OST
Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

Ethereum and Solidity

Thomas Bocek

07.10.2021

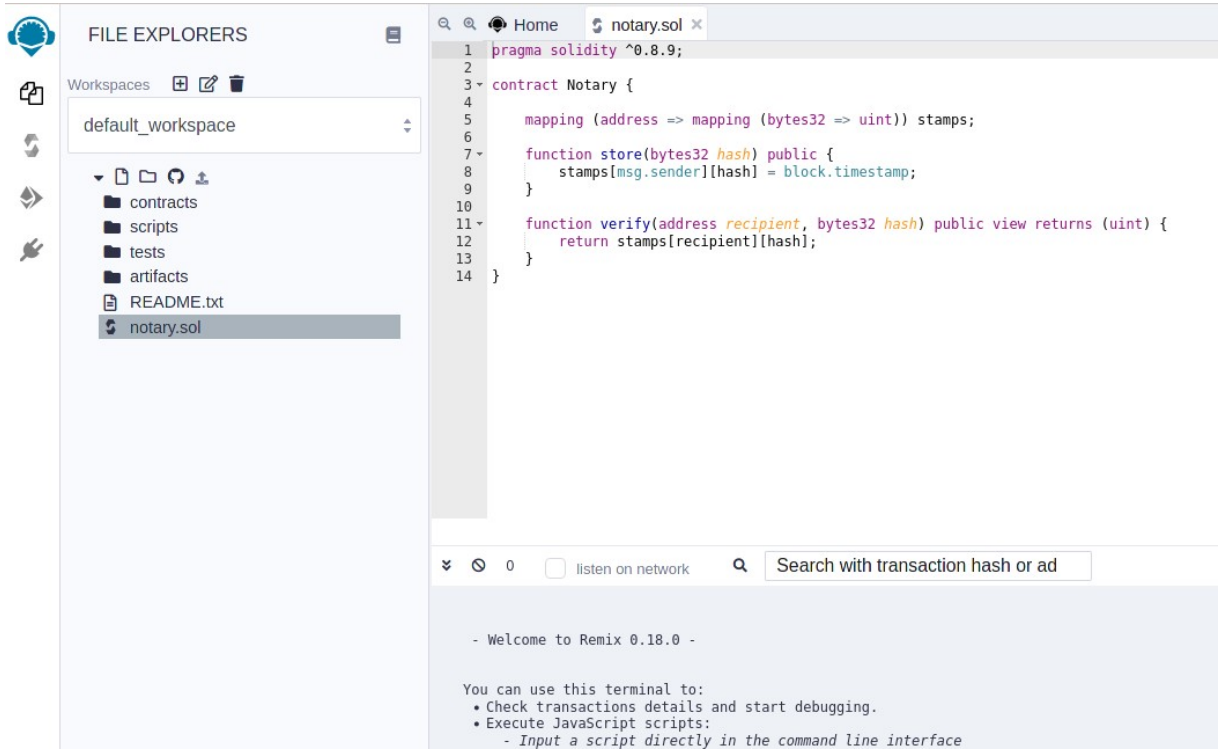
ERC-20

- ERC-20
  - Token contract keeps track of fungible tokens
  - Can be used as vault for NFTs
- Optional (highly recommended)
  - `function name() public view returns (string)`
  - `function symbol() public view returns (string)`
  - `function decimals() public view returns (uint8)`
- Events
  - `event Transfer(address indexed _from, address indexed _to, uint256 _value)`
  - `event Approval(address indexed _owner, address indexed _spender, uint256 _value)`
- Mandatory
  - `function totalSupply() public view returns (uint256)`
  - `function balanceOf(address _owner) public view returns (uint256 balance)`
  - `function transfer(address _to, uint256 _value) public returns (bool success)`
  - `function transferFrom(address _from, address _to, uint256 _value) public returns (bool success)`
  - `function approve(address _spender, uint256 _value) public returns (bool success)`
  - `function allowance(address _owner, address _spender) public view returns (uint256 remaining)`

OST

ERC-20 Implementation

- OpenZeppelin – many default contracts, very good source
  - Can be referenced
  - Let’s implement a simple ERC20 and deploy on testnet
    - Remix IDE



OST

ERC20 Dividends

- Loop over accounts does not work
- TotalDrop always increasing
  - Every account knows if bonus payed out
  - Call updateAccount() on every transfer
- User claims bonus

```
function claimBonus() public payable {
  Account storage account = updateAccount(msg.sender);
  uint256 sendValue = account.bonusWei;
  account.bonusWei = 0;
  msg.sender.transfer(sendValue);
}
```

```
uint256 public totalDrop = 0;
uint256 public rounding = 0;
struct Account {
  uint256 lastAirdropWei;
  uint256 bonusWei;
  uint256 valueToken;
}
mapping(address => Account) public accounts;

…

function() public payable {
  uint256 value = msg.value + rounding;
  rounding = value % totalSupply;
  uint256 weiPerToken = (value - rounding) / totalSupply;
  totalDrop += weiPerToken;
}

…

function updateAccount(address _addr) internal {
  Account storage account = accounts[_addr];
  uint256 weiPerToken = totalDrop - account.lastAirdropWei;
  if(weiPerToken != 0) {
    account.bonusWei += weiPerToken * account.valueToken;
    account.lastAirdropWei = totalDrop;
  }
}
```


OST

ERC20 – Considerations - Minting

- Transfer ownership (Ownable.sol)
  - Important for minting: after minting, set new owner so that the private key is never exposed
  - Private key never on the minting machine
- Token lockups
  - Vest tokens – make sure big investor don't dump
  - Team vesting – show the world that you believe in your token
- Minting
  - Lock contract during minting

```
function transferOwnership(address _newOwner) public {
        require(owner == msg.sender);
        owner = _newOwner;
}

mapping(address => uint256) lockups;

…

if (lockups[msg.sender] != 0) {
            require(now >= lockups[msg.sender]);
}

bool public mintingDone = false;

…

require(mintingDone == true);
```


