Create a .sol file on REMIX with:

```
pragma solidity ^0.4.18;

// ----------------------------------------------------------------------
// 'Oscar' 'Example Oscar Supply Token' token contract
//
// Symbol           : OSCAR
// Name              : Oscar Token
// Total supply: 1,000,000.000000000000000000
// Decimals        : 18
// ----------------------------------------------------------------------


// ----------------------------------------------------------------------
// Safe maths
// ----------------------------------------------------------------------
library SafeMath {
    function add(uint a, uint b) internal pure returns (uint c) {
        c = a + b;
        require(c >= a);
    }
    function sub(uint a, uint b) internal pure returns (uint c) {
        require(b <= a);
        c = a - b;
    }
    function mul(uint a, uint b) internal pure returns (uint c) {
        c = a * b;
        require(a == 0 || c / a == b);
    }
    function div(uint a, uint b) internal pure returns (uint c) {
        require(b > 0);
        c = a / b;
    }
}


// ----------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// ----------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint
```

```
remaining);
        function transfer(address to, uint tokens) public returns (bool success);
        function approve(address spender, uint tokens) public returns (bool success);
        function transferFrom(address from, address to, uint tokens) public returns (bool success);

        event Transfer(address indexed from, address indexed to, uint tokens);
        event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}


// ----------------------------------------------------------------------------
// Contract function to receive approval and execute function in one call
// ----------------------------------------------------------------------------
contract ApproveAndCallFallBack {
        function receiveApproval(address from, uint256 tokens, address token, bytes data) public;
}


// ----------------------------------------------------------------------------
// Owned contract
// ----------------------------------------------------------------------------
contract Owned {
        address public owner;
        address public newOwner;

        event OwnershipTransferred(address indexed _from, address indexed _to);

        function Owned() public {
                owner = msg.sender;
        }

        modifier onlyOwner {
                require(msg.sender == owner);
                _;
        }

        function transferOwnership(address _newOwner) public onlyOwner {
                newOwner = _newOwner;
        }
        function acceptOwnership() public {
                require(msg.sender == newOwner);
                OwnershipTransferred(owner, newOwner);
                owner = newOwner;
                newOwner = address(0);
```

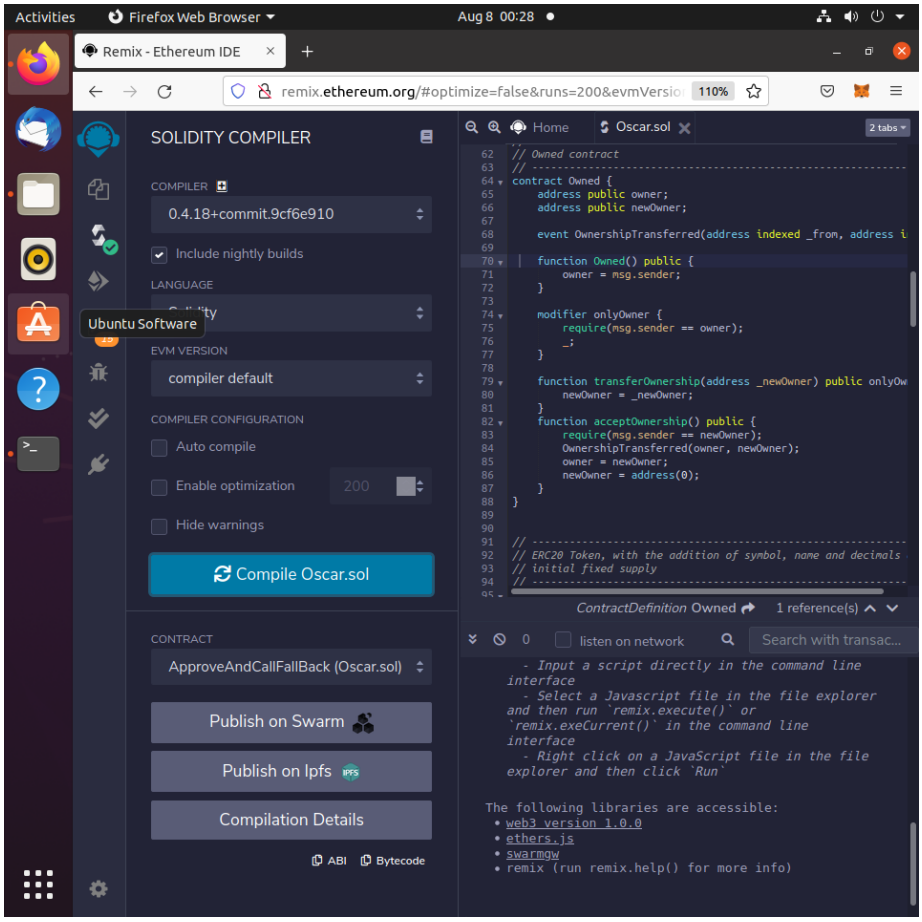
```
    }
}


// ----------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and an
// initial fixed supply
// ----------------------------------------------------------------------
contract OscarToken is ERC20Interface, Owned {
    using SafeMath for uint;

    string public symbol;
    string public    name;
    uint8 public decimals;
    uint public _totalSupply;

    mapping(address => uint) balances;
    mapping(address => mapping(address => uint)) allowed;


    // ------------------------------------------------------------------------
    // Constructor
    // ------------------------------------------------------------------------
    function OscarToken() public {
        symbol = "Oscar";
        name = "Oscar Token";
        decimals = 18;
        _totalSupply = 1000000 * 10**uint(decimals);
        balances[owner] = _totalSupply;
        Transfer(address(0), owner, _totalSupply);
    }


    // ------------------------------------------------------------------------
    // Total supply
