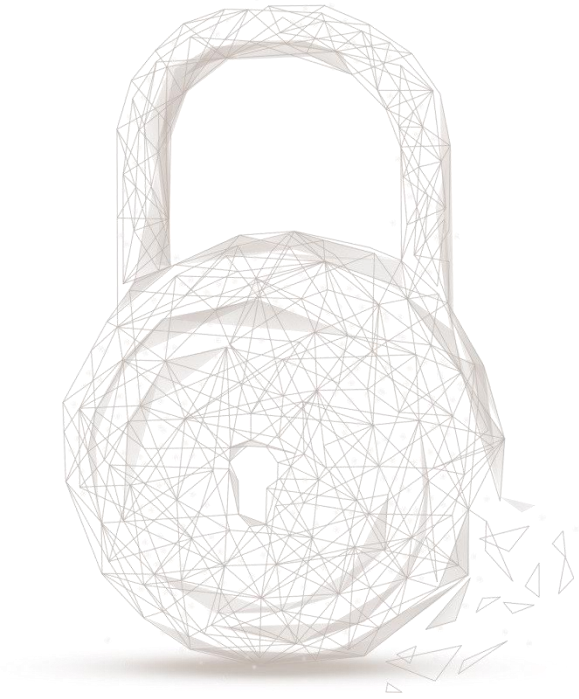
成都链安
BEOSIN

Smart contract security audit report



Audit Number：201908052016

Smart Contract Name：

Token: Token Prometeus Network (PROM)

Vault: PromTokenVault

Smart Contract Address：

Token: 0xfc82bb4ba86045Af6F327323a46E80412b91b27d

Vault: 0x3CDfA426726347a728b8e136bD33f914d3cc6d1c

Smart Contract Address Link：

Token: https://etherscan.io/address/0xfc82bb4ba86045af6f327323a46e80412b91b27d#code

Vault: https://etherscan.io/address/0x3cdfa426726347a728b8e136bd33f914d3cc6d1c#code

Start Date：2019.08.02

Completion Date：2019.08.05

Overall Result：Pass（Distinction）

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	ERC20 Token Standards	Pass
		Compiler Version Security	Pass
		Visibility Specifiers	Pass
		Gas Consumption	Pass
		SafeMath Features	Pass
		Fallback Usage	Pass
		tx.origin Usage	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		Overriding Variables	Pass
2	Function Call Audit	Authorization of Function Call	Pass

		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		selfdestruct Function Security	Pass
3	Business Security	Access Control of Owner	Pass
		Business Logics	Pass
		Business Implementations	Pass
4	Integer Overflow/Underflow	-	Pass
5	Reentrancy	-	Pass
6	Exceptional Reachable State	-	Pass
7	Transaction-Ordering Dependence	-	Pass
8	Block Properties Dependence	-	Pass
9	Pseudo-random Number Generator (PRNG)	-	Pass
10	DoS (Denial of Service)	-	Pass
11	Token Vesting Implementation	-	Pass
12	Fake Deposit	-	Pass
13	event security	-	Pass

Note: Audit results and suggestions in code comments

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of smart contract PROM and PromTokenVault, including Coding Standards, Security, and Business Logic. **PROM and PromTokenVault contract passed all audit items. The overall result is Pass (Distinction). The smart contract is able to function properly.** Please find below the basic information of the smart contract:

1、Basic Token Information

Token name	Token Prometeus Network
Token symbol	PROM
decimals	18
totalSupply	20 Million (Total Supply is constant)
Token type	ERC20

Table 1 – Basic Token Information

2、Token Vesting Information

The detailed token vesting information is shown in the Table 2 below:

Use	Query key	Locked Amount	Start Release Date	Release Amount
Liquidity Pool	0x2(0x30783200)	1,150,000 PROM	TGE (Jun-26-2019 01:53:17 PM +UTC).	1,064,997 PROM (the other 85003 PROM had been sent out)
Team & Advisors	0x3(0x30783300)	1,000,000 PROM	TGE+12 * MONTH	30,000PROM / MONTH
Company Reserve	0x4(0x30783400)	3,000,000 PROM	TGE+12 * MONTH	90,000 PROM / MONTH
Private Sale	0x5(0x30783500)	2,400,000 PROM	TGE+6 * MONTH	400,000 PROM / MONTH
Community / Minting	0x6(0x30783600)	9,000,000 PROM	TGE+6 * MONTH	9,000,000 PROM
Ecosystem	0x7(0x30783700)	1,000,000 PROM	TGE (Jun-26-2019 01:53:17 PM +UTC)	250,000 PROM /6 MONTH

