

CERTIK VERIFICATION REPORT FOR TETHER



Request Date: 2019-03-27
Revision Date: 2019-04-11



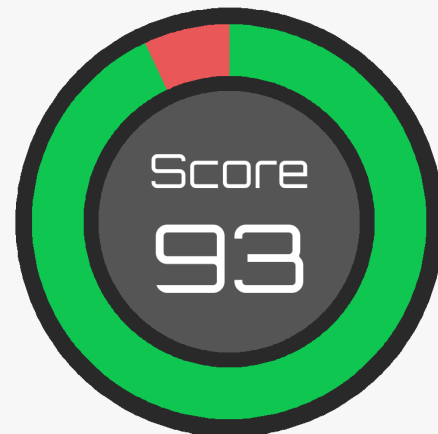
Disclaimer

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 Tether (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.

PASS

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

Apr 11, 2019



Summary

This audit report summarises the smart contract verification service requested by Tether. The goal of this security audit is to guarantee that the audited smart contracts are robust enough to avoid any potential security loopholes.

The result of this report is only a reflection of the source code that was determined in this scope, and of the source code at the time of the audit.

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	Com-	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	Ran-	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	for	tx.origin should not be used for authorization. Use msg.sender instead.	0	SWC-115
Delegatecall to Untrusted Callee	to	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Variable	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Default	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

Please see Manual Review Details for reference.

Manual Review Details

StandardTokenWithFees.sol

- **transfer()** – guard against the case where fee is higher than value.
- **transferFrom()** – guard against the case where fee is higher than value.

- **setParams()** – either check decimals against a constant max limit, or check `log(uint(10)**decimals)== decimals and (maximumFee/(uint(10)**decimals))== newMaxFee`.

The logic of this contract is as clear as its naming, it basically inherits the Standard-Token library from OpenZeppelin while adding the fee implementation given the transfer related functions: `transfer` and `transferFrom`. Users should be aware that whenever the token owner sets the fee to a number greater than 0, the transaction amount to the receiving party would be less than the original amount (less the fee). Given the fact that `basisPointsRate` is a public variable, users could invoke the getter function to confirm the current rate and decide whether to continue the action.

One thing to notice is that the version of OpenZeppelin is 1.4.0, which was released on Nov 23, 2017. Some new features were introduced since then, so we suggest to use the latest release together with updating the solidity pragma to latest so we could leverage a safer compiler.

TetherToken.sol

- **issue()** – check for `!deprecated` and no arithmetic overflow.
- **redeem()** – check for `!deprecated` and no arithmetic overflow.
- **deprecate()** – check `_upgradedAddress != upgradedAddress` and `!deprecated**`.
- **transfer()** – consider having a modifier for `require(!isBlackListed[msg.sender])` to make the code more decoupled and test friendly.
- **transferFrom()** – consider having a modifier for `require(!isBlackListed[_from])` to make the code more decoupled and test friendly.

TetherToken basically implements all the ERC20 interfaces to serve its functioning. It introduced the deprecated status, whereas all invocations will be redirected to the upgraded contract if the current token is deprecated (only the owner has the ability to set it to true and to provide a new address). From the code repository, we only see the interface definition of `UpgradedStandardToken`, thus we cannot guarantee the level of security when upgraded to a new token contract. This feature increases the extensibility of the TetherToken, however potentially if the new address for the token could be some contract not fully audited, this double-bladed feature will turn to be overpowerful centralization leading to uncontrollable situations. (however the chance is very low though). On the good side, since the TetherToken is owned by a multisig wallet, the chance for compromised ownership is extremely low.

TetherToken is not a payable contract, which means there is no function that has payable keyword or any fallback function that could receive value. This minimized the potential for hackers to attack given no value stored. The actual payable logic is handled by its `MultiSigWallet` contract, which is controlled by a group of admins to decide the execution of transactions.

MultiSigWallet.sol

- **changeRequirement ()** – require explicit `changeRequirement()` and avoid doing implicit requirement change in `removeOwner()`.

- **getConfirmations()** – consider tracking a confirmed counter so that `getConfirmations()` will not need to allocate for the size of the whole owner set.
- **getTransactionIds()** – consider tracking `from` and `to` transaction count so that `getTransactionIds()` will not need to allocate for the size of the whole transaction set; also should check for `from j= to`, ensure `from` is always less than or equal to `to`.
- **replaceOwner()** – consider adding the `validRequirement()` to ensure that at least one valid owner address.
- **ownerDoesNotExist()** – add check for invalid address (`0x0`)
- **modifiers** – suggest to use `require` instead of `if/throw`.

MultiSigWallet contract follows the common practices and standard workflows as the purpose and use case of a multisignature wallet. It provides enough transparency by exposing many getters for public to invoke, such as the list of owners or confirmations of any given transaction. We assume only the Tether organization administrators shall have the privilege to be invited as owner group and decide on which transactions to confirm and execute. All the state change functions must be sent from wallet address directly, this greatly increased the security level as each transaction must be agreed upon by owner group in the first place (thus malicious transactions will never get picked up). Regardless of whether the design goes against the decentralized trust or autonomy, Tether holders/users should be aware that the group of Tether organization administrators is the key base and the fundamental trust. Given the nature of stable coin, we believe this is the ideal and rational model to protect assets of end users by applying the centralized governing from Tether organization.

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.

Source Code with CertiK Labels

File BlackList.sol

```

1 pragma solidity ^0.4.18;
2
3 import "zeppelin-solidity/contracts/ownership/Ownable.sol";
4
5 contract BlackList is Ownable {
6
7     ////////// Getter to allow the same blacklist to be used also by other contracts (
8         including upgraded Tether) //////////
9     /*@CTK getBlackListStatus
10        @post __return == isBlackListed[_maker]
11    */
12    function getBlackListStatus(address _maker) external constant returns (bool) {
13        return isBlackListed[_maker];
14    }
15
16    mapping (address => bool) public isBlackListed;
17
18    /*@CTK addBlackList
19        @tag assume_completion
20        @post owner == msg.sender
21        @post __post.isBlackListed[_evilUser]
22    */
23    function addBlackList (address _evilUser) public onlyOwner {
24        isBlackListed[_evilUser] = true;
25        AddedBlackList(_evilUser);
26    }
27
28    /*@CTK removeBlackList
29        @tag assume_completion
30        @post owner == msg.sender
31        @post !__post.isBlackListed[_clearedUser]
32    */
33    function removeBlackList (address _clearedUser) public onlyOwner {
34        isBlackListed[_clearedUser] = false;
35        RemovedBlackList(_clearedUser);
36    }
37
38    event AddedBlackList(address indexed _user);
39
40    event RemovedBlackList(address indexed _user);
41 }

```

File StandardTokenWithFees.sol

```

1 pragma solidity ^0.4.18;
2
3 // import "zeppelin-solidity/contracts/token/StandardToken.sol";
4 import "zeppelin-solidity/contracts/token/ERC20/StandardToken.sol";
5 import "zeppelin-solidity/contracts/ownership/Ownable.sol";
6
7 contract StandardTokenWithFees is StandardToken, Ownable {
8
9     // Additional variables for use if transaction fees ever became necessary
10    uint256 public basisPointsRate = 0;

```

```

11  uint256 public maximumFee = 0;
12  uint256 constant MAX_SETTABLE_BASIS_POINTS = 20;
13  uint256 constant MAX_SETTABLE_FEE = 50;
14
15  string public name;
16  string public symbol;
17  uint8 public decimals;
18  uint public _totalSupply;
19
20  uint public constant MAX_UINT = 2**256 - 1;
21
22  /*@CTK calcFee_maximumFee
23   @tag assume_completion
24   @post __return <= maximumFee
25   */
26  /*@CTK calcFee
27   @tag assume_completion
28   @pre _value * basisPointsRate / 10000 <= maximumFee
29   @post __return == _value * basisPointsRate / 10000
30   */
31  function calcFee(uint _value) constant returns (uint) {
32    uint fee = (_value.mul(basisPointsRate)).div(10000);
33    if (fee > maximumFee) {
34      fee = maximumFee;
35    }
36    return fee;
37  }
38
39  /*@CTK transfer
40   @tag assume_completion
41   @pre _value * basisPointsRate / 10000 <= maximumFee
42   @pre msg.sender != _to && msg.sender != owner && _to != owner
43   @post __post.balances[msg.sender] == balances[msg.sender] - _value
44   @post __post.balances[_to] == balances[_to] +
45         (_value - _value * basisPointsRate / 10000)
46   @post __post.balances[owner] == balances[owner] +
47         _value * basisPointsRate / 10000
48   */
49  // CTK: transfer _value to _to. then transfer fee from msg.sender
50  function transfer(address _to, uint _value) public returns (bool) {
51    uint fee = calcFee(_value);
52    uint sendAmount = _value.sub(fee);
53
54    super.transfer(_to, sendAmount);
55    if (fee > 0) {
56      super.transfer(owner, fee);
57    }
58  }
59
60  /*@CTK transferFrom
61   @tag assume_completion
62   @pre allowed[_from][msg.sender] < MAX_UINT
63   @pre _from != _to && _from != owner && _to != owner
64   @pre _value * basisPointsRate / 10000 <= maximumFee
65   @post _to != address(0)
66   @post _value <= balances[_from]
67   @post _value <= allowed[_from][msg.sender]
68   @post __post.balances[_from] == balances[_from] - _value

```



```

69     @post __post.balances[_to] == balances[_to] +
70         (_value - _value * basisPointsRate / 10000)
71     @post __post.balances[owner] == balances[owner] +
72         _value * basisPointsRate / 10000
73     @post __post.allowed[_from][msg.sender] ==
74         allowed[_from][msg.sender] - _value
75     */
76     function transferFrom(address _from, address _to, uint256 _value) public returns (
77         bool) {
78         require(_to != address(0));
79         require(_value <= balances[_from]);
80         require(_value <= allowed[_from][msg.sender]);
81
82         uint fee = calcFee(_value);
83         uint sendAmount = _value.sub(fee);
84
85         balances[_from] = balances[_from].sub(_value);
86         balances[_to] = balances[_to].add(sendAmount);
87         if (allowed[_from][msg.sender] < MAX_UINT) {
88             allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
89         }
90         Transfer(_from, _to, sendAmount);
91         if (fee > 0) {
92             balances[owner] = balances[owner].add(fee);
93             Transfer(_from, owner, fee);
94         }
95         return true;
96     }
97     /*CTK setParams
98     @tag assume_completion
99     @pre decimals == 3
100    @post newBasisPoints < MAX_SETTABLE_BASIS_POINTS
101    @post newMaxFee < MAX_SETTABLE_FEE
102    @post owner == msg.sender
103    @post __post.basisPointsRate == newBasisPoints
104    @post __post.maximumFee == newMaxFee * 1000
105    */
106    function setParams(uint newBasisPoints, uint newMaxFee) public onlyOwner {
107        // Ensure transparency by hardcoding limit beyond which fees can never be added
108        require(newBasisPoints < MAX_SETTABLE_BASIS_POINTS);
109        require(newMaxFee < MAX_SETTABLE_FEE);
110
111        basisPointsRate = newBasisPoints;
112        maximumFee = newMaxFee.mul(uint(10)**decimals);
113
114        Params(basisPointsRate, maximumFee);
115    }
116
117    // Called if contract ever adds fees
118    event Params(uint feeBasisPoints, uint maxFee);
119
120 }

