

# CERTIK AUDIT REPORT FOR ONTOLOGY



Request Date: 2019-05-09  
Revision Date: 2019-05-14  
Platform Name: Ethereum



# Contents

|                                     |           |
|-------------------------------------|-----------|
| <b>Disclaimer</b>                   | <b>1</b>  |
| <b>Exective Summary</b>             | <b>2</b>  |
| <b>Vulnerability Classification</b> | <b>2</b>  |
| <b>Testing Summary</b>              | <b>3</b>  |
| Audit Score . . . . .               | 3         |
| Type of Issues . . . . .            | 3         |
| Vulnerability Details . . . . .     | 4         |
| <b>Formal Verification Results</b>  | <b>5</b>  |
| How to read . . . . .               | 5         |
| <b>Manual Review Notes</b>          | <b>28</b> |
| <b>Source Code</b>                  | <b>31</b> |
| <b>Source Code</b>                  | <b>36</b> |

## 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 Ontology(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.

## Executive Summary

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

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

## Vulnerability Classification

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

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

## Testing Summary

# PASS

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

May 14, 2019



### Type of Issues

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

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

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

## Vulnerability Details

### Critical

No issue found.

### Medium

No issue found.

### Low



No issue found.

# Formal Verification Results

## How to read

### Detail for Request 1

transferFrom to same address


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

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

|              |  |
|--------------|--|
| CERTIK label | <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> |

## Formal Verification Request 1

SafeMath sub

📅 13, May 2019

🕒 14.62 ms

```
50  /*@CTK "SafeMath sub"
51     @post (a < b) == __reverted
52     @post !__reverted -> __return == a - b
53     @post !__reverted -> !__has_overflow
54     @post !(__has_buf_overflow)
55     @post !(__has_assertion_failure)
56  */

57  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
58      require(b <= a);
59      uint256 c = a - b;
60
61      return c;
62  }
```

✅ The code meets the specification

## Formal Verification Request 2

SafeMath add

📅 13, May 2019

🕒 17.63 ms

```
67  /*@CTK "SafeMath add"
68     @post (a + b < a || a + b < b) == __reverted
69     @post !__reverted -> __return == a + b
70     @post !__reverted -> !__has_overflow
71     @post !(__has_buf_overflow)
72     @post !(__has_assertion_failure)
73  */

74  function add(uint256 a, uint256 b) internal pure returns (uint256) {
75      uint256 c = a + b;
76      require(c >= a);
77
78      return c;
79  }
```


✅ The code meets the specification



## Formal Verification Request 3

SafeMath div

 13, May 2019

 14.1 ms

```
85  /*@CTK "SafeMath div"
86     @post b != 0 -> !__reverted
87     @post !__reverted -> __return == a % b
88     @post !__reverted -> !__has_overflow
89     @post !(__has_buf_overflow)
90     @post !(__has_assertion_failure)
91  */


92  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
93      require(b != 0);
94      return a % b;
95  }
```

 The code meets the specification

## Formal Verification Request 4

If method completes, integer overflow would not happen.

 13, May 2019

 20.36 ms

```
106  //@CTK NO_OVERFLOW


114  constructor(address[] _owners, uint _required) public {
115      require(!initialized, "already initialized");
116      require(_owners.length >= _required);
117      require(_owners.length <= MAX_OWNER);
118
119      for (uint i = 0; i < _owners.length; i++) {
120          address owner = _owners[i];
121          require(owner != address(0));
122          require(!isOwner[owner]);
123          isOwner[owner] = true;
124      }
125
126      owners = _owners;
127      required = _required;
128      initialized = true;
129  }
```

 The code meets the specification

## Formal Verification Request 5

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

 13, May 2019

 0.6 ms

```
107 //@CTK_NO_BUF_OVERFLOW


114 constructor(address[] _owners, uint _required) public {
115     require(!initialized, "already initialized");
116     require(_owners.length >= _required);
117     require(_owners.length <= MAX_OWNER);
118
119     for (uint i = 0; i < _owners.length; i++) {
120         address owner = _owners[i];
121         require(owner != address(0));
122         require(!isOwner[owner]);
123         isOwner[owner] = true;
124     }
125
126     owners = _owners;
127     required = _required;
128     initialized = true;
129 }
```

 The code meets the specification

## Formal Verification Request 6

Method will not encounter an assertion failure.

 13, May 2019

 0.63 ms

```
108 //@CTK_NO_ASF

114 constructor(address[] _owners, uint _required) public {
115     require(!initialized, "already initialized");
116     require(_owners.length >= _required);
117     require(_owners.length <= MAX_OWNER);
118
119     for (uint i = 0; i < _owners.length; i++) {
120         address owner = _owners[i];
121         require(owner != address(0));
122         require(!isOwner[owner]);
123         isOwner[owner] = true;
124     }
125
126     owners = _owners;
127     required = _required;
128     initialized = true;
129 }
```

✔ The code meets the specification

## Formal Verification Request 7

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 19.91 ms

```
124 // @CTK_NO_OVERFLOW
132 function PAXT00EP4(address _from, string memory _ont_bas58_address, uint256 _value
    ) public {
133     require(msg.sender == _from, "not token owner");
134     // OEP-4 base58 address length == 34
135     require(bytes(_ont_bas58_address).length == 34);
136     require(_value > 0 && (_value % PAX_UNIT) == 0);
137     require(pax.transferFrom(_from, address(this), _value));
138     emit PAXT00EP4Event("PAXT00EP4", _from, _ont_bas58_address, _value);
139 }
```

✔ The code meets the specification

## Formal Verification Request 8

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

📅 13, May 2019

🕒 0.53 ms

```
125 // @CTK_NO_BUF_OVERFLOW
132 function PAXT00EP4(address _from, string memory _ont_bas58_address, uint256 _value
    ) public {
133     require(msg.sender == _from, "not token owner");
134     // OEP-4 base58 address length == 34
135     require(bytes(_ont_bas58_address).length == 34);
136     require(_value > 0 && (_value % PAX_UNIT) == 0);
137     require(pax.transferFrom(_from, address(this), _value));
138     emit PAXT00EP4Event("PAXT00EP4", _from, _ont_bas58_address, _value);
139 }
```

✔ The code meets the specification

## Formal Verification Request 9

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.52 ms

```
126 // @CTK NO_ASF
132 function PAXTOOEP4(address _from, string memory _ont_bas58_address, uint256 _value
    ) public {
133     require(msg.sender == _from, "not token owner");
134     // OEP-4 base58 address length == 34
135     require(bytes(_ont_bas58_address).length == 34);
136     require(_value > 0 && (_value % PAX_UNIT) == 0);
137     require(pax.transferFrom(_from, address(this), _value));
138     emit PAXTOOEP4Event("PAXTOOEP4", _from, _ont_bas58_address, _value);
139 }
```

✔ The code meets the specification

## Formal Verification Request 10

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 14.21 ms

```
143 // @CTK NO_OVERFLOW
150 function OEP4TOPAX(string _txId, address _to, uint256 _value, string memory
    _txhash, uint8[] _v, bytes32[] _r, bytes32[] _s) public {
151     require(_validWithdrawSig(_txId, pax, _to, _value, _v, _r, _s));
152     require(_withdraw(pax, _to, _value, _txhash));
153     emit OEP4TOPAXEvent("OEP4TOPAX", _to, _value, _txhash);
154 }
```

✔ The code meets the specification

## Formal Verification Request 11

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

📅 13, May 2019

🕒 0.57 ms


```
144 // @CTK NO_BUF_OVERFLOW
150 function OEP4TOPAX(string _txId, address _to, uint256 _value, string memory
    _txhash, uint8[] _v, bytes32[] _r, bytes32[] _s) public {
151     require(_validWithdrawSig(_txId, pax, _to, _value, _v, _r, _s));
152     require(_withdraw(pax, _to, _value, _txhash));
153     emit OEP4TOPAXEvent("OEP4TOPAX", _to, _value, _txhash);
154 }
```

✔ The code meets the specification

## Formal Verification Request 12

Method will not encounter an assertion failure.

