
```

The code meets the specification.

Formal Verification Request 73

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

02, Oct 2019
51.57 ms

Line 614 in File ez365.sol

```
614    //@CTK NO_BUF_OVERFLOW
```

Line 626-628 in File ez365.sol

```
626    function transfer(address _to, uint256 _value) public isTokenReleased returns (
           bool) {
627        return super.transfer(_to,_value);
628    }
```

The code meets the specification.

Formal Verification Request 74

Method will not encounter an assertion failure.

02, Oct 2019
244.95 ms

Line 615 in File ez365.sol

```
615    //@CTK NO_ASF
```

Line 626-628 in File ez365.sol

```
626    function transfer(address _to, uint256 _value) public isTokenReleased returns (
           bool) {
627        return super.transfer(_to,_value);
628    }
```

The code meets the specification.

Formal Verification Request 75

transfer correctness

02, Oct 2019
1311.37 ms

Line 616-625 in File ez365.sol

```
616   /*@CTK "transfer correctness"
617     @tag assume_completion
618     @post now >= _releaseTime || _owner == msg.sender
619     @post _to != 0x0
620     @post _value <= _balances[msg.sender]
621     @post _to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
          - _value
622     @post _to != msg.sender -> __post._balances[_to] == _balances[_to] + _value
623     @post _to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
624     @post __return == true
625    */
```

Line 626-628 in File ez365.sol

```
626   function transfer(address _to, uint256 _value) public isTokenReleased returns (
        bool) {
627       return super.transfer(_to,_value);
628   }
```

The code meets the specification.

Formal Verification Request 76

If method completes, integer overflow would not happen.

02, Oct 2019
686.73 ms

Line 630 in File ez365.sol

```
630   //@CTK NO_OVERFLOW
```

Line 644-646 in File ez365.sol

```
644   function transferFrom(address _from, address _to, uint256 _value) public
        isTokenReleased returns (bool) {
645     return super.transferFrom(_from, _to, _value);
646   }
```

The code meets the specification.

Formal Verification Request 77

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

02, Oct 2019
132.12 ms

Line 631 in File ez365.sol

```
631   //@CTK NO_BUF_OVERFLOW
```

Line 644-646 in File ez365.sol

```
644   function transferFrom(address _from, address _to, uint256 _value) public
          isTokenReleased returns (bool) {
645     return super.transferFrom(_from, _to, _value);
646   }
```

The code meets the specification.

Formal Verification Request 78

Method will not encounter an assertion failure.

02, Oct 2019
339.65 ms

Line 632 in File ez365.sol

```
632   //@CTK NO_ASF
```

Line 644-646 in File ez365.sol

```
644   function transferFrom(address _from, address _to, uint256 _value) public
          isTokenReleased returns (bool) {
645     return super.transferFrom(_from, _to, _value);
646   }
```

The code meets the specification.

Formal Verification Request 79

transferFrom correctness

02, Oct 2019
5483.25 ms

Line 633-643 in File ez365.sol

```
633   /*@CTK "transferFrom correctness"
634     @tag assume_completion
635     @post now >= _releaseTime || _owner == msg.sender
636     @post _to != 0x0
637     @post _value <= _balances[_from] && _value <= _allowed[_from][msg.sender]
638     @post _to != _from -> __post._balances[_from] == _balances[_from] - _value
639     @post _to != _from -> __post._balances[_to] == _balances[_to] + _value
640     @post _to == _from -> __post._balances[_from] == _balances[_from]
641     @post __post._allowed[_from][msg.sender] == _allowed[_from][msg.sender] - _value
642     @post __return == true
643    */
```

Line 644-646 in File ez365.sol

```
644   function transferFrom(address _from, address _to, uint256 _value) public
          isTokenReleased returns (bool) {
645     return super.transferFrom(_from, _to, _value);
646   }
```

The code meets the specification.

Formal Verification Request 80

If method completes, integer overflow would not happen.

02, Oct 2019
305.21 ms

Line 648 in File ez365.sol

```
648    //@CTK NO_OVERFLOW
```

Line 658-660 in File ez365.sol

```
658   function increaseAllowance(address _spender, uint _addedValue) public
          isTokenReleased returns (bool) {
659     return super.increaseAllowance(_spender, _addedValue);
660   }
```

The code meets the specification.

Formal Verification Request 81

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

02, Oct 2019
42.69 ms

Line 649 in File ez365.sol

```
649    //@CTK NO_BUF_OVERFLOW
```

Line 658-660 in File ez365.sol

```
658   function increaseAllowance(address _spender, uint _addedValue) public
          isTokenReleased returns (bool) {
659     return super.increaseAllowance(_spender, _addedValue);
660   }
```

The code meets the specification.

Formal Verification Request 82

Method will not encounter an assertion failure.

02, Oct 2019
66.96 ms

Line 650 in File ez365.sol

