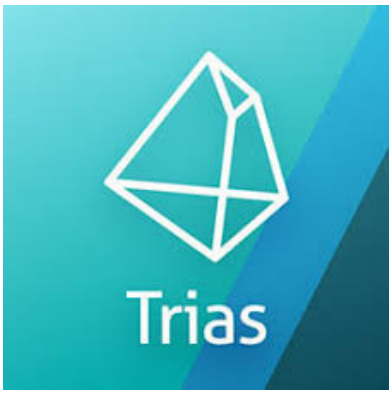
# CertiK Audit Report For Trias




Request Date: 2019-05-07
Revision Date: 2019-05-10
Platform Name: Ethereum

CERTIK

# 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 Trias(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.

# Exective Summary

This report has been prepared as product of the Smart Contract Audit request by Trias. This audit was conducted to discover issues and vulnerabilities in the source code of Trias'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 issues found, CertiK categorizes them into 3 buckets based on its risk level:

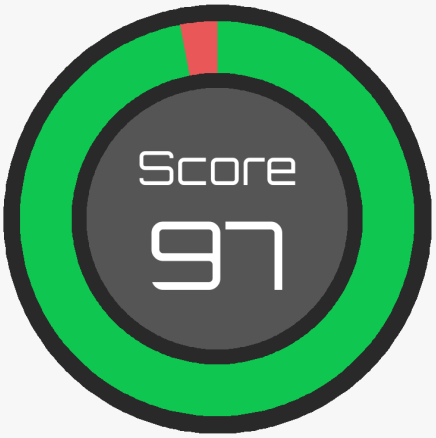
- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The code implementation is not a best practice, or use a suboptimal design pattern, which may lead to security vulnerability, but no concern found yet.

# Testing Summary

PASS

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

*May 10, 2019*




## Type of Issues

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

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

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

## Vulnerability Details

### Critical

No issue found.

### Medium

No issue found.

### Low

No issue found.

# Formal Verification Results

## How to read

### Detail for Request 1

transferFrom to same address

| | |
|---|---|
| *Verification date* | 20, Oct 2018 |
| *Verification timespan* | 395.38 ms |

CERTIK *label location*: Line 30-34 in File howtoread.sol

CERTIK *label*

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

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

*Raw code*

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

*Counterexample*: This code violates the specification

*Initial environment*

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

*Post environment*

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

## Formal Verification Request 1

### SafeMath sub

10, May 2019
23.78 ms

Line 10-16 in File TRYSimple.sol

```
10    /*@CTK "SafeMath sub"
11      @post (a < b) == __reverted
12      @post !__reverted -> __return == a - b
13      @post !__reverted -> !__has_overflow
14      @post !(__has_buf_overflow)
15      @post !(__has_assertion_failure)
16    */
```

Line 17-21 in File TRYSimple.sol

```
17    function safeSub(uint256 a, uint256 b) internal pure returns (uint256) {
18        require(b <= a, "SafeMath: subtraction overflow");
19        uint256 c = a - b;
20        return c;
21    }
```

The code meets the specification

## Formal Verification Request 2

### SafeMath add

10, May 2019
28.87 ms

Line 26-32 in File TRYSimple.sol

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

Line 33-37 in File TRYSimple.sol

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

The code meets the specification

## Formal Verification Request 3

**If method completes, integer overflow would not happen.**

10, May 2019
22.27 ms

Line 50 in File TRYSimple.sol

```
50    //@CTK NO_OVERFLOW
```

Line 55-61 in File TRYSimple.sol

```
55    constructor(uint256 _initialAmount,string _tokenName,uint8 _decimalUnits,string
          _tokenSymbol) public {
56        balances[msg.sender] = _initialAmount;
57        totalSupply = _initialAmount;
58        name = _tokenName;
59        decimals = _decimalUnits;
60        symbol = _tokenSymbol;
61    }
```

The code meets the specification

## Formal Verification Request 4

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

10, May 2019
0.54 ms

Line 51 in File TRYSimple.sol

```
51    //@CTK NO_BUF_OVERFLOW
```

Line 55-61 in File TRYSimple.sol

```
55    constructor(uint256 _initialAmount,string _tokenName,uint8 _decimalUnits,string
          _tokenSymbol) public {
56        balances[msg.sender] = _initialAmount;
57        totalSupply = _initialAmount;
58        name = _tokenName;
59        decimals = _decimalUnits;
60        symbol = _tokenSymbol;
61    }
```

