CertiK Audit Report For PlasmaPay

Request Date: 2019-05-06
Revision Date: 2019-05-08
Platform Name: Ethereum

CERTIK

Contents

Disclaimer

This Report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and PlasmaPay(the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This Report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This Report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

Exective Summary

This report has been prepared as product of the Smart Contract Audit request by PlasmaPay. This audit was conducted to discover issues and vulnerabilities in the source code of PlasmaPay'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:

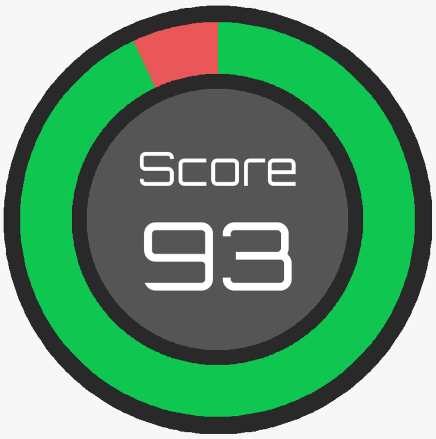
- Critical: The code implementation does not match the specification, or it could result in loss of funds for contract owner or users.
- Medium: The code implementation does not match the specification at certain condition, or it could affect the security standard by lost of access control.
- Low: The 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 08, 2019



Type of Issues

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

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

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

Vulnerability Details

Critical

No issue found.

Medium

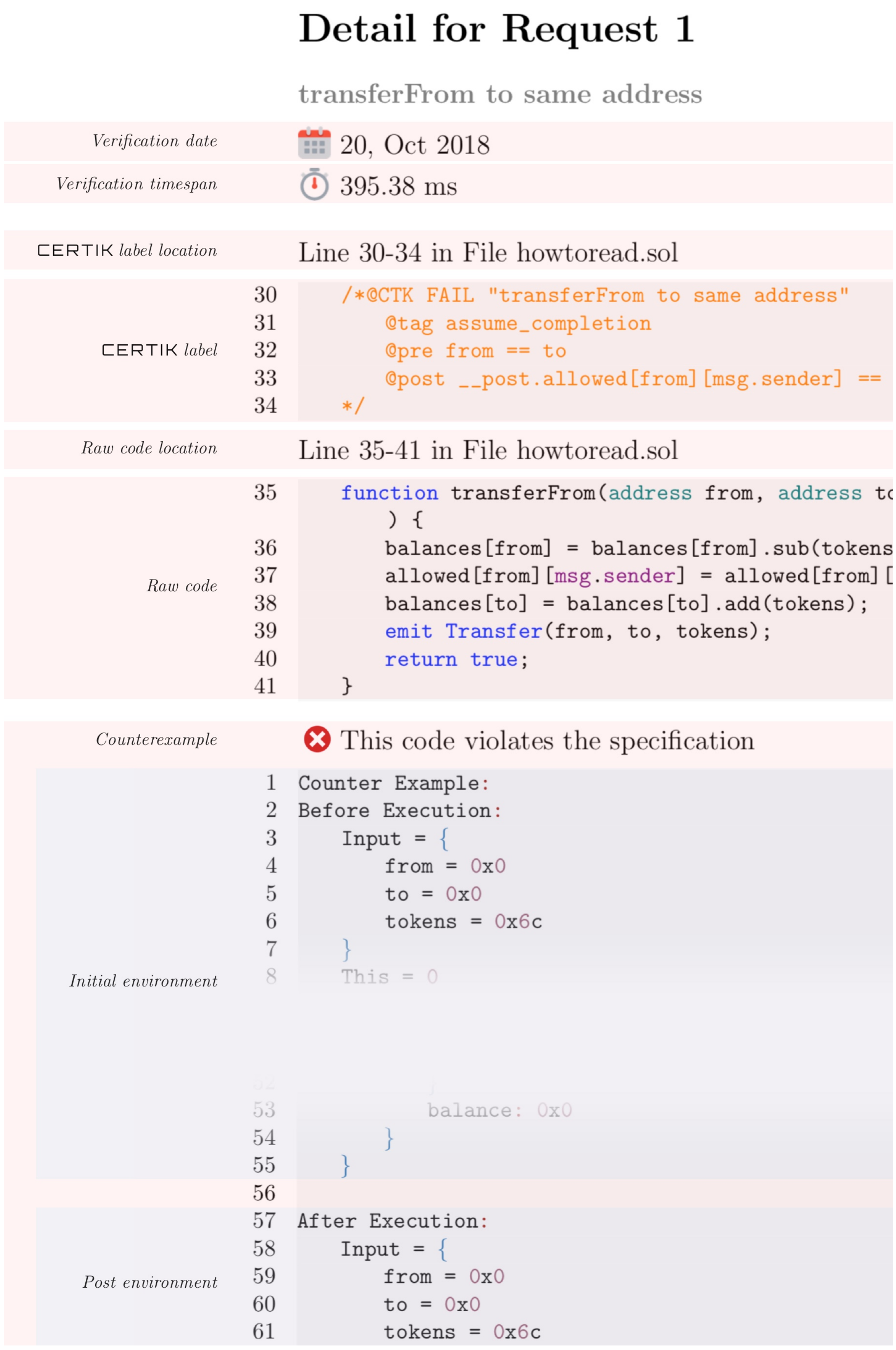
No issue found.