Note: 1 MONTH = 2592000 s

Table 2 –Token Vesting Information

Audited Source Code with Comments:

```
// Beosin (Chengdu LianAn) // ### PROMToken ###
/**
 *Submitted for verification at Etherscan.io on 2019-05-23
*/

pragma solidity ^0.4.23; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
```

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

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256 c) {
    // Gas optimization: this is cheaper than asserting 'a' not being zero, but the
    // benefit is lost if 'b' is also tested.
    // See: https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
    if (a == 0) {
      return 0;
    }

    c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  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 a / b;
  }

  /**
  * @dev Subtracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
  */
  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    assert(b <= a);
    return a - b;
  }

  /**
  * @dev Adds two numbers, throws on overflow.
  */
  function add(uint256 a, uint256 b) internal pure returns (uint256 c) {
    c = a + b;
    assert(c >= a);
```

```
        return c;
    }
}

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

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
contract BasicToken is ERC20Basic {
    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    mapping(address => uint256) balances; // Beosin (Chengdu LianAn) // Declare the mapping variable 'balances' for storing the token balance of corresponding address.

    uint256 totalSupply_; // Beosin (Chengdu LianAn) // Declare the variable 'totalSupply_' for storing the total supply of token.

    /**
    * @dev Total number of tokens in existence
    */
    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.
    */
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0), "Address must not be zero."); // Beosin (Chengdu LianAn) // The non-zero
```

```
address check for '_to'. Avoid losing transferred tokens.
        require(_value <= balances[msg.sender], "There is no enough balance."); // Beosin (Chengdu LianAn) // The balance check, require that the transaction value should be no greater than the balance of 'msg.sender'.

        balances[msg.sender] = balances[msg.sender].sub(_value); // Beosin (Chengdu LianAn) // Alter the token balance of 'msg.sender'.
        balances[_to] = balances[_to].add(_value); // Beosin (Chengdu LianAn) // Alter the token balance of '_to'.
        emit Transfer(msg.sender, _to, _value); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
        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.
    */
    function balanceOf(address _owner) public view returns (uint256) {
        return balances[_owner];
    }

}

/**
 * @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); // Beosin (Chengdu LianAn) // Define the interface of 'allowance' function.

    function transferFrom(address from, address to, uint256 value)
        public returns (bool); // Beosin (Chengdu LianAn) // Define the interface of 'transferFrom' function.

    function approve(address spender, uint256 value) public returns (bool); // Beosin (Chengdu LianAn) // Define the interface of 'approve' function.
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    ); // Beosin (Chengdu LianAn) // Declare the event 'Approval'.
}

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/issues/20
 * 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; // Beosin (Chengdu LianAn) // Declare the mapping variable 'allowed' for storing the allowance between two addresses.


  /**
  * @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
  */
  function transferFrom(
    address _from,
    address _to,
    uint256 _value
  )
    public
    returns (bool)
  {
    require(_to != address(0), "Address must not be zero."); // Beosin (Chengdu LianAn) // The non-zero address check for '_to'. Avoid losing transferred tokens.
    require(_value <= balances[_from], "There is no enough balance."); // Beosin (Chengdu LianAn) // The balance check, require that the transaction value should be no greater than the balance of '_from'.
    require(_value <= allowed[_from][msg.sender], "There is no enough allowed balance."); // Beosin (Chengdu LianAn) // The allowance check, require that the transaction value should be no greater than the allowance which '_from' allowed to 'msg.sender'.

    balances[_from] = balances[_from].sub(_value); // Beosin (Chengdu LianAn) // Alter the token balance of '_from'.
    balances[_to] = balances[_to].add(_value); // Beosin (Chengdu LianAn) // Alter the token balance of '_to'.
    allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value); // Beosin (Chengdu LianAn) // Update the allowance between two addresses.
    emit Transfer(_from, _to, _value); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
    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.
  */
```

