

CERTIK AUDIT REPORT FOR TAUSCHBLOC



TAUSCH BLOC
Shopping mall blockchain

Request Date: 2019-09-05
Revision Date: 2019-09-12
Platform Name: Ethereum



CERTIK

Contents

Disclaimer	1
About CertiK	2
Executive 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	42

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 TauschBloc(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 as the product of the Smart Contract Audit request by TauschBloc. This audit was conducted to discover issues and vulnerabilities in the source code of TauschBloc'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 vulnerabilities, but no concern found yet.

Testing Summary

PASS

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

Sep 06, 2019



Type of Issues

CertiK smart label engine applied 100% covered 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.	1	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can be influenced by minors to some degree.	0	SWC-120

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

Vulnerability Details

Critical

No issue found.

Medium

No issue found.

Low

No issue found.

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **tauschbloc.sol**
9d4a11a9cd90413d0524751196a18c6f3d56eb63b457756b6b9968b00e203f98

Summary

CertiK was chosen by TauschBloc to audit the design and implementation of its soon to be released 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 5 in File tauschbloc.sol

```
5 pragma solidity ^0.5.2;
```



 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
	52	}
	53	balance: 0x0
	54	}
	55	}
Post environment	57	After Execution:
	58	Input = {
	59	from = 0x0
	60	to = 0x0
	61	tokens = 0x6c

Formal Verification Request 1

SafeMath mul

📅 06, Sep 2019

🕒 418.55 ms

Line 34-40 in File tauschbloc.sol

```
34 /*@CTK "SafeMath mul"
35   @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
36   @post !__reverted -> __return == a * b
37   @post !__reverted == !__has_overflow
38   @post !(__has_buf_overflow)
39   @post !(__has_assertion_failure)
40 */
```

Line 41-53 in File tauschbloc.sol

```
41 function mul(uint256 a, uint256 b) internal pure returns (uint256) {
42     // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
43     // benefit is lost if 'b' is also tested.
44     // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
45     if (a == 0) {
46         return 0;
47     }
48
49     uint256 c = a * b;
50     require(c / a == b);
51
52     return c;
53 }
```

✅ The code meets the specification.

Formal Verification Request 2

SafeMath div

📅 06, Sep 2019

🕒 16.98 ms

Line 58-64 in File tauschbloc.sol

```
58 /*@CTK "SafeMath div"
59   @post b != 0 -> !__reverted
60   @post !__reverted -> __return == a / b
61   @post !__reverted -> !__has_overflow
62   @post !(__has_buf_overflow)
63   @post !(__has_assertion_failure)
64 */
```

Line 65-72 in File tauschbloc.sol

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

```

70
71     return c;
72 }

```

✓ The code meets the specification.

Formal Verification Request 3

SafeMath sub

📅 06, Sep 2019

🕒 14.76 ms

Line 77-83 in File tauschbloc.sol

```

77     /*@CTK "SafeMath sub"
78         @post (a < b) == __reverted
79         @post !__reverted -> __return == a - b
80         @post !__reverted -> !__has_overflow
81         @post !(__has_buf_overflow)
82         @post !(__has_assertion_failure)
83     */

```

Line 84-89 in File tauschbloc.sol

```

84     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
85         require(b <= a);
86         uint256 c = a - b;
87
88         return c;
89     }

```

✓ The code meets the specification.

Formal Verification Request 4

SafeMath add

📅 06, Sep 2019

🕒 19.94 ms

Line 94-100 in File tauschbloc.sol

```

94     /*@CTK "SafeMath add"
95         @post (a + b < a || a + b < b) == __reverted
96         @post !__reverted -> __return == a + b
97         @post !__reverted -> !__has_overflow
98         @post !(__has_buf_overflow)
99         @post !(__has_assertion_failure)
100     */

```

Line 101-106 in File tauschbloc.sol

```

101     function add(uint256 a, uint256 b) internal pure returns (uint256) {
102         uint256 c = a + b;
103         require(c >= a);
104     }

```

```
105     return c;  
106 }
```

✓ The code meets the specification.

Formal Verification Request 5

SafeMath div

📅 06, Sep 2019

🕒 15.0 ms

Line 112-118 in File tauschbloc.sol

```
112  /*@CTK "SafeMath div"  
113     @post b != 0 -> !__reverted  
114     @post !__reverted -> __return == a % b  
115     @post !__reverted -> !__has_overflow  
116     @post !(__has_buf_overflow)  
117     @post !(__has_assertion_failure)  
118  */
```

Line 119-122 in File tauschbloc.sol

```
119  function mod(uint256 a, uint256 b) internal pure returns (uint256) {  
120      require(b != 0);  
121      return a % b;  
122  }
```

✓ The code meets the specification.

Formal Verification Request 6

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 5.79 ms

Line 138 in File tauschbloc.sol

```
138  //@CTK NO_OVERFLOW
```

Line 144-146 in File tauschbloc.sol

```
144  function totalSupply() public view returns (uint256) {  
145      return _totalSupply;  
146  }
```

✓ The code meets the specification.

Formal Verification Request 7

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

📅 06, Sep 2019

🕒 0.4 ms

Line 139 in File tauschbloc.sol

```
139 // @CTK NO_BUF_OVERFLOW
```

Line 144-146 in File tauschbloc.sol

```
144 function totalSupply() public view returns (uint256) {  
145     return _totalSupply;  
146 }
```

✓ The code meets the specification.

Formal Verification Request 8

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.41 ms

Line 140 in File tauschbloc.sol

```
140 // @CTK NO_ASF
```

Line 144-146 in File tauschbloc.sol

```
144 function totalSupply() public view returns (uint256) {  
145     return _totalSupply;  
146 }
```

✓ The code meets the specification.

Formal Verification Request 9

totalSupply correctness

📅 06, Sep 2019

🕒 0.41 ms

Line 141-143 in File tauschbloc.sol

```
141 /* @CTK "totalSupply correctness"  
142     @post __return == _totalSupply  
143 */
```

Line 144-146 in File tauschbloc.sol

```
144 function totalSupply() public view returns (uint256) {  
145     return _totalSupply;  
146 }
```

✓ The code meets the specification.

Formal Verification Request 10

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 5.8 ms

Line 153 in File tauschbloc.sol

```
153 // @CTK_NO_OVERFLOW
```

Line 159-161 in File tauschbloc.sol

```
159 function balanceOf(address owner) public view returns (uint256) {  
160     return _balances[owner];  
161 }
```

✅ The code meets the specification.

Formal Verification Request 11

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

📅 06, Sep 2019

🕒 0.42 ms

Line 154 in File tauschbloc.sol

```
154 // @CTK_NO_BUF_OVERFLOW
```

Line 159-161 in File tauschbloc.sol

```
159 function balanceOf(address owner) public view returns (uint256) {  
160     return _balances[owner];  
161 }
```

✅ The code meets the specification.

Formal Verification Request 12

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.47 ms

Line 155 in File tauschbloc.sol

```
155 // @CTK_NO_ASF
```

Line 159-161 in File tauschbloc.sol

```
159 function balanceOf(address owner) public view returns (uint256) {  
160     return _balances[owner];  
161 }
```

✅ The code meets the specification.