Low

No issue found.

Formal Verification Results

How to read



Detail for Request 1

transferFrom to same address

Verification date	20, Oct 2018
Verification timespan	395.38 ms
CERTIK *label location*	Line 30-34 in File howtoread.sol

CERTIK *label*

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

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

Raw code

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

Counterexample — This code violates the specification

Initial environment

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

53              balance: 0x0
54          }
55      }
56
```

Post environment

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

Formal Verification Request 1

SafeMath mul

📅 08, May 2019
⏱ 327.34 ms

Line 18-23 in File pbktoken.sol

```
/*@CTK "SafeMath mul"
  @post (a > 0) && (((a * b) / a) != b) -> __reverted
  @post __reverted -> (a > 0) && (((a * b) / a) != b)
  @post !__reverted -> __return == a * b
  @post !__reverted == !__has_overflow
*/
```

Line 24-31 in File pbktoken.sol

```
function mul(uint256 a, uint256 b) internal pure returns (uint256) {
  if (a == 0) {
    return 0;
  }
  uint256 c = a * b;
  assert(c / a == b);
  return c;
}
```

✓ The code meets the specification

Formal Verification Request 2

SafeMath div

📅 08, May 2019
⏱ 11.85 ms

Line 36-40 in File pbktoken.sol

```
/*@CTK "SafeMath div"
  @post b != 0 -> !__reverted
  @post !__reverted -> __return == a / b
  @post !__reverted -> !__has_overflow
*/
```

Line 41-46 in File pbktoken.sol

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

✓ The code meets the specification

Formal Verification Request 3

SafeMath sub

📅 08, May 2019
⏱ 28.95 ms

Line 51-55 in File pbktoken.sol

```
/*@CTK "SafeMath sub"
  @post (a < b) == __reverted
  @post !__reverted -> __return == a - b
  @post !__reverted -> !__has_overflow
*/
```

Line 56-59 in File pbktoken.sol

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

✓ The code meets the specification

Formal Verification Request 4

SafeMath add

📅 08, May 2019
⏱ 29.9 ms

Line 64-68 in File pbktoken.sol

```
/*@CTK "SafeMath add"
  @post (a + b < a || a + b < b) == __reverted
  @post !__reverted -> __return == a + b
  @post !__reverted -> !__has_overflow
*/
```

Line 69-73 in File pbktoken.sol

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

✓ The code meets the specification

Formal Verification Request 5

totalSupply

📅 08, May 2019
⏱ 13.31 ms

Line 106-108 in File pbktoken.sol

```
/*@CTK totalSupply
  @post __return == totalSupply_
 */
```

Line 109-111 in File pbktoken.sol

```
function totalSupply() public view returns (uint256) {
  return totalSupply_;
}
```

✓ The code meets the specification

Formal Verification Request 6

transfer

08, May 2019
236.93 ms

Line 118-124 in File pbktoken.sol

```
/*@CTK transfer
  @tag assume_completion
  @pre _to != msg.sender
  @post _to != address(0)
  @post __post.balances[msg.sender] == balances[msg.sender] - _value
  @post __post.balances[_to] == balances[_to] + _value
*/
```

Line 125-134 in File pbktoken.sol

```
function transfer(address _to, uint256 _value) public returns (bool) {
  require(_to != address(0));
  require(_value <= balances[msg.sender]);

  // SafeMath.sub will throw if there is not enough balance.
  balances[msg.sender] = balances[msg.sender].sub(_value);
  balances[_to] = balances[_to].add(_value);
  Transfer(msg.sender, _to, _value);
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 7

balanceOf

08, May 2019
7.08 ms

Line 141-143 in File pbktoken.sol

```
/*@CTK balanceOf
  @post balance == balances[_owner]
 */
```

Line 144-146 in File pbktoken.sol

```
function balanceOf(address _owner) public view returns (uint256 balance) {
  return balances[_owner];
}
```

✓ The code meets the specification

Formal Verification Request 8

burn

08, May 2019
239.02 ms

Line 164-169 in File pbktoken.sol

```
/*@CTK burn
  @tag assume_completion
  @post _value <= balances[msg.sender]
  @post __post.balances[msg.sender] == balances[msg.sender] - _value
  @post __post.totalSupply_ == totalSupply_ - _value
 */
```