```
    // Beosin (Chengdu LianAn) // Using function 'increaseApproval' and 'decreaseApproval' to alter allowance is recommended.
    function approve(address _spender, uint256 _value) public returns (bool) {
        allowed[msg.sender][_spender] = _value; // Beosin (Chengdu LianAn) // The allowance which 'msg.sender' allowed to '_spender' is set to '_value'.
        emit Approval(msg.sender, _spender, _value); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        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.
    */
    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.
    */
    function increaseApproval(
        address _spender,
        uint256 _addedValue
    )
        public
        returns (bool)
    {
        allowed[msg.sender][_spender] = (
        allowed[msg.sender][_spender].add(_addedValue)); // Beosin (Chengdu LianAn) // Increase the allowance which 'msg.sender' allowed to '_spender', altering value is '_addedValue'.
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        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.
    */
    function decreaseApproval(
        address _spender,
        uint256 _subtractedValue
    )
        public
        returns (bool)
    {
        uint256 oldValue = allowed[msg.sender][_spender]; // Beosin (Chengdu LianAn) // Get the previous allowance.
        if (_subtractedValue > oldValue) {
            allowed[msg.sender][_spender] = 0; // Beosin (Chengdu LianAn) // If the '_subtractedValue' is greater than previous allowance, decrease allowance to 0 directly.
        } else {
            allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue); // Beosin (Chengdu LianAn) // Otherwise, decrease the allowance, altering value is '_subtractedValue '.
        }
        emit Approval(msg.sender, _spender, allowed[msg.sender][_spender]); // Beosin (Chengdu LianAn) // Trigger the event 'Approval'.
        return true;
    }

}

contract PROMToken is StandardToken {
    string public name = "Token Prometeus Network"; // Beosin (Chengdu LianAn) // The token name is 'Token Prometeus Network'.
    string public symbol = "PROM"; // Beosin (Chengdu LianAn) // The token symbol is 'PROM'.
    uint8 public decimals = 18; // Beosin (Chengdu LianAn) // The token decimals is 18.
    uint256 public INITIAL_SUPPLY = 20000000  * 10 ** uint256(decimals); // Beosin (Chengdu LianAn) // The initial supply of token is 20000000.
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic information of token.
    constructor() public {
        totalSupply_ = INITIAL_SUPPLY;
        balances[msg.sender] = totalSupply_; // Beosin (Chengdu LianAn) // Send all tokens to the deployer of this contract.
        emit Transfer(0x0, msg.sender, INITIAL_SUPPLY); // Beosin (Chengdu LianAn) // Trigger the event 'Transfer'.
```

```
    }
}


// Beosin (Chengdu LianAn) // ### PromTokenVault ###


/**
 *Submitted for verification at Etherscan.io on 2019-08-01
*/

pragma solidity ^0.4.25; // Beosin (Chengdu LianAn) // Fixing compiler version is recommended.
/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {
    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;
    }

    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;
    }

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

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

/**
 * @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; // Beosin (Chengdu LianAn) // Declare variable 'owner' for storing the address of contract owner.


    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner); // Beosin (Chengdu LianAn) // Declare the event 'OwnershipTransferred'.


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

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

    /**
    * @dev Allows the current owner to transfer control of the contract to a newOwner.
    * @param newOwner The address to transfer ownership to.
    */
    function transferOwnership(address newOwner) public onlyOwner {
        require(newOwner != address(0)); // Beosin (Chengdu LianAn) // The non-zero address check for 'newOwner'. Avoid losing ownership.
        emit OwnershipTransferred(owner, newOwner); // Beosin (Chengdu LianAn) // Trigger the event 'OwnershipTransferred'.
        owner = newOwner; // Beosin (Chengdu LianAn) // Change the ownership to 'newOwner'.
    }
}

/**
 * @title ERC20Basic
 */
contract ERC20Basic {
    uint256 public totalSupply; // Beosin (Chengdu LianAn) // Declare the variable 'totalSupply' for storing the total supply of token.
    function balanceOf(address who) public view returns (uint256); // Beosin (Chengdu LianAn) // Define the interface of 'balanceOf' function.
```

```
    function transfer(address to, uint256 value) public returns (bool); // Beosin (Chengdu LianAn) // Define the interface of 'transfer' function.
    event Transfer(address indexed from, address indexed to, uint256 value); // Beosin (Chengdu LianAn) // Declare the event 'Transfer'.
}