Formal Verification Request 13

balanceOf correctness

📅 06, Sep 2019

🕒 0.64 ms

Line 156-158 in File tauschbloc.sol

```
156 /*@CTK "balanceOf correctness"
157    @post __return == _balances[owner]
158 */
```

Line 159-161 in File tauschbloc.sol

```
159 function balanceOf(address owner) public view returns (uint256) {
160     return _balances[owner];
161 }
```

✅ The code meets the specification.

Formal Verification Request 14

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 8.29 ms

Line 169 in File tauschbloc.sol

```
169 //@CTK NO_OVERFLOW
```

Line 175-177 in File tauschbloc.sol

```
175 function allowance(address owner, address spender) public view returns (uint256) {
176     return _allowed[owner][spender];
177 }
```

✅ The code meets the specification.

Formal Verification Request 15

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

📅 06, Sep 2019

🕒 0.36 ms

Line 170 in File tauschbloc.sol

```
170 //@CTK NO_BUF_OVERFLOW
```

Line 175-177 in File tauschbloc.sol

```
175 function allowance(address owner, address spender) public view returns (uint256) {
176     return _allowed[owner][spender];
177 }
```

✅ The code meets the specification.

Formal Verification Request 16

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.46 ms

Line 171 in File tauschbloc.sol

```
171 // @CTK NO_ASF
```

Line 175-177 in File tauschbloc.sol

```
175 function allowance(address owner, address spender) public view returns (uint256) {  
176     return _allowed[owner][spender];  
177 }
```

✅ The code meets the specification.

Formal Verification Request 17

allowance correctness

📅 06, Sep 2019

🕒 0.53 ms

Line 172-174 in File tauschbloc.sol

```
172 /* @CTK "allowance correctness"  
173     @post __return == _allowed[owner][spender]  
174 */
```

Line 175-177 in File tauschbloc.sol

```
175 function allowance(address owner, address spender) public view returns (uint256) {  
176     return _allowed[owner][spender];  
177 }
```

✅ The code meets the specification.

Formal Verification Request 18

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 162.2 ms

Line 184 in File tauschbloc.sol

```
184 // @CTK NO_OVERFLOW
```

Line 187-190 in File tauschbloc.sol

```
187 function transfer(address to, uint256 value) public returns (bool) {  
188     _transfer(msg.sender, to, value);  
189     return true;  
190 }
```

✅ The code meets the specification.

Formal Verification Request 19

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

📅 06, Sep 2019

🕒 4.31 ms

Line 185 in File tauschbloc.sol

```
185 // @CTK NO_BUF_OVERFLOW
```

Line 187-190 in File tauschbloc.sol

```
187 function transfer(address to, uint256 value) public returns (bool) {  
188     _transfer(msg.sender, to, value);  
189     return true;  
190 }
```

✅ The code meets the specification.

Formal Verification Request 20

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 3.78 ms

Line 186 in File tauschbloc.sol

```
186 // @CTK NO_ASF
```

Line 187-190 in File tauschbloc.sol

```
187 function transfer(address to, uint256 value) public returns (bool) {  
188     _transfer(msg.sender, to, value);  
189     return true;  
190 }
```

✅ The code meets the specification.

Formal Verification Request 21

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 63.93 ms

Line 201 in File tauschbloc.sol

```
201 // @CTK NO_OVERFLOW
```

Line 204-207 in File tauschbloc.sol

```
204 function approve(address spender, uint256 value) public returns (bool) {  
205     _approve(msg.sender, spender, value);  
206     return true;  
207 }
```

✅ The code meets the specification.

Formal Verification Request 22

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

📅 06, Sep 2019

🕒 0.71 ms

Line 202 in File tauschbloc.sol

202 `//@CTK NO_BUF_OVERFLOW`

Line 204-207 in File tauschbloc.sol

```
204     function approve(address spender, uint256 value) public returns (bool) {
205         _approve(msg.sender, spender, value);
206         return true;
207     }
```

✅ The code meets the specification.

Formal Verification Request 23

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.66 ms

Line 203 in File tauschbloc.sol

203 `//@CTK NO_ASF`

Line 204-207 in File tauschbloc.sol

```
204     function approve(address spender, uint256 value) public returns (bool) {
205         _approve(msg.sender, spender, value);
206         return true;
207     }
```

✅ The code meets the specification.

Formal Verification Request 24

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 152.64 ms

Line 217 in File tauschbloc.sol

217 `//@CTK NO_OVERFLOW`

Line 220-224 in File tauschbloc.sol

```
220     function transferFrom(address from, address to, uint256 value) public returns (
221         bool) {
222         _transfer(from, to, value);
223         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
224         return true;
225     }
```

✓ The code meets the specification.

Formal Verification Request 25

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

📅 06, Sep 2019

🕒 7.41 ms

Line 218 in File tauschbloc.sol

218 `//@CTK NO_BUF_OVERFLOW`

Line 220-224 in File tauschbloc.sol

```
220     function transferFrom(address from, address to, uint256 value) public returns (
221         bool) {
222         _transfer(from, to, value);
223         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
224         return true;
225     }
```

✓ The code meets the specification.

Formal Verification Request 26

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 7.57 ms

Line 219 in File tauschbloc.sol

219 `//@CTK NO_ASF`

Line 220-224 in File tauschbloc.sol

```
220     function transferFrom(address from, address to, uint256 value) public returns (
221         bool) {
222         _transfer(from, to, value);
223         _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
224         return true;
225     }
```

✓ The code meets the specification.

Formal Verification Request 27

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 73.82 ms

Line 236 in File tauschbloc.sol

236 `//@CTK NO_OVERFLOW`

Line 239-242 in File tauschbloc.sol

```
239     function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
240         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
241         return true;
242     }
```

✓ The code meets the specification.

Formal Verification Request 28

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

📅 06, Sep 2019

🕒 0.97 ms

Line 237 in File tauschbloc.sol

237 `//@CTK NO_BUF_OVERFLOW`

Line 239-242 in File tauschbloc.sol

```
239     function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
240         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
241         return true;
242     }
```

✓ The code meets the specification.

Formal Verification Request 29

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 1.01 ms

Line 238 in File tauschbloc.sol

238 `//@CTK NO_ASF`

Line 239-242 in File tauschbloc.sol

```
239     function increaseAllowance(address spender, uint256 addedValue) public returns (
        bool) {
240         _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
241         return true;
242     }
```

✓ The code meets the specification.

Formal Verification Request 30

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 67.58 ms

Line 254 in File tauschbloc.sol

254 `//@CTK_NO_OVERFLOW`

Line 257-260 in File tauschbloc.sol

```
257     function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
258         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue)
            );
259         return true;
260     }
```

✅ The code meets the specification.

Formal Verification Request 31

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

📅 06, Sep 2019

🕒 1.04 ms

Line 255 in File tauschbloc.sol

255 `//@CTK_NO_BUF_OVERFLOW`

Line 257-260 in File tauschbloc.sol

```
257     function decreaseAllowance(address spender, uint256 subtractedValue) public
        returns (bool) {
258         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue)
            );
259         return true;
260     }
```

✅ The code meets the specification.