OST

ERC20 - Considerations

- Utility Token – ERC 223 / 677
  - ERC 677 fully backwards compatible, but tokens still can be lost
    - One call instead approve/transferFrom
  - Eliminates the problem of lost tokens
    - QTUM, $1,204,273 lost
    - EOS, $1,015,131 lost
  - Allows developers to handle incoming token transactions
  - Backwards compatible?
    - ERC 223: throw if transferring to a contract that does not implement tokenFallback… (corner cases)
  - ERC 777: Register to get notified on tokensReceived()

```
contract ERC677 {
    event Transfer(address indexed _from, address indexed _to, uint256
_value, bytes _data);
    function transferAndCall(address _to, uint _value, bytes _data)
public returns (bool success);
}

contract ERC677Receiver {
    function tokenFallback(address _from, uint _value, bytes _data)
public;
}

// ERC677 functionality
function transferAndCall(address _to, uint _value, bytes _data) public
returns (bool) {
require(transfer(_to, _value));
    …
   if (isContract(_to)) {
            ERC677Receiver receiver = ERC677Receiver(_to);
            receiver.tokenFallback(msg.sender, _value, _data);
}
//ERC223
function transfer(address _to, uint _value) public returns (bool
success) {
    bytes memory empty;
    if(isContract(_to)) {
        return transferToContract(_to, _value, empty);
    }
    else {
        return transferToAddress(_to, _value, empty);
    }
}
```


OST

Contract - Considerations

- Minting done in batches
  - Around 120/170 until max gas used
- Only owner can mint
  - Once minting finished no one can ever mint
- Check maxSupply
  - Don’t mint more tokens, check important, as minter needs to respect limits
- Emit from:0
  - Discussing if form is 0 or contract address

```
function mint(address[] _recipients, uint256[] _amounts) public {
        require(owner == msg.sender);
        require(mintingDone == false);
        require(_recipients.length == _amounts.length);
        require(_recipients.length <= 256);

        for (uint8 i = 0; i < _recipients.length; i++) {
            address recipient = _recipients[i];
            uint256 amount = _amounts[i];

            // enforce maximum token supply
            require(totalSupply_.add(amount) <= maxSupply);

            balances[recipient] = balances[recipient].add(amount);
            totalSupply_ = totalSupply_.add(amount);

            emit Transfer(0, recipient, amount);
        }
}
```


OST

Contract - Considerations

- ERC865: Pay transfers in tokens instead of gas, in one transaction
  - Delegate transfer of tokens to a third party
  - UX: pay service in tokens – user needs to have ETH and tokens
    - Create account at exchange, KYC, Bank transfer – and wait
- Off-chain service to get signature from 3rd party service
- Use signature to check if transfer tokens allowed
- But, ERC677 – add delegatedTransferAndCall()

```
function delegatedTransfer(
     uint256 _nonce,
     address _from,
     address _to,
     uint256 _value,
     uint256 _fee,
     uint8 _v,
     bytes32 _r,
     bytes32 _s) public returns (bool) {

uint256 total = _value.add(_fee);
require(_from != address(0));
require(_to != address(0));
require(total <= balances[_from]);
require(_nonce > nonces[_from]);

address delegate = msg.sender;
address token = address(this);
bytes32 delegatedTxnHash = keccak256(delegate, token, _nonce,
_from, _to, _value, _fee);
address signatory = ecrecover(delegatedTxnHash, _v, _r, _s);
require(signatory == _from);

balances[_from] = balances[_from].sub(total);
balances[_to] = balances[_to].add(_value);
balances[delegate] = balances[delegate].add(_fee);
nonces[_from] = _nonce;

DelegatedTransfer(_from, _to, delegate, value, fee);
return true;
}
```


OST

Random Numbers

- There is no random number in Ethereum
- Every EVM needs to come to the same result
- Random Numbers from Oracle (External source)
- Commitment schemes, solidity
- Coinflip
  - Step 4) here you can try to cheat!

1) Alice flips coin, adds it to a random number, e.g., tail#randomnumber1234
   hashes it:
   commitment = sha256(tail#randomnumber1234)
2) Alice sends the commitment to Bob and tells bob to flip the coin
3) Bob flips coin and sends head to Alice
4) Alice reveals the random number, Bob can verify that the commitment was tail
5) Both same, Alice wins, both different, Bob wins
6) Alice: tail, Bob: head → Bob wins

OST

Random Numbers

- Use future blockhash
- Only up to 256 blockhashes from the past can be accessed.
- Deduct / add tokens/ETH from the past address
- Miner can influence the random value in a sense
- Value needs to be low (miner gets 2 ETH)

```
function transfer(address _to, uint256 _value) public returns (bool) {
    …
        //since this is a lucky coin, the value transferred is not what you expect
        luckyTransfer();
        previousTransferBlockNr = block.number;
        previousTransferAddress = msg.sender;
        …
        uint256 val -= potIncrease;
        pot += potIncrease;
}

function luckyTransfer() private {
        if(block.number != previousTransferBlockNr &&
          (block.number - previousTransferBlockNr) < 256 ) {
            uint256 rnd = uint256(keccak256(block.blockhash(previousTransferBlockNr)));
            if(rnd % 200 == 0) { //.5%
                balances[previousTransferAddress] += pot;
                Emit Transfer(this, previousTransferAddress, pot);
                //tokens are from pot, thus no tokens are created from thin air
                pot = 0;
                previousTransferBlockNr = 0;
                previousTransferAddress = 0;
            }
        }
}
```


OST