/**
 * @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); // Beosin (Chengdu LianAn) // Define the interface of 'allowance' function.
    function transferFrom(address from, address to, uint256 value) public returns (bool); // Beosin (Chengdu LianAn) // Define the interface of 'transferFrom' function.
    function approve(address spender, uint256 value) public returns (bool); // Beosin (Chengdu LianAn) // Define the interface of 'approve' function.
    event Approval(address indexed owner, address indexed spender, uint256 value); // Beosin (Chengdu LianAn) // Declare the event 'Approval'.
}

/**
 * @title Basic token
 * @dev Basic version of StandardToken, with no allowances.
 */
// Beosin (Chengdu LianAn) // Note: This contract is not inherited by any contracts, it is recommended to delete it.
contract BasicToken is ERC20Basic, Ownable {

    using SafeMath for uint256;

    mapping(address => uint256) balances;

    /**
    * @dev transfer token for a specified address
    * @param _to The address to transfer to.
    * @param _value The amount to be transferred.
    */
    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);
        emit 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.
    */
    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }

}


/** Smart contract for controlling of where and when tokens are distributed duting crowdsale
 * Stores addresses of wallets which will receive tokens according to different parts of crowdsale for furthet distribution
 * Also it controls when certain number of tokens is unfrozen
 * */
contract PromTokenVault is Ownable{

    using SafeMath for uint256; // Beosin (Chengdu LianAn) // Use the SafeMath library for mathematical operation. Avoid integer overflow/underflow.

    // amount of seconds in 1 month
    // uint256 MONTH = 3;
    uint256 MONTH = 2592000;
    // address of token smart contract
    address token_;
    // Beosin (Chengdu LianAn) // Variables for storing the flag of addresses. Note: In fact, each value of these flags(string type) is auto-transformed and stored as bytes4 type, when using the function interface 'getAlreadyWithdrawn' to query something, the input value is different with the value shown in code. E.g. "0x4" is stored as 0x30783400 actually.
    bytes4 publicKey = "0x1"; // Beosin (Chengdu LianAn) // Note: Unused variable, it is recommended to delete.
    bytes4 liquidityKey = "0x2";
    bytes4 teamKey = "0x3";
    bytes4 companyKey = "0x4";
    bytes4 privateKey = "0x5";
    bytes4 communityKey = "0x6";
    bytes4 ecosystemKey = "0x7";

    // these are addresses which will receive tokens according to respective parts of crowdsale
    address liquidity_;
    address team_;
    address company_;
    address private_;
    address community_;
    address ecosystem_;
```

```
    // stores data on the amount of tokens released according to different parts of crowdsale
    mapping (bytes4=>uint256) alreadyWithdrawn;

    // time when Token Generation Event occurs
    uint256 TGE_timestamp;
    ERC20 token; // Beosin (Chengdu LianAn) // Create an instance of token contract.
    // Beosin (Chengdu LianAn) // Constructor, initialize the basic information of the vault contract.
    constructor(
            address _token,
            address _private,
            address _ecosystem,
            address _liquidity,
            address _team,
            address _company,
            address _community
            ) public {
        token = ERC20(_token);
        private_ = _private;
        ecosystem_ = _ecosystem;
        liquidity_ = _liquidity;
        team_ = _team;
        company_ = _company;
        community_ = _community;
        TGE_timestamp = block.timestamp; // Beosin (Chengdu LianAn) // Set the deployment time as the value
of 'TGE_timestamp'.
    }

    /**
     * @dev Get the addresses to which tokens intended for respective part tokensale can be transferred;
    */
    function getLiqudityAddress() public view returns(address){
        return liquidity_;
    }

    function getTeamAddress() public view returns(address){
        return team_;
    }

    function getCompanyAddress() public view returns(address){
        return company_;
    }

    function getPrivateAddress() public view returns(address){
        return private_;
    }

    function getCommunityAddress() public view returns(address){
```

```
    return community_;
  }

  function getEcosystemAddress() public view returns(address){
    return ecosystem_;
  }

  /**
   * @dev Set the address to which tokens intended for part of tokensale can be transferred; Usable only by owner of smart contract;
  */
  function setLiquidityAddress(address _liquidity) public onlyOwner{
    liquidity_ = _liquidity;
  }

  function setTeamAddress(address _team) public onlyOwner{
    team_ = _team;
  }

  function setCompanyAddress(address _company) public onlyOwner{
    company_ = _company;
  }

  function setPrivateAddress(address _private) public onlyOwner{
    private_ = _private;
  }

  function setCommunityAddress(address _community) public onlyOwner{
    community_ = _community;
  }
  function setEcosystemAddress(address _ecosystem) public onlyOwner{
    ecosystem_ = _ecosystem;
  }