 13, May 2019

 0.58 ms

```

145 //@CTK NO_ASF

150 function OEP4TOPAX(string _txId, address _to, uint256 _value, string memory
    _txhash, uint8[] _v, bytes32[] _r, bytes32[] _s) public {
151     require(!_validWithdrawSig(_txId, pax, _to, _value, _v, _r, _s));
152     require(!_withdraw(pax, _to, _value, _txhash));
153     emit OEP4TOPAXEvent("OEP4TOPAX", _to, _value, _txhash);
154 }


```

 The code meets the specification

## Formal Verification Request 13

If method completes, integer overflow would not happen.

 13, May 2019

 130.98 ms

```

206 //@CTK NO_OVERFLOW

215 function ChangeOwner(string _txId, uint8[] _v, bytes32[] _r, bytes32[] _s, address
    [] _owners, uint _required) public {
216     require(bytes(_txId).length == 36);
217
218     bytes32 txHash = _getTxHash(_txId);
219
220     require(!txExist[txHash]);
221     require(_v.length == _r.length && _r.length == _s.length);
222     require(_v.length >= required);
223
224     uint confirmed = 0;
225
226     for (uint i = 0; i < required; i++) {
227
228         address signer = _getChangeOwnerSigner(_txId, _owners, _required, _v[i], _r
            [i], _s[i]);
229         require(isOwner[signer]);
230         require(!confirmations[txHash][signer]);
231         confirmed++;
232         confirmations[txHash][signer] = true;
233     }
234     require(confirmed >= required);
235
236     require(_owners.length >= _required);
237     require(_owners.length <= MAX_OWNER);
238     owners = _owners;

```

```
239     required = _required;
240 }
```

✔ The code meets the specification

## Formal Verification Request 14

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

📅 13, May 2019

🕒 0.8 ms

```
207 // @CTK NO_BUF_OVERFLOW

215 function ChangeOwner(string _txId, uint8[] _v, bytes32[] _r, bytes32[] _s, address
    [] _owners, uint _required) public {
216     require(bytes(_txId).length == 36);
217
218     bytes32 txHash = _getTxHash(_txId);
219
220     require(!txExist[txHash]);
221     require(_v.length == _r.length && _r.length == _s.length);
222     require(_v.length >= required);
223
224     uint confirmed = 0;
225
226     for (uint i = 0; i < required; i++) {
227
228         address signer = _getChangeOwnerSigner(_txId, _owners, _required, _v[i], _r
            [i], _s[i]);
229         require(isOwner[signer]);
230         require(!confirmations[txHash][signer]);
231         confirmed++;
232         confirmations[txHash][signer] = true;
233     }
234     require(confirmed >= required);
235
236     require(_owners.length >= _required);
237     require(_owners.length <= MAX_OWNER);
238     owners = _owners;
239     required = _required;
240 }
```

✔ The code meets the specification

## Formal Verification Request 15

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.78 ms

```
208 // @CTK_NO_ASF

215 function ChangeOwner(string _txId, uint8[] _v, bytes32[] _r, bytes32[] _s, address
    [] _owners, uint _required) public {
216     require(bytes(_txId).length == 36);
217
218     bytes32 txHash = _getTxHash(_txId);
219
220     require(!txExist[txHash]);
221     require(_v.length == _r.length && _r.length == _s.length);
222     require(_v.length >= required);
223
224     uint confirmed = 0;
225
226     for (uint i = 0; i < required; i++) {
227
228         address signer = _getChangeOwnerSigner(_txId, _owners, _required, _v[i], _r
            [i], _s[i]);
229         require(isOwner[signer]);
230         require(!confirmations[txHash][signer]);
231         confirmed++;
232         confirmations[txHash][signer] = true;
233     }
234     require(confirmed >= required);
235
236     require(_owners.length >= _required);
237     require(_owners.length <= MAX_OWNER);
238     owners = _owners;
239     required = _required;
240 }
```

✔ The code meets the specification

## Formal Verification Request 16

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 32.69 ms

```
224 // @CTK_NO_OVERFLOW

231 function _validWithdrawSig(string _txId, address _pax, address _to, uint256 _value
    , uint8[] _v, bytes32[] _r, bytes32[] _s) internal returns(bool) {
232     require(bytes(_txId).length == 36);
233
234     bytes32 txHash = _getTxHash(_txId);
235
236     require(!txExist[txHash]);
237     require(_v.length == _r.length && _r.length == _s.length);
238     require(_v.length >= required);
239     require(_to != address(0));
240
241     uint confirmed = 0;
242 }
```

```

243     for (uint i = 0; i < required; i++) {
244         address signer = _getWithdrawSigner(_txId, _pax, _to, _value, _v[i], _r[i],
            _s[i]);
245         require(isOwner[signer]);
246         require(!confirmations[txHash][signer]);
247         confirmed++;
248         confirmations[txHash][signer] = true;
249     }
250     require(confirmed >= required);
251     return true;
252 }

```

✓ The code meets the specification

## Formal Verification Request 17

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

📅 13, May 2019

🕒 0.72 ms

```

225 // @CTK_NO_BUF_OVERFLOW

231 function _validWithdrawSig(string _txId, address _pax, address _to, uint256 _value
    , uint8[] _v, bytes32[] _r, bytes32[] _s) internal returns(bool) {
232     require(bytes(_txId).length == 36);
233
234     bytes32 txHash = _getTxHash(_txId);
235
236     require(!txExist[txHash]);
237     require(_v.length == _r.length && _r.length == _s.length);
238     require(_v.length >= required);
239     require(_to != address(0));
240
241     uint confirmed = 0;
242
243     for (uint i = 0; i < required; i++) {
244         address signer = _getWithdrawSigner(_txId, _pax, _to, _value, _v[i], _r[i],
            _s[i]);
245         require(isOwner[signer]);
246         require(!confirmations[txHash][signer]);
247         confirmed++;
248         confirmations[txHash][signer] = true;
249     }
250     require(confirmed >= required);
251     return true;
252 }

```


✓ The code meets the specification

## Formal Verification Request 18

Method will not encounter an assertion failure.

📅 13, May 2019




 0.68 ms

```
226 // @CTK NO_ASF
231 function _validWithdrawSig(string _txId, address _pax, address _to, uint256 _value
    , uint8[] _v, bytes32[] _r, bytes32[] _s) internal returns(bool) {
232     require(bytes(_txId).length == 36);
233
234     bytes32 txHash = _getTxHash(_txId);
235
236     require(!txExist[txHash]);
237     require(_v.length == _r.length && _r.length == _s.length);
238     require(_v.length >= required);
239     require(_to != address(0));
240
241     uint confirmed = 0;
242
243     for (uint i = 0; i < required; i++) {
244         address signer = _getWithdrawSigner(_txId, _pax, _to, _value, _v[i], _r[i],
            _s[i]);
245         require(isOwner[signer]);
246         require(!confirmations[txHash][signer]);
247         confirmed++;
248         confirmations[txHash][signer] = true;
249     }
250     require(confirmed >= required);
251     return true;
252 }
```

 The code meets the specification

## Formal Verification Request 19

If method completes, integer overflow would not happen.

 13, May 2019 132.62 ms


```
235 // @CTK NO_OVERFLOW
250 function _withdraw(PaxInterface _pax, address _to, uint256 _value, string memory
    _txhash) internal returns(bool){
251     //ontology transacton hash length == 64
252     require(bytes(_txhash).length == 64);
253     require(_value > 0 && (_value % PAX_UNIT) == 0);
254     //Duplication tx check
255     require(!TxHashs[_txhash]);
256     require(_pax.transfer(_to, _value));
257     TxHashs[_txhash] = true;
258     return true;
259 }
```

 The code meets the specification

## Formal Verification Request 20

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

 13, May 2019

 5.53 ms

```
236 //@CTK_NO_BUF_OVERFLOW


250 function _withdraw(PaxInterface _pax, address _to, uint256 _value, string memory
    _txhash) internal returns(bool){
251 //ontology transacton hash length == 64
252 require(bytes(_txhash).length == 64);
253 require(_value > 0 && (_value % PAX_UNIT) == 0);
254 //Duplication tx check
255 require(!TxHashs[_txhash]);
256 require(_pax.transfer(_to, _value));
257 TxHashs[_txhash] = true;
258 return true;
259 }
```

 The code meets the specification

## Formal Verification Request 21

Method will not encounter an assertion failure.