The code meets the specification

## Formal Verification Request 5

**Method will not encounter an assertion failure.**

10, May 2019
0.51 ms

Line 52 in File TRYSimple.sol

```
52    //@CTK NO_ASF
```

Line 55-61 in File TRYSimple.sol

```
55    constructor(uint256 _initialAmount,string _tokenName,uint8 _decimalUnits,string
          _tokenSymbol) public {
56        balances[msg.sender] = _initialAmount;
57        totalSupply = _initialAmount;
58        name = _tokenName;
59        decimals = _decimalUnits;
60        symbol = _tokenSymbol;
61    }
```

✅ The code meets the specification

## Formal Verification Request 6

**If method completes, integer overflow would not happen.**

📅 10, May 2019
⏱ 6.17 ms

Line 73 in File TRYSimple.sol

```
73    //@CTK NO_OVERFLOW
```

Line 81-83 in File TRYSimple.sol

```
81    function balanceOf(address _owner) public constant returns (uint256 balance) {//
          constant==view
82        return balances[_owner];
83    }
```

✅ The code meets the specification

## Formal Verification Request 7

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

📅 10, May 2019
⏱ 0.45 ms

Line 74 in File TRYSimple.sol

```
74    //@CTK NO_BUF_OVERFLOW
```

Line 81-83 in File TRYSimple.sol

```
81    function balanceOf(address _owner) public constant returns (uint256 balance) {//
          constant==view
82        return balances[_owner];
83    }
```

✅ The code meets the specification

## Formal Verification Request 8

**Method will not encounter an assertion failure.**

10, May 2019
0.42 ms

Line 75 in File TRYSimple.sol

```
75    //@CTK NO_ASF
```

Line 81-83 in File TRYSimple.sol

```
81    function balanceOf(address _owner) public constant returns (uint256 balance) {//
          constant==view
82        return balances[_owner];
83    }
```

The code meets the specification

## Formal Verification Request 9

**balanceOf**

10, May 2019
0.42 ms

Line 76-80 in File TRYSimple.sol

```
76    /*@CTK "balanceOf"
77      @post (__reverted) == (false)
78      @post (balance) == (balances[_owner])
79      @post (this) == (__post)
80    */
```

Line 81-83 in File TRYSimple.sol

```
81    function balanceOf(address _owner) public constant returns (uint256 balance) {//
          constant==view
82        return balances[_owner];
83    }
```

The code meets the specification

## Formal Verification Request 10

**If method completes, integer overflow would not happen.**

10, May 2019
168.48 ms

Line 89 in File TRYSimple.sol

```
89    //@CTK NO_OVERFLOW
```

Line 105-116 in File TRYSimple.sol

```
105     function transfer(address _to, uint256 _value) public returns (bool success) {
106         require(_to != 0x0, 'cannot transfer to the zero address');
107         //  totalSupply                         (2^256 - 1).
108         require(balances[msg.sender] >= _value);
109         balances[msg.sender] = safeSub(balances[msg.sender], _value);
110         //                  token_value
111         balances[_to] = safeAdd(balances[_to], _value);
112         //            token_value
113         emit Transfer(msg.sender, _to, _value);
114         //
115         return true;
116     }
```

The code meets the specification

## Formal Verification Request 11

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

10, May 2019
12.16 ms

Line 90 in File TRYSimple.sol

```
90      //@CTK NO_BUF_OVERFLOW
```

Line 105-116 in File TRYSimple.sol

```
105     function transfer(address _to, uint256 _value) public returns (bool success) {
106         require(_to != 0x0, 'cannot transfer to the zero address');
107         //  totalSupply                         (2^256 - 1).
108         require(balances[msg.sender] >= _value);
109         balances[msg.sender] = safeSub(balances[msg.sender], _value);
110         //                  token_value
111         balances[_to] = safeAdd(balances[_to], _value);
112         //            token_value
113         emit Transfer(msg.sender, _to, _value);
114         //
115         return true;
116     }
```

The code meets the specification

## Formal Verification Request 12

**Method will not encounter an assertion failure.**

10, May 2019
11.71 ms

Line 91 in File TRYSimple.sol

```
91      //@CTK NO_ASF
```

Line 105-116 in File TRYSimple.sol

```
105     function transfer(address _to, uint256 _value) public returns (bool success) {
106         require(_to != 0x0, 'cannot transfer to the zero address');
107         //  totalSupply                         (2^256 - 1).
108         require(balances[msg.sender] >= _value);
109         balances[msg.sender] = safeSub(balances[msg.sender], _value);
110         //                  token_value
111         balances[_to] = safeAdd(balances[_to], _value);
112         //            token_value
113         emit Transfer(msg.sender, _to, _value);
114         //
115         return true;
116     }
```