```

File TetherToken.sol

```

1 pragma solidity ^0.4.18;
2
3 import "./StandardTokenWithFees.sol";

```

```

4 import "zeppelin-solidity/contracts/lifecycle/Pausable.sol";
5 import "../UpgradedStandardToken.sol";
6 import "../BlackList.sol";
7
8
9 contract TetherToken is Pausable, StandardTokenWithFees, BlackList {
10
11     address public upgradedAddress;
12     bool public deprecated;
13
14     // The contract can be initialized with a number of tokens
15     // All the tokens are deposited to the owner address
16     //
17     // @param _balance Initial supply of the contract
18     // @param _name Token Name
19     // @param _symbol Token symbol
20     // @param _decimals Token decimals
21     /*@CTK TetherToken
22         @post __post._totalSupply == _initialSupply
23         @post __post.name == _name
24         @post __post.symbol == _symbol
25         @post __post.decimals == _decimals
26         @post __post.balances[owner] == _initialSupply
27         @post __post.deprecated == false
28     */
29     function TetherToken(uint _initialSupply, string _name, string _symbol, uint8
        _decimals) public {
30         _totalSupply = _initialSupply;
31         name = _name;
32         symbol = _symbol;
33         decimals = _decimals;
34         balances[owner] = _initialSupply;
35         deprecated = false;
36     }
37
38     // Forward ERC20 methods to upgraded contract if this one is deprecated
39     /*@CTK transfer
40         @tag assume_completion
41         @post !isBlackListed[msg.sender]
42         @post !paused
43     */
44     function transfer(address _to, uint _value) public whenNotPaused returns (bool) {
45         require(!isBlackListed[msg.sender]);
46         if (deprecated) {
47             return UpgradedStandardToken(upgradedAddress).transferByLegacy(msg.sender,
                _to, _value);
48         } else {
49             return super.transfer(_to, _value);
50         }
51     }
52
53     // Forward ERC20 methods to upgraded contract if this one is deprecated
54     /*@CTK transferFrom
55         @tag assume_completion
56         @post !isBlackListed[msg.sender]
57         @post !paused
58     */
59     function transferFrom(address _from, address _to, uint _value) public

```

```
        whenNotPaused returns (bool) {
60         require(!isBlackListed[_from]);
61         if (deprecated) {
62             return UpgradedStandardToken(upgradedAddress).transferFromByLegacy(msg.
                sender, _from, _to, _value);
63         } else {
64             return super.transferFrom(_from, _to, _value);
65         }
66     }
67
68     // Forward ERC20 methods to upgraded contract if this one is deprecated
69     function balanceOf(address who) public constant returns (uint) {
70         if (deprecated) {
71             return UpgradedStandardToken(upgradedAddress).balanceOf(who);
72         } else {
73             return super.balanceOf(who);
74         }
75     }
76
77     // Allow checks of balance at time of deprecation
78     /*@CTK balanceOf
79     @pre deprecated
80     @post __return == balances[who]
81     */
82     function oldBalanceOf(address who) public constant returns (uint) {
83         if (deprecated) {
84             return super.balanceOf(who);
85         }
86     }
87
88     // Forward ERC20 methods to upgraded contract if this one is deprecated
89     /*@CTK approve
90     @tag assume_completion
91     @pre !deprecated
92     @post !paused
93     @post __post.allowed[msg.sender][_spender] == _value
94     */
95     function approve(address _spender, uint _value) public whenNotPaused returns (bool
        ) {
96         if (deprecated) {
97             return UpgradedStandardToken(upgradedAddress).approveByLegacy(msg.sender,
                _spender, _value);
98         } else {
99             return super.approve(_spender, _value);
100        }
101    }
102
103    /*@CTK increaseApproval
104    @tag assume_completion
105    @pre !deprecated
106    @post __post.allowed[msg.sender][_spender] ==
107        allowed[msg.sender][_spender] + _addedValue
108    */
109    function increaseApproval(address _spender, uint _addedValue) public whenNotPaused
        returns (bool) {
110        if (deprecated) {
111            return UpgradedStandardToken(upgradedAddress).increaseApprovalByLegacy(msg.
                sender, _spender, _addedValue);
```

```

112     } else {
113         return super.increaseApproval(_spender, _addedValue);
114     }
115 }
116
117 /*@CTK decreaseApproval
118  @tag assume_completion
119  @pre !deprecated
120  @post allowed[msg.sender][_spender] >= _subtractedValue ->
121  __post.allowed[msg.sender][_spender] ==
122  allowed[msg.sender][_spender] - _subtractedValue
123  @post allowed[msg.sender][_spender] < _subtractedValue ->
124  __post.allowed[msg.sender][_spender] == 0
125  */
126 function decreaseApproval(address _spender, uint _subtractedValue) public
127     whenNotPaused returns (bool) {
128     if (deprecated) {
129         return UpgradedStandardToken(upgradedAddress).decreaseApprovalByLegacy(msg.
130             sender, _spender, _subtractedValue);
131     } else {
132         return super.decreaseApproval(_spender, _subtractedValue);
133     }
134 }
135
136 // Forward ERC20 methods to upgraded contract if this one is deprecated
137 /*@CTK allowance
138  @pre !deprecated
139  @post remaining == allowed[_owner][_spender]
140  */
141 function allowance(address _owner, address _spender) public constant returns (uint
142     remaining) {
143     if (deprecated) {
144         return StandardToken(upgradedAddress).allowance(_owner, _spender);
145     } else {
146         return super.allowance(_owner, _spender);
147     }
148 }
149
150 // deprecate current contract in favour of a new one
151 /*@CTK deprecate
152  @tag assume_completion
153  @post owner == msg.sender
154  @post _upgradedAddress != address(0)
155  @post __post.upgradedAddress == _upgradedAddress
156  @post __post.deprecated
157  */
158 function deprecate(address _upgradedAddress) public onlyOwner {
159     require(_upgradedAddress != address(0));
160     deprecated = true;
161     upgradedAddress = _upgradedAddress;
162     Deprecate(_upgradedAddress);
163 }
164
165 // deprecate current contract if favour of a new one
166 /*@CTK totalSupply
167  @pre !deprecated
168  @post __return == _totalSupply
169  */

```

```

167 function totalSupply() public constant returns (uint) {
168     if (deprecated) {
169         return StandardToken(upgradedAddress).totalSupply();
170     } else {
171         return _totalSupply;
172     }
173 }
174
175 // Issue a new amount of tokens
176 // these tokens are deposited into the owner address
177 //
178 // @param _amount Number of tokens to be issued
179 /*@CTK issue
180     @tag assume_completion
181     @post owner == msg.sender
182     @post __post.balances[owner] == balances[owner] + amount
183     @post __post._totalSupply == _totalSupply + amount
184 */
185 function issue(uint amount) public onlyOwner {
186     balances[owner] = balances[owner].add(amount);
187     _totalSupply = _totalSupply.add(amount);
188     Issue(amount);
189     Transfer(address(0), owner, amount);
190 }
191
192 // Redeem tokens.
193 // These tokens are withdrawn from the owner address
194 // if the balance must be enough to cover the redeem
195 // or the call will fail.
196 // @param _amount Number of tokens to be issued
197 /*@CTK redeem
198     @tag assume_completion
199     @post owner == msg.sender
200     @post __post._totalSupply == _totalSupply - amount
201     @post __post.balances[owner] == balances[owner] - amount
202 */
203 function redeem(uint amount) public onlyOwner {
204     _totalSupply = _totalSupply.sub(amount);
205     balances[owner] = balances[owner].sub(amount);
206     Redeem(amount);
207     Transfer(owner, address(0), amount);
208 }
209
210 /*@CTK destroyBlackFunds
211     @tag assume_completion
212     @post owner == msg.sender
213     @post isBlackListed[_blackListedUser]
214     @post __post._totalSupply == _totalSupply - balances[_blackListedUser]
215 */
216 function destroyBlackFunds (address _blackListedUser) public onlyOwner {
217     require(isBlackListed[_blackListedUser]);
218     uint dirtyFunds = balanceOf(_blackListedUser);
219     balances[_blackListedUser] = 0;
220     _totalSupply = _totalSupply.sub(dirtyFunds);
221     DestroyedBlackFunds(_blackListedUser, dirtyFunds);
222 }
223
224 event DestroyedBlackFunds(address indexed _blackListedUser, uint _balance);
    
```

```

225
226 // Called when new token are issued
227 event Issue(uint amount);
228
229 // Called when tokens are redeemed
230 event Redeem(uint amount);
231
232 // Called when contract is deprecated
233 event Deprecate(address newAddress);
234
235 }

```

File MultiSigWallet.sol

```

1 pragma solidity ^0.4.10;
2
3 /// @title Multisignature wallet - Allows multiple parties to agree on transactions
  before execution.
4 /// @author Stefan George - <stefan.george@consensys.net>
5 contract MultiSigWallet {
6
7     uint constant public MAX_OWNER_COUNT = 50;
8
9     event Confirmation(address indexed sender, uint indexed transactionId);
10    event Revocation(address indexed sender, uint indexed transactionId);
11    event Submission(uint indexed transactionId);
12    event Execution(uint indexed transactionId);
13    event ExecutionFailure(uint indexed transactionId);
14    event Deposit(address indexed sender, uint value);
15    event OwnerAddition(address indexed owner);
16    event OwnerRemoval(address indexed owner);
17    event RequirementChange(uint required);
18
19    mapping (uint => Transaction) public transactions;
20    mapping (uint => mapping (address => bool)) public confirmations;
21    mapping (address => bool) public isOwner;
22    address[] public owners;
23    uint public required;
24    uint public transactionCount;
25
26    struct Transaction {
27        address destination;
28        uint value;
29        bytes data;
30        bool executed;
31    }
32
33    modifier onlyWallet() {
34        if (msg.sender != address(this))
35            throw;
36        _;
37    }
38
39    modifier ownerDoesNotExist(address owner) {
40        if (isOwner[owner])
41            throw;
42        _;
43    }
44

```

```

45     modifier ownerExists(address owner) {
46         if (!isOwner[owner])
47             throw;
48         _;
49     }
50
51     modifier transactionExists(uint transactionId) {
52         if (transactions[transactionId].destination == 0)
53             throw;
54         _;
55     }
56
57     modifier confirmed(uint transactionId, address owner) {
58         if (!confirmations[transactionId][owner])
59             throw;
60         _;
61     }
62
63     modifier notConfirmed(uint transactionId, address owner) {
64         if (confirmations[transactionId][owner])
65             throw;
66         _;
67     }
68
69     modifier notExecuted(uint transactionId) {
70         if (transactions[transactionId].executed)
71             throw;
72         _;
73     }
74
75     modifier notNull(address _address) {
76         if (_address == 0)
77             throw;
78         _;
79     }
80
81     modifier validRequirement(uint ownerCount, uint _required) {
82         if ( ownerCount > MAX_OWNER_COUNT
83             || _required > ownerCount
84             || _required == 0
85             || ownerCount == 0)
86             throw;
87         _;
88     }
89
90     /// @dev Fallback function allows to deposit ether.
91     function()
92         payable
93     {
94         if (msg.value > 0)
95             Deposit(msg.sender, msg.value);
96     }
97
98     /*
99     * Public functions
100    */
101    /// @dev Contract constructor sets initial owners and required number of
        confirmations.