 13, May 2019

 5.33 ms

```
237 //@CTK_NO_ASF


250 function _withdraw(PaxInterface _pax, address _to, uint256 _value, string memory
    _txhash) internal returns(bool){
251 //ontology transacton hash length == 64
252 require(bytes(_txhash).length == 64);
253 require(_value > 0 && (_value % PAX_UNIT) == 0);
254 //Duplication tx check
255 require(!TxHashs[_txhash]);
256 require(_pax.transfer(_to, _value));
257 TxHashs[_txhash] = true;
258 return true;
259 }
```

 The code meets the specification

## Formal Verification Request 22

If method completes, integer overflow would not happen.

 13, May 2019

 87.73 ms

```
254 // @CTK_NO_OVERFLOW

264 function _getWithdrawSigner(string _txId, address _token, address _destination,
    uint _value, uint8 v, bytes32 r, bytes32 s) internal pure returns (address
    signer){
265     bytes32 argsHash = keccak256(abi.encodePacked(_txId, _addressToString(_token),
        _addressToString(_destination), _uint2str(_value)));
266     bytes memory prefix = "\x19Ethereum Signed Message:\n66";
267     bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
        argsHash)));
268     return ecrecover(signedHash, v, r, s);
269 }
```

✔ The code meets the specification

## Formal Verification Request 23

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

📅 13, May 2019

🕒 1.07 ms

```
255 // @CTK_NO_BUF_OVERFLOW

264 function _getWithdrawSigner(string _txId, address _token, address _destination,
    uint _value, uint8 v, bytes32 r, bytes32 s) internal pure returns (address
    signer){
265     bytes32 argsHash = keccak256(abi.encodePacked(_txId, _addressToString(_token),
        _addressToString(_destination), _uint2str(_value)));
266     bytes memory prefix = "\x19Ethereum Signed Message:\n66";
267     bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
        argsHash)));
268     return ecrecover(signedHash, v, r, s);
269 }
```

✔ The code meets the specification

## Formal Verification Request 24

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 1.04 ms

```
256 // @CTK_NO_ASF
```

```

264 function _getWithdrawSigner(string _txId, address _token, address _destination,
    uint _value, uint8 v, bytes32 r, bytes32 s) internal pure returns (address
    signer){
265 bytes32 argsHash = keccak256(abi.encodePacked(_txId, _addressToString(_token),
    _addressToString(_destination), _uint2str(_value)));
266 bytes memory prefix = "\x19Ethereum Signed Message:\n66";
267 bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
    argsHash)));
268 return ecrecover(signedHash, v, r, s);
269 }

```

✓ The code meets the specification

## Formal Verification Request 25

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 48.53 ms

```

268 // @CTK NO_OVERFLOW
277 function _getChangeOwnerSigner(string _txId, address[] _owners, uint _required,
    uint8 v, bytes32 r, bytes32 s) internal pure returns (address signer){
278 string memory encodeOwner = "";
279 for (uint i = 0; i < _owners.length; i++) {
280     encodeOwner = string(abi.encodePacked(encodeOwner, _addressToString(_owners
    [i])));
281 }
282
283 bytes32 argsHash = keccak256(abi.encodePacked(_txId, encodeOwner, _uint2str(
    _required)));
284 bytes memory prefix = "\x19Ethereum Signed Message:\n66";
285 bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
    argsHash)));
286 return ecrecover(signedHash, v, r, s);
287 }

```

✓ The code meets the specification

## Formal Verification Request 26

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

📅 13, May 2019

🕒 0.79 ms

```

269 // @CTK NO_BUF_OVERFLOW

```

```

277     function _getChangeOwnerSigner(string _txId, address[] _owners, uint _required,
278         uint8 v, bytes32 r, bytes32 s) internal pure returns (address signer){
279         string memory encodeOwner = "";
280         for (uint i = 0; i < _owners.length; i++) {
281             encodeOwner = string(abi.encodePacked(encodeOwner, _addressToString(_owners
282                 [i])));
283         }
284         bytes32 argsHash = keccak256(abi.encodePacked(_txId, encodeOwner, _uint2str(
285             _required)));
286         bytes memory prefix = "\x19Ethereum Signed Message:\n66";
287         bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
288             argsHash)));
289         return ecrecover(signedHash, v, r, s);
290     }

```

✔ The code meets the specification

## Formal Verification Request 27

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.76 ms

```

270     // @CTK_NO_ASF
271
272     function _getChangeOwnerSigner(string _txId, address[] _owners, uint _required,
273         uint8 v, bytes32 r, bytes32 s) internal pure returns (address signer){
274         string memory encodeOwner = "";
275         for (uint i = 0; i < _owners.length; i++) {
276             encodeOwner = string(abi.encodePacked(encodeOwner, _addressToString(_owners
277                 [i])));
278         }
279         bytes32 argsHash = keccak256(abi.encodePacked(_txId, encodeOwner, _uint2str(
280             _required)));
281         bytes memory prefix = "\x19Ethereum Signed Message:\n66";
282         bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
283             argsHash)));
284         return ecrecover(signedHash, v, r, s);
285     }

```

✔ The code meets the specification

## Formal Verification Request 28

renouncePauser correctness

📅 13, May 2019

🕒 3.75 ms

```

271  /*@CTK "renouncePauser correctness"
272     @post msg.sender == 0x0 -> __reverted
273     @post !_pausers.bearer[msg.sender] -> __reverted
274     @post msg.sender != 0x0 && _pausers.bearer[msg.sender]
275         -> !__reverted && !_post._pausers.bearer[msg.sender]
276     */

277  function _getChangeOwnerSigner(string _txId, address[] _owners, uint _required,
278     uint8 v, bytes32 r, bytes32 s) internal pure returns (address signer){
279     string memory encodeOwner = "";
280     for (uint i = 0; i < _owners.length; i++) {
281         encodeOwner = string(abi.encodePacked(encodeOwner, _addressToString(_owners
282             [i])));
283     }
284     bytes32 argsHash = keccak256(abi.encodePacked(_txId, encodeOwner, _uint2str(
285         _required)));
286     bytes memory prefix = "\x19Ethereum Signed Message:\n66";
287     bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
288         argsHash)));
289     return ecrecover(signedHash, v, r, s);
290 }

```

✔ The code meets the specification

## Formal Verification Request 29

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 5.37 ms

```

298  //@CTK NO_OVERFLOW

304  function _bytes32ToString(bytes32 data) internal pure returns (string) {
305     bytes memory bytesString = new bytes(66);
306     bytesString[0] = '0';
307     bytesString[1] = 'x';
308     for (uint j = 0; j < 32; j++) {
309         byte char = byte(bytes32(uint(data) * 2 ** (8 * j)));
310         bytesString[j * 2 + 2] = _uintToAscii(uint(char) / 16);
311         bytesString[j * 2 + 3] = _uintToAscii(uint(char) % 16);
312     }
313     return string(bytesString);
314 }

```

✔ The code meets the specification

## Formal Verification Request 30

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

📅 13, May 2019

🕒 0.41 ms

```
299 //@CTK_NO_BUF_OVERFLOW

304 function _bytes32ToString(bytes32 data) internal pure returns (string) {
305     bytes memory bytesString = new bytes(66);
306     bytesString[0] = '0';
307     bytesString[1] = 'x';
308     for (uint j = 0; j < 32; j++) {
309         byte char = byte(bytes32(uint(data) * 2 ** (8 * j)));
310         bytesString[j * 2 + 2] = _uintToAscii(uint(char) / 16);
311         bytesString[j * 2 + 3] = _uintToAscii(uint(char) % 16);
312     }
313     return string(bytesString);
314 }
```