Line 170-179 in File pbktoken.sol

```
function burn(uint256 _value) public {
  require(_value <= balances[msg.sender]);
  // no need to require value <= totalSupply, since that would imply the
  // sender's balance is greater than the totalSupply, which *should* be an
      assertion failure

  address burner = msg.sender;
  balances[burner] = balances[burner].sub(_value);
  totalSupply_ = totalSupply_.sub(_value);
  Burn(burner, _value);
}
```

✓ The code meets the specification

Formal Verification Request 9

Ownable

08, May 2019
38.7 ms

Line 201-204 in File pbktoken.sol

```
/*@CTK Ownable
  @post __post.owner == msg.sender
  @post __post.owner2 == owner2_address
 */
```

Line 205-208 in File pbktoken.sol

```
function Ownable() public {
  owner = msg.sender;
  owner2 = owner2_address;
}
```

✓ The code meets the specification

Formal Verification Request 10

transferOwnership

08, May 2019
83.37 ms

Line 227-232 in File pbktoken.sol

```
227  /*@CTK transferOwnership
228    @tag assume_completion
229    @post owner == msg.sender || owner2 == msg.sender
230    @post newOwner != address(0)
231    @post __post.owner == newOwner
232   */
```

Line 233-237 in File pbktoken.sol

```
233  function transferOwnership(address newOwner) public onlyOwner {
234    require(newOwner != address(0));
235    OwnershipTransferred(owner, newOwner);
236    owner = newOwner;
237  }
```

✓ The code meets the specification

Formal Verification Request 11

transferOwnership2

08, May 2019
73.79 ms

Line 243-248 in File pbktoken.sol

```
243  /*@CTK transferOwnership2
244    @tag assume_completion
245    @post owner2 == msg.sender
246    @post newOwner != address(0)
247    @post __post.owner2 == newOwner
248   */
```

Line 249-253 in File pbktoken.sol

```
249  function transferOwnership2(address newOwner) public onlyOwner2 {
250    require(newOwner != address(0));
251    OwnershipTransferred(owner2, newOwner);
252    owner2 = newOwner;
253  }
```

✓ The code meets the specification

Formal Verification Request 12

transferFrom

📅 08, May 2019
⏱ 487.23 ms

Line 290-299 in File pbktoken.sol

```
/*@CTK transferFrom
  @tag assume_completion
  @pre _to != _from
  @post _to != address(0)
  @post _value <= balances[_from]
  @post _value <= allowed[_from][msg.sender]
  @post __post.balances[_from] == balances[_from] - _value
  @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
  @post __post.balances[_to] == balances[_to] + _value
*/
```

Line 300-310 in File pbktoken.sol

```
function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool) {
  require(_to != address(0));
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);

  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  Transfer(_from, _to, _value);
  return true;
}
```

✅ The code meets the specification

Formal Verification Request 13

approve

📅 08, May 2019
⏱ 17.85 ms

Line 322-324 in File pbktoken.sol

```
/*@CTK approve
  @post __post.allowed[msg.sender][_spender] == _value
*/
```

Line 325-329 in File pbktoken.sol

```
function approve(address _spender, uint256 _value) public returns (bool) {
  allowed[msg.sender][_spender] = _value;
  Approval(msg.sender, _spender, _value);
  return true;
}
```

✅ The code meets the specification

Formal Verification Request 14

allowance

📅 08, May 2019
⏱ 9.04 ms

Line 337-339 in File pbktoken.sol

```
/*@CTK allowance
  @post __return == allowed[_owner][_spender]
*/
```

Line 340-342 in File pbktoken.sol

```
function allowance(address _owner, address _spender) public view returns (uint256) {
  return allowed[_owner][_spender];
}
```

✓ The code meets the specification

Formal Verification Request 15

increaseApproval

📅 08, May 2019
⏱ 67.45 ms

Line 354-357 in File pbktoken.sol

```
/*@CTK increaseApproval
  @tag assume_completion
  @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] +
      _addedValue
 */
```

Line 358-362 in File pbktoken.sol

```
function increaseApproval(address _spender, uint _addedValue) public returns (bool)
    {
  allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
  Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 16

decreaseApproval0

📅 08, May 2019
⏱ 114.71 ms

Line 374-378 in File pbktoken.sol

```
/*@CTK decreaseApproval0
  @tag assume_completion
  @pre allowed[msg.sender][_spender] <= _subtractedValue
  @post __post.allowed[msg.sender][_spender] == 0
 */
```

Line 384-393 in File pbktoken.sol

```
function decreaseApproval(address _spender, uint _subtractedValue) public returns (
    bool) {
  uint oldValue = allowed[msg.sender][_spender];
  if (_subtractedValue > oldValue) {
    allowed[msg.sender][_spender] = 0;
  } else {
    allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
  }
  Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 17

decreaseApproval

📅 08, May 2019
⏱ 18.01 ms

Line 379-383 in File pbktoken.sol

```
/*@CTK decreaseApproval
  @tag assume_completion
  @pre allowed[msg.sender][_spender] > _subtractedValue
  @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] -
      _subtractedValue
 */
```