The code meets the specification

## Formal Verification Request 13

**transfer2_same**

10, May 2019
59.4 ms

Line 92-97 in File TRYSimple.sol

```
92      /*@CTK "transfer2_same"
93        @pre (__reverted) == (false)
94        @pre (_to) == (msg.sender)
95        @post (__post.balances[_to]) == (balances[_to])
96        @post (success) == (true)
97      */
```

Line 105-116 in File TRYSimple.sol

```
105     function transfer(address _to, uint256 _value) public returns (bool success) {
106         require(_to != 0x0, 'cannot transfer to the zero address');
107         //  totalSupply                         (2^256 - 1).
108         require(balances[msg.sender] >= _value);
109         balances[msg.sender] = safeSub(balances[msg.sender], _value);
110         //                  token_value
111         balances[_to] = safeAdd(balances[_to], _value);
112         //            token_value
113         emit Transfer(msg.sender, _to, _value);
114         //
115         return true;
116     }
```

The code meets the specification

## Formal Verification Request 14

**transfer2**

10, May 2019
177.84 ms

Line 98-104 in File TRYSimple.sol

```
98    /*@CTK "transfer2"
99      @pre (__reverted) == (false)
100     @pre (_to) != (msg.sender)
101     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
102     @post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
103     @post (success) == (true)
104   */
```

Line 105-116 in File TRYSimple.sol

```
105   function transfer(address _to, uint256 _value) public returns (bool success) {
106       require(_to != 0x0, 'cannot transfer to the zero address');
107       //  totalSupply                          (2^256 - 1).
108       require(balances[msg.sender] >= _value);
109       balances[msg.sender] = safeSub(balances[msg.sender], _value);
110       //                  token_value
111       balances[_to] = safeAdd(balances[_to], _value);
112       //            token_value
113       emit Transfer(msg.sender, _to, _value);
114       //
115       return true;
116   }
```

The code meets the specification

## Formal Verification Request 15

**If method completes, integer overflow would not happen.**

10, May 2019
162.5 ms

Line 123 in File TRYSimple.sol

```
123   //@CTK NO_OVERFLOW
```

Line 139-153 in File TRYSimple.sol

```
139   function transferFrom(address _from, address _to, uint256 _value) public returns (
          bool success){
140       require(_to != 0x0, 'cannot transfer to the zero address');
141       require(_value > 0);
142       require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143
144       balances[_to] = safeAdd(balances[_to], _value);
145       //          token_value
146       balances[_from] = safeSub(balances[_from], _value);
147       //          _fromtoken_value
148       allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149       //                          _from_value
150       emit Transfer(_from, _to, _value);
151       //
152       return true;
153   }
```

The code meets the specification

## Formal Verification Request 16

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

10, May 2019
46.59 ms

Line 124 in File TRYSimple.sol

```
124     //@CTK NO_BUF_OVERFLOW
```

Line 139-153 in File TRYSimple.sol

```
139     function transferFrom(address _from, address _to, uint256 _value) public returns (
            bool success){
140         require(_to != 0x0, 'cannot transfer to the zero address');
141         require(_value > 0);
142         require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143 
144         balances[_to] = safeAdd(balances[_to], _value);
145         //          token_value
146         balances[_from] = safeSub(balances[_from], _value);
147         //          _fromtoken_value
148         allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149         //                          _from_value
150         emit Transfer(_from, _to, _value);
151         //
152         return true;
153     }
```

The code meets the specification

## Formal Verification Request 17

**Method will not encounter an assertion failure.**

10, May 2019
48.42 ms

Line 125 in File TRYSimple.sol

```
125     //@CTK NO_ASF
```

Line 139-153 in File TRYSimple.sol

```
139     function transferFrom(address _from, address _to, uint256 _value) public returns (
            bool success){
140         require(_to != 0x0, 'cannot transfer to the zero address');
141         require(_value > 0);
142         require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143 
144         balances[_to] = safeAdd(balances[_to], _value);
145         //          token_value
146         balances[_from] = safeSub(balances[_from], _value);
147         //          _fromtoken_value
148         allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149         //                          _from_value
150         emit Transfer(_from, _to, _value);
151         //
```

```
152         return true;
153     }
```