✔ The code meets the specification

## Formal Verification Request 31

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.41 ms

```
300 //@CTK_NO_ASF

304 function _bytes32ToString(bytes32 data) internal pure returns (string) {
305     bytes memory bytesString = new bytes(66);
306     bytesString[0] = '0';
307     bytesString[1] = 'x';
308     for (uint j = 0; j < 32; j++) {
309         byte char = byte(bytes32(uint(data) * 2 ** (8 * j)));
310         bytesString[j * 2 + 2] = _uintToAscii(uint(char) / 16);
311         bytesString[j * 2 + 3] = _uintToAscii(uint(char) % 16);
312     }
313     return string(bytesString);
314 }
```

✔ The code meets the specification

## Formal Verification Request 32

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 5.71 ms

```
311 //@CTK_NO_OVERFLOW
```

```
317     function _addressToString(address _addr) internal pure returns (string) {
318         bytes32 value = bytes32(uint256(_addr));
319         bytes memory alphabet = "0123456789abcdef";
320
321         bytes memory str = new bytes(42);
322         str[0] = '0';
323         str[1] = 'x';
324         for (uint i = 0; i < 20; i++) {
325             str[2 + i * 2] = alphabet[uint(value[i + 12] >> 4)];
326             str[3 + i * 2] = alphabet[uint(value[i + 12] & 0x0f)];
327         }
328         return string(str);
329     }
```

✔ The code meets the specification

## Formal Verification Request 33

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

📅 13, May 2019

🕒 0.4 ms

```
312     //@CTK_NO_BUF_OVERFLOW
317     function _addressToString(address _addr) internal pure returns (string) {
318         bytes32 value = bytes32(uint256(_addr));
319         bytes memory alphabet = "0123456789abcdef";
320
321         bytes memory str = new bytes(42);
322         str[0] = '0';
323         str[1] = 'x';
324         for (uint i = 0; i < 20; i++) {
325             str[2 + i * 2] = alphabet[uint(value[i + 12] >> 4)];
326             str[3 + i * 2] = alphabet[uint(value[i + 12] & 0x0f)];
327         }
328         return string(str);
329     }
```

✔ The code meets the specification

## Formal Verification Request 34

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.38 ms

```
313     //@CTK_NO_ASF
```



```
317     function _addressToString(address _addr) internal pure returns (string) {
318         bytes32 value = bytes32(uint256(_addr));
319         bytes memory alphabet = "0123456789abcdef";
320
321         bytes memory str = new bytes(42);
322         str[0] = '0';
323         str[1] = 'x';
324         for (uint i = 0; i < 20; i++) {
325             str[2 + i * 2] = alphabet[uint(value[i + 12] >> 4)];
326             str[3 + i * 2] = alphabet[uint(value[i + 12] & 0x0f)];
327         }
328         return string(str);
329     }
```

✔ The code meets the specification

## Formal Verification Request 35

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 128.0 ms

```
340     //@CTK NO_OVERFLOW
352     function _bytesToAddress(bytes _address) internal pure returns (address) {
353         uint160 m = 0;
354         uint160 b = 0;
355
356         for (uint8 i = 0; i < 20; i++) {
357             m *= 256;
358             b = uint160(_address[i]);
359             m += (b);
360         }
361
362         return address(m);
363     }
```

✔ The code meets the specification

## Formal Verification Request 36

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

📅 13, May 2019

🕒 1.36 ms

```
341     //@CTK NO_BUF_OVERFLOW
```

```
352     function _bytesToAddress(bytes _address) internal pure returns (address) {
353         uint160 m = 0;
354         uint160 b = 0;
355
356         for (uint8 i = 0; i < 20; i++) {
357             m *= 256;
358             b = uint160(_address[i]);
359             m += (b);
360         }
361
362         return address(m);
363     }
```

✔ The code meets the specification

## Formal Verification Request 37

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 1.3 ms

```
342     //@CTK NO_ASF
352     function _bytesToAddress(bytes _address) internal pure returns (address) {
353         uint160 m = 0;
354         uint160 b = 0;
355
356         for (uint8 i = 0; i < 20; i++) {
357             m *= 256;
358             b = uint160(_address[i]);
359             m += (b);
360         }
361
362         return address(m);
363     }
```

✔ The code meets the specification

## Formal Verification Request 38

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 83.61 ms

```
360     //@CTK NO_OVERFLOW
372     function _uint2str(uint _i) internal pure returns (string memory _uintAsString) {
373         if (_i == 0) {
374             return "0";
375         }
376     }
```

```
375     }
376     uint j = _i;
377     uint len;
378     while (j != 0) {
379         len++;
380         j /= 10;
381     }
382     bytes memory bstr = new bytes(len);
383     uint k = len - 1;
384     while (_i != 0) {
385         bstr[k--] = byte(uint8(48 + _i % 10));
386         _i /= 10;
387     }
388     return string(bstr);
389 }
```

✔ The code meets the specification

## Formal Verification Request 39

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

📅 13, May 2019

🕒 1.38 ms

```
361 // @CTK_NO_BUF_OVERFLOW

372 function _uint2str(uint _i) internal pure returns (string memory _uintAsString) {
373     if (_i == 0) {
374         return "0";
375     }
376     uint j = _i;
377     uint len;
378     while (j != 0) {
379         len++;
380         j /= 10;
381     }
382     bytes memory bstr = new bytes(len);
383     uint k = len - 1;
384     while (_i != 0) {
385         bstr[k--] = byte(uint8(48 + _i % 10));
386         _i /= 10;
387     }
388     return string(bstr);
389 }
```

✔ The code meets the specification

## Formal Verification Request 40

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 1.32 ms

```
362 // @CTK NO_ASF
372 function _uint2str(uint _i) internal pure returns (string memory _uintAsString) {
373     if (_i == 0) {
374         return "0";
375     }
376     uint j = _i;
377     uint len;
378     while (j != 0) {
379         len++;
380         j /= 10;
381     }
382     bytes memory bstr = new bytes(len);
383     uint k = len - 1;
384     while (_i != 0) {
385         bstr[k--] = byte(uint8(48 + _i % 10));
386         _i /= 10;
387     }
388     return string(bstr);
389 }
```

✔ The code meets the specification

## Formal Verification Request 41

If method completes, integer overflow would not happen.

📅 13, May 2019

🕒 5.44 ms

```
408 // @CTK NO_OVERFLOW
414 function _uintToAscii(uint number) internal pure returns (byte) {
415     if (number < 10) {
416         return byte(48 + number);
417     } else if (number < 16) {
418         return byte(87 + number);
419     } else {
420         revert();
421     }
422 }
```

✔ The code meets the specification

## Formal Verification Request 42

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

📅 13, May 2019

🕒 0.38 ms

```
409 //@CTK_NO_BUF_OVERFLOW

414 function _uintToAscii(uint number) internal pure returns (byte) {
415     if (number < 10) {
416         return byte(48 + number);
417     } else if (number < 16) {
418         return byte(87 + number);
419     } else {
420         revert();
421     }
422 }
```

✔ The code meets the specification

## Formal Verification Request 43

Method will not encounter an assertion failure.

📅 13, May 2019

🕒 0.42 ms

```
410 //@CTK_NO_ASF

414 function _uintToAscii(uint number) internal pure returns (byte) {
415     if (number < 10) {
416         return byte(48 + number);
417     } else if (number < 16) {
418         return byte(87 + number);
419     } else {
420         revert();
421     }
422 }
```

✔ The code meets the specification

# Manual Review Notes

## Review Details

### Source Code SHA-256 Checksum

- `pax-lock-multi-sig.sol` 68708e035cfb5d0620e710f4a34c08b84afebbb39ee66a00bbaa444685cc20e7
- `PAX-OEP4.py` 430614eeb680e824e94b1213bee12d27d618cbd5cda5342b80c047cf378eb591

### Summary

CertiK team is invited by The Ontology Network team to audit the design and implementations of ontology's first stablecoin project Paxos Standard(PAX), an ERC20 token based on Ontology's OEP-4 token standard, and the source code has been analyzed under different perspectives and with different tools such as CertiK formal verification checking as well as manual reviews by smart contract experts. We have been actively interacting with client-side engineers when there was any potential loopholes or recommended design changes during the audit process, and Ontology Network team has been actively giving us updates for the source code and feedback about the business logic.