```

```

102  /// @param _owners List of initial owners.
103  /// @param _required Number of required confirmations.
104  function MultiSigWallet(address[] _owners, uint _required)
105      public
106      validRequirement(_owners.length, _required)
107  {
108      for (uint i=0; i<_owners.length; i++) {
109          if (isOwner[_owners[i]] || _owners[i] == 0)
110              throw;
111          isOwner[_owners[i]] = true;
112      }
113      owners = _owners;
114      required = _required;
115  }
116
117  /// @dev Allows to add a new owner. Transaction has to be sent by wallet.
118  /// @param owner Address of new owner.
119  /*@CTK addOwner
120   @tag assume_completion
121   @post msg.sender == address(this)
122   @post owner != address(0)
123   @post !isOwner[owner]
124   @post __post.isOwner[owner]
125   @post __post.owners[owners.length] == owner
126   @post required != 0
127   @post required <= __post.owners.length
128  */
129  function addOwner(address owner)
130      public
131      onlyWallet
132      ownerDoesNotExist(owner)
133      notNull(owner)
134      validRequirement(owners.length + 1, required)
135  {
136      isOwner[owner] = true;
137      owners.push(owner);
138      OwnerAddition(owner);
139  }
140
141  /// @dev Allows to remove an owner. Transaction has to be sent by wallet.
142  /// @param owner Address of owner.
143  function removeOwner(address owner)
144      public
145      onlyWallet
146      ownerExists(owner)
147  {
148      isOwner[owner] = false;
149      for (uint i=0; i<owners.length - 1; i++)
150          if (owners[i] == owner) {
151              owners[i] = owners[owners.length - 1];
152              break;
153          }
154      owners.length -= 1;
155      if (required > owners.length)
156          changeRequirement(owners.length);
157      OwnerRemoval(owner);
158  }
159

```



```

160     /// @dev Allows to replace an owner with a new owner. Transaction has to be sent
161     /// by wallet.
162     /// @param owner Address of owner to be replaced.
163     /// @param newOwner Address of new owner.
164     function replaceOwner(address owner, address newOwner)
165     public
166     onlyWallet
167     ownerExists(owner)
168     ownerDoesNotExist(newOwner)
169     {
170     for (uint i=0; i<owners.length; i++)
171     if (owners[i] == owner) {
172     owners[i] = newOwner;
173     break;
174     }
175     isOwner[owner] = false;
176     isOwner[newOwner] = true;
177     OwnerRemoval(owner);
178     OwnerAddition(newOwner);
179     }
180     /// @dev Allows to change the number of required confirmations. Transaction has to
181     /// be sent by wallet.
182     /// @param _required Number of required confirmations.
183     /*@CTK changeRequirement
184     @tag assume_completion
185     @post __post.required == _required
186     */
187     function changeRequirement(uint _required)
188     public
189     onlyWallet
190     validRequirement(owners.length, _required)
191     {
192     required = _required;
193     RequirementChange(_required);
194     }
195     /// @dev Allows an owner to submit and confirm a transaction.
196     /// @param destination Transaction target address.
197     /// @param value Transaction ether value.
198     /// @param data Transaction data payload.
199     /// @return Returns transaction ID.
200     function submitTransaction(address destination, uint value, bytes data)
201     public
202     returns (uint transactionId)
203     {
204     transactionId = addTransaction(destination, value, data);
205     confirmTransaction(transactionId);
206     }
207     /// @dev Allows an owner to confirm a transaction.
208     /// @param transactionId Transaction ID.
209     /*@CTK confirmTransaction
210     @post !isOwner[msg.sender]
211     @post transactions[transactionId].destination != address(0)
212     @post !confirmations[transactionId][msg.sender]
213     @post __post.confirmations[transactionId][msg.sender]
214     */
215     */

```

```

216 function confirmTransaction(uint transactionId)
217     public
218     ownerExists(msg.sender)
219     transactionExists(transactionId)
220     notConfirmed(transactionId, msg.sender)
221     {
222         confirmations[transactionId][msg.sender] = true;
223         Confirmation(msg.sender, transactionId);
224         executeTransaction(transactionId);
225     }
226
227     /// @dev Allows an owner to revoke a confirmation for a transaction.
228     /// @param transactionId Transaction ID.
229     function revokeConfirmation(uint transactionId)
230         public
231         ownerExists(msg.sender)
232         confirmed(transactionId, msg.sender)
233         notExecuted(transactionId)
234         {
235             confirmations[transactionId][msg.sender] = false;
236             Revocation(msg.sender, transactionId);
237         }
238
239     /// @dev Allows anyone to execute a confirmed transaction.
240     /// @param transactionId Transaction ID.
241     function executeTransaction(uint transactionId)
242         public
243         notExecuted(transactionId)
244         {
245             if (isConfirmed(transactionId)) {
246                 Transaction tx = transactions[transactionId];
247                 tx.executed = true;
248                 if (tx.destination.call.value(tx.value)(tx.data))
249                     Execution(transactionId);
250             } else {
251                 ExecutionFailure(transactionId);
252                 tx.executed = false;
253             }
254         }
255     }
256
257     /// @dev Returns the confirmation status of a transaction.
258     /// @param transactionId Transaction ID.
259     /// @return Confirmation status.
260     function isConfirmed(uint transactionId)
261         public
262         constant
263         returns (bool)
264         {
265             uint count = 0;
266             for (uint i=0; i<owners.length; i++) {
267                 if (confirmations[transactionId][owners[i]])
268                     count += 1;
269                 if (count == required)
270                     return true;
271             }
272         }
273

```

```

274  /*
275  * Internal functions
276  */
277  /// @dev Adds a new transaction to the transaction mapping, if transaction does
    not exist yet.
278  /// @param destination Transaction target address.
279  /// @param value Transaction ether value.
280  /// @param data Transaction data payload.
281  /// @return Returns transaction ID.
282  /*@CTK addTransaction
283  @tag assume_completion
284  @post destination != address(0)
285  @post __post.transactions[transactionCount].destination == destination
286  @post __post.transactions[transactionCount].value == value
287  @post __post.transactions[transactionCount].data == data
288  @post __post.transactions[transactionCount].executed == false
289  @post __post.transactionCount == transactionCount + 1
290  @post transactionId == transactionCount
291  */
292  function addTransaction(address destination, uint value, bytes data)
293  internal
294  notNull(destination)
295  returns (uint transactionId)
296  {
297  transactionId = transactionCount;
298  transactions[transactionId].destination = destination;
299  transactions[transactionId].value = value;
300  transactions[transactionId].data = data;
301  transactions[transactionId].executed = false;
302  // transactions[transactionId] = Transaction({
303  //   destination: destination,
304  //   value: value,
305  //   data: data,
306  //   executed: false
307  // });
308  transactionCount += 1;
309  Submission(transactionId);
310  }
311
312  /*
313  * Web3 call functions
314  */
315  /// @dev Returns number of confirmations of a transaction.
316  /// @param transactionId Transaction ID.
317  /// @return Number of confirmations.
318  function getConfirmationCount(uint transactionId)
319  public
320  constant
321  returns (uint count)
322  {
323  for (uint i=0; i<owners.length; i++)
324  if (confirmations[transactionId][owners[i]])
325  count += 1;
326  }
327
328  /// @dev Returns total number of transactions after filters are applied.
329  /// @param pending Include pending transactions.
330  /// @param executed Include executed transactions.

```

```

331     /// @return Total number of transactions after filters are applied.
332     function getTransactionCount(bool pending, bool executed)
333         public
334         constant
335         returns (uint count)
336     {
337         for (uint i=0; i<transactionCount; i++)
338             if ( pending && !transactions[i].executed
339                 || executed && transactions[i].executed)
340                 count += 1;
341     }
342
343     /// @dev Returns list of owners.
344     /// @return List of owner addresses.
345     /*@CTK getOwners
346     @post __return == owners
347     */
348     function getOwners()
349         public
350         constant
351         returns (address[])
352     {
353         return owners;
354     }
355
356     /// @dev Returns array with owner addresses, which confirmed transaction.
357     /// @param transactionId Transaction ID.
358     /// @return Returns array of owner addresses.
359     function getConfirmations(uint transactionId)
360         public
361         constant
362         returns (address[] _confirmations)
363     {
364         address[] memory confirmationsTemp = new address[](owners.length);
365         uint count = 0;
366         uint i;
367         for (i=0; i<owners.length; i++)
368             if (confirmations[transactionId][owners[i]]) {
369                 confirmationsTemp[count] = owners[i];
370                 count += 1;
371             }
372         _confirmations = new address[](count);
373         for (i=0; i<count; i++)
374             _confirmations[i] = confirmationsTemp[i];
375     }
376
377     /// @dev Returns list of transaction IDs in defined range.
378     /// @param from Index start position of transaction array.
379     /// @param to Index end position of transaction array.
380     /// @param pending Include pending transactions.
381     /// @param executed Include executed transactions.
382     /// @return Returns array of transaction IDs.
383     function getTransactionIds(uint from, uint to, bool pending, bool executed)
384         public
385         constant
386         returns (uint[] _transactionIds)
387     {
388         uint[] memory transactionIdsTemp = new uint[](transactionCount);

```

```

389     uint count = 0;
390     uint i;
391     for (i=0; i<transactionCount; i++)
392         if ( pending && !transactions[i].executed
393             || executed && transactions[i].executed)
394         {
395             transactionIdsTemp[count] = i;
396             count += 1;
397         }
398     _transactionIds = new uint[](to - from);
399     for (i=from; i<to; i++)
400         _transactionIds[i - from] = transactionIdsTemp[i];
401 }
402 }

```

File UpgradedTokenTest.sol

```

1  pragma solidity ^0.4.18;
2
3  import "zeppelin-solidity/contracts/ownership/Ownable.sol";
4  import "./UpgradedStandardToken.sol";
5
6  contract PreviousTokenInterface {
7      /*@CTK oldBalanceOf
8         @tag spec
9         @post __return == 1
10        @post __reverted == false
11        */
12     function oldBalanceOf(address who) public constant returns (uint);
13 }
14
15
16 contract UpgradedTokenTest is Ownable, UpgradedStandardToken {
17
18     mapping (address => bool) userMigrated;
19     address public oldAddress;
20
21     // Ugly example of ensuring balances are continuous - warning possible
22     // vulnerabilities
23     /*@CTK userMigrate
24        @post __post.userMigrated[user]
25        */
26     function userMigrate(address user) public {
27         if(!userMigrated[user]) {
28             balances[user] = PreviousTokenInterface(oldAddress).oldBalanceOf(user);
29             userMigrated[user] = true;
30         }
31     }
32
33     /*@CTK balanceOf
34        @post __post.userMigrated[who]
35        @post __return == __post.balances[who]
36        */
37     function balanceOf(address who) public constant returns (uint) {
38         userMigrate(who);
39         return super.balanceOf(who);
40     }
41
42     // Example method, do not copy! In the real world it should