Line 384-393 in File pbktoken.sol

```
function decreaseApproval(address _spender, uint _subtractedValue) public returns (
    bool) {
  uint oldValue = allowed[msg.sender][_spender];
  if (_subtractedValue > oldValue) {
    allowed[msg.sender][_spender] = 0;
  } else {
    allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
  }
  Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 18

mint

08, May 2019
251.57 ms

Line 423-429 in File pbktoken.sol

```
/*@CTK mint
  @tag assume_completion
  @post owner == msg.sender || owner2 == msg.sender
  @post !mintingFinished
  @post __post.totalSupply_ == totalSupply_ + _amount
  @post __post.balances[_to] == balances[_to] + _amount
 */
```

Line 430-436 in File pbktoken.sol

```
function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool)
    {
  totalSupply_ = totalSupply_.add(_amount);
  balances[_to] = balances[_to].add(_amount);
  Mint(_to, _amount);
  Transfer(address(0), _to, _amount);
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 19

finishMinting

08, May 2019
70.02 ms

Line 442-447 in File pbktoken.sol

```
/*@CTK finishMinting
  @tag assume_completion
  @post owner == msg.sender || owner2 == msg.sender
  @post !mintingFinished
  @post __post.mintingFinished
 */
```

Line 448-452 in File pbktoken.sol

```
function finishMinting() onlyOwner canMint public returns (bool) {
  mintingFinished = true;
  MintFinished();
  return true;
}
```

✓ The code meets the specification

Formal Verification Request 20

pause

08, May 2019
97.53 ms

Line 487-492 in File pbktoken.sol

```
/*@CTK pause
  @tag assume_completion
  @post !paused
  @post owner == msg.sender || owner2 == msg.sender
  @post __post.paused
 */
```

Line 493-496 in File pbktoken.sol

```
function pause() onlyOwner whenNotPaused public {
  paused = true;
  Pause();
}
```

✓ The code meets the specification

Formal Verification Request 21

pause

08, May 2019
55.93 ms

Line 501-506 in File pbktoken.sol

```
/*@CTK pause
  @tag assume_completion
  @post paused
  @post owner == msg.sender || owner2 == msg.sender
  @post !__post.paused
 */
```

Line 507-510 in File pbktoken.sol

```
function unpause() onlyOwner whenPaused public {
  paused = false;
  Unpause();
}
```

✓ The code meets the specification

Formal Verification Request 22

TokenTimelock

08, May 2019
93.42 ms

Line 583-589 in File pbktoken.sol

```
/*@CTK TokenTimelock
  @tag assume_completion
  @post _releaseTime > now
  @post __post.token == _token
  @post __post.beneficiary == _beneficiary
  @post __post.releaseTime == _releaseTime
 */
```

Line 590-595 in File pbktoken.sol

```
function TokenTimelock(ERC20Basic _token, address _beneficiary, uint256 _releaseTime
    ) public {
  require(_releaseTime > now);
  token = _token;
  beneficiary = _beneficiary;
  releaseTime = _releaseTime;
}
```

✓ The code meets the specification

Formal Verification Request 23

grantBurner

📅 08, May 2019
⏱ 126.33 ms

Line 667-671 in File pbktoken.sol

```
/*@CTK grantBurner
  @tag assume_completion
  @post owner == msg.sender || owner2 == msg.sender
  @post __post.isBurner[_burner] == _value
 */
```

Line 672-674 in File pbktoken.sol

```
function grantBurner(address _burner, bool _value) public onlyOwner {
    isBurner[_burner] = _value;
}
```

✓ The code meets the specification

Formal Verification Request 24

burn

📅 08, May 2019
⏱ 748.61 ms

Line 689-695 in File pbktoken.sol

```
/*@CTK burn
  @tag assume_completion
  @post isBurner[msg.sender]
  @post _value <= balances[msg.sender]
  @post __post.balances[msg.sender] == balances[msg.sender] - _value
```

```
694     @post __post.totalSupply_ == totalSupply_ - _value
695    */
```

Line 696-698 in File pbktoken.sol

```
696  function burn(uint256 _value) public onlyBurner {
697      super.burn(_value);
698  }
```

✓ The code meets the specification

Static Analysis Results

INSECURE_COMPILER_VERSION

Line 5 in File pbktoken.sol

```
pragma solidity ^0.4.18;
```

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

TIMESTAMP_DEPENDENCY

Line 591 in File pbktoken.sol

```
    require(_releaseTime > now);
```

"now" can be influenced by minors to some degree

TIMESTAMP_DEPENDENCY

Line 601 in File pbktoken.sol

```
    require(now >= releaseTime);
```

"now" can be influenced by minors to some degree

Manual Review Notes

Review Details

Source Code SHA-256 Checksum

- **PBKtoken.sol** `9cf3a1bb72d97c96dc109d20954c5d0d8be126e67a1d4dd88150a7bd97d6e63e`
- **Etherscan** 0x560a20eddeddf84217221aef0d5ca7d7ae7ae798