In Ontology April's press release, it explains the intention of these works is to bring the convenience and accelerate the distributed businesses process to the traditional institutions and individuals to do business in fiat terms in the Ontology ecosystem. Ontology team mentioned, the token (Paxos Standard) will continue to use the same symbol `PAX` as its ticker, based on Ontology's OEP-4 token standard.

Ontology team presents following approach to fulfill the business enhancements:

- `pax-lock-multi-sig.sol`: a ERC20 Smart Contract that act as atomic swapper between `PAX-ERC20` and `PAX-OEP4`, following functionalities are supporting in the contract:
  - `PAXTOOEP4()`: allow token holder to deposit and convert from `PAX-ERC20` to `PAX-OPE4` token on Ontology blockchain
  - `OEP4TOPAX()`: allow token holder to withdraw and convert `PAX-OEP4` token from Ontology blockchain to `PAX-ERC20` at given Ethereum wallet
  - `ChangeOwner()`: allow to interchange the owner(s) in the contract
- `PAX-OEP4.py`: a python sematic smart contract, implements with the tool `Ontology Neptune`, a Python Smart Contract Compiler, allow to compile Python files into `.avm` format for usage in the Ontology blockchain. Following capabilities are supporting by the contract:
  - `IncreasePAX()`: allow Supply Controller to increase supply token to given receiver address
  - `DecreasePAX()`: allow Supply Controller to decrease supply token to given receiver address

- `transferOwnership()`: allow transfer contract ownership from old owner to new owner account
- `freeze()`: allow Enforcement Role admin to freeze a specific address
- `unfreeze()`: allow Enforcement Role admin to unfreeze a specific address
- `wipeFrozenAddress()`: allow Enforcement Role admin to deduct the frozen account balance to 0
- `pause()`: allow contract owner to pause all the transfer and approve activities in the contract
- `unpause()`: allow contract owner to unpause the contract status

Overall we found the `PAX-OEP4.py` contract follows good practices, with a reasonable amount of features on top of the ERC20 related to administrative privileged controls by the token issuer. With the final update of source code and delivery of the audit report, we conclude that the contract is not vulnerable to any classically known anti-patterns or security issues. The audit report itself is not necessarily a guarantee of correctness or trustworthiness, and we always recommend seeking multiple opinions, more test coverage and sandbox deployments before the mainnet release.

## Recommendations

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

### `pax-lock-multi-sig.sol`

- **PaxLock** – for contract variable “`txExist`”, there’re only read operations but no writes. Recommend to reconfirm the usage of this variable.
- recommend to remove unused contract variable “`owner`” and “`supplyController`”
- **PaxLock.constructor** – `require(!initialized, “already initialized”)` can be removed since constructor function will only be called once
- **PaxLock.PAXTOOEP4** – recommend to remove the first function parameter `_from` and honor `msg.sender` instead
- **PaxLock.ChangeOwner** – recommend to refactor contract variable “`confirmations`” to be a local variable, since we don’t actually need to persist the result, checking `txHash` exists is sufficient

### `PAX-OEP4.py`

- recommend to create a helper function **`isValidAmount`** for amount verification, the function should check both variable type to be an integer and its value is greater than 0.
- **IncreasePAX** – recommend to replace `assert(amount > 0)` with stricter `assert(isValidAmount(amount))`
- **IncreasePAX** – recommend to add an assertion for receiver address, `assert(isAddress(receiver))`

- **DecreasePAX** – recommend to replace `assert(amount > 0)` with stricter `assert(isValidAmount(amount))`
- **transfer** – recommend to replace `assert(amount > 0)` with stricter `assert(isValidAmount(amount))`
- **transferFrom** – recommend to replace `assert(amount > 0)` with stricter `assert(isValidAmount(amount))`
- **transferFrom** – recommend to reuse “allowance” function instead of calling `Get` directly between line 524-525
- **approve** – recommend to replace `assert(amount > 0)` with stricter `assert(isValidAmount(amount))`
- **wipeFrozenAddress** – recommend to assert the address is frozen before wiping out, `assert(isFrozen(address))`
- **balanceOf** – recommend to return default value 0 by adding “or 0” after `Get` operation since the value can be undefined
- **allowance** – recommend to return default value 0 by adding “or 0” after `Get` operation since the value can be undefined
- **Mul** – recommend to remove the unused function
- **Div** – recommend to remove the unused function



## Solidity Source Code

File pax-lock-multi-sig.sol

```
1 pragma solidity ^0.4.24;
2
3 /**
4  * @title SafeMath
5  * @dev Math operations with safety checks that throw on error
6  */
7 library SafeMath {
8     /**
9      * @dev Subtracts two numbers, reverts on overflow (i.e. if subtrahend is greater
10     than minuend).
11     */
12     function sub(uint256 a, uint256 b) internal pure returns (uint256) {
13         require(b <= a);
14         uint256 c = a - b;
15
16         return c;
17     }
18
19     /**
20     * @dev Adds two numbers, reverts on overflow.
21     */
22     function add(uint256 a, uint256 b) internal pure returns (uint256) {
23         uint256 c = a + b;
24         require(c >= a);
25
26         return c;
27     }
28 }
29 interface PaxInterface {
30     function transfer(address _to, uint256 _value) external returns (bool);
31     function transferFrom(address _from, address _to, uint _value) external returns (
32     bool success);
33 }
34 contract PaxLock {
35
36     using SafeMath for uint256;
37
38     //CONTRACT ROLE
39     address public owner;
40     address public supplyController;
41
42     //CONTRACT STATE
43     bool private initialized = false;
44     bool public paused = false;
45
46     //ONTOLOGY TRANSACTION HASH RECORDS
47     mapping (string => bool) internal TxHashs;
48
49     uint256 public constant PAX_UNIT = 1000000000000000000;
50
51     //PAX ERC-20 Token Address on Ethereum ropsten testnet,
```

```

52 //it need to be replaced the mainnet address: 0
    x8e870d67f660d95d5be530380d0ec0bd388289e1
53 PaxInterface constant public pax = PaxInterface(0
    xb88afaecfe9fb9bda50a978645690bf282ed0490);
54
55 //Signature
56 uint public required;
57 address[] public owners;
58 mapping(address => bool) isOwner;
59 uint constant MAX_OWNER = 50;
60
61 //Multi-Sig transaction record
62 mapping(bytes32 => mapping(address => bool)) confirmations;
63 mapping(bytes32 => bool) txExist;
64
65 //PAXTOOEP4 Event
66 event PAXTOOEP4Event(string indexed topic, address ERC_address, string
    ont_bas58_address, uint256 amount);
67 event OEP4TOPAXEvent(string indexed topic, address ERC_address, uint256 amount,
    string ont_txhash);
68
69 constructor(address[] _owners, uint _required) public {
70     require(!initialized, "already initialized");
71     require(_owners.length >= _required);
72     require(_owners.length <= MAX_OWNER);
73
74     for (uint i = 0; i < _owners.length; i++) {
75         address owner = _owners[i];
76         require(owner != address(0));
77         require(!isOwner[owner]);
78         isOwner[owner] = true;
79     }
80
81     owners = _owners;
82     required = _required;
83     initialized = true;
84 }
85
86 /**
87 * PAX token owner transfer PAX to current contract, then ontology pax-gateway
    will
88 * transfer PAX-OEP4 Token to token holder on ontology chain.
89 * the owner must approve target amount PAX token to current contract, otherwise,
90 * the contract can't transfer holder token.
91 */
92 function PAXTOOEP4(address _from, string memory _ont_bas58_address, uint256 _value
    ) public {
93     require(msg.sender == _from, "not token owner");
94     //OEP-4 base58 address length == 34
95     require(bytes(_ont_bas58_address).length == 34);
96     require(_value > 0 && (_value % PAX_UNIT) == 0);
97     require(pax.transferFrom(_from, address(this), _value));
98     emit PAXTOOEP4Event("PAXTOOEP4", _from, _ont_bas58_address, _value);
99 }
100
101 /**
102 * called by multi-owners, this function will transfer target amount PAX to
    specific token holder.