The code meets the specification

## Formal Verification Request 18

**transferFrom**

10, May 2019
187.48 ms

Line 126-132 in File TRYSimple.sol

```
126     /*@CTK "transferFrom"
127       @pre (__reverted) == (false)
128       @pre (_from) != (_to)
129       @post (success) == (true)
130       @post (__post.balances[_to]) == ((balances[_to]) + (_value))
131       @post (__post.balances[_from]) == ((balances[_from]) - (_value))
132     */
```

Line 139-153 in File TRYSimple.sol

```
139     function transferFrom(address _from, address _to, uint256 _value) public returns (
            bool success){
140         require(_to != 0x0, 'cannot transfer to the zero address');
141         require(_value > 0);
142         require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143 
144         balances[_to] = safeAdd(balances[_to], _value);
145         //          token_value
146         balances[_from] = safeSub(balances[_from], _value);
147         //          _fromtoken_value
148         allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149         //                          _from_value
150         emit Transfer(_from, _to, _value);
151         //
152         return true;
153     }
```

The code meets the specification

## Formal Verification Request 19

**transferFrom_same**

10, May 2019
192.39 ms

Line 133-138 in File TRYSimple.sol

```
133     /*@CTK "transferFrom_same"
134       @pre (__reverted) == (false)
135       @pre (_from) == (_to)
136       @post (success) == (true)
```

```
137     @post (__post.balances[_to]) == (balances[_to])
138   */
```

Line 139-153 in File TRYSimple.sol

```
139   function transferFrom(address _from, address _to, uint256 _value) public returns (
          bool success){
140       require(_to != 0x0, 'cannot transfer to the zero address');
141       require(_value > 0);
142       require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143
144       balances[_to] = safeAdd(balances[_to], _value);
145       //          token_value
146       balances[_from] = safeSub(balances[_from], _value);
147       //          _fromtoken_value
148       allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149       //                          _from_value
150       emit Transfer(_from, _to, _value);
151       //
152       return true;
153   }
```

The code meets the specification

## Formal Verification Request 20

**If method completes, integer overflow would not happen.**

10, May 2019
11.26 ms

Line 159 in File TRYSimple.sol

```
159   //@CTK NO_OVERFLOW
```

Line 166-170 in File TRYSimple.sol

```
166   function approve(address _spender, uint256 _value) public returns (bool success){
167       allowance[msg.sender][_spender] = _value;
168       emit Approval(msg.sender, _spender, _value);
169       return true;
170   }
```

The code meets the specification

## Formal Verification Request 21

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

10, May 2019
0.46 ms

Line 160 in File TRYSimple.sol

```
160   //@CTK NO_BUF_OVERFLOW
```

Line 166-170 in File TRYSimple.sol

```
166     function approve(address _spender, uint256 _value) public returns (bool success){
167         allowance[msg.sender][_spender] = _value;
168         emit Approval(msg.sender, _spender, _value);
169         return true;
170     }
```

The code meets the specification

## Formal Verification Request 22

**Method will not encounter an assertion failure.**

10, May 2019
0.45 ms

Line 161 in File TRYSimple.sol

```
161     //@CTK NO_ASF
```

Line 166-170 in File TRYSimple.sol

```
166     function approve(address _spender, uint256 _value) public returns (bool success){
167         allowance[msg.sender][_spender] = _value;
168         emit Approval(msg.sender, _spender, _value);
169         return true;
170     }
```

The code meets the specification

## Formal Verification Request 23

**approve**

10, May 2019
1.55 ms

Line 162-165 in File TRYSimple.sol

```
162     /*@CTK approve
163       @post __post.allowance[msg.sender][_spender] == _value
164       @post success == true
165      */
```

Line 166-170 in File TRYSimple.sol

```
166     function approve(address _spender, uint256 _value) public returns (bool success){
167         allowance[msg.sender][_spender] = _value;
168         emit Approval(msg.sender, _spender, _value);
169         return true;
170     }
```

The code meets the specification

## Formal Verification Request 24

**If method completes, integer overflow would not happen.**

10, May 2019
8.44 ms

Line 176 in File TRYSimple.sol

```
176     //@CTK NO_OVERFLOW
```

Line 182-184 in File TRYSimple.sol

```
182     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining){
183         return allowance[_owner][_spender];
184     }
```

The code meets the specification

## Formal Verification Request 25

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

10, May 2019
0.45 ms

Line 177 in File TRYSimple.sol

```
177     //@CTK NO_BUF_OVERFLOW
```

Line 182-184 in File TRYSimple.sol

```
182     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining){
183         return allowance[_owner][_spender];
184     }
```