  /**
   * @dev Get how many tokens are still available for releaseal according to emmision rules of public part of tokensale
  */
  function getLiquidityAvailable() public view returns(uint256){
    return getLiquidityReleasable().sub(alreadyWithdrawn[liquidityKey]);
  }
  function getTeamAvailable() public view returns(uint256){
    return getTeamReleasable().sub(alreadyWithdrawn[teamKey]);
  }
  function getCompanyAvailable() public view returns(uint256){
    return getCompanyReleasable().sub(alreadyWithdrawn[companyKey]);
  }
```

```
    function getPrivateAvailable() public view returns(uint256){
        return getPrivateReleasable().sub(alreadyWithdrawn[privateKey]);
    }
    function getCommunityAvailable() public view returns(uint256){
        return getCommunityReleasable().sub(alreadyWithdrawn[communityKey]);
    }
    function getEcosystemAvailable() public view returns(uint256){
        return getEcosystemReleasable().sub(alreadyWithdrawn[ecosystemKey]);
    }

    /**
     * @dev Get the total amount of tokens emitted which are intended for respective part of crowdsale
    */
    function getPercentReleasable(uint256 _part, uint256 _full) internal pure returns(uint256){
        if(_part >= _full){
            _part = _full; // Beosin (Chengdu LianAn) // If '_part' is no less than '_full', set '_part' to '_full'.
        }
        return _part;
    }
    // Beosin (Chengdu LianAn) // Internal function 'getMonthsPassed' for calculating the counts of how many months passed since a specified timestamp.
    function getMonthsPassed(uint256 _since) internal view returns(uint256){
        return (block.timestamp.sub(_since)).div(MONTH);
    }

    // LIQUIDITY TOKENSALE; 5.75% for liquidity sale; available after TGE
    function getLiquidityReleasable() public view returns(uint256){
        if(block.timestamp >= TGE_timestamp){
            return token.totalSupply().div(10000).mul(575) - (85000 + 3) * 10 ** uint256(18); // Beosin (Chengdu LianAn) // Note: This part of tokens (85003 PROM) had been in circulation. Therefore, it is subtracted here. Although there is no overflow, using SafeMath is still recommended.
        }else{
            return 0; // Beosin (Chengdu LianAn) // Note: Theoretically, it is impossible because 'block.timestamp' is greater than 'TGE_timestamp' constantly. It is recommended to delete it.
        }
    }

    // TEAM & ADVISORS; 5% for team & advisors; 100% are locked for 12 month, then vesting 3% per month
    function getTeamReleasable() public view returns(uint256){
        uint256 unlockDate = TGE_timestamp.add(MONTH.mul(12)); // Beosin (Chengdu LianAn) // Declare the local variable 'unlockDate' for getting the unlock date of locked tokens which used for TEAM & ADVISORS. Lock for 12 months.
        if(block.timestamp >= unlockDate){ // Beosin (Chengdu LianAn) // Do contents below if the current time is later than 'unlockDate'.
            uint256 totalReleasable = token.totalSupply().div(100).mul(5); // Beosin (Chengdu LianAn) // Declare the local variable 'totalReleasable' for recording the total locked amount of tokens.
            uint256 monthPassed = getMonthsPassed(unlockDate)+1; // Beosin (Chengdu LianAn) // Declare the local variable 'monthPassed' for getting the count(s) of past months since 'unlockDate'.
```

```
        return totalReleasable.div(100).mul(getPercentReleasable((monthPassed.mul(3)),100)); // Beosin (Chengdu LianAn) // Return the corresponding amount of releasable tokens.
      }else{
        return 0; // Beosin (Chengdu LianAn) // Otherwise, return zero.
      }
    }