Formal Verification Request 32

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 1.01 ms

Line 256 in File tauschbloc.sol

256 `//@CTK_NO_ASF`

Line 257-260 in File tauschbloc.sol

```
257     function decreaseAllowance(address spender, uint256 subtractedValue) public
258         returns (bool) {
259         _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
260             ));
259         return true;
260     }
```

✓ The code meets the specification.

Formal Verification Request 33

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 8.33 ms

Line 268 in File tauschbloc.sol

```
268     //@CTK NO_OVERFLOW
```

Line 278-284 in File tauschbloc.sol

```
278     function _transfer(address from, address to, uint256 value) internal {
279         require(to != address(0));
280
281         _balances[from] = _balances[from].sub(value);
282         _balances[to] = _balances[to].add(value);
283         emit Transfer(from, to, value);
284     }
```

✓ The code meets the specification.

Formal Verification Request 34

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

📅 06, Sep 2019

🕒 3.83 ms

Line 269 in File tauschbloc.sol

```
269     //@CTK NO_BUF_OVERFLOW
```

Line 278-284 in File tauschbloc.sol

```
278     function _transfer(address from, address to, uint256 value) internal {
279         require(to != address(0));
280
281         _balances[from] = _balances[from].sub(value);
282         _balances[to] = _balances[to].add(value);
283         emit Transfer(from, to, value);
284     }
```

✓ The code meets the specification.

Formal Verification Request 35

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 3.54 ms

Line 270 in File tauschbloc.sol

270 `//@CTK NO_ASF`

Line 278-284 in File tauschbloc.sol

```

278     function _transfer(address from, address to, uint256 value) internal {
279         require(to != address(0));
280
281         _balances[from] = _balances[from].sub(value);
282         _balances[to] = _balances[to].add(value);
283         emit Transfer(from, to, value);
284     }

```

✅ The code meets the specification.

Formal Verification Request 36

`_transfer correctness`

📅 06, Sep 2019

🕒 81.61 ms

Line 271-277 in File tauschbloc.sol

```

271     /*@CTK "_transfer correctness"
272         @tag assume_completion
273         @post to != 0x0
274         @post to != from -> __post._balances[from] == _balances[from] - value
275         @post to != from -> __post._balances[to] == _balances[to] + value
276         @post to == from -> __post._balances[from] == _balances[from]
277     */

```

Line 278-284 in File tauschbloc.sol

```

278     function _transfer(address from, address to, uint256 value) internal {
279         require(to != address(0));
280
281         _balances[from] = _balances[from].sub(value);
282         _balances[to] = _balances[to].add(value);
283         emit Transfer(from, to, value);
284     }

```

✅ The code meets the specification.

Formal Verification Request 37

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 80.4 ms

Line 293 in File tauschbloc.sol

```
293 // @CTK_NO_OVERFLOW
```

Line 302-308 in File tauschbloc.sol

```
302 function _mint(address account, uint256 value) internal {
303     require(account != address(0));
304
305     _totalSupply = _totalSupply.add(value);
306     _balances[account] = _balances[account].add(value);
307     emit Transfer(address(0), account, value);
308 }
```

✓ The code meets the specification.

Formal Verification Request 38

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

📅 06, Sep 2019

🕒 4.05 ms

Line 294 in File tauschbloc.sol

```
294 // @CTK_NO_BUF_OVERFLOW
```

Line 302-308 in File tauschbloc.sol

```
302 function _mint(address account, uint256 value) internal {
303     require(account != address(0));
304
305     _totalSupply = _totalSupply.add(value);
306     _balances[account] = _balances[account].add(value);
307     emit Transfer(address(0), account, value);
308 }
```

✓ The code meets the specification.

Formal Verification Request 39

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 3.46 ms

Line 295 in File tauschbloc.sol

```
295 // @CTK_NO_ASF
```

Line 302-308 in File tauschbloc.sol

```
302 function _mint(address account, uint256 value) internal {
303     require(account != address(0));
304
305     _totalSupply = _totalSupply.add(value);
306     _balances[account] = _balances[account].add(value);
307     emit Transfer(address(0), account, value);
308 }
```

✓ The code meets the specification.

Formal Verification Request 40

`_mint correctness`

📅 06, Sep 2019

🕒 23.39 ms

Line 296-301 in File tauschbloc.sol

```
296 /*@CTK "_mint correctness"
297    @tag assume_completion
298    @post account != 0x0
299    @post __post._balances[account] == _balances[account] + value
300    @post __post._totalSupply == _totalSupply + value
301 */
```

Line 302-308 in File tauschbloc.sol

```
302 function _mint(address account, uint256 value) internal {
303     require(account != address(0));
304
305     _totalSupply = _totalSupply.add(value);
306     _balances[account] = _balances[account].add(value);
307     emit Transfer(address(0), account, value);
308 }
```

✓ The code meets the specification.

Formal Verification Request 41

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 71.11 ms

Line 316 in File tauschbloc.sol

```
316 //@CTK NO_OVERFLOW
```

Line 325-331 in File tauschbloc.sol

```
325 function _burn(address account, uint256 value) internal {
326     require(account != address(0));
327
328     _totalSupply = _totalSupply.sub(value);
329     _balances[account] = _balances[account].sub(value);
330     emit Transfer(account, address(0), value);
331 }
```

✓ The code meets the specification.

Formal Verification Request 42

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

📅 06, Sep 2019

🕒 3.45 ms

Line 317 in File tauschbloc.sol

317 `//@CTK_NO_BUF_OVERFLOW`

Line 325-331 in File tauschbloc.sol

```
325     function _burn(address account, uint256 value) internal {
326         require(account != address(0));
327
328         _totalSupply = _totalSupply.sub(value);
329         _balances[account] = _balances[account].sub(value);
330         emit Transfer(account, address(0), value);
331     }
```

✅ The code meets the specification.

Formal Verification Request 43

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 3.42 ms

Line 318 in File tauschbloc.sol

318 `//@CTK_NO_ASF`

Line 325-331 in File tauschbloc.sol

```
325     function _burn(address account, uint256 value) internal {
326         require(account != address(0));
327
328         _totalSupply = _totalSupply.sub(value);
329         _balances[account] = _balances[account].sub(value);
330         emit Transfer(account, address(0), value);
331     }
```

✅ The code meets the specification.

Formal Verification Request 44

_burn correctness

📅 06, Sep 2019

🕒 45.16 ms

Line 319-324 in File tauschbloc.sol

```

319  /*@CTK "_burn correctness"
320    @tag assume_completion
321    @post account != 0x0
322    @post __post._balances[account] == _balances[account] - value
323    @post __post._totalSupply == _totalSupply - value
324  */

```

Line 325-331 in File tauschbloc.sol

```

325  function _burn(address account, uint256 value) internal {
326    require(account != address(0));
327
328    _totalSupply = _totalSupply.sub(value);
329    _balances[account] = _balances[account].sub(value);
330    emit Transfer(account, address(0), value);
331  }

```

✓ The code meets the specification.