Summary

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

The PBKToken source code has been deployed to ethereum mainnet at address `0x560a20eddeddf84217221aef0d5ca7d7ae7ae798` by March 20, 2018. It compiled with solidity compiler version ***v0.4.20+commit.3155dd80***. The `PBKToken.sol` is a standard ERC20 token along with some additional operations:

- `Ownable`: Change the owner & owner2 to the new owner(s)
- `Pausable`: Pause the contract for emergency incidents
- `Burnable`: Authorize Burner has capability to perform the burn operations to destroy specific amount of tokens
- `Mintable`: The total token supply is minted by various token distribution or token release plans
  - Private Token Sale
  - PreToken Sale Reserve
  - Token Sale Reserve
  - Reserve For Bonus
  - Reserve For Bounty
  - Reserve For Early Birds
  - Team Options Reserve Address
  - Frozen For Institutional Sales
  - Reserve For Advisors
  - Foundation Reserve
  - Frozen For Management
  - Frozen For Token Sale 2020
- `TimeLock`: allow a beneficiary to extract the tokens after certain time period

At this point the PlasmaPay team didn't provide other repositories sources as testing and documentation reference. We recommend to have more unit tests coverage together with documentation to simulate potential use cases and walk through the functionalites to token holders, especially those super admin privileges that may impact the decentralized nature.

Overall we found the `PBKtoken` contract follows good practices, with 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.

PBKtoken.sol

- **function() payable** – We assume the intention from client to have fallback payable is for accepting ethers, however currently the function type is `private`, consider change to `public` or `external`.

Source Code with CertiK Labels

File pbktoken.sol

```
/**
 * Source Code first verified at https://etherscan.io on Tuesday, March 20, 2018
 (UTC) */

pragma solidity ^0.4.18;

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

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  /*@CTK "SafeMath mul"
    @post (a > 0) && (((a * b) / a) != b) -> __reverted
    @post __reverted -> (a > 0) && (((a * b) / a) != b)
    @post !__reverted -> __return == a * b
    @post !__reverted == !__has_overflow
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

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

  /**
  * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater
      than minuend).
  */
  /*@CTK "SafeMath sub"
    @post (a < b) == __reverted
    @post !__reverted -> __return == a - b
    @post !__reverted -> !__has_overflow
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  /*@CTK "SafeMath add"
    @post (a + b < a || a + b < b) == __reverted
    @post !__reverted -> __return == a + b
    @post !__reverted -> !__has_overflow
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    assert(c >= a);
    return c;
  }
}

// File: zeppelin-solidity/contracts/token/ERC20/ERC20Basic.sol

/**
 * @title ERC20Basic
 * @dev Simpler version of ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/179
 */
contract ERC20Basic {
  function totalSupply() public view returns (uint256);
  function balanceOf(address who) public view returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  event Transfer(address indexed from, address indexed to, uint256 value);
}

// File: zeppelin-solidity/contracts/token/ERC20/BasicToken.sol

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
  using SafeMath for uint256;

  mapping(address => uint256) balances;

  uint256 totalSupply_;

  /**
  * @dev total number of tokens in existence
  */
  /*@CTK totalSupply
    @post __return == totalSupply_
   */
  function totalSupply() public view returns (uint256) {
    return totalSupply_;
  }

  /**
  * @dev transfer token for a specified address
  * @param _to The address to transfer to.
  * @param _value The amount to be transferred.
  */
  /*@CTK transfer
    @tag assume_completion
    @pre _to != msg.sender
    @post _to != address(0)
    @post __post.balances[msg.sender] == balances[msg.sender] - _value
    @post __post.balances[_to] == balances[_to] + _value
  */