    // TODO: unlocks after 13 month, not 12
    // COMPANY; 15% for company reserve; 100% are locked for 12 month, then vesting 3% per month
    function getCompanyReleasable() public view returns(uint256){
      uint256 unlockDate = TGE_timestamp.add(MONTH.mul(12)); // Beosin (Chengdu LianAn) // Declare the local variable 'unlockDate' for getting the unlock date of locked tokens which used for COMPANY. Lock for 12 months.
      if(now >= unlockDate){ // Beosin (Chengdu LianAn) // Do contents below if the current time is later than 'unlockDate'.
        uint256 totalReleasable = token.totalSupply().div(100).mul(15); // Beosin (Chengdu LianAn) // Declare the local variable 'totalReleasable' for recording the total locked amount of tokens.
        uint256 monthPassed = getMonthsPassed(unlockDate)+1; // Beosin (Chengdu LianAn) // Declare the local variable 'monthPassed' for getting the count(s) of past months since 'unlockDate'.
        return totalReleasable.div(100).mul(getPercentReleasable(monthPassed.mul(3),100)); // Beosin (Chengdu LianAn) // Return the corresponding amount of releasable tokens.
      }else{
        return 0; // Beosin (Chengdu LianAn) // Otherwise, return zero.
      }
    }

    //PRIVATE SALE; 20% for private sale; 40% at listing, 60% of tokens are locked for 6 month, then vested by 10% per month
    function getPrivateReleasable() public view returns(uint256){
      uint256 totalReleasable = token.totalSupply().div(100).mul(20); // Beosin (Chengdu LianAn) // Declare the local variable 'totalReleasable' for recording the total locked amount of tokens.
      uint256 firstPart = totalReleasable.div(100).mul(40); // Beosin (Chengdu LianAn) // Declare the local variable 'firstPart' for getting the amount of tokens which at listing(unlocked).
      uint256 currentlyReleasable = firstPart; // Beosin (Chengdu LianAn) // Declare the local variable 'currentlyReleasable' whose value is set to 'firstPart'.
      uint256 unlockDate = TGE_timestamp.add(MONTH.mul(6)); // Beosin (Chengdu LianAn) // Declare the local variable 'unlockDate' for getting the unlock date of locked tokens which used for PRIVATE SALE. Lock for 6 months.

      if(now >= unlockDate){ // Beosin (Chengdu LianAn) // Do contents below if the current time is later than 'unlockDate'.
        uint256 monthPassed = getMonthsPassed(unlockDate)+1; // Beosin (Chengdu LianAn) // Declare the local variable 'monthPassed' for getting the count(s) of past months since 'unlockDate'.
        uint256 secondPart = totalReleasable.div(100).mul(getPercentReleasable(monthPassed.mul(10),60)); // Beosin (Chengdu LianAn) // Declare the local variable 'secondPart' for getting the amount of tokens which need to be released.
        currentlyReleasable = firstPart.add(secondPart); // Beosin (Chengdu LianAn) // Calculate the amount of releasable tokens after 'unlockDate'.
```

```
        }
        return currentlyReleasable; // Beosin (Chengdu LianAn) // Return the corresponding amount of releasable tokens.
    }

    // COMMUNITY; 45% for community minting; frozen for 6 month
    function getCommunityReleasable() public view returns(uint256){
        uint256 unfreezeTimestamp = TGE_timestamp.add(MONTH.mul(6)); // Beosin (Chengdu LianAn) // Declare the local variable 'unfreezeTimestamp' for getting the unfreeze date of frozen tokens which used for COMMUNITY. Frozen for 6 months.
        if(now >= unfreezeTimestamp){ // Beosin (Chengdu LianAn) // Do contents below if the current time is later than 'unfreezeTimestamp'.
            return token.totalSupply().div(100).mul(45); // Beosin (Chengdu LianAn) // Return the corresponding amount of releasable tokens.
        }else{
            return 0; // Beosin (Chengdu LianAn) // Otherwise, return zero.
        }
    }