Formal Verification Request 45

_approve

📅 06, Sep 2019

🕒 2.76 ms

Line 339-344 in File tauschbloc.sol

```

339  /*@CTK _approve
340    @tag assume_completion
341    @post spender != 0x0
342    @post owner != 0x0
343    @post __post._allowed[owner][spender] == value
344  */

```

Line 345-351 in File tauschbloc.sol

```

345  function _approve(address owner, address spender, uint256 value) internal {
346    require(spender != address(0));
347    require(owner != address(0));
348
349    _allowed[owner][spender] = value;
350    emit Approval(owner, spender, value);
351  }

```

✓ The code meets the specification.

Formal Verification Request 46

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 142.23 ms

Line 361 in File tauschbloc.sol

```

361  //@CTK NO_OVERFLOW

```

Line 364-367 in File tauschbloc.sol

```
364     function _burnFrom(address account, uint256 value) internal {
365         _burn(account, value);
366         _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
367     }
```

✓ The code meets the specification.

Formal Verification Request 47

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

📅 06, Sep 2019

🕒 7.15 ms

Line 362 in File tauschbloc.sol

```
362     //@CTK NO_BUF_OVERFLOW
```

Line 364-367 in File tauschbloc.sol

```
364     function _burnFrom(address account, uint256 value) internal {
365         _burn(account, value);
366         _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
367     }
```

✓ The code meets the specification.

Formal Verification Request 48

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 6.2 ms

Line 363 in File tauschbloc.sol

```
363     //@CTK NO_ASF
```

Line 364-367 in File tauschbloc.sol

```
364     function _burnFrom(address account, uint256 value) internal {
365         _burn(account, value);
366         _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
367     }
```

✓ The code meets the specification.

Formal Verification Request 49

Roles has correctness

📅 06, Sep 2019

🕒 15.61 ms

Line 405-408 in File tauschbloc.sol

```
405 /*@CTK "Roles has correctness"
406     @post account == 0x0 -> __reverted
407     @post account != 0x0 -> (!__reverted) && (__return == role.bearer[account])
408 */
```

Line 409-412 in File tauschbloc.sol

```
409 function has(Role storage role, address account) internal view returns (bool) {
410     require(account != address(0));
411     return role.bearer[account];
412 }
```

✓ The code meets the specification.

Formal Verification Request 50

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 19.3 ms

Line 423 in File tauschbloc.sol

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

Line 426-428 in File tauschbloc.sol

```
426 constructor () internal {
427     _addMinter(msg.sender);
428 }
```

✓ The code meets the specification.

Formal Verification Request 51

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

📅 06, Sep 2019

🕒 0.52 ms

Line 424 in File tauschbloc.sol

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

Line 426-428 in File tauschbloc.sol

```
426 constructor () internal {
427     _addMinter(msg.sender);
428 }
```

✓ The code meets the specification.

Formal Verification Request 52

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.45 ms

Line 425 in File tauschbloc.sol

425 `//@CTK NO_ASF`

Line 426-428 in File tauschbloc.sol

```
426     constructor () internal {  
427         _addMinter(msg.sender);  
428     }
```

✅ The code meets the specification.

Formal Verification Request 53

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 34.77 ms

Line 434 in File tauschbloc.sol

434 `//@CTK NO_OVERFLOW`

Line 441-443 in File tauschbloc.sol

```
441     function isMinter(address account) public view returns (bool) {  
442         return _minters.has(account);  
443     }
```

✅ The code meets the specification.

Formal Verification Request 54

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

📅 06, Sep 2019

🕒 0.62 ms

Line 435 in File tauschbloc.sol

435 `//@CTK NO_BUF_OVERFLOW`

Line 441-443 in File tauschbloc.sol

```
441     function isMinter(address account) public view returns (bool) {  
442         return _minters.has(account);  
443     }
```

✅ The code meets the specification.

Formal Verification Request 55

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.63 ms

Line 436 in File tauschbloc.sol

436 `//@CTK NO_ASF`

Line 441-443 in File tauschbloc.sol

```
441 function isMinter(address account) public view returns (bool) {  
442 return _minters.has(account);  
443 }
```

✅ The code meets the specification.

Formal Verification Request 56

isMinter correctness

📅 06, Sep 2019

🕒 1.61 ms

Line 437-440 in File tauschbloc.sol

```
437 /*CTK "isMinter correctness"  
438 @post account == 0x0 -> __reverted  
439 @post account != 0x0 -> !__reverted && __return == _minters.bearer[account]  
440 */
```

Line 441-443 in File tauschbloc.sol

```
441 function isMinter(address account) public view returns (bool) {  
442 return _minters.has(account);  
443 }
```

✅ The code meets the specification.

Formal Verification Request 57

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 66.4 ms

Line 444 in File tauschbloc.sol

444 `//@CTK NO_OVERFLOW`

Line 447-449 in File tauschbloc.sol

```
447 function addMinter(address account) public onlyMinter {  
448 _addMinter(account);  
449 }
```

✅ The code meets the specification.

Formal Verification Request 58

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

📅 06, Sep 2019

🕒 0.93 ms

Line 445 in File tauschbloc.sol

```
445 // @CTK NO_BUF_OVERFLOW
```

Line 447-449 in File tauschbloc.sol

```
447 function addMinter(address account) public onlyMinter {  
448     _addMinter(account);  
449 }
```

✅ The code meets the specification.

Formal Verification Request 59

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.9 ms

Line 446 in File tauschbloc.sol

```
446 // @CTK NO_ASF
```

Line 447-449 in File tauschbloc.sol

```
447 function addMinter(address account) public onlyMinter {  
448     _addMinter(account);  
449 }
```

✅ The code meets the specification.

Formal Verification Request 60

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 18.81 ms

Line 450 in File tauschbloc.sol

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

Line 453-455 in File tauschbloc.sol

```
453 function renounceMinter() public {  
454     _removeMinter(msg.sender);  
455 }
```

✅ The code meets the specification.

Formal Verification Request 61

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

📅 06, Sep 2019

🕒 0.49 ms

Line 451 in File tauschbloc.sol

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

Line 453-455 in File tauschbloc.sol

```
453 function renounceMinter() public {  
454     _removeMinter(msg.sender);  
455 }
```

✅ The code meets the specification.

Formal Verification Request 62

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.47 ms

Line 452 in File tauschbloc.sol

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

Line 453-455 in File tauschbloc.sol

```
453 function renounceMinter() public {  
454     _removeMinter(msg.sender);  
455 }
```

✅ The code meets the specification.

Formal Verification Request 63

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 257.43 ms

Line 480 in File tauschbloc.sol

```
480 // @CTK NO_OVERFLOW
```

Line 489-492 in File tauschbloc.sol

```
489 function mint(address to, uint256 value) public onlyMinter returns (bool) {  
490     _mint(to, value);  
491     return true;  
492 }
```

✅ The code meets the specification.

Formal Verification Request 64

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

📅 06, Sep 2019

🕒 5.63 ms

Line 481 in File tauschbloc.sol

```
481 // @CTK NO_BUF_OVERFLOW
```

Line 489-492 in File tauschbloc.sol

```
489 function mint(address to, uint256 value) public onlyMinter returns (bool) {  
490     _mint(to, value);  
491     return true;  
492 }
```

✅ The code meets the specification.