```
650    //@CTK NO_ASF
```

Line 658-660 in File ez365.sol

```
658   function increaseAllowance(address _spender, uint _addedValue) public
          isTokenReleased returns (bool) {
659     return super.increaseAllowance(_spender, _addedValue);
660   }
```

The code meets the specification.

Formal Verification Request 83

increaseAllowance correctness

02, Oct 2019
84.17 ms

Line 651-657 in File ez365.sol

```
651   /*@CTK "increaseAllowance correctness"
652     @tag assume_completion
653     @post now >= _releaseTime || _owner == msg.sender
654     @post _spender != 0x0
655     @post __post._allowed[msg.sender][_spender] == _allowed[msg.sender][_spender] +
          _addedValue
656     @post __return == true
657   */
```

Line 658-660 in File ez365.sol

```
658   function increaseAllowance(address _spender, uint _addedValue) public
          isTokenReleased returns (bool) {
659     return super.increaseAllowance(_spender, _addedValue);
660   }
```

The code meets the specification.

Formal Verification Request 84

If method completes, integer overflow would not happen.

02, Oct 2019
250.73 ms

Line 662 in File ez365.sol

```
662   //@CTK NO_OVERFLOW
```

Line 672-674 in File ez365.sol

```
672   function decreaseAllowance(address _spender, uint _subtractedValue) public
          isTokenReleased returns (bool) {
673     return super.decreaseAllowance(_spender, _subtractedValue);
674   }
```

The code meets the specification.

Formal Verification Request 85

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

02, Oct 2019
43.34 ms

Line 663 in File ez365.sol

```
663    //@CTK NO_BUF_OVERFLOW
```

Line 672-674 in File ez365.sol

```
672   function decreaseAllowance(address _spender, uint _subtractedValue) public
          isTokenReleased returns (bool) {
673     return super.decreaseAllowance(_spender, _subtractedValue);
674   }
```

The code meets the specification.

Formal Verification Request 86

Method will not encounter an assertion failure.

02, Oct 2019
36.8 ms

Line 664 in File ez365.sol

```
664    //@CTK NO_ASF
```

Line 672-674 in File ez365.sol

```
672   function decreaseAllowance(address _spender, uint _subtractedValue) public
          isTokenReleased returns (bool) {
673     return super.decreaseAllowance(_spender, _subtractedValue);
674   }
```

The code meets the specification.

Formal Verification Request 87

decreaseAllowance correctness

02, Oct 2019
277.53 ms

Line 665-671 in File ez365.sol

```
665    /*@CTK "decreaseAllowance correctness"
666      @tag assume_completion
667      @post now >= _releaseTime || _owner == msg.sender
668      @post _spender != 0x0
669      @post __post._allowed[msg.sender][_spender] == _allowed[msg.sender][_spender] -
             _subtractedValue
670      @post __return == true
671     */
```

Line 672-674 in File ez365.sol

```
672   function decreaseAllowance(address _spender, uint _subtractedValue) public
          isTokenReleased returns (bool) {
673     return super.decreaseAllowance(_spender, _subtractedValue);
674   }
```

The code meets the specification.

Source Code with CertiK Labels

File ez365.sol

```
pragma solidity ^0.5.11;


/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
    address private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract to the sender
     * account.
     */
    /*@CTK Ownable
      @post __post._owner == msg.sender
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @return the address of the owner.
     */
    /*@CTK owner
      @post __return == _owner
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner());
        _;
    }

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    /*@CTK isOwner
      @post __return == (msg.sender == _owner)
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
```

