Sikke Creator

Abstract

The meaning of the word sikke in old Turkish is equivalent to the word coin, but in modern Turkish, this word creates some problems, for example, jokes such as "paying sikke sikke, either now or later, sikke sikke" advertise this word, while attracting attention from 250 million Turkish speaking people around the world. is a statement. The coin is not only the Turkish money, but the general name of all the coins of that time ("Byzantine coin", "Seljuk coin").

In this project, a dapp has been designed where everyone can generate their own coin using smart contracts. The solidty code used is public and the generated coin belongs to its owner with all its transparency. The SikkeCompany does not make any claims on the coins and contract you create.

The ICO of this project will be carried out on the token called Sikke(Sikkoin(0xdab2733ac19a8d827b48760A5cC18331AD2A810F)(BSC)).

Idea

For a non-technical person or a company to launch their own ico, a functional code that is transparent and readable is required. You can find this code only as bytecode using current token creators, but this code is not safe and sufficient for the ico owner because markets and scanlers want the plain code from the ico owner. In this project, it is aimed to solve this problem.

Method

A contract deployer was designed using the Web3 library. It is first sent to the solccomp service by changing the necessary variables in example_code. Then the contract is signed, gas is paid and the block is processed. After the block is processed, all created tokens are identified to the address of the signing account. The system gives the generated smart contract address and codes to the owner. The owner can use these codes and generated coins on any platform using them. Example Code is coded on ERC20 standards. The SikkeCompany undertakes not to make any copyright or claim on example_code.

example_code

```
contract @contractname {
    string public constant name = "@name";
    string public constant symbol = "@symbol";
    uint8 public constant decimals = @decimals;
    string public asd="";

    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
    event Transfer(address indexed from, address indexed to, uint tokens);
    mapping(address => uint256) balances;
    mapping(address => mapping (address => uint256)) allowed;
    uint256 totalSupply_;
    using SafeMath for uint256;

   constructor() public {
       totalSupply_ = @total;
       balances[msg.sender] = totalSupply_;
    }

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

    function balanceOf(address tokenOwner) public view returns (uint) {
        return balances[tokenOwner];
    }
    function transfer(address receiver, uint numTokens) public returns (bool) {
        require(numTokens <= balances[msg.sender]);
        balances[msg.sender] = balances[msg.sender].sub(numTokens);
        balances[receiver] = balances[receiver].add(numTokens);
        emit Transfer(msg.sender, receiver, numTokens);
        return true;
    }
    function approve(address delegate, uint numTokens) public returns (bool) {
        allowed[msg.sender][delegate] = numTokens;
        emit Approval(msg.sender, delegate, numTokens);
        return true;
    }
    function allowance(address owner, address delegate) public view returns (uint) {
        return allowed[owner][delegate];
    }
    function transferFrom(address owner, address buyer, uint numTokens) public returns
(bool) {
        require(numTokens <= balances[owner]);
        require(numTokens <= allowed[owner][msg.sender]);
        balances[owner] = balances[owner].sub(numTokens);
        allowed[owner][msg.sender] = allowed[owner][msg.sender].sub(numTokens);
        balances[buyer] = balances[buyer].add(numTokens);
        emit Transfer(owner, buyer, numTokens);
        return true;
    }
}
library SafeMath {
    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;
    }
}
```

ICO

The ICO of this project will be carried out on the token called Sikke(Sikkoin).

The ICO of this project will take place in two phases. In the first stage, 100 million of the total 513 million coins will be distributed, the first stage will take place between October 1-31. The second stage will be held between January 1st and January 31st, and in the second stage, 100 million will be distributed.

All rights of 13 million coins belong to Ömer Fatih AĞİN. The SikkeCompany can burn 100 million Coins at any time, but undertakes not to sell or change hands until 1 June 2023 and to remain at 0x379faF2bB1C222077e1A7892d6aA2f6D5a70EcB5. The SikkeCompany undertakes that it will not sell the remaining 200 million Sikkoin and will only burn it on the dates to be announced.

The SikkeCompany can sell its own 100 million coins at any time after July 1, 2022. Before this date it can only burn.

The SikkeCompany undertakes to present 2*10^-6 times the amount of SikkoinV2 to its current coin shareholders for the SikkoinV2 project, which it will launch on January 1, 2023, for each sale of 1 coin of its own. if he sells it, he will have given a total of 400 million SikkoinV2 gifts to all shareholders in the sale process.)

The SikkeCompany reserves all rights of Ico.

License