Formal Verification Request 65

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 5.43 ms

Line 482 in File tauschbloc.sol

```
482 // @CTK NO_ASF
```

Line 489-492 in File tauschbloc.sol

```
489 function mint(address to, uint256 value) public onlyMinter returns (bool) {  
490     _mint(to, value);  
491     return true;  
492 }
```

✅ The code meets the specification.

Formal Verification Request 66

mint

📅 06, Sep 2019

🕒 174.98 ms

Line 483-488 in File tauschbloc.sol

```
483 /* @CTK mint  
484     @tag assume_completion  
485     @post to != 0  
486     @post __post._totalSupply == _totalSupply + value  
487     @post __post._balances[to] == _balances[to] + value  
488 */
```

Line 489-492 in File tauschbloc.sol

```

489     function mint(address to, uint256 value) public onlyMinter returns (bool) {
490         _mint(to, value);
491         return true;
492     }

```

✓ The code meets the specification.

Formal Verification Request 67

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 149.63 ms

Line 501 in File tauschbloc.sol

```

501     //@CTK NO_OVERFLOW

```

Line 504-506 in File tauschbloc.sol

```

504     function burn(uint256 value) public {
505         _burn(msg.sender, value);
506     }

```

✓ The code meets the specification.

Formal Verification Request 68

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

📅 06, Sep 2019

🕒 3.88 ms

Line 502 in File tauschbloc.sol

```

502     //@CTK NO_BUF_OVERFLOW

```

Line 504-506 in File tauschbloc.sol

```

504     function burn(uint256 value) public {
505         _burn(msg.sender, value);
506     }

```

✓ The code meets the specification.

Formal Verification Request 69

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 3.87 ms

Line 503 in File tauschbloc.sol

```

503     //@CTK NO_ASF

```

Line 504-506 in File tauschbloc.sol

```
504     function burn(uint256 value) public {
505         _burn(msg.sender, value);
506     }
```

✓ The code meets the specification.

Formal Verification Request 70

If method completes, integer overflow would not happen.



06, Sep 2019



311.59 ms

Line 513 in File tauschbloc.sol

```
513     //@CTK_NO_OVERFLOW
```

Line 516-518 in File tauschbloc.sol

```
516     function burnFrom(address from, uint256 value) public {
517         _burnFrom(from, value);
518     }
```

✓ The code meets the specification.

Formal Verification Request 71

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



06, Sep 2019



7.26 ms

Line 514 in File tauschbloc.sol

```
514     //@CTK_NO_BUF_OVERFLOW
```

Line 516-518 in File tauschbloc.sol

```
516     function burnFrom(address from, uint256 value) public {
517         _burnFrom(from, value);
518     }
```

✓ The code meets the specification.

Formal Verification Request 72

Method will not encounter an assertion failure.



06, Sep 2019



6.61 ms

Line 515 in File tauschbloc.sol

```
515     //@CTK_NO_ASF
```

Line 516-518 in File tauschbloc.sol

```
516     function burnFrom(address from, uint256 value) public {
517         _burnFrom(from, value);
518     }
```

✓ The code meets the specification.

Formal Verification Request 73

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 10.36 ms

Line 526 in File tauschbloc.sol

```
526     //@CTK NO_OVERFLOW
```

Line 534-538 in File tauschbloc.sol

```
534     constructor (string memory name, string memory symbol, uint8 decimals) public {
535         _name = name;
536         _symbol = symbol;
537         _decimals = decimals;
538     }
```

✓ The code meets the specification.

Formal Verification Request 74

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

📅 06, Sep 2019

🕒 0.41 ms

Line 527 in File tauschbloc.sol

```
527     //@CTK NO_BUF_OVERFLOW
```

Line 534-538 in File tauschbloc.sol

```
534     constructor (string memory name, string memory symbol, uint8 decimals) public {
535         _name = name;
536         _symbol = symbol;
537         _decimals = decimals;
538     }
```

✓ The code meets the specification.

Formal Verification Request 75

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.41 ms

Line 528 in File tauschbloc.sol

```
528 // @CTK NO_ASF
```

Line 534-538 in File tauschbloc.sol

```
534 constructor (string memory name, string memory symbol, uint8 decimals) public {  
535     _name = name;  
536     _symbol = symbol;  
537     _decimals = decimals;  
538 }
```

✓ The code meets the specification.

Formal Verification Request 76

ERC20Detailed constructor correctness

📅 06, Sep 2019

🕒 0.82 ms

Line 529-533 in File tauschbloc.sol

```
529 /* @CTK "ERC20Detailed constructor correctness"  
530     @post __post._name == name  
531     @post __post._symbol == symbol  
532     @post __post._decimals == decimals  
533 */
```

Line 534-538 in File tauschbloc.sol

```
534 constructor (string memory name, string memory symbol, uint8 decimals) public {  
535     _name = name;  
536     _symbol = symbol;  
537     _decimals = decimals;  
538 }
```

✓ The code meets the specification.

Formal Verification Request 77

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 7.03 ms

Line 543 in File tauschbloc.sol

```
543 // @CTK NO_OVERFLOW
```

Line 549-551 in File tauschbloc.sol

```
549 function name() public view returns (string memory) {  
550     return _name;  
551 }
```

✓ The code meets the specification.

Formal Verification Request 78

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

📅 06, Sep 2019

🕒 0.45 ms

Line 544 in File tauschbloc.sol

```
544 // @CTK NO_BUF_OVERFLOW
```

Line 549-551 in File tauschbloc.sol

```
549 function name() public view returns (string memory) {  
550     return _name;  
551 }
```

✅ The code meets the specification.

Formal Verification Request 79

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.41 ms

Line 545 in File tauschbloc.sol

```
545 // @CTK NO_ASF
```

Line 549-551 in File tauschbloc.sol

```
549 function name() public view returns (string memory) {  
550     return _name;  
551 }
```

✅ The code meets the specification.

Formal Verification Request 80

ERC20Detailed name correctness

📅 06, Sep 2019

🕒 0.45 ms

Line 546-548 in File tauschbloc.sol

```
546 /* @CTK "ERC20Detailed name correctness"  
547     @post __return == _name  
548 */
```

Line 549-551 in File tauschbloc.sol

```
549 function name() public view returns (string memory) {  
550     return _name;  
551 }
```

✅ The code meets the specification.

Formal Verification Request 81

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 6.45 ms

Line 556 in File tauschbloc.sol

```
556 // @CTK_NO_OVERFLOW
```

Line 562-564 in File tauschbloc.sol

```
562 function symbol() public view returns (string memory) {  
563     return _symbol;  
564 }
```

✅ The code meets the specification.

Formal Verification Request 82

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

📅 06, Sep 2019

🕒 0.48 ms

Line 557 in File tauschbloc.sol

```
557 // @CTK_NO_BUF_OVERFLOW
```

Line 562-564 in File tauschbloc.sol

```
562 function symbol() public view returns (string memory) {  
563     return _symbol;  
564 }
```

✅ The code meets the specification.

Formal Verification Request 83

Method will not encounter an assertion failure.

📅 06, Sep 2019

🕒 0.44 ms

Line 558 in File tauschbloc.sol

```
558 // @CTK_NO_ASF
```

Line 562-564 in File tauschbloc.sol

```
562 function symbol() public view returns (string memory) {  
563     return _symbol;  
564 }
```

✅ The code meets the specification.