    // ECOSYSTEM; 5% for ecosystem; 25% after TGE, then 25% per 6 month
    function getEcosystemReleasable() public view returns(uint256){
        uint256 currentlyReleasable = 0;
        if(block.timestamp >= TGE_timestamp){ // Beosin (Chengdu LianAn) // Do contents below if the current time is later than 'TGE_timestamp'.
            uint256 totalReleasable = token.totalSupply().div(100).mul(5); // Beosin (Chengdu LianAn) // Declare the local variable 'totalReleasable' for recording the total amount of releasable tokens.
            uint256 firstPart = totalReleasable.div(100).mul(25); // Beosin (Chengdu LianAn) // Declare the local variable 'firstPart' for getting the amount of tokens which is unlocked (available after 'TGE_timestamp').
            uint256 monthPassed = getMonthsPassed(TGE_timestamp); // Beosin (Chengdu LianAn) // Declare the local variable 'monthPassed' for getting the count(s) of past months since 'TGE_timestamp'.
            uint256 releases = monthPassed.div(6); // Beosin (Chengdu LianAn) // Declare the local variable 'releases' for getting the count(s) of release times.
            uint256 secondPart = totalReleasable.div(100).mul(getPercentReleasable(releases.mul(25),75)); // Beosin (Chengdu LianAn) // Declare the local variable 'secondPart' for getting the amount of tokens which need to be released.
            currentlyReleasable = firstPart.add(secondPart); // Beosin (Chengdu LianAn) // Calculate the amount of releasable tokens according to the past months.
        }
        return currentlyReleasable; // Beosin (Chengdu LianAn) // Return the corresponding amount of releasable tokens.
    }
    // Beosin (Chengdu LianAn) // Internal function 'incrementReleased' for recording the number of released tokens of corresponding '_key' match to a specified address.
    function incrementReleased(bytes4 _key, uint256 _amount) internal{
        alreadyWithdrawn[_key]=alreadyWithdrawn[_key].add(_amount);
    }
    /**
     * @dev Transfer all tokens intended for respective part of crowdsale to the wallet which will be distribute these
```

```
tokens
    */
```

// Beosin (Chengdu LianAn) // The functions below are used for release tokens to the corresponding addresses.

// Beosin (Chengdu LianAn) // The logic of functions below are following steps here: 1). Check whether the vault contract has sufficient token balance (greater than available); 2). Get the amount of tokens to release; 3). Call function 'incrementReleased' to record the released amount of corresponding key; 4). Transfer tokens and check return value.

```
function releaseLiqudity() public{
    require(token.balanceOf(address(this))>=getLiquidityAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getLiquidityAvailable();
    incrementReleased(liquidityKey,toSend);
    require(token.transfer(liquidity_, toSend),'Token Transfer returned false');
}
function releaseTeam() public{
    require(token.balanceOf(address(this))>=getTeamAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getTeamAvailable();
    incrementReleased(teamKey,toSend);
    require(token.transfer(team_, toSend),'Token Transfer returned false');
}
function releaseCompany() public{
    require(token.balanceOf(address(this))>=getCompanyAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getCompanyAvailable();
    incrementReleased(companyKey,toSend);
    require(token.transfer(company_, toSend),'Token Transfer returned false');
}
function releasePrivate() public{
    require(token.balanceOf(address(this))>=getPrivateAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getPrivateAvailable();
    incrementReleased(privateKey,toSend);
    require(token.transfer(private_, toSend),'Token Transfer returned false');
}
function releaseCommunity() public{
    require(token.balanceOf(address(this))>=getCommunityAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getCommunityAvailable();
    incrementReleased(communityKey,toSend);
    require(token.transfer(community_, toSend),'Token Transfer returned false');
}
function releaseEcosystem() public{
    require(token.balanceOf(address(this))>=getEcosystemAvailable(),'Vault does not have enough tokens');
    uint256 toSend = getEcosystemAvailable();
    incrementReleased(ecosystemKey,toSend);
    require(token.transfer(ecosystem_, toSend),'Token Transfer returned false');
}
```

// Beosin (Chengdu LianAn) // Function 'getAlreadyWithdrawn' for querying the released token amount according to a specified key. Note: Input value is different with the value shown in code. E.g. "0x4" is stored as 0x30783400 actually.

```
function getAlreadyWithdrawn(bytes4 _key) public view returns(uint256){
```

```
        return alreadyWithdrawn[_key];
    }
    // function testReduceTGEByMonth() public{
    //     TGE_timestamp=TGE_timestamp-MONTH;
    // }
    // function testIncreaseTGEByMonth() public{
    //     TGE_timestamp=TGE_timestamp+MONTH;
    // }

    // function testGetTGE() public view returns(uint256){
    //     return TGE_timestamp;
    // }
    // function testGetTimespan() public view returns(uint256){
    //     return getMonthsPassed(TGE_timestamp);
    // }
}
```

成都链安

BEOSIN

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/LianAnTech