The code meets the specification

## Formal Verification Request 26

**Method will not encounter an assertion failure.**

10, May 2019
0.45 ms

Line 178 in File TRYSimple.sol

```
178     //@CTK NO_ASF
```

Line 182-184 in File TRYSimple.sol

```
182     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining){
183         return allowance[_owner][_spender];
184     }
```

The code meets the specification

# Formal Verification Request 27

## allowance

10, May 2019
0.46 ms

Line 179-181 in File TRYSimple.sol

```
179     /*@CTK allowance
180       @post remaining == allowance[_owner][_spender]
181      */
```

Line 182-184 in File TRYSimple.sol

```
182     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining){
183         return allowance[_owner][_spender];
184     }
```

The code meets the specification

# Static Analysis Results

## INSECURE_COMPILER_VERSION

Line 1 in File TRYSimple.sol

```
1 pragma solidity ^0.4.24;
```

Only these compiler versions are safe to compile your code: 0.4.25

# Manual Review Notes

## Source Code SHA-256 Checksum

- **TRYSimple.sol** `98630c15e26c4417134aab9681ed66367ec4a96d29bab77da006318adbdb46b8`

## Summary

CertiK team is invited by Trias team to audit the design and implementations of its to be released ERC20 based smart contract, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checking as well as manual reviews by smart contract experts. We have been actively interacting with Trias engineers when there was any potential loopholes or recommended design changes during the audit process, and Trias team has been actively giving us updates for the source code and feedback about the business logic.

Overall we found the Trias team is very professional, and engaged in the discussion. The `TRYSimple` contract follow good practices of a ERC20 based smart contract features. The business logic and intentions are well-defined, straight-forward, and following industrial standards.

At this point the Trias team didn't provide other repositories sources as testing and documentation reference. We highly recommend the team to have more unit tests coverage together with documentation to simulate potential use cases and walk through all the functionalities. With the final update of source code and delivery of the audit report, we conclude that the contract is 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 seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

## Recommendations

Items in this section are low impact to the overall aspects of the smart contracts, thus will let client to decide whether to have those reflected in the final deployed version of source codes.