Formal Verification Request 84

ERC20Detailed symbol correctness

📅 06, Sep 2019

🕒 0.51 ms

Line 559-561 in File tauschbloc.sol

```
559 /*@CTK "ERC20Detailed symbol correctness"
560     @post __return == _symbol
561 */
```

Line 562-564 in File tauschbloc.sol

```
562 function symbol() public view returns (string memory) {
563     return _symbol;
564 }
```

✅ The code meets the specification.

Formal Verification Request 85

If method completes, integer overflow would not happen.

📅 06, Sep 2019

🕒 5.75 ms

Line 569 in File tauschbloc.sol

```
569 //@CTK NO_OVERFLOW
```

Line 575-577 in File tauschbloc.sol

```
575 function decimals() public view returns (uint8) {
576     return _decimals;
577 }
```

✅ The code meets the specification.

Formal Verification Request 86

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

📅 06, Sep 2019

🕒 0.42 ms

Line 570 in File tauschbloc.sol

```
570 //@CTK NO_BUF_OVERFLOW
```

Line 575-577 in File tauschbloc.sol


```
575 function decimals() public view returns (uint8) {
576     return _decimals;
577 }
```

✅ The code meets the specification.

Formal Verification Request 87

Method will not encounter an assertion failure.

 06, Sep 2019

 0.4 ms

Line 571 in File tauschbloc.sol

```
571 // @CTK NO_ASF
```

Line 575-577 in File tauschbloc.sol


```
575 function decimals() public view returns (uint8) {
576     return _decimals;
577 }
```

 The code meets the specification.

Formal Verification Request 88

ERC20Detailed decimals correctness

 06, Sep 2019

 0.4 ms

Line 572-574 in File tauschbloc.sol

```
572 /* @CTK "ERC20Detailed decimals correctness"
573     @post __return == _decimals
574 */
```

Line 575-577 in File tauschbloc.sol

```
575 function decimals() public view returns (uint8) {
576     return _decimals;
577 }
```

 The code meets the specification.

Source Code with CertiK Labels

File tauschbloc.sol

```

1  /**
2   *Submitted for verification at Etherscan.io on 2019-05-27
3   */
4
5  pragma solidity ^0.5.2;
6
7  /**
8   * @title ERC20 interface
9   * @dev see https://eips.ethereum.org/EIPS/eip-20
10  */
11 interface IERC20 {
12     function transfer(address to, uint256 value) external returns (bool);
13
14     function approve(address spender, uint256 value) external returns (bool);
15
16     function transferFrom(address from, address to, uint256 value) external returns (
17         bool);
18
19     function totalSupply() external view returns (uint256);
20
21     function balanceOf(address who) external view returns (uint256);
22
23     function allowance(address owner, address spender) external view returns (uint256)
24         ;
25
26     event Transfer(address indexed from, address indexed to, uint256 value);
27
28     event Approval(address indexed owner, address indexed spender, uint256 value);
29 }
30
31 library SafeMath {
32     /**
33     * @dev Multiplies two unsigned integers, reverts on overflow.
34     */
35     /*@CTK "SafeMath mul"
36     @post (((a) > (0)) && (((a) * (b)) / (a)) != (b))) == (__reverted)
37     @post !__reverted -> __return == a * b
38     @post !__reverted == !__has_overflow
39     @post !(__has_buf_overflow)
40     @post !(__has_assertion_failure)
41     */
42     function mul(uint256 a, uint256 b) internal pure returns (uint256) {
43         // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
44         // benefit is lost if 'b' is also tested.
45         // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
46         if (a == 0) {
47             return 0;
48         }
49
50         uint256 c = a * b;
51         require(c / a == b);
52
53         return c;

```

```

53 }
54
55 /**
56  * @dev Integer division of two unsigned integers truncating the quotient, reverts
    on division by zero.
57  */
58 /*@CTK "SafeMath div"
59  @post b != 0 -> !__reverted
60  @post !__reverted -> __return == a / b
61  @post !__reverted -> !__has_overflow
62  @post !(__has_buf_overflow)
63  @post !(__has_assertion_failure)
64  */
65 function div(uint256 a, uint256 b) internal pure returns (uint256) {
66     // Solidity only automatically asserts when dividing by 0
67     require(b > 0);
68     uint256 c = a / b;
69     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
70
71     return c;
72 }
73
74 /**
75  * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend
    is greater than minuend).
76  */
77 /*@CTK "SafeMath sub"
78  @post (a < b) == __reverted
79  @post !__reverted -> __return == a - b
80  @post !__reverted -> !__has_overflow
81  @post !(__has_buf_overflow)
82  @post !(__has_assertion_failure)
83  */
84 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
85     require(b <= a);
86     uint256 c = a - b;
87
88     return c;
89 }
90
91 /**
92  * @dev Adds two unsigned integers, reverts on overflow.
93  */
94 /*@CTK "SafeMath add"
95  @post (a + b < a || a + b < b) == __reverted
96  @post !__reverted -> __return == a + b
97  @post !__reverted -> !__has_overflow
98  @post !(__has_buf_overflow)
99  @post !(__has_assertion_failure)
100  */
101 function add(uint256 a, uint256 b) internal pure returns (uint256) {
102     uint256 c = a + b;
103     require(c >= a);
104
105     return c;
106 }
107
108 /**

```

```

109     * @dev Divides two unsigned integers and returns the remainder (unsigned integer
      modulo),
110     * reverts when dividing by zero.
111     */
112     /*@CTK "SafeMath div"
113         @post b != 0 -> !__reverted
114         @post !__reverted -> __return == a % b
115         @post !__reverted -> !__has_overflow
116         @post !(__has_buf_overflow)
117         @post !(__has_assertion_failure)
118     */
119     function mod(uint256 a, uint256 b) internal pure returns (uint256) {
120         require(b != 0);
121         return a % b;
122     }
123 }
124
125
126 contract ERC20 is IERC20 {
127     using SafeMath for uint256;
128
129     mapping (address => uint256) private _balances;
130
131     mapping (address => mapping (address => uint256)) private _allowed;
132
133     uint256 private _totalSupply;
134
135     /**
136     * @dev Total number of tokens in existence
137     */
138     //@CTK NO_OVERFLOW
139     //@CTK NO_BUF_OVERFLOW
140     //@CTK NO_ASF
141     /*@CTK "totalSupply correctness"
142         @post __return == _totalSupply
143     */
144     function totalSupply() public view returns (uint256) {
145         return _totalSupply;
146     }
147
148     /**
149     * @dev Gets the balance of the specified address.
150     * @param owner The address to query the balance of.
151     * @return A uint256 representing the amount owned by the passed address.
152     */
153     //@CTK NO_OVERFLOW
154     //@CTK NO_BUF_OVERFLOW
155     //@CTK NO_ASF
156     /*@CTK "balanceOf correctness"
157         @post __return == _balances[owner]
158     */
159     function balanceOf(address owner) public view returns (uint256) {
160         return _balances[owner];
161     }
162
163     /**
164     * @dev Function to check the amount of tokens that an owner allowed to a spender.
165     * @param owner address The address which owns the funds.