```

```

42 // ensure that 'msg.sender' is the legacy contract
43 /*@CTK transferByLegacy
44   @tag assume_completion
45   @pre from != to
46   @pre userMigrated[from]
47   @pre userMigrated[to]
48   @post __post.userMigrated[from]
49   @post __post.userMigrated[to]
50   @post __post.balances[from] == balances[from] - value
51   @post __post.balances[to] == balances[to] + value
52 */
53 function transferByLegacy(address from, address to, uint value) public returns (
54   bool) {
55   userMigrate(from);
56   userMigrate(to);
57   balances[from] = balances[from].sub(value);
58   balances[to] = balances[to].add(value);
59   Transfer(from, to, value);
60 }
61 // Example method, do not copy! In the real world it should
62 // ensure that 'msg.sender' is the legacy contract
63 /*@CTK approveByLegacy
64   @tag assume_completion
65   @post value != 0
66   @post allowed[from][spender] != 0
67   @post __post.allowed[from][spender] == value
68 */
69 function approveByLegacy(address from, address spender, uint value) public returns
70   (bool) {
71   require((value != 0) && (allowed[from][spender] != 0));
72   allowed[from][spender] = value;
73   Approval(from, spender, value);
74 }
75 // Example method, do not copy! In the real world it should
76 // ensure that 'msg.sender' is the legacy contract
77 /*@CTK increaseApprovalByLegacy
78   @post __post.allowed[from][spender] == allowed[from][spender] + addedValue
79 */
80 function increaseApprovalByLegacy(address from, address spender, uint addedValue)
81   public returns (bool) {
82   allowed[from][spender] = allowed[from][spender] + addedValue;
83   Approval(from, spender, allowed[from][spender]);
84 }
85 // Example method, do not copy! In the real world it should
86 // ensure that 'msg.sender' is the legacy contract
87 /*@CTK decreaseApprovalByLegacy
88   @post __post.allowed[from][spender] == allowed[from][spender] - subtractedValue
89 */
90 function decreaseApprovalByLegacy(address from, address spender, uint
91   subtractedValue) public returns (bool) {
92   allowed[from][spender] = allowed[from][spender] - subtractedValue;
93   Approval(from, spender, allowed[from][spender]);
94 }
95

```

```

96 // Example method, do not copy! In the real world it should
97 // ensure that 'msg.sender' is the legacy contract
98 /*@CTK transferFromByLegacy
99   @tag assume_completion
100   @pre to != from
101   @post __post.balances[to] == balances[to] + value
102   @post __post.balances[from] == balances[from] - value
103   @post __post.allowed[from][sender] == allowed[from][sender] - value
104   */
105 function transferFromByLegacy(address sender, address from, address to, uint value
   ) public returns (bool) {
106     var _allowance = allowed[from][sender];
107
108     balances[to] = balances[to].add(value);
109     balances[from] = balances[from].sub(value);
110     allowed[from][sender] = _allowance.sub(value);
111     Transfer(from, to, value);
112 }
113
114 /*@CTK UpgradedTokenTest
115   @post __post.oldAddress == _oldAddress
116   @post __post._totalSupply == _initialSupply
117   */
118 function UpgradedTokenTest(address _oldAddress, uint _initialSupply) public {
119     oldAddress = _oldAddress;
120     _totalSupply = _initialSupply;
121 }
122
123 /*@CTK totalSupply
124   @post __post._totalSupply == _totalSupply
125   */
126 function totalSupply() public constant returns (uint) {
127     return _totalSupply;
128 }
129 }

```

File Migrations.sol

```

1 pragma solidity ^0.4.4;
2 /* solhint-disable var-name-mixedcase */
3
4
5 contract Migrations {
6     address public owner;
7     uint public last_completed_migration;
8
9     modifier restricted() {
10         if (msg.sender == owner) _;
11     }
12
13     /*@CTK Migrations
14       @post __post.owner == msg.sender
15       */
16     function Migrations() public {
17         owner = msg.sender;
18     }
19
20     /*@CTK setCompleted
21       @pre msg.sender == owner

```

```

22     @post __post.last_completed_migration == completed
23     */
24     function setCompleted(uint completed) public restricted {
25         last_completed_migration = completed;
26     }
27
28     function upgrade(address newAddress) public restricted {
29         Migrations upgraded = Migrations(newAddress);
30         upgraded.setCompleted(last_completed_migration);
31     }
32 }

```

File zeppelin-solidity/contracts/token/ERC20/MintableToken.sol

```

1  pragma solidity ^0.4.24;
2
3  import "./StandardToken.sol";
4  import "../ownership/Ownable.sol";
5
6
7  /**
8   * @title Mintable token
9   * @dev Simple ERC20 Token example, with mintable token creation
10  * Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
11  */
12  contract MintableToken is StandardToken, Ownable {
13      event Mint(address indexed to, uint256 amount);
14      event MintFinished();
15
16      bool public mintingFinished = false;
17
18      modifier canMint() {
19          require(!mintingFinished);
20          _;
21      }
22
23      modifier hasMintPermission() {
24          require(msg.sender == owner);
25          _;
26      }
27
28      /**
29       * @dev Function to mint tokens
30       * @param _to The address that will receive the minted tokens.
31       * @param _amount The amount of tokens to mint.
32       * @return A boolean that indicates if the operation was successful.
33       */
34      /*CTK mint
35       @tag assume_completion
36       @post msg.sender == owner
37       @post mintingFinished == false
38       @post __post.totalSupply_ == totalSupply_ + _amount
39       @post __post.balances[_to] == balances[_to] + _amount
40       */
41      function mint(
42          address _to,
43          uint256 _amount
44      )

```



```

45     public
46     hasMintPermission
47     canMint
48     returns (bool)
49     {
50         totalSupply_ = totalSupply_.add(_amount);
51         balances[_to] = balances[_to].add(_amount);
52         emit Mint(_to, _amount);
53         emit Transfer(address(0), _to, _amount);
54         return true;
55     }
56
57     /**
58     * @dev Function to stop minting new tokens.
59     * @return True if the operation was successful.
60     */
61     /*@CTK finishMinting
62     @tag assume_completion
63     @post mintingFinished == false
64     @post __post.mintingFinished == true
65     */
66     function finishMinting() public onlyOwner canMint returns (bool) {
67         mintingFinished = true;
68         emit MintFinished();
69         return true;
70     }
71 }

```

File zeppelin-solidity/contracts/lifecycle/Pausable.sol

```

1  pragma solidity ^0.4.21;
2
3
4  import "../ownership/Ownable.sol";
5
6
7  /**
8   * @title Pausable
9   * @dev Base contract which allows children to implement an emergency stop mechanism.
10  */
11  contract Pausable is Ownable {
12     event Pause();
13     event Unpause();
14
15     bool public paused = false;
16
17
18     /**
19     * @dev Modifier to make a function callable only when the contract is not paused.
20     */
21     modifier whenNotPaused() {
22         require(!paused);
23         _;
24     }
25
26     /**
27     * @dev Modifier to make a function callable only when the contract is paused.
28     */
29     modifier whenPaused() {

```

```
30     require(paused);
31     _;
32 }
33
34 /**
35  * @dev called by the owner to pause, triggers stopped state
36  */
37 //@CTK NO_OVERFLOW
38 //@CTK NO_ASF
39 function pause() onlyOwner whenNotPaused public {
40     paused = true;
41     emit Pause();
42 }
43
44 /**
45  * @dev called by the owner to unpause, returns to normal state
46  */
47 //@CTK NO_OVERFLOW
48 //@CTK NO_ASF
49 function unpause() onlyOwner whenPaused public {
50     paused = false;
51     emit Unpause();
52 }
53 }
```

File zeppelin-solidity/contracts/math/SafeMath.sol

```
1 pragma solidity ^0.4.21;
2
3
4 /**
5  * @title SafeMath
6  * @dev Math operations with safety checks that throw on error
7  */
8 library SafeMath {
9
10     /**
11     * @dev Multiplies two numbers, throws on overflow.
12     */
13     //@CTK NO_OVERFLOW
14     //@CTK FAIL NO_ASF
15     /*@CTK SafeMath_mul
16     @tag spec
17     @post __reverted == __has_assertion_failure
18     @post __has_assertion_failure == __has_overflow
19     @post __reverted == false -> c == a * b
20     @post msg == msg__post
21     @post (a > 0 && (a * b / a != b)) == __has_assertion_failure
22     @post __addr_map == __addr_map__post
23     */
24     function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
25         if (a == 0) {
26             return 0;
27         }
28         c = a * b;
29         assert(c / a == b);
30         return c;
31     }
32 }
```

```

33  /**
34  * @dev Integer division of two numbers, truncating the quotient.
35  */
36  //@CTK NO_OVERFLOW
37  //@CTK FAIL NO_ASF
38  /*@CTK SafeMath_div
39   @tag spec
40   @post __reverted == __has_assertion_failure
41   @post __has_overflow == true -> __has_assertion_failure == true
42   @post __reverted == false -> __return == a / b
43   @post msg == msg__post
44   @post (b == 0) == __has_assertion_failure
45   @post __addr_map == __addr_map__post
46  */
47  function div(uint256 a, uint256 b) internal pure returns (uint256) {
48   // assert(b > 0); // Solidity automatically throws when dividing by 0
49   // uint256 c = a / b;
50   // assert(a == b * c + a % b); // There is no case in which this doesn't hold
51   return a / b;
52  }
53
54  /**
55  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than
56   minuent).
57  */
58  //@CTK NO_OVERFLOW
59  //@CTK FAIL NO_ASF
60  /*@CTK SafeMath_sub
61   @tag spec
62   @post __reverted == __has_assertion_failure
63   @post __has_overflow == true -> __has_assertion_failure == true
64   @post __reverted == false -> __return == a - b
65   @post msg == msg__post
66   @post (a < b) == __has_assertion_failure
67   @post __addr_map == __addr_map__post
68  */
69  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
70   assert(b <= a);
71   return a - b;
72  }
73
74  /**
75  * @dev Adds two numbers, throws on overflow.
76  */
77  //@CTK NO_OVERFLOW
78  //@CTK FAIL NO_ASF
79  /*@CTK SafeMath_add
80   @tag spec
81   @post __reverted == __has_assertion_failure
82   @post __has_assertion_failure == __has_overflow
83   @post __reverted == false -> c == a + b
84   @post msg == msg__post
85   @post ((a + b < a) || (a + b < b)) == __has_assertion_failure
86   @post __addr_map == __addr_map__post
87  */
88  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
89   c = a + b;
90   assert(c >= a);