  function transfer(address _to, uint256 _value) public returns (bool) {
    require(_to != address(0));
    require(_value <= balances[msg.sender]);

    // SafeMath.sub will throw if there is not enough balance.
    balances[msg.sender] = balances[msg.sender].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(msg.sender, _to, _value);
    return true;
  }

  /**
  * @dev Gets the balance of the specified address.
  * @param _owner The address to query the the balance of.
  * @return An uint256 representing the amount owned by the passed address.
  */
  /*@CTK balanceOf
    @post balance == balances[_owner]
   */
  function balanceOf(address _owner) public view returns (uint256 balance) {
    return balances[_owner];
  }

}

// File: zeppelin-solidity/contracts/token/ERC20/BurnableToken.sol

/**
 * @title Burnable Token
 * @dev Token that can be irreversibly burned (destroyed).
 */
contract BurnableToken is BasicToken {

  event Burn(address indexed burner, uint256 value);

  /**
   * @dev Burns a specific amount of tokens.
   * @param _value The amount of token to be burned.
   */
  /*@CTK burn
    @tag assume_completion
    @post _value <= balances[msg.sender]
    @post __post.balances[msg.sender] == balances[msg.sender] - _value
    @post __post.totalSupply_ == totalSupply_ - _value
   */
  function burn(uint256 _value) public {
    require(_value <= balances[msg.sender]);
    // no need to require value <= totalSupply, since that would imply the
    // sender's balance is greater than the totalSupply, which *should* be an
        assertion failure

    address burner = msg.sender;
    balances[burner] = balances[burner].sub(_value);
    totalSupply_ = totalSupply_.sub(_value);
    Burn(burner, _value);
  }
}

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

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization
     control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;
  address public owner2;

  address private owner2_address = 0x615B255EEE9cdb8BF1FA7db3EE101106673E8DCB;

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

  /**
   * @dev The Ownable constructor sets the original `owner` of the contract to the
       sender
   * account.
   */
  /*@CTK Ownable
    @post __post.owner == msg.sender
    @post __post.owner2 == owner2_address
   */
  function Ownable() public {
    owner = msg.sender;
    owner2 = owner2_address;
  }

  /**
   * @dev Throws if called by any account other than the owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner || msg.sender == owner2);
    _;
  }

  modifier onlyOwner2() {
    require(msg.sender == owner2);
    _;
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  /*@CTK transferOwnership
    @tag assume_completion
    @post owner == msg.sender || owner2 == msg.sender
    @post newOwner != address(0)
    @post __post.owner == newOwner
   */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

  /**
   * @dev Allows the current owner to transfer control of the contract to a newOwner.
   * @param newOwner The address to transfer ownership to.
   */
  /*@CTK transferOwnership2
    @tag assume_completion
    @post owner2 == msg.sender
    @post newOwner != address(0)
    @post __post.owner2 == newOwner
   */
  function transferOwnership2(address newOwner) public onlyOwner2 {
    require(newOwner != address(0));
    OwnershipTransferred(owner2, newOwner);
    owner2 = newOwner;
  }

}

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

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
contract ERC20 is ERC20Basic {
  function allowance(address owner, address spender) public view returns (uint256);
  function transferFrom(address from, address to, uint256 value) public returns (bool)
      ;
  function approve(address spender, uint256 value) public returns (bool);
  event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: zeppelin-solidity/contracts/token/ERC20/StandardToken.sol

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * @dev https://github.com/ethereum/EIPs/issues/20
 * @dev Based on code by FirstBlood: https://github.com/Firstbloodio/token/blob/master
     /smart_contract/FirstBloodToken.sol
 */
contract StandardToken is ERC20, BasicToken {