```

```

166 * @param spender address The address which will spend the funds.
167 * @return A uint256 specifying the amount of tokens still available for the
    spender.
168 */
169 //@CTK NO_OVERFLOW
170 //@CTK NO_BUF_OVERFLOW
171 //@CTK NO_ASF
172 /*@CTK "allowance correctness"
173   @post __return == _allowed[owner][spender]
174 */
175 function allowance(address owner, address spender) public view returns (uint256) {
176     return _allowed[owner][spender];
177 }
178
179 /**
180  * @dev Transfer token to a specified address
181  * @param to The address to transfer to.
182  * @param value The amount to be transferred.
183  */
184 //@CTK NO_OVERFLOW
185 //@CTK NO_BUF_OVERFLOW
186 //@CTK NO_ASF
187 function transfer(address to, uint256 value) public returns (bool) {
188     _transfer(msg.sender, to, value);
189     return true;
190 }
191
192 /**
193  * @dev Approve the passed address to spend the specified amount of tokens on
    behalf of msg.sender.
194  * Beware that changing an allowance with this method brings the risk that someone
    may use both the old
195  * and the new allowance by unfortunate transaction ordering. One possible
    solution to mitigate this
196  * race condition is to first reduce the spender's allowance to 0 and set the
    desired value afterwards:
197  * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
198  * @param spender The address which will spend the funds.
199  * @param value The amount of tokens to be spent.
200  */
201 //@CTK NO_OVERFLOW
202 //@CTK NO_BUF_OVERFLOW
203 //@CTK NO_ASF
204 function approve(address spender, uint256 value) public returns (bool) {
205     _approve(msg.sender, spender, value);
206     return true;
207 }
208
209 /**
210  * @dev Transfer tokens from one address to another.
211  * Note that while this function emits an Approval event, this is not required as
    per the specification,
212  * and other compliant implementations may not emit the event.
213  * @param from address The address which you want to send tokens from
214  * @param to address The address which you want to transfer to
215  * @param value uint256 the amount of tokens to be transferred
216  */
217 //@CTK NO_OVERFLOW

```

```

218 //@CTK NO_BUF_OVERFLOW
219 //@CTK NO_ASF
220 function transferFrom(address from, address to, uint256 value) public returns (
    bool) {
221     _transfer(from, to, value);
222     _approve(from, msg.sender, _allowed[from][msg.sender].sub(value));
223     return true;
224 }
225
226 /**
227  * @dev Increase the amount of tokens that an owner allowed to a spender.
228  * approve should be called when _allowed[msg.sender][spender] == 0. To increment
229  * allowed value is better to use this function to avoid 2 calls (and wait until
230  * the first transaction is mined)
231  * From MonolithDAO Token.sol
232  * Emits an Approval event.
233  * @param spender The address which will spend the funds.
234  * @param addedValue The amount of tokens to increase the allowance by.
235  */
236 //@CTK NO_OVERFLOW
237 //@CTK NO_BUF_OVERFLOW
238 //@CTK NO_ASF
239 function increaseAllowance(address spender, uint256 addedValue) public returns (
    bool) {
240     _approve(msg.sender, spender, _allowed[msg.sender][spender].add(addedValue));
241     return true;
242 }
243
244 /**
245  * @dev Decrease the amount of tokens that an owner allowed to a spender.
246  * approve should be called when _allowed[msg.sender][spender] == 0. To decrement
247  * allowed value is better to use this function to avoid 2 calls (and wait until
248  * the first transaction is mined)
249  * From MonolithDAO Token.sol
250  * Emits an Approval event.
251  * @param spender The address which will spend the funds.
252  * @param subtractedValue The amount of tokens to decrease the allowance by.
253  */
254 //@CTK NO_OVERFLOW
255 //@CTK NO_BUF_OVERFLOW
256 //@CTK NO_ASF
257 function decreaseAllowance(address spender, uint256 subtractedValue) public
    returns (bool) {
258     _approve(msg.sender, spender, _allowed[msg.sender][spender].sub(subtractedValue
    ));
259     return true;
260 }
261
262 /**
263  * @dev Transfer token for a specified addresses
264  * @param from The address to transfer from.
265  * @param to The address to transfer to.
266  * @param value The amount to be transferred.
267  */
268 //@CTK NO_OVERFLOW
269 //@CTK NO_BUF_OVERFLOW
270 //@CTK NO_ASF
271 /*@CTK "_transfer correctness"

```



```

272     @tag assume_completion
273     @post to != 0x0
274     @post to != from -> __post._balances[from] == _balances[from] - value
275     @post to != from -> __post._balances[to] == _balances[to] + value
276     @post to == from -> __post._balances[from] == _balances[from]
277     */
278     function _transfer(address from, address to, uint256 value) internal {
279         require(to != address(0));
280
281         _balances[from] = _balances[from].sub(value);
282         _balances[to] = _balances[to].add(value);
283         emit Transfer(from, to, value);
284     }
285
286     /**
287     * @dev Internal function that mints an amount of the token and assigns it to
288     * an account. This encapsulates the modification of balances such that the
289     * proper events are emitted.
290     * @param account The account that will receive the created tokens.
291     * @param value The amount that will be created.
292     */
293     //@CTK NO_OVERFLOW
294     //@CTK NO_BUF_OVERFLOW
295     //@CTK NO_ASF
296     /*@CTK "_mint correctness"
297     @tag assume_completion
298     @post account != 0x0
299     @post __post._balances[account] == _balances[account] + value
300     @post __post._totalSupply == _totalSupply + value
301     */
302     function _mint(address account, uint256 value) internal {
303         require(account != address(0));
304
305         _totalSupply = _totalSupply.add(value);
306         _balances[account] = _balances[account].add(value);
307         emit Transfer(address(0), account, value);
308     }
309
310     /**
311     * @dev Internal function that burns an amount of the token of a given
312     * account.
313     * @param account The account whose tokens will be burnt.
314     * @param value The amount that will be burnt.
315     */
316     //@CTK NO_OVERFLOW
317     //@CTK NO_BUF_OVERFLOW
318     //@CTK NO_ASF
319     /*@CTK "_burn correctness"
320     @tag assume_completion
321     @post account != 0x0
322     @post __post._balances[account] == _balances[account] - value
323     @post __post._totalSupply == _totalSupply - value
324     */
325     function _burn(address account, uint256 value) internal {
326         require(account != address(0));
327
328         _totalSupply = _totalSupply.sub(value);
329         _balances[account] = _balances[account].sub(value);