```

```

103  */
104  function OEP4TOPAX(string _txId, address _to, uint256 _value, string memory
    _txhash, uint8[] _v, bytes32[] _r, bytes32[] _s) public {
105      require(_validWithdrawSig(_txId, pax, _to, _value, _v, _r, _s));
106      require(_withdraw(pax, _to, _value, _txhash));
107      emit OEP4TOPAXEvent("OEP4TOPAX", _to, _value, _txhash);
108  }
109
110  /**
111   * called by owners, change old owners to new owners.
112   */
113  function ChangeOwner(string _txId, uint8[] _v, bytes32[] _r, bytes32[] _s, address
    [] _owners, uint _required) public {
114      require(bytes(_txId).length == 36);
115
116      bytes32 txHash = _getTxHash(_txId);
117
118      require(!txExist[txHash]);
119      require(_v.length == _r.length && _r.length == _s.length);
120      require(_v.length >= required);
121
122      uint confirmed = 0;
123
124      for (uint i = 0; i < required; i++) {
125
126          address signer = _getChangeOwnerSigner(_txId, _owners, _required, _v[i], _r
            [i], _s[i]);
127          require(isOwner[signer]);
128          require(!confirmations[txHash][signer]);
129          confirmed++;
130          confirmations[txHash][signer] = true;
131      }
132      require(confirmed >= required);
133
134      require(_owners.length >= _required);
135      require(_owners.length <= MAX_OWNER);
136      owners = _owners;
137      required = _required;
138  }
139
140  function _validWithdrawSig(string _txId, address _pax, address _to, uint256 _value
    , uint8[] _v, bytes32[] _r, bytes32[] _s) internal returns(bool) {
141      require(bytes(_txId).length == 36);
142
143      bytes32 txHash = _getTxHash(_txId);
144
145      require(!txExist[txHash]);
146      require(_v.length == _r.length && _r.length == _s.length);
147      require(_v.length >= required);
148      require(_to != address(0));
149
150      uint confirmed = 0;
151
152      for (uint i = 0; i < required; i++) {
153          address signer = _getWithdrawSigner(_txId, _pax, _to, _value, _v[i], _r[i],
            _s[i]);
154          require(isOwner[signer]);
155          require(!confirmations[txHash][signer]);

```

```
156         confirmed++;
157         confirmations[txHash][signer] = true;
158     }
159     require(confirmed >= required);
160     return true;
161 }
162
163 function _withdraw(PaxInterface _pax, address _to, uint256 _value, string memory
    _txhash) internal returns(bool){
164     //ontology transacton hash length == 64
165     require(bytes(_txhash).length == 64);
166     require(_value > 0 && (_value % PAX_UNIT) == 0);
167     //Duplication tx check
168     require(!TxHashs[_txhash]);
169     require(_pax.transfer(_to, _value));
170     TxHashs[_txhash] = true;
171     return true;
172 }
173
174 function _getTxHash(string _txId) internal pure returns (bytes32){
175     return keccak256(abi.encodePacked(_txId));
176 }
177
178 function _getWithdrawSigner(string _txId, address _token, address _destination,
    uint _value, uint8 v, bytes32 r, bytes32 s) internal pure returns (address
    signer){
179     bytes32 argsHash = keccak256(abi.encodePacked(_txId, _addressToString(_token),
        _addressToString(_destination), _uint2str(_value)));
180     bytes memory prefix = "\x19Ethereum Signed Message:\n66";
181     bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
        argsHash)));
182     return ecrecover(signedHash, v, r, s);
183 }
184
185 function _getChangeOwnerSigner(string _txId, address[] _owners, uint _required,
    uint8 v, bytes32 r, bytes32 s) internal pure returns (address signer){
186     string memory encodeOwner = "";
187     for (uint i = 0; i < _owners.length; i++) {
188         encodeOwner = string(abi.encodePacked(encodeOwner, _addressToString(_owners
            [i])));
189     }
190
191     bytes32 argsHash = keccak256(abi.encodePacked(_txId, encodeOwner, _uint2str(
        _required)));
192     bytes memory prefix = "\x19Ethereum Signed Message:\n66";
193     bytes32 signedHash = keccak256(abi.encodePacked(prefix, _bytes32ToString(
        argsHash)));
194     return ecrecover(signedHash, v, r, s);
195 }
196
197 function _bytes32ToString(bytes32 data) internal pure returns (string) {
198     bytes memory bytesString = new bytes(66);
199     bytesString[0] = '0';
200     bytesString[1] = 'x';
201     for (uint j = 0; j < 32; j++) {
202         byte char = byte(bytes32(uint(data) * 2 ** (8 * j)));
203         bytesString[j * 2 + 2] = _uintToAscii(uint(char) / 16);
204         bytesString[j * 2 + 3] = _uintToAscii(uint(char) % 16);
```

```
205     }
206     return string(bytesString);
207 }
208
209 function _addressToString(address _addr) internal pure returns (string) {
210     bytes32 value = bytes32(uint256(_addr));
211     bytes memory alphabet = "0123456789abcdef";
212
213     bytes memory str = new bytes(42);
214     str[0] = '0';
215     str[1] = 'x';
216     for (uint i = 0; i < 20; i++) {
217         str[2 + i * 2] = alphabet[uint(value[i + 12] >> 4)];
218         str[3 + i * 2] = alphabet[uint(value[i + 12] & 0x0f)];
219     }
220     return string(str);
221 }
222
223 function _bytesToAddress(bytes _address) internal pure returns (address) {
224     uint160 m = 0;
225     uint160 b = 0;
226
227     for (uint8 i = 0; i < 20; i++) {
228         m *= 256;
229         b = uint160(_address[i]);
230         m += (b);
231     }
232
233     return address(m);
234 }
235
236 function _uint2str(uint _i) internal pure returns (string memory _uintAsString) {
237     if (_i == 0) {
238         return "0";
239     }
240     uint j = _i;
241     uint len;
242     while (j != 0) {
243         len++;
244         j /= 10;
245     }
246     bytes memory bstr = new bytes(len);
247     uint k = len - 1;
248     while (_i != 0) {
249         bstr[k--] = byte(uint8(48 + _i % 10));
250         _i /= 10;
251     }
252     return string(bstr);
253 }
254
255 function _uintToAscii(uint number) internal pure returns (byte) {
256     if (number < 10) {
257         return byte(48 + number);
258     } else if (number < 16) {
259         return byte(87 + number);
260     } else {
261         revert();
262     }
263 }
```



```
47 DecreasePAXEvent = RegisterAction("DecreasePAX", "OEP-4 Address", "ERC-20 Address", "
    amount")
48 SetAutoSupplyControllerEvent = RegisterAction("SetAutoSupplyController", "
    newController")
49 SetManualSupplyControllerEvent = RegisterAction("SetManualSupplyController", "
    newController")
50 UpgradeContractEvent = RegisterAction("UpgradeContract")
51
52 def Main(operation, args):
53
54     if operation == "init":
55         return init()
56
57     if operation == 'name':
58         return name()
59
60     if operation == 'symbol':
61         return symbol()
62
63     if operation == 'decimals':
64         return decimals()
65
66     if operation == 'totalSupply':
67         return totalSupply()
68
69     if operation == 'balanceOf':
70         acct = args[0]
71         return balanceOf(acct)
72
73     if operation == 'transfer':
74         from_acct = args[0]
75         to_acct = args[1]
76         amount = args[2]
77         return transfer(from_acct, to_acct, amount)
78
79     if operation == 'transferMulti':
80         return transferMulti(args)
81
82     if operation == 'transferFrom':
83         spender = args[0]
84         from_acct = args[1]
85         to_acct = args[2]
86         amount = args[3]
87         return transferFrom(spender, from_acct, to_acct, amount)
88
89     if operation == 'approve':
90         owner = args[0]
91         spender = args[1]
92         amount = args[2]
93         return approve(owner, spender, amount)
94
95     if operation == 'allowance':
96         owner = args[0]
97         spender = args[1]
98         return allowance(owner, spender)
99
100    if operation == 'IncreasePAX':
101        receiver = args[0]
```

```
102     amount = args[1]
103     txhash = args[2]
104     return IncreasePAX(receiver, amount, txhash)
105
106     if operation == 'DecreasePAX':
107         deducter = args[0]
108         amount = args[1]
109         ERC20_address = args[2]
110         return DecreasePAX(deducter, amount, ERC20_address)
111
112     if operation == 'setLawEnforcementRole':
113         newRole = args[0]
114         return setLawEnforcementRole(newRole)
115
116     if operation == 'getEnforcementRole':
117         return getEnforcementRole()
118
119     if operation == 'transferOwnership':
120         newOwner = args[0]
121         return transferOwnership(newOwner)
122
123     if operation == 'getOwner':
124         return getOwner()
125
126     if operation == 'freeze':
127         address = args[0]
128         return freeze(address)
129
130     if operation == 'unfreeze':
131         address = args[0]
132         return unfreeze(address)
133
134     if operation == 'wipeFrozenAddress':
135         address = args[0]
136         return wipeFrozenAddress(address)
137
138     if operation == 'setAutoSupplyController':
139         address = args[0]
140         return setAutoSupplyController(address)
141
142     if operation == 'setManualSupplyController':
143         address = args[0]
144         return setManualSupplyController(address)
145
146     if operation == 'getAutoSupplyController':
147         return getAutoSupplyController()
148
149     if operation == 'getManualSupplyController':
150         return getManualSupplyController()
151
152     if operation == 'pause':
153         return pause()
154
155     if operation == 'unpause':
156         return unpause()
157
158     if operation == 'isPaused':
159         return isPaused()
```



```

160
161     if operation == 'isInitialized':
162         return isInitialized()
163
164     if operation == 'isFrozen':
165         address = args[0]
166         return isFrozen(address)
167
168     if operation == "upgrade":
169         code = args[0]
170         return upgrade(code)
171
172 def init():
173     """
174     Initialize smart contract.
175
176     :return: True or raise exception.
177     """
178     assert (CheckWitness(Owner))
179     assert (not isInitialized())
180
181     Put(ctx, INITIALIZED, True)
182     Put(ctx, TOTAL_SUPPLY_KEY, 0)
183     Put(ctx, OWNER_KEY, Owner)
184     Put(ctx, ENFORCEMENT_ROLE_KEY, EnforcementRole)
185     Put(ctx, AUTO_SUPPLY_CONTROLLER_KEY, AutoSupplyController)
186     Put(ctx, MANUAL_SUPPLY_CONTROLLER_KEY, ManualSupplyController)
187
188     return True
189
190 def IncreasePAX(receiver, amount, txHash):
191     """
192     Increase supply token to receiver address.
193     :param amount: Increase token amount.
194     :return: True or raise exception.
195     """
196     assert(amount > 0)
197     assert (CheckWitness(getAutoSupplyController()) or CheckWitness(
198         getManualSupplyController()))
199     assert (len(txHash) == 64 or len(txHash) == 66)
200
201     if Get(ctx, concat(PAX_ETH_TXHASH_KEY, txHash)):
202         raise Exception("duplicated transaction")
203
204     Put(ctx, concat(BALANCE_PREFIX, receiver), Add(balanceOf(receiver), amount))
205     Put(ctx, TOTAL_SUPPLY_KEY, Add(totalSupply(), amount))
206     Put(ctx, concat(PAX_ETH_TXHASH_KEY, txHash), 1)
207
208     IncreasePAXEvent(receiver, amount, txHash)
209     return True
210
211 def DecreasePAX(deducter, amount, ERC20_address):
212     """
213     Decrease token supply from deducter address.
214     :param amount: decreased token amount.
215     :return:
216     """
217     assert (amount > 0)