```

```
mapping (address => mapping (address => uint256)) internal allowed;


/**
 * @dev Transfer tokens from one address to another
 * @param _from address The address which you want to send tokens from
 * @param _to address The address which you want to transfer to
 * @param _value uint256 the amount of tokens to be transferred
 */
/*@CTK transferFrom
  @tag assume_completion
  @pre _to != _from
  @post _to != address(0)
  @post _value <= balances[_from]
  @post _value <= allowed[_from][msg.sender]
  @post __post.balances[_from] == balances[_from] - _value
  @post __post.allowed[_from][msg.sender] == allowed[_from][msg.sender] - _value
  @post __post.balances[_to] == balances[_to] + _value
*/
function transferFrom(address _from, address _to, uint256 _value) public returns (
    bool) {
  require(_to != address(0));
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);

  balances[_from] = balances[_from].sub(_value);
  balances[_to] = balances[_to].add(_value);
  allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);
  Transfer(_from, _to, _value);
  return true;
}

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

/**
 * @dev Function to check the amount of tokens that an owner allowed to a spender.
 * @param _owner address The address which owns the funds.
```

```
   * @param _spender address The address which will spend the funds.
   * @return A uint256 specifying the amount of tokens still available for the spender
       .
   */
  /*@CTK allowance
    @post __return == allowed[_owner][_spender]
  */
  function allowance(address _owner, address _spender) public view returns (uint256) {
    return allowed[_owner][_spender];
  }

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

  /**
   * @dev Decrease the amount of tokens that an owner allowed to a spender.
   *
   * approve should be called when allowed[_spender] == 0. To decrement
   * allowed value is better to use this function to avoid 2 calls (and wait until
   * the first transaction is mined)
   * From MonolithDAO Token.sol
   * @param _spender The address which will spend the funds.
   * @param _subtractedValue The amount of tokens to decrease the allowance by.
   */
  /*@CTK decreaseApproval0
    @tag assume_completion
    @pre allowed[msg.sender][_spender] <= _subtractedValue
    @post __post.allowed[msg.sender][_spender] == 0
   */
  /*@CTK decreaseApproval
    @tag assume_completion
    @pre allowed[msg.sender][_spender] > _subtractedValue
    @post __post.allowed[msg.sender][_spender] == allowed[msg.sender][_spender] -
        _subtractedValue
   */
  function decreaseApproval(address _spender, uint _subtractedValue) public returns (
      bool) {
    uint oldValue = allowed[msg.sender][_spender];
    if (_subtractedValue > oldValue) {
      allowed[msg.sender][_spender] = 0;
    } else {
      allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
    }
    Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
    return true;
  }

}

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

/**
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
 * Based on code by TokenMarketNet: https://github.com/TokenMarketNet/ico/blob/master/
    contracts/MintableToken.sol
 */
contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;


  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  /*@CTK mint
    @tag assume_completion
    @post owner == msg.sender || owner2 == msg.sender
    @post !mintingFinished
    @post __post.totalSupply_ == totalSupply_ + _amount
    @post __post.balances[_to] == balances[_to] + _amount
   */
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool)
     {
    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    Mint(_to, _amount);
    Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  /*@CTK finishMinting
```

```
    @tag assume_completion
    @post owner == msg.sender || owner2 == msg.sender
    @post !mintingFinished
    @post __post.mintingFinished
   */
  function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    MintFinished();
    return true;
  }
}

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

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;


  /**
   * @dev Modifier to make a function callable only when the contract is not paused.
   */
  modifier whenNotPaused() {
    require(!paused);
    _;
  }

  /**
   * @dev Modifier to make a function callable only when the contract is paused.
   */
  modifier whenPaused() {
    require(paused);
    _;
  }

  /**
   * @dev called by the owner to pause, triggers stopped state
   */
  /*@CTK pause
    @tag assume_completion
    @post !paused
    @post owner == msg.sender || owner2 == msg.sender
    @post __post.paused
   */
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  /*@CTK pause
    @tag assume_completion
    @post paused
    @post owner == msg.sender || owner2 == msg.sender
    @post !__post.paused
   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    Unpause();
  }
}

// File: zeppelin-solidity/contracts/token/ERC20/PausableToken.sol

/**
 * @title Pausable token
 * @dev StandardToken modified with pausable transfers.
 **/
contract PausableToken is StandardToken, Pausable {

  function transfer(address _to, uint256 _value) public whenNotPaused returns (bool) {
    return super.transfer(_to, _value);
  }

  function transferFrom(address _from, address _to, uint256 _value) public
      whenNotPaused returns (bool) {
    return super.transferFrom(_from, _to, _value);
  }

  function approve(address _spender, uint256 _value) public whenNotPaused returns (
      bool) {
    return super.approve(_spender, _value);
  }

  function increaseApproval(address _spender, uint _addedValue) public whenNotPaused
      returns (bool success) {
    return super.increaseApproval(_spender, _addedValue);
  }

  function decreaseApproval(address _spender, uint _subtractedValue) public
      whenNotPaused returns (bool success) {
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}

// File: zeppelin-solidity/contracts/token/ERC20/SafeERC20.sol

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure.
 * To use this library you can add a ‘using SafeERC20 for ERC20;‘ statement to your
     contract,
 * which allows you to call the safe operations as ‘token.safeTransfer(...)‘, etc.
 */
library SafeERC20 {
  function safeTransfer(ERC20Basic token, address to, uint256 value) internal {
    assert(token.transfer(to, value));
  }

  function safeTransferFrom(ERC20 token, address from, address to, uint256 value)
      internal {
    assert(token.transferFrom(from, to, value));
  }

  function safeApprove(ERC20 token, address spender, uint256 value) internal {
    assert(token.approve(spender, value));
  }
}

// File: zeppelin-solidity/contracts/token/ERC20/TokenTimelock.sol

/**
 * @title TokenTimelock
 * @dev TokenTimelock is a token holder contract that will allow a
 * beneficiary to extract the tokens after a given release time
 */
contract TokenTimelock {
  using SafeERC20 for ERC20Basic;

  // ERC20 basic token contract being held
  ERC20Basic public token;

  // beneficiary of tokens after they are released
  address public beneficiary;

  // timestamp when token release is enabled
  uint256 public releaseTime;

  /*@CTK TokenTimelock
    @tag assume_completion
    @post _releaseTime > now
    @post __post.token == _token
    @post __post.beneficiary == _beneficiary
    @post __post.releaseTime == _releaseTime
   */
  function TokenTimelock(ERC20Basic _token, address _beneficiary, uint256 _releaseTime
      ) public {
    require(_releaseTime > now);
    token = _token;
    beneficiary = _beneficiary;
    releaseTime = _releaseTime;
  }

  /**
   * @notice Transfers tokens held by timelock to beneficiary.
   */
  function release() public {
    require(now >= releaseTime);

    uint256 amount = token.balanceOf(this);
    require(amount > 0);

    token.safeTransfer(beneficiary, amount);
  }
}