# Source Code with CertiK Labels

File TRYSimple.sol

```
1  pragma solidity ^0.4.24;
2  /**
3   * Math operations with safety checks
4   */
5  contract SafeMath {
6
7      /**
8       * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if subtrahend
            is greater than minuend).
9       */
10     /*@CTK "SafeMath sub"
11       @post (a < b) == __reverted
12       @post !__reverted -> __return == a - b
13       @post !__reverted -> !__has_overflow
14       @post !(__has_buf_overflow)
15       @post !(__has_assertion_failure)
16     */
17     function safeSub(uint256 a, uint256 b) internal pure returns (uint256) {
18         require(b <= a, "SafeMath: subtraction overflow");
19         uint256 c = a - b;
20         return c;
21     }
22
23     /**
24      * @dev Adds two unsigned integers, reverts on overflow.
25      */
26     /*@CTK "SafeMath add"
27       @post (a + b < a || a + b < b) == __reverted
28       @post !__reverted -> __return == a + b
29       @post !__reverted -> !__has_overflow
30       @post !(__has_buf_overflow)
31       @post !(__has_assertion_failure)
32     */
33     function safeAdd(uint256 a, uint256 b) internal pure returns (uint256) {
34         uint256 c = a + b;
35         require(c >= a, "SafeMath: addition overflow");
36         return c;
37     }
38 }
39
40 /**
41  * TRY Token by Trias
42  * As ERC20 standard
43  */
44 contract TRY is SafeMath{
45     uint256 public totalSupply;
46     string public name;
47     string public symbol;
48     uint8 public decimals;
49
50     //@CTK NO_OVERFLOW
51     //@CTK NO_BUF_OVERFLOW
52     //@CTK NO_ASF
```

```
53    //
                                    tokenpublicgettertotalSupply
          ().
54    //
55    constructor(uint256 _initialAmount,string _tokenName,uint8 _decimalUnits,string
          _tokenSymbol) public {
56        balances[msg.sender] = _initialAmount;
57        totalSupply = _initialAmount;
58        name = _tokenName;
59        decimals = _decimalUnits;
60        symbol = _tokenSymbol;
61    }
62
63    mapping(address => uint256) balances; //
64    mapping(address => mapping(address => uint256)) public allowance; //
65
66    //
67    event Transfer(address indexed from, address indexed to, uint256 value);
68    //    approve    (address _spender, uint256 _value)
69    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
70    // event Approval(address _owner,address _spender,uint256 _value);
71
72    ///             _ownertoken
73    //@CTK NO_OVERFLOW
74    //@CTK NO_BUF_OVERFLOW
75    //@CTK NO_ASF
76    /*@CTK "balanceOf"
77      @post (__reverted) == (false)
78      @post (balance) == (balances[_owner])
79      @post (this) == (__post)
80    */
81    function balanceOf(address _owner) public constant returns (uint256 balance) {//
          constant==view
82        return balances[_owner];
83    }
84
85    /**
86     * Send coins
87     * owner
88     */
89    //@CTK NO_OVERFLOW
90    //@CTK NO_BUF_OVERFLOW
91    //@CTK NO_ASF
92    /*@CTK "transfer2_same"
93      @pre (__reverted) == (false)
94      @pre (_to) == (msg.sender)
95      @post (__post.balances[_to]) == (balances[_to])
96      @post (success) == (true)
97    */
98    /*@CTK "transfer2"
99      @pre (__reverted) == (false)
100     @pre (_to) != (msg.sender)
101     @post (__post.balances[_to]) == ((balances[_to]) + (_value))
102     @post (__post.balances[msg.sender]) == ((balances[msg.sender]) - (_value))
103     @post (success) == (true)
104   */
105   function transfer(address _to, uint256 _value) public returns (bool success) {
106       require(_to != 0x0, 'cannot transfer to the zero address');
```

```
107         //  totalSupply                         (2^256 - 1).
108         require(balances[msg.sender] >= _value);
109         balances[msg.sender] = safeSub(balances[msg.sender], _value);
110         //                 token_value
111         balances[_to] = safeAdd(balances[_to], _value);
112         //           token_value
113         emit Transfer(msg.sender, _to, _value);
114         //
115         return true;
116     }
117 
118     /**
119      * A contract attempts to get the coins
120      *                           _from_to_valuetokenapprove
121      *           tokentoken
122      */
123     //@CTK NO_OVERFLOW
124     //@CTK NO_BUF_OVERFLOW
125     //@CTK NO_ASF
126     /*@CTK "transferFrom"
127       @pre (__reverted) == (false)
128       @pre (_from) != (_to)
129       @post (success) == (true)
130       @post (__post.balances[_to]) == ((balances[_to]) + (_value))
131       @post (__post.balances[_from]) == ((balances[_from]) - (_value))
132     */
133     /*@CTK "transferFrom_same"
134       @pre (__reverted) == (false)
135       @pre (_from) == (_to)
136       @post (success) == (true)
137       @post (__post.balances[_to]) == (balances[_to])
138     */
139     function transferFrom(address _from, address _to, uint256 _value) public returns (
        bool success){
140         require(_to != 0x0, 'cannot transfer to the zero address');
141         require(_value > 0);
142         require(balances[_from] >= _value && allowance[_from][msg.sender] >= _value);
143 
144         balances[_to] = safeAdd(balances[_to], _value);
145         //           token_value
146         balances[_from] = safeSub(balances[_from], _value);
147         //           _fromtoken_value
148         allowance[_from][msg.sender] = safeSub(allowance[_from][msg.sender], _value);
149         //                          _from_value
150         emit Transfer(_from, _to, _value);
151         //
152         return true;
153     }
154 
155     /**
156      * Allow another contract to spend some tokens in your behalf
157      *                                 _spender_valuetoken
158      */
159     //@CTK NO_OVERFLOW
160     //@CTK NO_BUF_OVERFLOW
161     //@CTK NO_ASF
162     /*@CTK approve
163       @post __post.allowance[msg.sender][_spender] == _value
```

```
164       @post success == true
165     */
166     function approve(address _spender, uint256 _value) public returns (bool success){
167         allowance[msg.sender][_spender] = _value;
168         emit Approval(msg.sender, _spender, _value);
169         return true;
170     }
171 
172     /**
173      *                    _spender_ownertoken
174      *                                                ,        Token
175      */
176     //@CTK NO_OVERFLOW
177     //@CTK NO_BUF_OVERFLOW
178     //@CTK NO_ASF
179     /*@CTK allowance
180       @post remaining == allowance[_owner][_spender]
181      */
182     function allowance(address _owner, address _spender) public constant returns (
            uint256 remaining){
183         return allowance[_owner][_spender];
184     }
185 
186 }
```