```

```

217     assert (CheckWitness(deducter))
218     #eth address format:0x673dfa9caf9145fdbef98e9d9874f36e63d8a5b4,length is 42
219     assert (len(erc20_address) == 42 or len(erc20_address) == 40)
220
221     Put(ctx, concat(BALANCE_PREFIX, deducter), Sub(balanceOf(deducter), amount))
222     Put(ctx, TOTAL_SUPPLY_KEY, Sub(totalSupply(), amount))
223
224     DecreasePAXEvent(deducter, erc20_address, amount)
225     return True
226
227 def setLawEnforcementRole(newEnforceRole):
228     """
229     Set Enforment role address.
230     :param newEnforceRole: new enforcement role account.
231     :return:
232     """
233     assert (isAddress(newEnforceRole))
234     assert (CheckWitness(getEnforcementRole() or CheckWitness(getOwner())))
235
236     Put(ctx, ENFORCEMENT_ROLE_KEY, newEnforceRole)
237     return True
238
239 def getEnforcementRole():
240     """
241     Get current enforcement role account.
242     :return: enforcement role.
243     """
244     enforcementRole = Get(ctx, ENFORCEMENT_ROLE_KEY)
245
246     if not enforcementRole:
247         return getOwner()
248
249     return enforcementRole
250
251 def transferOwnership(newOwner):
252     """
253     transfer contract ownership from current owner to new owner account.
254     :param newOwner: new smart contract owner.
255     :return:True or raise exception.
256     """
257     assert(isAddress(newOwner))
258     assert(CheckWitness(getOwner()))
259
260     Put(ctx, OWNER_KEY, newOwner)
261     TransferOwnerEvent(getOwner(), newOwner)
262     return True
263
264 def getOwner():
265     """
266     Get contract owner.
267     :return:smart contract owner.
268     """
269     return Get(ctx, OWNER_KEY)
270
271 def freeze(address):
272     """
273     Freeze specific acccount, it will not be traded unless it will be unfreez.
274     :param address: Frozen account.

```

```

275     :return: True or raise exception.
276     """
277     assert(isAddress(address))
278     assert (CheckWitness(getEnforcementRole()))
279
280     Put(ctx, concat(FROZEN_PREFIX, address), True)
281     FrozenEvent(address)
282     return True
283
284 def unfreeze(address):
285     """
286     Unfreeze specific account, this account will be re-traded
287     :param address: Unfrozen account.
288     :return: True or raise exception.
289     """
290
291     assert (isAddress(address))
292     assert (CheckWitness(getEnforcementRole()))
293
294     Delete(ctx, concat(FROZEN_PREFIX, address))
295     UnfrozenEvent(address)
296     return True
297
298 def wipeFrozenAddress(address):
299     """
300     Deduct the balance of the frozen account to 0.
301     :param address: frozen account.
302     :return: True or raise exception.
303     """
304     assert(isAddress(address))
305     assert(CheckWitness(getEnforcementRole()))
306     balance = balanceOf(address)
307     total = totalSupply()
308     Put(ctx, TOTAL_SUPPLY_KEY, Sub(total, balance))
309     Put(ctx, concat(BALANCE_PREFIX, address), 0)
310
311     WipeFrozenEvent(address)
312     return True
313
314 def setAutoSupplyController(address):
315     """
316     Set new supply controller account.
317     :param address: new supply controller account.
318     :return:
319     """
320     assert (isAddress(address))
321     assert(CheckWitness(getOwner()))
322
323     Put(ctx, AUTO_SUPPLY_CONTROLLER_KEY, address)
324     SetAutoSupplyControllerEvent(address)
325     return True
326
327 def setManualSupplyController(address):
328     """
329     Set new supply controller account.
330     :param address: new supply controller account.
331     :return:
332     """