```
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     * @notice Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    /*@CTK renounceOwnership
      @tag assume_completion
      @post _owner == msg.sender
      @post __post._owner == address(0)
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Allows the current owner to transfer control of the contract to a newOwner
        .
     * @param newOwner The address to transfer ownership to.
     */
    /*@CTK transferOwnership
      @tag assume_completion
      @post _owner == msg.sender
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    /*@CTK _transferOwnership
      @tag assume_completion
      @post newOwner != address(0)
      @post __post._owner == newOwner
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error.
 */
library SafeMath {
    /**
     * @dev Multiplies two unsigned integers, reverts on overflow.
     */
    /*@CTK "SafeMath mul"
      @post (((a) > (0)) && ((((a) * (b)) / (a)) != (b))) == (__reverted)
      @post !__reverted -> __return == a * b
```

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

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

       return c;
   }

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

       return c;
   }

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

       return c;
   }

   /**
    * @dev Adds two unsigned integers, reverts on overflow.
```

```
     */
    /*@CTK "SafeMath add"
      @post (a + b < a || a + b < b) == __reverted
      @post !__reverted -> __return == a + b
      @post !__reverted -> !__has_overflow
      @post !(__has_buf_overflow)
      @post !(__has_assertion_failure)
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
     * @dev Divides two unsigned integers and returns the remainder (unsigned integer
         modulo),
     * reverts when dividing by 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) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}
/**
 * @title ERC20 interface
 * @dev see https://eips.ethereum.org/EIPS/eip-20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external returns (
        bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

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

    event Transfer(address indexed from, address indexed to, uint256 value);

    event Approval(address indexed owner, address indexed spender, uint256 value);
}

/**
 * @title Standard ERC20 token
```

```
 *
 * @dev Implementation of the basic standard token.
 * https://eips.ethereum.org/EIPS/eip-20
 * Originally based on code by FirstBlood:
 * https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBloodToken.sol
 *
 * This implementation emits additional Approval events, allowing applications to reconstruct the allowance status for
 * all accounts just by listening to said events. Note that this isn't required by the specification, and other
 * compliant implementations may not do it.
 */
contract ERC20 is IERC20, Ownable {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

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

    uint256 private _totalSupply;

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

    /**
     * @dev Gets the balance of the specified address.
     * @param owner The address to query the balance of.
     * @return A uint256 representing the amount owned by the passed address.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "balanceOf correctness"
      @post __return == _balances[owner]
     */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @dev Function to check the amount of tokens that an owner allowed to a spender.
     * @param owner address The address which owns the funds.
     * @param spender address The address which will spend the funds.
     * @return A uint256 specifying the amount of tokens still available for the spender.
     */
    //@CTK NO_OVERFLOW
```

```
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "allowance correctness"
      @post __return == _allowed[owner][spender]
     */
    function allowance(address owner, address spender) public view returns (uint256) {
        return _allowed[owner][spender];
    }

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

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

```

```
    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required as per the specification,
     * and other compliant implementations may not emit the event.
     * @param from address The address which you want to send tokens from
     * @param to address The address which you want to transfer to
     * @param value uint256 the amount of tokens to be transferred
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "transferFrom correctness"
      @tag assume_completion
      @post to != 0x0
      @post value <= _balances[from] && value <= _allowed[from][msg.sender]
      @post to != from -> __post._balances[from] == _balances[from] - value
      @post to != from -> __post._balances[to] == _balances[to] + value
      @post to == from -> __post._balances[from] == _balances[from]
      @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
      @post __return == true
     */
    function transferFrom(address from, address to, uint256 value) public returns (bool) {
        _transfer(from, to, value);
        _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
        return true;
    }

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

    /**
     * @dev Decrease the amount of tokens that an owner allowed to a spender.
     * approve should be called when _allowed[msg.sender][spender] == 0. To decrement
```

```
     * allowed value is better to use this function to avoid 2 calls (and wait until
     * the first transaction is mined)
     * From MonolithDAO Token.sol
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param subtractedValue The amount of tokens to decrease the allowance by.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "decreaseAllowance correctness"
      @tag assume_completion
      @post spender != 0x0
      @post __post._allowed[msg.sender][spender] == _allowed[msg.sender][spender] -
          subtractedValue
      @post __return == true
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
        _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
            ));
        return true;
    }

    /**
     * @dev Transfer token for a specified addresses.
     * @param from The address to transfer from.
     * @param to The address to transfer to.
     * @param value The amount to be transferred.
     */
    function _transfer(address from, address to, uint256 value) internal {
        require(to != address(0));

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

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

        _totalSupply = _totalSupply.add(value);
```

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

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

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

    /**
     * @dev Approve an address to spend another addresses' tokens.
     * @param owner The address that owns the tokens.
     * @param spender The address that will spend the tokens.
     * @param value The number of tokens that can be spent.
     */
    function _approve(address owner, address spender, uint256 value) internal {
        require(spender != address(0));
        require(owner != address(0));

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

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burnFrom(address account, uint256 value) internal {
        _burn(account, value);
        _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
    }
}


/**
 * @title ERC20Detailed token
 * @dev The decimals are only for visualization purposes.
 * All the operations are done using the smallest and indivisible token unit,
 * just as on Ethereum all the operations are done in wei.
 */
contract ERC20Detailed is ERC20 {
    string constant private _name = "EZ365";
    string constant private _symbol = "EZ365";
    uint256 constant private _decimals = 18;

```