```

```

90     return c;
91   }
92 }

```

File zeppelin-solidity/contracts/ownership/Ownable.sol

```

1  pragma solidity ^0.4.21;
2
3
4  /**
5   * @title Ownable
6   * @dev The Ownable contract has an owner address, and provides basic authorization
7     control
8   * functions, this simplifies the implementation of "user permissions".
9   */
9 contract Ownable {
10   address public owner;
11
12
13   event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
14
15
16   /**
17    * @dev The Ownable constructor sets the original 'owner' of the contract to the
18      sender
19      account.
20    */
21   // @CTK NO_OVERFLOW
22   // @CTK NO_ASF
23   function Ownable() public {
24     owner = msg.sender;
25   }
26
27   /**
28    * @dev Throws if called by any account other than the owner.
29    */
30   modifier onlyOwner() {
31     require(msg.sender == owner);
32     _;
33   }
34
35   /**
36    * @dev Allows the current owner to transfer control of the contract to a newOwner.
37    * @param newOwner The address to transfer ownership to.
38    */
39   // @CTK NO_OVERFLOW
40   // @CTK NO_ASF
41   function transferOwnership(address newOwner) public onlyOwner {
42     require(newOwner != address(0));
43     emit OwnershipTransferred(owner, newOwner);
44     owner = newOwner;
45   }
46 }

```

File zeppelin-solidity/contracts/ownership/HasNoContracts.sol

```

1  pragma solidity ^0.4.21;
2
3  import "./Ownable.sol";

```

```

4
5
6 /**
7  * @title Contracts that should not own Contracts
8  * @dev Should contracts (anything Ownable) end up being owned by this contract, it
9  *   allows the owner
10  * of this contract to reclaim ownership of the contracts.
11  */
12 contract HasNoContracts is Ownable {
13
14  /**
15  * @dev Reclaim ownership of Ownable contracts
16  * @param contractAddr The address of the Ownable to be reclaimed.
17  */
18  //@CTK_NO_OVERFLOW
19  //@CTK_NO_ASF
20  function reclaimContract(address contractAddr) external onlyOwner {
21  Ownable contractInst = Ownable(contractAddr);
22  contractInst.transferOwnership(owner);
23  }

```

File zeppelin-solidity/contracts/ownership/HasNoEther.sol

```

1 pragma solidity ^0.4.21;
2
3 import "./Ownable.sol";
4
5
6 /**
7  * @title Contracts that should not own Ether
8  * @dev This tries to block incoming ether to prevent accidental loss of Ether. Should
9  *   Ether end up
10  * in the contract, it will allow the owner to reclaim this ether.
11  * @notice Ether can still be sent to this contract by:
12  * calling functions labeled 'payable'
13  * 'selfdestruct(contract_address)'
14  * mining directly to the contract address
15  */
16 contract HasNoEther is Ownable {
17
18  /**
19  * @dev Constructor that rejects incoming Ether
20  * @dev The 'payable' flag is added so we can access 'msg.value' without compiler
21  *   warning. If we
22  * leave out payable, then Solidity will allow inheriting contracts to implement a
23  * payable
24  * constructor. By doing it this way we prevent a payable constructor from working.
25  * Alternatively
26  * we could use assembly to access msg.value.
27  */
28  //@CTK_NO_OVERFLOW
29  //@CTK_NO_ASF
30  function HasNoEther() public payable {
31  require(msg.value == 0);
32  }
33
34  /**
35  * @dev Disallows direct send by settings a default function without the 'payable'

```

```

    flag.
32  */
33  //@CTK NO_OVERFLOW
34  //@CTK NO_ASF
35  function() external {
36  }
37
38  /**
39   * @dev Transfer all Ether held by the contract to the owner.
40   */
41  //@CTK NO_OVERFLOW
42  //@CTK NO_ASF
43  function reclaimEther() external onlyOwner {
44      // solium-disable-next-line security/no-send
45      // assert(owner.send(address(this).balance));
46  }
47 }

```

File zeppelin-solidity/contracts/ownership/Claimable.sol

```

1  pragma solidity ^0.4.21;
2
3
4  import "./Ownable.sol";
5
6
7  /**
8   * @title Claimable
9   * @dev Extension for the Ownable contract, where the ownership needs to be claimed.
10  * This allows the new owner to accept the transfer.
11  */
12  contract Claimable is Ownable {
13      address public pendingOwner;
14
15      /**
16       * @dev Modifier throws if called by any account other than the pendingOwner.
17       */
18      modifier onlyPendingOwner() {
19          require(msg.sender == pendingOwner);
20          _;
21      }
22
23      /**
24       * @dev Allows the current owner to set the pendingOwner address.
25       * @param newOwner The address to transfer ownership to.
26       */
27      //@CTK NO_OVERFLOW
28      //@CTK NO_ASF
29      function transferOwnership(address newOwner) onlyOwner public {
30          pendingOwner = newOwner;
31      }
32
33      /**
34       * @dev Allows the pendingOwner address to finalize the transfer.
35       */
36      //@CTK NO_OVERFLOW
37      //@CTK NO_ASF
38      function claimOwnership() onlyPendingOwner public {
39          emit OwnershipTransferred(owner, pendingOwner);

```

```
40     owner = pendingOwner;  
41     pendingOwner = address(0);  
42 }  
43 }
```

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

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

Raw code	<pre> 35 function transferFrom(address from, address to 36) { 37 balances[from] = balances[from].sub(tokens 38 allowed[from] [msg.sender] = allowed[from] [39 balances[to] = balances[to].add(tokens); 40 emit Transfer(from, to, tokens); 41 return true; 42 } </pre>
----------	---

Counterexample	 This code violates the specification
----------------	--

Initial environment	<pre> 1 Counter Example: 2 Before Execution: 3 Input = { 4 from = 0x0 5 to = 0x0 6 tokens = 0x6c 7 } 8 This = 0 </pre>
Post environment	<pre> 52 } 53 balance: 0x0 54 } 55 } 56 57 After Execution: 58 Input = { 59 from = 0x0 60 to = 0x0 61 tokens = 0x6c </pre>




Static Analysis Request

Formal Verification Request 1

getBlackListStatus

 11, Apr 2019

 17.88 ms

Line 8-10 in File BlackList.sol

```
8  /*@CTK getBlackListStatus
9     @post __return == isBlackListed[_maker]
10 */
```

Line 11-13 in File BlackList.sol


```
11 function getBlackListStatus(address _maker) external constant returns (bool) {
12     return isBlackListed[_maker];
13 }
```

 The code meets the specification

Formal Verification Request 2

addBlackList

 11, Apr 2019

 30.84 ms

Line 17-21 in File BlackList.sol

```
17 /*@CTK addBlackList
18     @tag assume_completion
19     @post owner == msg.sender
20     @post __post.isBlackListed[_evilUser]
21 */
```

Line 22-25 in File BlackList.sol


```
22 function addBlackList (address _evilUser) public onlyOwner {
23     isBlackListed[_evilUser] = true;
24     AddedBlackList(_evilUser);
25 }
```

 The code meets the specification

Formal Verification Request 3

removeBlackList

 11, Apr 2019

 23.41 ms

Line 27-31 in File BlackList.sol

```

27  /*@CTK removeBlackList
28     @tag assume_completion
29     @post owner == msg.sender
30     @post !__post.isBlackListed[_clearedUser]
31  */

```

Line 32-35 in File BlackList.sol

```

32  function removeBlackList (address _clearedUser) public onlyOwner {
33      isBlackListed[_clearedUser] = false;
34      RemovedBlackList(_clearedUser);
35  }


```

✓ The code meets the specification

Formal Verification Request 4

calcFee_maximumFee

 11, Apr 2019

 28.84 ms

Line 22-25 in File StandardTokenWithFees.sol

```

22  /*@CTK calcFee_maximumFee
23     @tag assume_completion
24     @post __return <= maximumFee
25  */

```

Line 31-37 in File StandardTokenWithFees.sol

```

31  function calcFee(uint _value) constant returns (uint) {
32      uint fee = (_value.mul(basisPointsRate)).div(10000);
33      if (fee > maximumFee) {
34          fee = maximumFee;
35      }
36      return fee;
37  }


```

✓ The code meets the specification

Formal Verification Request 5

calcFee

 11, Apr 2019

 2.57 ms

Line 26-30 in File StandardTokenWithFees.sol

```

26  /*@CTK calcFee
27     @tag assume_completion
28     @pre _value * basisPointsRate / 10000 <= maximumFee
29     @post __return == _value * basisPointsRate / 10000
30  */

```

Line 31-37 in File StandardTokenWithFees.sol

```

31  function calcFee(uint _value) constant returns (uint) {
32      uint fee = (_value.mul(basisPointsRate)).div(10000);
33      if (fee > maximumFee) {
34          fee = maximumFee;
35      }
36      return fee;
37  }

```

✓ The code meets the specification

Formal Verification Request 6

transfer

📅 11, Apr 2019

🕒 2010.25 ms

Line 39-48 in File StandardTokenWithFees.sol

```

39  /*@CTK transfer
40      @tag assume_completion
41      @pre _value * basisPointsRate / 10000 <= maximumFee
42      @pre msg.sender != _to && msg.sender != owner && _to != owner
43      @post __post.balances[msg.sender] == balances[msg.sender] - _value
44      @post __post.balances[_to] == balances[_to] +
45              (_value - _value * basisPointsRate / 10000)
46      @post __post.balances[owner] == balances[owner] +
47              _value * basisPointsRate / 10000
48  */

```

Line 50-58 in File StandardTokenWithFees.sol

```

50  function transfer(address _to, uint _value) public returns (bool) {
51      uint fee = calcFee(_value);
52      uint sendAmount = _value.sub(fee);
53
54      super.transfer(_to, sendAmount);
55      if (fee > 0) {
56          super.transfer(owner, fee);
57      }
58  }

```

✓ The code meets the specification

Formal Verification Request 7

transferFrom

📅 11, Apr 2019

🕒 4740.71 ms

Line 60-75 in File StandardTokenWithFees.sol

```

60  /*@CTK transferFrom
61     @tag assume_completion
62     @pre allowed[_from][msg.sender] < MAX_UINT
63     @pre _from != _to && _from != owner && _to != owner
64     @pre _value * basisPointsRate / 10000 <= maximumFee
65     @post _to != address(0)
66     @post _value <= balances[_from]
67     @post _value <= allowed[_from][msg.sender]
68     @post __post.balances[_from] == balances[_from] - _value
69     @post __post.balances[_to] == balances[_to] +
70         (_value - _value * basisPointsRate / 10000)
71     @post __post.balances[owner] == balances[owner] +
72         _value * basisPointsRate / 10000
73     @post __post.allowed[_from][msg.sender] ==
74         allowed[_from][msg.sender] - _value
75  */

```

Line 76-95 in File StandardTokenWithFees.sol

```

76  function transferFrom(address _from, address _to, uint256 _value) public returns (
77     bool) {
78     require(_to != address(0));
79     require(_value <= balances[_from]);
80     require(_value <= allowed[_from][msg.sender]);
81
82     uint fee = calcFee(_value);
83     uint sendAmount = _value.sub(fee);
84
85     balances[_from] = balances[_from].sub(_value);
86     balances[_to] = balances[_to].add(sendAmount);
87     if (allowed[_from][msg.sender] < MAX_UINT) {
88         allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
89     }
90     Transfer(_from, _to, sendAmount);
91     if (fee > 0) {
92         balances[owner] = balances[owner].add(fee);
93         Transfer(_from, owner, fee);
94     }
95     return true;
96 }