```

```

333     assert (isAddress(address))
334     assert(CheckWitness(getOwner()))
335
336     Put(ctx, MANUAL_SUPPLY_CONTROLLER_KEY, address)
337     SetManualSupplyControllerEvent(address)
338     return True
339
340 def getAutoSupplyController():
341     """
342     Get current contract supply controller account.
343     :return: supply controller account.
344     """
345     return Get(ctx, AUTO_SUPPLY_CONTROLLER_KEY)
346
347 def getManualSupplyController():
348     """
349     Get current contract supply controller account.
350     :return: supply controller account.
351     """
352     return Get(ctx, MANUAL_SUPPLY_CONTROLLER_KEY)
353
354 def pause():
355     """
356     Set the smart contract to paused state, the token can not be transfered, approved.
357     Just can invoke some get interface, like getOwner.
358     :return: True or raise exception.
359     """
360     assert(CheckWitness(getOwner()))
361
362     Put(ctx, PAUSED, True)
363     PauseEvent()
364     return True
365
366 def unpaused():
367     """
368     Resume the smart contract to normal state, all the function can be invoked.
369     :return: True or raise exception.
370     """
371     assert(CheckWitness(getOwner()))
372
373     Put(ctx, PAUSED, False)
374     UnpauseEvent()
375     return True
376
377 def isPaused():
378     """
379     Confirm whether the contract is paused or not.
380     :return: True or False
381     """
382     return Get(ctx, PAUSED)
383
384 def isInitialized():
385     """
386     Confir whether the contract is initialized or not.
387     :return: True or False
388     """
389     return Get(ctx, INITIALIZED)
390

```

```

391 def isFrozen(address):
392     """
393     Confir whether specific account is frozen or not.
394     :param address:confirmed account.
395     :return:True or False.
396     """
397     return Get(ctx, concat(FROZEN_PREFIX, address))
398
399 def name():
400     """
401     :return: name of the token
402     """
403     return NAME
404
405
406 def symbol():
407     """
408     :return: symbol of the token
409     """
410     return SYMBOL
411
412
413 def decimals():
414     """
415     :return: the decimals of the token
416     """
417     return DECIMALS
418
419
420 def totalSupply():
421     """
422     :return: the total supply of the token
423     """
424     return Get(ctx, TOTAL_SUPPLY_KEY)
425
426
427 def balanceOf(account):
428     """
429     :param account:
430     :return: the token balance of account
431     """
432     return Get(ctx, concat(BALANCE_PREFIX, account))
433
434
435 def transfer(from_acct, to_acct, amount):
436     """
437     Transfer amount of tokens from from_acct to to_acct
438     :param from_acct: the account from which the amount of tokens will be transferred
439     :param to_acct: the account to which the amount of tokens will be transferred
440     :param amount: the amount of the tokens to be transferred, >= 0
441     :return: True means success, False or raising exception means failure.
442     """
443     assert(not isPaused())
444     assert(amount > 0)
445     assert(isAddress(to_acct))
446     assert(CheckWitness(from_acct))
447     assert(not isFrozen(from_acct))
448     assert(not isFrozen(to_acct))

```

```

449
450
451     fromKey = concat(BALANCE_PREFIX, from_acct)
452     fromBalance = balanceOf(from_acct)
453     if amount > fromBalance:
454         return False
455     if amount == fromBalance:
456         Delete(ctx, fromKey)
457     else:
458         Put(ctx, fromKey, Sub(fromBalance, amount))
459
460     toKey = concat(BALANCE_PREFIX, to_acct)
461     toBalance = balanceOf(to_acct)
462     Put(ctx, toKey, Add(toBalance, amount))
463
464     TransferEvent(from_acct, to_acct, amount)
465
466     return True
467
468 def transferMulti(args):
469     """
470     :param args: the parameter is 'transfer' function parameter array, like [from, to,
471                 amount]
472     :return: True or raising exception.
473     """
474     for p in args:
475         assert (len(p) == 3)
476         assert (transfer(p[0], p[1], p[2]))
477
478     return True
479
480 def approve(owner, spender, amount):
481     """
482     owner allow spender to spend amount of token from owner account
483     Note here, the amount should be less than the balance of owner right now.
484     :param owner:
485     :param spender:
486     :param amount: amount>=0
487     :return: True means success, False or raising exception means failure.
488     """
489     assert (amount > 0)
490     assert (not isPaused())
491     assert (not isFrozen(owner))
492     assert (not isFrozen(spender))
493
494     assert (isAddress(spender))
495     assert (CheckWitness(owner))
496     assert (balanceOf(owner) >= amount)
497
498     Put(ctx, concat(concat(APPROVE_PREFIX, owner), spender), amount)
499
500     ApproveEvent(owner, spender, amount)
501
502     return True
503
504 def transferFrom(spender, from_acct, to_acct, amount):
505     """

```

```

505     spender spends amount of tokens on the behalf of from_acct, spender makes a
        transaction of amount of tokens
506     from from_acct to to_acct
507     :param spender:
508     :param from_acct:
509     :param to_acct:
510     :param amount:
511     :return:
512     """
513     assert (amount > 0)
514     assert (not isPaused())
515     assert (isAddress(from_acct) and isAddress(to_acct))
516     assert (CheckWitness(spender))
517     assert (not isFrozen(from_acct))
518     assert (not isFrozen(to_acct))
519
520     fromKey = concat(BALANCE_PREFIX, from_acct)
521     fromBalance = balanceOf(from_acct)
522     assert (fromBalance >= amount)
523
524     approveKey = concat(concat(APPROVE_PREFIX, from_acct), spender)
525     approvedAmount = Get(ctx, approveKey)
526
527     if amount > approvedAmount:
528         return False
529     elif amount == approvedAmount:
530         Delete(ctx, approveKey)
531         Put(ctx, fromKey, Sub(fromBalance, amount))
532     else:
533         Put(ctx, approveKey, Sub(approvedAmount, amount))
534         Put(ctx, fromKey, Sub(fromBalance, amount))
535
536     toBalance = balanceOf(to_acct)
537     Put(ctx, concat(BALANCE_PREFIX, to_acct), Add(toBalance, amount))
538
539     TransferFromEvent(spender, from_acct, to_acct, amount)
540     return True
541
542 def allowance(owner, spender):
543     """
544     check how many token the spender is allowed to spend from owner account
545     :param owner: token owner
546     :param spender: token spender
547     :return: the allowed amount of tokens
548     """
549     key = concat(concat(APPROVE_PREFIX, owner), spender)
550     return Get(ctx, key)
551
552 def upgrade(code):
553     """
554     upgrade current smart contract to new smart contract.
555     :param code: new smart contract avm code.
556     :return: True or raise exception.
557     """
558     owner = getOwner()
559     assert(CheckWitness(owner))
560
561     ongBalance = Invoke(0, ONG_ADDRESS, 'balanceOf', state(CONTRACT_ADDRESS))

```

```

562     res = Invoke(0, ONG_ADDRESS, "transfer", [state(CONTRACT_ADDRESS, owner,
563         ontBalance)])
564     if res != b'\x01':
565         assert(False)
566
567     ontBalance = Invoke(0, ONT_ADDRESS, 'balanceOf', state(CONTRACT_ADDRESS))
568     res = Invoke(0, ONT_ADDRESS, "transfer", [state(CONTRACT_ADDRESS, owner,
569         ontBalance)])
570     if res != b'\x01':
571         assert (False)
572
573     #upgrade smart contract
574     res = Migrate(code, "", "", "", "", "", "")
575     if not res:
576         assert (False)
577
578     UpgradeContractEvent()
579
580     return True
581
582 def Add(a, b):
583     """
584     Adds two numbers, throws on overflow.
585     :param a:operand a
586     :param b:operand b
587     :return:
588     """
589     c = a + b
590     assert (c >= a)
591     return c
592
593 def Sub(a, b):
594     """
595     Substracts two numbers, throws on overflow (i.e. if subtrahend is greater than
596     minuend).
597     :param a: operand a
598     :param b: operand b
599     :return: a - b if a - b > 0 or revert the transaction.
600     """
601     assert(a>=b)
602     return a-b
603
604 def Mul(a, b):
605     """
606     Multiplies two numbers, throws on overflow.
607     :param a: operand a
608     :param b: operand b
609     :return:
610     """
611     if a == 0:
612         return 0
613     c = a * b
614     assert(c / a == b)
615     return c
616
617 def Div(a, b):
618     """

```



```
617 Integer division of two numbers, truncating the quotient.
618 :param a: operand a
619 :param b: operand b
620 :return:
621 """
622
623     assert (b > 0)
624     c = a / b
625     return c
626
627 def isAddress(address):
628     """
629     check the address is legal address.
630     :param address:
631     :return: True or raise exception.
632     """
633     assert (len(address) == 20 and address != ZERO_ADDRESS)
634     return True
```