```

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

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

    /**
     * @return the number of decimals of the token.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "ERC20Detailed decimals correctness"
      @post __return == _decimals
     */
    function decimals() public pure returns (uint256) {
        return _decimals;
    }
}
contract EZ365Token is ERC20Detailed {

    uint256 public _releaseTime;
    constructor() public {
        uint256 totalSupply = 200000000 * (10 ** decimals()); //1 Billion
        _mint(msg.sender,totalSupply);
        _releaseTime = block.timestamp;
    }
     /**
     * @dev Burns a specific amount of tokens.
     * @param value The amount of token to be burned.
     */
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "burn correctness"
      @tag assume_completion
      @post msg.sender != 0x0
```

```
     @post value <= _balances[msg.sender]
     @post __post._balances[msg.sender] == _balances[msg.sender] - value
     @post __post._totalSupply == _totalSupply - value
    */
   function burn(uint256 value) public {
       _burn(msg.sender, value);
   }



   /**
    * @dev Burns a specific amount of tokens from the target address and decrements
        allowance.
    * @param from address The account whose tokens will be burned.
    * @param value uint256 The amount of token to be burned.
    */
   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@CTK "burnFrom correctness"
     @tag assume_completion
     @post from != 0x0
     @post value <= _balances[from] && value <= _allowed[from][msg.sender]
     @post __post._balances[from] == _balances[from] - value
     @post __post._totalSupply == _totalSupply - value
     @post __post._allowed[from][msg.sender] == _allowed[from][msg.sender] - value
    */
   function burnFrom(address from, uint256 value) public {
       _burnFrom(from, value);
   }


   //@CTK NO_OVERFLOW
   //@CTK NO_BUF_OVERFLOW
   //@CTK NO_ASF
   /*@CTK "updateReleaseTokenTime correctness"
     @post _owner != msg.sender -> __reverted
     @post _owner == msg.sender -> __post._releaseTime == tokenTime
    */
   function updateReleaseTokenTime(uint256 tokenTime) public onlyOwner {
       _releaseTime = tokenTime;
   }

   modifier isTokenReleased () {
       if (isOwner()){
          _;
       }else{
           require(block.timestamp >= _releaseTime);
           _;
       }
   }
   /**
    * @dev Function to mint tokens
    * @param to The address that will receive the minted tokens.
    * @param value The amount of tokens to mint.
    * @return A boolean that indicates if the operation was successful.
    */
   function mint(address to, uint256 value) public onlyOwner returns (bool) {
```

```
        _mint(to, value);
        return true;
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "transfer correctness"
      @tag assume_completion
      @post now >= _releaseTime || _owner == msg.sender
      @post _to != 0x0
      @post _value <= _balances[msg.sender]
      @post _to != msg.sender -> __post._balances[msg.sender] == _balances[msg.sender] - _value
      @post _to != msg.sender -> __post._balances[_to] == _balances[_to] + _value
      @post _to == msg.sender -> __post._balances[msg.sender] == _balances[msg.sender]
      @post __return == true
     */
    function transfer(address _to, uint256 _value) public isTokenReleased returns (bool) {
        return super.transfer(_to,_value);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "transferFrom correctness"
      @tag assume_completion
      @post now >= _releaseTime || _owner == msg.sender
      @post _to != 0x0
      @post _value <= _balances[_from] && _value <= _allowed[_from][msg.sender]
      @post _to != _from -> __post._balances[_from] == _balances[_from] - _value
      @post _to != _from -> __post._balances[_to] == _balances[_to] + _value
      @post _to == _from -> __post._balances[_from] == _balances[_from]
      @post __post._allowed[_from][msg.sender] == _allowed[_from][msg.sender] - _value
      @post __return == true
     */
    function transferFrom(address _from, address _to, uint256 _value) public isTokenReleased returns (bool) {
      return super.transferFrom(_from, _to, _value);
    }

    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "increaseAllowance correctness"
      @tag assume_completion
      @post now >= _releaseTime || _owner == msg.sender
      @post _spender != 0x0
      @post __post._allowed[msg.sender][_spender] == _allowed[msg.sender][_spender] + _addedValue
      @post __return == true
     */
    function increaseAllowance(address _spender, uint _addedValue) public isTokenReleased returns (bool) {
      return super.increaseAllowance(_spender, _addedValue);
    }

```

```
    //@CTK NO_OVERFLOW
    //@CTK NO_BUF_OVERFLOW
    //@CTK NO_ASF
    /*@CTK "decreaseAllowance correctness"
      @tag assume_completion
      @post now >= _releaseTime || _owner == msg.sender
      @post _spender != 0x0
      @post __post._allowed[msg.sender][_spender] == _allowed[msg.sender][_spender] -
          _subtractedValue
      @post __return == true
     */
   function decreaseAllowance(address _spender, uint _subtractedValue) public
       isTokenReleased returns (bool) {
     return super.decreaseAllowance(_spender, _subtractedValue);
    }
}
```