```

```

330     emit Transfer(account, address(0), value);
331 }
332
333 /**
334  * @dev Approve an address to spend another addresses' tokens.
335  * @param owner The address that owns the tokens.
336  * @param spender The address that will spend the tokens.
337  * @param value The number of tokens that can be spent.
338  */
339 /*@CTK _approve
340   @tag assume_completion
341   @post spender != 0x0
342   @post owner != 0x0
343   @post __post._allowed[owner][spender] == value
344 */
345 function _approve(address owner, address spender, uint256 value) internal {
346     require(spender != address(0));
347     require(owner != address(0));
348
349     _allowed[owner][spender] = value;
350     emit Approval(owner, spender, value);
351 }
352
353 /**
354  * @dev Internal function that burns an amount of the token of a given
355  * account, deducting from the sender's allowance for said account. Uses the
356  * internal burn function.
357  * Emits an Approval event (reflecting the reduced allowance).
358  * @param account The account whose tokens will be burnt.
359  * @param value The amount that will be burnt.
360  */
361 // @CTK NO_OVERFLOW
362 // @CTK NO_BUF_OVERFLOW
363 // @CTK NO_ASF
364 function _burnFrom(address account, uint256 value) internal {
365     _burn(account, value);
366     _approve(account, msg.sender, _allowed[account][msg.sender].sub(value));
367 }
368 }
369
370
371 library Roles {
372     struct Role {
373         mapping (address => bool) bearer;
374     }
375
376     /**
377     * @dev give an account access to this role
378     */
379     function add(Role storage role, address account) internal {
380         require(account != address(0));
381         require(!role.bearer[account]);
382
383         role.bearer[account] = true;
384     }
385 }
386
387 /**

```

```

388     * @dev remove an account's access to this role
389     */
390     function remove(Role storage role, address account) internal {
391         require(account != address(0));
392         require(role.bearer[account]);
393
394         role.bearer[account] = false;
395     }
396
397     /**
398     * @dev check if an account has this role
399     * @return bool
400     */
401     /*@CTK "Roles has correctness"
402     @post account == 0x0 -> __reverted
403     @post account != 0x0 -> (!__reverted) && (__return == role.bearer[account])
404     */
405     function has(Role storage role, address account) internal view returns (bool) {
406         require(account != address(0));
407         return role.bearer[account];
408     }
409 }
410
411
412 contract MinterRole {
413     using Roles for Roles.Role;
414
415     event MinterAdded(address indexed account);
416     event MinterRemoved(address indexed account);
417
418     Roles.Role private _minters;
419     /*@CTK NO_OVERFLOW
420     /*@CTK NO_BUF_OVERFLOW
421     /*@CTK NO_ASF
422     constructor () internal {
423         _addMinter(msg.sender);
424     }
425
426     modifier onlyMinter() {
427         require(isMinter(msg.sender));
428         _;
429     }
430     /*@CTK NO_OVERFLOW
431     /*@CTK NO_BUF_OVERFLOW
432     /*@CTK NO_ASF
433     /*@CTK "isMinter correctness"
434     @post account == 0x0 -> __reverted
435     @post account != 0x0 -> !__reverted && __return == _minters.bearer[account]
436     */
437     function isMinter(address account) public view returns (bool) {
438         return _minters.has(account);
439     }
440     /*@CTK NO_OVERFLOW
441     /*@CTK NO_BUF_OVERFLOW
442     /*@CTK NO_ASF
443     function addMinter(address account) public onlyMinter {
444         _addMinter(account);
445     }

```

```

446 //CTK NO_OVERFLOW
447 //CTK NO_BUF_OVERFLOW
448 //CTK NO_ASF
449 function renounceMinter() public {
450     _removeMinter(msg.sender);
451 }
452
453 function _addMinter(address account) internal {
454     _minters.add(account);
455     emit MinterAdded(account);
456 }
457
458 function _removeMinter(address account) internal {
459     _minters.remove(account);
460     emit MinterRemoved(account);
461 }
462 }
463
464
465 contract ERC20Mintable is ERC20, MinterRole {
466     /**
467      * @dev Function to mint tokens
468      * @param to The address that will receive the minted tokens.
469      * @param value The amount of tokens to mint.
470      * @return A boolean that indicates if the operation was successful.
471      */
472     //CTK NO_OVERFLOW
473     //CTK NO_BUF_OVERFLOW
474     //CTK NO_ASF
475     /*CTK mint
476         @tag assume_completion
477         @post to != 0
478         @post __post._totalSupply == _totalSupply + value
479         @post __post._balances[to] == _balances[to] + value
480     */
481     function mint(address to, uint256 value) public onlyMinter returns (bool) {
482         _mint(to, value);
483         return true;
484     }
485 }
486
487
488 contract ERC20Burnable is ERC20 {
489     /**
490      * @dev Burns a specific amount of tokens.
491      * @param value The amount of token to be burned.
492      */
493     //CTK NO_OVERFLOW
494     //CTK NO_BUF_OVERFLOW
495     //CTK NO_ASF
496     function burn(uint256 value) public {
497         _burn(msg.sender, value);
498     }
499
500     /**
501      * @dev Burns a specific amount of tokens from the target address and decrements
502          allowance
503      * @param from address The account whose tokens will be burned.

```

```

503     * @param value uint256 The amount of token to be burned.
504     */
505     //@CTK NO_OVERFLOW
506     //@CTK NO_BUF_OVERFLOW
507     //@CTK NO_ASF
508     function burnFrom(address from, uint256 value) public {
509         _burnFrom(from, value);
510     }
511 }
512
513
514 contract ERC20Detailed is IERC20 {
515     string private _name;
516     string private _symbol;
517     uint8 private _decimals;
518     //@CTK NO_OVERFLOW
519     //@CTK NO_BUF_OVERFLOW
520     //@CTK NO_ASF
521     /*@CTK "ERC20Detailed constructor correctness"
522         @post __post._name == name
523         @post __post._symbol == symbol
524         @post __post._decimals == decimals
525     */
526     constructor (string memory name, string memory symbol, uint8 decimals) public {
527         _name = name;
528         _symbol = symbol;
529         _decimals = decimals;
530     }
531
532     /**
533     * @return the name of the token.
534     */
535     //@CTK NO_OVERFLOW
536     //@CTK NO_BUF_OVERFLOW
537     //@CTK NO_ASF
538     /*@CTK "ERC20Detailed name correctness"
539         @post __return == _name
540     */
541     function name() public view returns (string memory) {
542         return _name;
543     }
544
545     /**
546     * @return the symbol of the token.
547     */
548     //@CTK NO_OVERFLOW
549     //@CTK NO_BUF_OVERFLOW
550     //@CTK NO_ASF
551     /*@CTK "ERC20Detailed symbol correctness"
552         @post __return == _symbol
553     */
554     function symbol() public view returns (string memory) {
555         return _symbol;
556     }
557
558     /**
559     * @return the number of decimals of the token.
560     */

```

```

561 // @CTK NO_OVERFLOW
562 // @CTK NO_BUF_OVERFLOW
563 // @CTK NO_ASF
564 /* @CTK "ERC20Detailed decimals correctness"
565     @post __return == _decimals
566 */
567 function decimals() public view returns (uint8) {
568     return _decimals;
569 }
570 }
571
572 contract TauschToken is ERC20Mintable, ERC20Burnable, ERC20Detailed {
573
574     string private _name = "TauschToken";
575     string private _symbol = "TUC";
576     uint8 private _decimals = 10;
577
578     uint256 public constant INITIAL_SUPPLY = 50 * (10 ** 18);
579
580     address account = msg.sender;
581
582     constructor()
583         ERC20Detailed(_name, _symbol, _decimals)
584         ERC20Burnable()
585         ERC20Mintable()
586     public {
587         _mint(account, INITIAL_SUPPLY);
588     }
589 }

```