```

✔ The code meets the specification

Formal Verification Request 8

TetherToken

📅 11, Apr 2019

🕒 44.16 ms

Line 21-28 in File TetherToken.sol

```

21  /*@CTK TetherToken
22     @post __post._totalSupply == _initialSupply
23     @post __post.name == _name
24     @post __post.symbol == _symbol
25     @post __post.decimals == _decimals

```

```

26     @post __post.balances[owner] == _initialSupply
27     @post __post.deprecated == false
28     */

```

Line 29-36 in File TetherToken.sol

```

29     function TetherToken(uint _initialSupply, string _name, string _symbol, uint8
        _decimals) public {
30         _totalSupply = _initialSupply;
31         name = _name;
32         symbol = _symbol;
33         decimals = _decimals;
34         balances[owner] = _initialSupply;
35         deprecated = false;
36     }

```

✔ The code meets the specification

Formal Verification Request 9

balanceOf

📅 11, Apr 2019

🕒 40.83 ms

Line 78-81 in File TetherToken.sol

```

78     /*@CTK balanceOf
79     @pre deprecated
80     @post __return == balances[who]
81     */

```

Line 82-86 in File TetherToken.sol

```

82     function oldBalanceOf(address who) public constant returns (uint) {
83         if (deprecated) {
84             return super.balanceOf(who);
85         }
86     }

```

✔ The code meets the specification

Formal Verification Request 10

approve

📅 11, Apr 2019

🕒 99.41 ms

Line 89-94 in File TetherToken.sol

```

89     /*@CTK approve
90     @tag assume_completion
91     @pre !deprecated
92     @post !paused
93     @post __post.allowed[msg.sender][_spender] == _value
94     */

```

Line 95-101 in File TetherToken.sol

```
95     function approve(address _spender, uint _value) public whenNotPaused returns (bool
96         ) {
97         if (deprecated) {
98             return UpgradedStandardToken(upgradedAddress).approveByLegacy(msg.sender,
99                 _spender, _value);
100         } else {
101             return super.approve(_spender, _value);
102         }
103     }
```

✔ The code meets the specification

Formal Verification Request 11

increaseApproval

📅 11, Apr 2019

🕒 174.69 ms

Line 103-108 in File TetherToken.sol

```
103     /*@CTK increaseApproval
104         @tag assume_completion
105         @pre !deprecated
106         @post __post.allowed[msg.sender][_spender] ==
107             allowed[msg.sender][_spender] + _addedValue
108     */
```

Line 109-115 in File TetherToken.sol

```
109     function increaseApproval(address _spender, uint _addedValue) public whenNotPaused
110         returns (bool) {
111         if (deprecated) {
112             return UpgradedStandardToken(upgradedAddress).increaseApprovalByLegacy(msg.
113                 sender, _spender, _addedValue);
114         } else {
115             return super.increaseApproval(_spender, _addedValue);
116         }
117     }
```

✔ The code meets the specification

Formal Verification Request 12

decreaseApproval

📅 11, Apr 2019

🕒 204.31 ms

Line 117-125 in File TetherToken.sol

```

117  /*@CTK decreaseApproval
118     @tag assume_completion
119     @pre !deprecated
120     @post allowed[msg.sender][_spender] >= _subtractedValue ->
121         __post.allowed[msg.sender][_spender] ==
122         allowed[msg.sender][_spender] - _subtractedValue
123     @post allowed[msg.sender][_spender] < _subtractedValue ->
124         __post.allowed[msg.sender][_spender] == 0
125  */

```

Line 126-132 in File TetherToken.sol

```

126  function decreaseApproval(address _spender, uint _subtractedValue) public
    whenNotPaused returns (bool) {
127      if (deprecated) {
128          return UpgradedStandardToken(upgradedAddress).decreaseApprovalByLegacy(msg.
                sender, _spender, _subtractedValue);
129      } else {
130          return super.decreaseApproval(_spender, _subtractedValue);
131      }
132  }

```

✔ The code meets the specification

Formal Verification Request 13

allownace

📅 11, Apr 2019

🕒 65.01 ms

Line 135-138 in File TetherToken.sol

```

135  /*@CTK allownace
136     @pre !deprecated
137     @post remaining == allowed[_owner][_spender]
138  */

```

Line 139-145 in File TetherToken.sol

```

139  function allowance(address _owner, address _spender) public constant returns (uint
    remaining) {
140      if (deprecated) {
141          return StandardToken(upgradedAddress).allowance(_owner, _spender);
142      } else {
143          return super.allowance(_owner, _spender);
144      }
145  }

```

✔ The code meets the specification

Formal Verification Request 14

deprecate

📅 11, Apr 2019

🕒 36.56 ms

Line 148-154 in File TetherToken.sol

```

148  /*@CTK deprecate
149     @tag assume_completion
150     @post owner == msg.sender
151     @post _upgradedAddress != address(0)
152     @post __post.upgradedAddress == _upgradedAddress
153     @post __post.deprecated
154  */

```

Line 155-160 in File TetherToken.sol

```

155  function deprecate(address _upgradedAddress) public onlyOwner {
156      require(_upgradedAddress != address(0));
157      deprecated = true;
158      upgradedAddress = _upgradedAddress;
159      Deprecate(_upgradedAddress);
160  }

```

✔ The code meets the specification

Formal Verification Request 15

totalSupply

📅 11, Apr 2019

🕒 41.04 ms

Line 163-166 in File TetherToken.sol

```

163  /*@CTK totalSupply
164     @pre !deprecated
165     @post __return == _totalSupply
166  */

```

Line 167-173 in File TetherToken.sol

```

167  function totalSupply() public constant returns (uint) {
168      if (deprecated) {
169          return StandardToken(upgradedAddress).totalSupply();
170      } else {
171          return _totalSupply;
172      }
173  }

```

✔ The code meets the specification

Formal Verification Request 16

issue

📅 11, Apr 2019

🕒 139.9 ms

Line 179-184 in File TetherToken.sol

```

179  /*@CTK issue
180     @tag assume_completion
181     @post owner == msg.sender
182     @post __post.balances[owner] == balances[owner] + amount
183     @post __post._totalSupply == _totalSupply + amount
184  */

```

Line 185-190 in File TetherToken.sol

```

185  function issue(uint amount) public onlyOwner {
186      balances[owner] = balances[owner].add(amount);
187      _totalSupply = _totalSupply.add(amount);
188      Issue(amount);
189      Transfer(address(0), owner, amount);
190  }


```

✔ The code meets the specification

Formal Verification Request 17

redeem

 11, Apr 2019

 121.6 ms

Line 197-202 in File TetherToken.sol

```

197  /*@CTK redeem
198     @tag assume_completion
199     @post owner == msg.sender
200     @post __post._totalSupply == _totalSupply - amount
201     @post __post.balances[owner] == balances[owner] - amount
202  */

```

Line 203-208 in File TetherToken.sol

```

203  function redeem(uint amount) public onlyOwner {
204      _totalSupply = _totalSupply.sub(amount);
205      balances[owner] = balances[owner].sub(amount);
206      Redeem(amount);
207      Transfer(owner, address(0), amount);
208  }


```

✔ The code meets the specification

Formal Verification Request 18

addOwner

 11, Apr 2019

 73.36 ms

Line 119-128 in File MultiSigWallet.sol

```

119  /*@CTK addOwner
120     @tag assume_completion
121     @post msg.sender == address(this)
122     @post owner != address(0)
123     @post !isOwner[owner]
124     @post __post.isOwner[owner]
125     @post __post.owners[owners.length] == owner
126     @post required != 0
127     @post required <= __post.owners.length
128  */

```

Line 129-139 in File MultiSigWallet.sol

```

129  function addOwner(address owner)
130      public
131      onlyWallet
132      ownerDoesNotExist(owner)
133      notNull(owner)
134      validRequirement(owners.length + 1, required)
135  {
136      isOwner[owner] = true;
137      owners.push(owner);
138      OwnerAddition(owner);
139  }

```

✔ The code meets the specification

Formal Verification Request 19

changeRequirement

📅 11, Apr 2019

🕒 26.22 ms

Line 182-185 in File MultiSigWallet.sol

```

182  /*@CTK changeRequirement
183     @tag assume_completion
184     @post __post.required == _required
185  */

```

Line 186-193 in File MultiSigWallet.sol

```

186  function changeRequirement(uint _required)
187      public
188      onlyWallet
189      validRequirement(owners.length, _required)
190  {
191      required = _required;
192      RequirementChange(_required);
193  }


```

✔ The code meets the specification

Formal Verification Request 20

addTransaction

 11, Apr 2019

 93.13 ms

Line 282-291 in File MultiSigWallet.sol

```
282  /*@CTK addTransaction
283     @tag assume_completion
284     @post destination != address(0)
285     @post __post.transactions[transactionCount].destination == destination
286     @post __post.transactions[transactionCount].value == value
287     @post __post.transactions[transactionCount].data == data
288     @post __post.transactions[transactionCount].executed == false
289     @post __post.transactionCount == transactionCount + 1
290     @post transactionId == transactionCount
291  */
```

Line 292-310 in File MultiSigWallet.sol


```
292  function addTransaction(address destination, uint value, bytes data)
293     internal
294     notNull(destination)
295     returns (uint transactionId)
296  {
297     transactionId = transactionCount;
298     transactions[transactionId].destination = destination;
299     transactions[transactionId].value = value;
300     transactions[transactionId].data = data;
301     transactions[transactionId].executed = false;
302     // transactions[transactionId] = Transaction({
303     //     destination: destination,
304     //     value: value,
305     //     data: data,
306     //     executed: false
307     // });
308     transactionCount += 1;
309     Submission(transactionId);
310 }
```

 The code meets the specification

Formal Verification Request 21

getOwners

 11, Apr 2019

 6.36 ms

Line 345-347 in File MultiSigWallet.sol

```
345  /*@CTK getOwners
346     @post __return == owners
347  */
```

Line 348-354 in File MultiSigWallet.sol

```
348     function getOwners()
349         public
350         constant
351         returns (address[])
352     {
353         return owners;
354     }
```

✔ The code meets the specification

Formal Verification Request 22

userMigrate

📅 11, Apr 2019

🕒 53.27 ms

Line 22-24 in File UpgradedTokenTest.sol

```
22     /*@CTK userMigrate
23         @post __post.userMigrated[user]
24     */
```

Line 25-30 in File UpgradedTokenTest.sol

```
25     function userMigrate(address user) public {
26         if(!userMigrated[user]) {
27             balances[user] = PreviousTokenInterface(oldAddress).oldBalanceOf(user);
28             userMigrated[user] = true;
29         }
30     }
```

✔ The code meets the specification

Formal Verification Request 23

balanceOf

📅 11, Apr 2019

🕒 101.4 ms

Line 32-35 in File UpgradedTokenTest.sol

```
32     /*@CTK balanceOf
33         @post __post.userMigrated[who]
34         @post __return == __post.balances[who]
35     */
```