```

```
// File: contracts/PBKtoken.sol

contract PBKtoken is MintableToken, PausableToken, BurnableToken {
  string public name = "PlasmaBank token";
  string public symbol = "PBK";
  uint public decimals = 2;

  /// @dev whether an address is permitted to perform burn operations.
  mapping(address => bool) public isBurner;

  event ReceivedEther(address from, uint256 value);
  event WithdrewEther(address to, uint256 value);

  address PlasmaPrivateTokenSale        = 0xec0767B180C05B261A23744cCF8EB89b677dFeE1;
  address PlasmaPreTokenSaleReserve     = 0x2910dB084a467131C121626987b3F8b69ebaE82A;
  address PlasmaTokenSaleReserve        = 0x516154A8e9d365dC976f977E6815710b94B8C9f6;
  address PlasmaReserveForBonus         = 0x47e061914750f0Ee7C7675da0D62A59e2bd27dc4;
  address PlasmaReserveForBounty        = 0xdbf81Af07e37ec855653de1dB152E578d847f215;
  address PlasmaReserveForEarlyBirds    = 0x831360b8Dd93692d1A0Bdf7fdE8C037BaB1CE631;
  address PlasmaTeamOptionsReserveAddress = 0x04D20280B1E870688B7552E14171923215D3411C;
  address PlasmaFrozenForInstitutionalSales = 0x88bF0Ae762B801943190D1B7D757103BA9Dd6eAb;
  address PlasmaReserveForAdvisors      = 0x6Df994BdCA65f6bdAb66c72cd3fE3666cc183E37;
  address PlasmaFoundationReserve       = 0xF0dbBDb93344Bc679F8f0CffAE187D324917F44b;
  address PlasmaFrozenForTopManagement  = 0x5ed22d37BB1A16a15E9a2dD6F46b9C891164916B;
  address PlasmaFrozenForTokenSale2020  = 0x67F585f3EB7363E26744aA19E8f217D70e7E0001;

  function PBKtoken() public {
    mint(PlasmaPrivateTokenSale,        500000000 * (10 ** decimals));
    mint(PlasmaPreTokenSaleReserve,     300000000 * (10 ** decimals));
    mint(PlasmaTokenSaleReserve,        3200000000 * (10 ** decimals));
    mint(PlasmaReserveForBonus,         100000000 * (10 ** decimals));
    mint(PlasmaReserveForBounty,        100000000 * (10 ** decimals));
    mint(PlasmaReserveForEarlyBirds,    200000000 * (10 ** decimals));
    mint(PlasmaTeamOptionsReserveAddress, 800000000 * (10 ** decimals));
    mint(PlasmaFrozenForInstitutionalSales, 500000000 * (10 ** decimals));
    mint(PlasmaReserveForAdvisors,      300000000 * (10 ** decimals));
    mint(PlasmaFoundationReserve,       1000000000 * (10 ** decimals));
    mint(PlasmaFrozenForTopManagement,  1500000000 * (10 ** decimals));
    mint(PlasmaFrozenForTokenSale2020,  1500000000 * (10 ** decimals));

    assert(totalSupply_ == 10000000000 * (10 ** decimals));

    finishMinting();
  }

  function transferTimelocked(address _to, uint256 _amount, uint256 _releaseTime) public
    returns (TokenTimelock) {

    TokenTimelock timelock = new TokenTimelock(this, _to, _releaseTime);
    transferFrom(msg.sender, timelock, _amount);

    return timelock;
  }

  /**
```

```
     * @dev Grant or remove burn permissions. Only owner can do that!
     */
    /*@CTK grantBurner
      @tag assume_completion
      @post owner == msg.sender || owner2 == msg.sender
      @post __post.isBurner[_burner] == _value
     */
    function grantBurner(address _burner, bool _value) public onlyOwner {
        isBurner[_burner] = _value;
    }

    /**
     * @dev Throws if called by any account other than the burner.
     */
    modifier onlyBurner() {
        require(isBurner[msg.sender]);
        _;
    }

    /**
     * @dev Burns a specific amount of tokens.
     * Only an address listed in `isBurner` can do this.
     * @param _value The amount of token to be burned.
     */
    /*@CTK burn
      @tag assume_completion
      @post isBurner[msg.sender]
      @post _value <= balances[msg.sender]
      @post __post.balances[msg.sender] == balances[msg.sender] - _value
      @post __post.totalSupply_ == totalSupply_ - _value
     */
    function burn(uint256 _value) public onlyBurner {
        super.burn(_value);
    }

    // transfer balance to owner
    function withdrawEther(uint256 amount) public onlyOwner {
      owner.transfer(amount);
      WithdrewEther(msg.sender, amount);
    }

    // can accept ether
    function() payable private {
      ReceivedEther(msg.sender, msg.value);
    }
  }
```