Line 36-39 in File UpgradedTokenTest.sol

```
36     function balanceOf(address who) public constant returns (uint) {
37         userMigrate(who);
38         return super.balanceOf(who);
39     }
```

✓ The code meets the specification

Formal Verification Request 24

transferByLegacy

📅 11, Apr 2019

🕒 284.3 ms

Line 43-52 in File UpgradedTokenTest.sol

```

43  /*@CTK transferByLegacy
44     @tag assume_completion
45     @pre from != to
46     @pre userMigrated[from]
47     @pre userMigrated[to]
48     @post __post.userMigrated[from]
49     @post __post.userMigrated[to]
50     @post __post.balances[from] == balances[from] - value
51     @post __post.balances[to] == balances[to] + value
52  */

```

Line 53-59 in File UpgradedTokenTest.sol

```

53  function transferByLegacy(address from, address to, uint value) public returns (
54     bool) {
55     userMigrate(from);
56     userMigrate(to);
57     balances[from] = balances[from].sub(value);
58     balances[to] = balances[to].add(value);
59     Transfer(from, to, value);
60  }

```

✓ The code meets the specification

Formal Verification Request 25

approveByLegacy

📅 11, Apr 2019

🕒 22.58 ms

Line 63-68 in File UpgradedTokenTest.sol

```

63  /*@CTK approveByLegacy
64     @tag assume_completion
65     @post value != 0
66     @post allowed[from][spender] != 0
67     @post __post.allowed[from][spender] == value
68  */

```

Line 69-74 in File UpgradedTokenTest.sol

```

69     function approveByLegacy(address from, address spender, uint value) public returns
       (bool) {
70         require((value != 0) && (allowed[from][spender] != 0));
71
72         allowed[from][spender] = value;
73         Approval(from, spender, value);
74     }

```

✔ The code meets the specification

Formal Verification Request 26

increaseApprovalByLegacy

📅 11, Apr 2019

🕒 13.64 ms

Line 78-80 in File UpgradedTokenTest.sol

```

78     /*@CTK increaseApprovalByLegacy
79         @post __post.allowed[from][spender] == allowed[from][spender] + addedValue
80     */

```

Line 81-84 in File UpgradedTokenTest.sol

```

81     function increaseApprovalByLegacy(address from, address spender, uint addedValue)
       public returns (bool) {
82         allowed[from][spender] = allowed[from][spender] + addedValue;
83         Approval(from, spender, allowed[from][spender]);
84     }

```

✔ The code meets the specification

Formal Verification Request 27

decreaseApprovalByLegacy

📅 11, Apr 2019

🕒 15.09 ms

Line 88-90 in File UpgradedTokenTest.sol

```

88     /*@CTK decreaseApprovalByLegacy
89         @post __post.allowed[from][spender] == allowed[from][spender] - subtractedValue
90     */

```

Line 91-94 in File UpgradedTokenTest.sol

```

91     function decreaseApprovalByLegacy(address from, address spender, uint
       subtractedValue) public returns (bool) {
92         allowed[from][spender] = allowed[from][spender] - subtractedValue;
93         Approval(from, spender, allowed[from][spender]);
94     }


```

✔ The code meets the specification

Formal Verification Request 28

transferFromByLegacy

 11, Apr 2019

 116.01 ms

Line 98-104 in File UpgradedTokenTest.sol

```
98  /*@CTK transferFromByLegacy
99     @tag assume_completion
100     @pre to != from
101     @post __post.balances[to] == balances[to] + value
102     @post __post.balances[from] == balances[from] - value
103     @post __post.allowed[from][sender] == allowed[from][sender] - value
104  */
```

Line 105-112 in File UpgradedTokenTest.sol


```
105  function transferFromByLegacy(address sender, address from, address to, uint value
    ) public returns (bool) {
106      var _allowance = allowed[from][sender];
107
108      balances[to] = balances[to].add(value);
109      balances[from] = balances[from].sub(value);
110      allowed[from][sender] = _allowance.sub(value);
111      Transfer(from, to, value);
112  }
```

 The code meets the specification

Formal Verification Request 29

UpgradedTokenTest

 11, Apr 2019

 12.12 ms

Line 114-117 in File UpgradedTokenTest.sol

```
114  /*@CTK UpgradedTokenTest
115     @post __post.oldAddress == _oldAddress
116     @post __post._totalSupply == _initialSupply
117  */
```

Line 118-121 in File UpgradedTokenTest.sol


```
118  function UpgradedTokenTest(address _oldAddress, uint _initialSupply) public {
119      oldAddress = _oldAddress;
120      _totalSupply = _initialSupply;
121  }
```

 The code meets the specification

Formal Verification Request 30

totalSupply

 11, Apr 2019

 6.6 ms

Line 123-125 in File UpgradedTokenTest.sol

```
123  /*@CTK totalSupply
124     @post __post._totalSupply == _totalSupply
125  */
```

Line 126-128 in File UpgradedTokenTest.sol


```
126  function totalSupply() public constant returns (uint) {
127      return _totalSupply;
128  }
```

 The code meets the specification

Formal Verification Request 31

Migrations

 11, Apr 2019

 5.6 ms

Line 13-15 in File Migrations.sol

```
13  /*@CTK Migrations
14     @post __post.owner == msg.sender
15  */
```

Line 16-18 in File Migrations.sol


```
16  function Migrations() public {
17      owner = msg.sender;
18  }
```

 The code meets the specification

Formal Verification Request 32

setCompleted

 11, Apr 2019

 8.96 ms

Line 20-23 in File Migrations.sol

```
20  /*@CTK setCompleted
21     @pre msg.sender == owner
22     @post __post.last_completed_migration == completed
23  */
```

Line 24-26 in File Migrations.sol

```
24     function setCompleted(uint completed) public restricted {
25         last_completed_migration = completed;
26     }
```

✓ The code meets the specification

Formal Verification Request 33

mint

📅 11, Apr 2019

🕒 187.82 ms

Line 34-40 in File MintableToken.sol

```
34     /*@CTK mint
35         @tag assume_completion
36         @post msg.sender == owner
37         @post mintingFinished == false
38         @post __post.totalSupply_ == totalSupply_ + _amount
39         @post __post.balances[_to] == balances[_to] + _amount
40     */
```

Line 41-55 in File MintableToken.sol

```
41     function mint(
42         address _to,
43         uint256 _amount
44     )
45     public
46     hasMintPermission
47     canMint
48     returns (bool)
49     {
50         totalSupply_ = totalSupply_.add(_amount);
51         balances[_to] = balances[_to].add(_amount);
52         emit Mint(_to, _amount);
53         emit Transfer(address(0), _to, _amount);
54         return true;
55     }
```

✓ The code meets the specification

Formal Verification Request 34

finishMinting

📅 11, Apr 2019

🕒 30.36 ms

Line 61-65 in File MintableToken.sol

```

61  /*@CTK finishMinting
62     @tag assume_completion
63     @post mintingFinished == false
64     @post __post.mintingFinished == true
65  */

```

Line 66-70 in File MintableToken.sol

```

66  function finishMinting() public onlyOwner canMint returns (bool) {
67      mintingFinished = true;
68      emit MintFinished();
69      return true;
70  }

```

✔ The code meets the specification

Formal Verification Request 35

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 30.23 ms

Line 37 in File Pausable.sol

```

37  //@CTK_NO_OVERFLOW

```

Line 39-42 in File Pausable.sol

```

39  function pause() onlyOwner whenNotPaused public {
40      paused = true;
41      emit Pause();
42  }

```

✔ The code meets the specification

Formal Verification Request 36

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.63 ms

Line 38 in File Pausable.sol

```

38  //@CTK_NO_ASF

```

Line 39-42 in File Pausable.sol

```

39  function pause() onlyOwner whenNotPaused public {
40      paused = true;
41      emit Pause();
42  }


```

✔ The code meets the specification

Formal Verification Request 37

If method completes, integer overflow would not happen.

 11, Apr 2019

 28.32 ms

Line 47 in File Pausable.sol

```
47 // @CTK_NO_OVERFLOW
```

Line 49-52 in File Pausable.sol


```
49 function unpause() onlyOwner whenPaused public {  
50     paused = false;  
51     emit Unpause();  
52 }
```

 The code meets the specification

Formal Verification Request 38

Method will not encounter an assertion failure.

 11, Apr 2019

 0.95 ms

Line 48 in File Pausable.sol

```
48 // @CTK_NO_ASF
```

Line 49-52 in File Pausable.sol


```
49 function unpause() onlyOwner whenPaused public {  
50     paused = false;  
51     emit Unpause();  
52 }
```

 The code meets the specification

Formal Verification Request 39

If method completes, integer overflow would not happen.

 11, Apr 2019

 49.27 ms

Line 13 in File SafeMath.sol

```
13 // @CTK_NO_OVERFLOW
```

Line 24-31 in File SafeMath.sol

```

24 function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
25     if (a == 0) {
26         return 0;
27     }
28     c = a * b;
29     assert(c / a == b);
30     return c;
31 }

```

✔ The code meets the specification

Formal Verification Request 40

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 8.96 ms

Line 14 in File SafeMath.sol

```
14 //@CTK FAIL NO_ASF
```

Line 24-31 in File SafeMath.sol

```

24 function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
25     if (a == 0) {
26         return 0;
27     }
28     c = a * b;
29     assert(c / a == b);
30     return c;
31 }

```

✘ This code violates the specification

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 2
5     b = 128
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19   Other = {
20     block = {
21       "number": 0,
22       "timestamp": 0
23     }

```

```


24     c = 0
25   }
26   Address_Map = [
27     {
28       "key": "ALL_OTHERS",
29       "value": "EmptyAddress"
30     }
31   ]
32
33 Function invocation is reverted.

```

Formal Verification Request 41

SafeMath_mul

 11, Apr 2019

 389.97 ms

Line 15-23 in File SafeMath.sol

```

15  /*@CTK SafeMath_mul
16     @tag spec
17     @post __reverted == __has_assertion_failure
18     @post __has_assertion_failure == __has_overflow
19     @post __reverted == false -> c == a * b
20     @post msg == msg__post
21     @post (a > 0 && (a * b / a != b)) == __has_assertion_failure
22     @post __addr_map == __addr_map__post
23  */

```

Line 24-31 in File SafeMath.sol

```

24  function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
25     if (a == 0) {
26         return 0;
27     }
28     c = a * b;
29     assert(c / a == b);
30     return c;
31 }


```

 The code meets the specification

Formal Verification Request 42

If method completes, integer overflow would not happen.

 11, Apr 2019

 8.63 ms

Line 36 in File SafeMath.sol

```

36  //@CTK NO_OVERFLOW

```

Line 47-52 in File SafeMath.sol

```

47 function div(uint256 a, uint256 b) internal pure returns (uint256) {
48     // assert(b > 0); // Solidity automatically throws when dividing by 0
49     // uint256 c = a / b;
50     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
51     return a / b;
52 }

```

✔ The code meets the specification

Formal Verification Request 43

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.43 ms

Line 37 in File SafeMath.sol

```

37 //CTK FAIL NO_ASF

```

Line 47-52 in File SafeMath.sol

```

47 function div(uint256 a, uint256 b) internal pure returns (uint256) {
48     // assert(b > 0); // Solidity automatically throws when dividing by 0
49     // uint256 c = a / b;
50     // assert(a == b * c + a % b); // There is no case in which this doesn't hold
51     return a / b;
52 }

```

✘ This code violates the specification

```

1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 0
5     b = 0
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27  {

```

```


28     "key": "ALL_OTHERS",
29     "value": "EmptyAddress"
30   }
31 ]
32
33 Function invocation is reverted.

```

Formal Verification Request 44

SafeMath_div

 11, Apr 2019

 0.39 ms

Line 38-46 in File SafeMath.sol

```

38 /*@CTK SafeMath_div
39   @tag spec
40   @post __reverted == __has_assertion_failure
41   @post __has_overflow == true -> __has_assertion_failure == true
42   @post __reverted == false -> __return == a / b
43   @post msg == msg__post
44   @post (b == 0) == __has_assertion_failure
45   @post __addr_map == __addr_map__post
46 */

```

Line 47-52 in File SafeMath.sol

```

47 function div(uint256 a, uint256 b) internal pure returns (uint256) {
48   // assert(b > 0); // Solidity automatically throws when dividing by 0
49   // uint256 c = a / b;
50   // assert(a == b * c + a % b); // There is no case in which this doesn't hold
51   return a / b;
52 }


```

 The code meets the specification

Formal Verification Request 45

If method completes, integer overflow would not happen.

 11, Apr 2019

 15.14 ms

Line 57 in File SafeMath.sol

```

57 // @CTK NO_OVERFLOW

```

Line 68-71 in File SafeMath.sol

```

68 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
69   assert(b <= a);
70   return a - b;
71 }


```

 The code meets the specification

Formal Verification Request 46

Method will not encounter an assertion failure.

 11, Apr 2019

 1.44 ms

Line 58 in File SafeMath.sol

```
58 // @CTK FAIL NO_ASF
```

Line 68-71 in File SafeMath.sol

```
68 function sub(uint256 a, uint256 b) internal pure returns (uint256) {
69     assert(b <= a);
70     return a - b;
71 }
```


 This code violates the specification

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 0
5     b = 1
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    __return = 0
21    block = {
22      "number": 0,
23      "timestamp": 0
24    }
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.
```

Formal Verification Request 47

SafeMath_sub

 11, Apr 2019

 1.19 ms

Line 59-67 in File SafeMath.sol

```
59  /*@CTK SafeMath_sub
60     @tag spec
61     @post __reverted == __has_assertion_failure
62     @post __has_overflow == true -> __has_assertion_failure == true
63     @post __reverted == false -> __return == a - b
64     @post msg == msg__post
65     @post (a < b) == __has_assertion_failure
66     @post __addr_map == __addr_map__post
67  */
```

Line 68-71 in File SafeMath.sol


```
68  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
69     assert(b <= a);
70     return a - b;
71 }
```

 The code meets the specification

Formal Verification Request 48

If method completes, integer overflow would not happen.

 11, Apr 2019

 20.99 ms

Line 76 in File SafeMath.sol

```
76  //@CTK_NO_OVERFLOW
```

Line 87-91 in File SafeMath.sol


```
87  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
88     c = a + b;
89     assert(c >= a);
90     return c;
91 }
```

 The code meets the specification

Formal Verification Request 49

Method will not encounter an assertion failure.

 11, Apr 2019

 2.81 ms

Line 77 in File SafeMath.sol

```
77 // @CTK FAIL NO_ASF
```

Line 87-91 in File SafeMath.sol

```
87 function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
88     c = a + b;
89     assert(c >= a);
90     return c;
91 }
```


✘ This code violates the specification

```
1 Counter Example:
2 Before Execution:
3   Input = {
4     a = 131
5     b = 125
6   }
7   Internal = {
8     __has_assertion_failure = false
9     __has_buf_overflow = false
10    __has_overflow = false
11    __has_returned = false
12    __reverted = false
13    msg = {
14      "gas": 0,
15      "sender": 0,
16      "value": 0
17    }
18  }
19  Other = {
20    block = {
21      "number": 0,
22      "timestamp": 0
23    }
24    c = 0
25  }
26  Address_Map = [
27    {
28      "key": "ALL_OTHERS",
29      "value": "EmptyAddress"
30    }
31  ]
32
33 Function invocation is reverted.
```

Formal Verification Request 50

SafeMath_add

 11, Apr 2019

 7.07 ms

Line 78-86 in File SafeMath.sol

```
78  /*@CTK SafeMath_add
79  @tag spec
80  @post __reverted == __has_assertion_failure
81  @post __has_assertion_failure == __has_overflow
82  @post __reverted == false -> c == a + b
83  @post msg == msg__post
84  @post ((a + b < a) || (a + b < b)) == __has_assertion_failure
85  @post __addr_map == __addr_map__post
86  */
```

Line 87-91 in File SafeMath.sol

```
87  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
88    c = a + b;
89    assert(c >= a);
90    return c;
91  }
```

✔ The code meets the specification

Formal Verification Request 51

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 8.2 ms

Line 20 in File Ownable.sol

```
20  //@CTK_NO_OVERFLOW
```

Line 22-24 in File Ownable.sol

```
22  function Ownable() public {
23    owner = msg.sender;
24  }
```

✔ The code meets the specification

Formal Verification Request 52

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.86 ms

Line 21 in File Ownable.sol

```
21  //@CTK_NO_ASF
```

Line 22-24 in File Ownable.sol


```
22  function Ownable() public {
23    owner = msg.sender;
24  }
```

✔ The code meets the specification

Formal Verification Request 53

If method completes, integer overflow would not happen.

 11, Apr 2019

 28.3 ms

Line 38 in File Ownable.sol

```
38 // @CTK_NO_OVERFLOW
```

Line 40-44 in File Ownable.sol


```
40 function transferOwnership(address newOwner) public onlyOwner {
41     require(newOwner != address(0));
42     emit OwnershipTransferred(owner, newOwner);
43     owner = newOwner;
44 }
```

 The code meets the specification

Formal Verification Request 54

Method will not encounter an assertion failure.

 11, Apr 2019

 0.68 ms

Line 39 in File Ownable.sol

```
39 // @CTK_NO_ASF
```

Line 40-44 in File Ownable.sol


```
40 function transferOwnership(address newOwner) public onlyOwner {
41     require(newOwner != address(0));
42     emit OwnershipTransferred(owner, newOwner);
43     owner = newOwner;
44 }
```

 The code meets the specification

Formal Verification Request 55

If method completes, integer overflow would not happen.

 11, Apr 2019

 91.11 ms

Line 17 in File HasNoContracts.sol

```
17 // @CTK_NO_OVERFLOW
```

Line 19-22 in File HasNoContracts.sol

```

19 function reclaimContract(address contractAddr) external onlyOwner {
20     Ownable contractInst = Ownable(contractAddr);
21     contractInst.transferOwnership(owner);
22 }

```

✓ The code meets the specification

Formal Verification Request 56

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.95 ms

Line 18 in File HasNoContracts.sol

```

18 // @CTK_NO_ASF

```

Line 19-22 in File HasNoContracts.sol

```

19 function reclaimContract(address contractAddr) external onlyOwner {
20     Ownable contractInst = Ownable(contractAddr);
21     contractInst.transferOwnership(owner);
22 }

```

✓ The code meets the specification

Formal Verification Request 57

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 15.67 ms

Line 24 in File HasNoEther.sol

```

24 // @CTK_NO_OVERFLOW

```

Line 26-28 in File HasNoEther.sol

```

26 function HasNoEther() public payable {
27     require(msg.value == 0);
28 }

```

✓ The code meets the specification

Formal Verification Request 58

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.47 ms

Line 25 in File HasNoEther.sol

```
25 // @CTK_NO_ASF
```

Line 26-28 in File HasNoEther.sol

```
26 function HasNoEther() public payable {  
27     require(msg.value == 0);  
28 }
```

✔ The code meets the specification

Formal Verification Request 59

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 3.56 ms

Line 33 in File HasNoEther.sol

```
33 // @CTK_NO_OVERFLOW
```

Line 35-36 in File HasNoEther.sol

```
35 function() external {  
36 }
```

✔ The code meets the specification

Formal Verification Request 60

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.48 ms

Line 34 in File HasNoEther.sol

```
34 // @CTK_NO_ASF
```

Line 35-36 in File HasNoEther.sol

```
35 function() external {  
36 }
```

✔ The code meets the specification

Formal Verification Request 61

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 14.52 ms

Line 41 in File HasNoEther.sol

41 `//@CTK_NO_OVERFLOW`

Line 43-46 in File HasNoEther.sol

```
43 function reclaimEther() external onlyOwner {
44     // solium-disable-next-line security/no-send
45     // assert(owner.send(address(this).balance));
46 }
```

✓ The code meets the specification

Formal Verification Request 62

Method will not encounter an assertion failure.

📅 11, Apr 2019

🕒 0.58 ms

Line 42 in File HasNoEther.sol

42 `//@CTK_NO_ASF`

Line 43-46 in File HasNoEther.sol

```
43 function reclaimEther() external onlyOwner {
44     // solium-disable-next-line security/no-send
45     // assert(owner.send(address(this).balance));
46 }
```

✓ The code meets the specification

Formal Verification Request 63

If method completes, integer overflow would not happen.

📅 11, Apr 2019

🕒 18.29 ms

Line 27 in File Claimable.sol

27 `//@CTK_NO_OVERFLOW`

Line 29-31 in File Claimable.sol


```
29 function transferOwnership(address newOwner) onlyOwner public {
30     pendingOwner = newOwner;
31 }
```

✓ The code meets the specification

Formal Verification Request 64

Method will not encounter an assertion failure.

 11, Apr 2019

 0.53 ms

Line 28 in File Claimable.sol

```
28 // @CTK_NO_ASF
```

Line 29-31 in File Claimable.sol


```
29 function transferOwnership(address newOwner) onlyOwner public {  
30     pendingOwner = newOwner;  
31 }
```

 The code meets the specification

Formal Verification Request 65

If method completes, integer overflow would not happen.

 11, Apr 2019

 20.81 ms

Line 36 in File Claimable.sol

```
36 // @CTK_NO_OVERFLOW
```

Line 38-42 in File Claimable.sol


```
38 function claimOwnership() onlyPendingOwner public {  
39     emit OwnershipTransferred(owner, pendingOwner);  
40     owner = pendingOwner;  
41     pendingOwner = address(0);  
42 }
```

 The code meets the specification

Formal Verification Request 66

Method will not encounter an assertion failure.

 11, Apr 2019

 0.77 ms

Line 37 in File Claimable.sol

```
37 // @CTK_NO_ASF
```

Line 38-42 in File Claimable.sol

```
38 function claimOwnership() onlyPendingOwner public {  
39     emit OwnershipTransferred(owner, pendingOwner);  
40     owner = pendingOwner;  
41     pendingOwner = address(0);  
42 }
```



✔ The code meets the specification