    // ------------------------------------------------------------------------
    function totalSupply() public constant returns (uint) {
        return _totalSupply    - balances[address(0)];
    }


    // ------------------------------------------------------------------------
    // Get the token balance for account `tokenOwner`
    // ------------------------------------------------------------------------
```

```
function balanceOf(address tokenOwner) public constant returns (uint balance) {
        return balances[tokenOwner];
}


// ------------------------------------------------------------------------
// Transfer the balance from token owner's account to `to` account
// - Owner's account must have sufficient balance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
function transfer(address to, uint tokens) public returns (bool success) {
        balances[msg.sender] = balances[msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(msg.sender, to, tokens);
        return true;
}


// ------------------------------------------------------------------------
// Token owner can approve for `spender` to transferFrom(...) `tokens`
// from the token owner's account
//
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// recommends that there are no checks for the approval double-spend attack
// as this should be implemented in user interfaces
// ------------------------------------------------------------------------
function approve(address spender, uint tokens) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        return true;
}


// ------------------------------------------------------------------------
// Transfer `tokens` from the `from` account to the `to` account
//
// The calling account must already have sufficient tokens approve(...)-d
// for spending from the `from` account and
// - From account must have sufficient balance to transfer
// - Spender must have sufficient allowance to transfer
// - 0 value transfers are allowed
// ------------------------------------------------------------------------
function transferFrom(address from, address to, uint tokens) public returns (bool success) {
        balances[from] = balances[from].sub(tokens);
```

```
        allowed[from][msg.sender] = allowed[from][msg.sender].sub(tokens);
        balances[to] = balances[to].add(tokens);
        Transfer(from, to, tokens);
        return true;
    }


    // ------------------------------------------------------------------------
    // Returns the amount of tokens approved by the owner that can be
    // transferred to the spender's account
    // ------------------------------------------------------------------------
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining) {
        return allowed[tokenOwner][spender];
    }


    // ------------------------------------------------------------------------
    // Token owner can approve for `spender` to transferFrom(...) `tokens`
    // from the token owner's account. The `spender` contract function
    // `receiveApproval(...)` is then executed
    // ------------------------------------------------------------------------
    function approveAndCall(address spender, uint tokens, bytes data) public returns (bool success) {
        allowed[msg.sender][spender] = tokens;
        Approval(msg.sender, spender, tokens);
        ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
        return true;
    }


    // ------------------------------------------------------------------------
    // Don't accept ETH
    // ------------------------------------------------------------------------
    function () public payable {
        revert();
    }


    // ------------------------------------------------------------------------
    // Owner can transfer out any accidentally sent ERC20 tokens
    // ------------------------------------------------------------------------
    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner returns (bool success) {
```

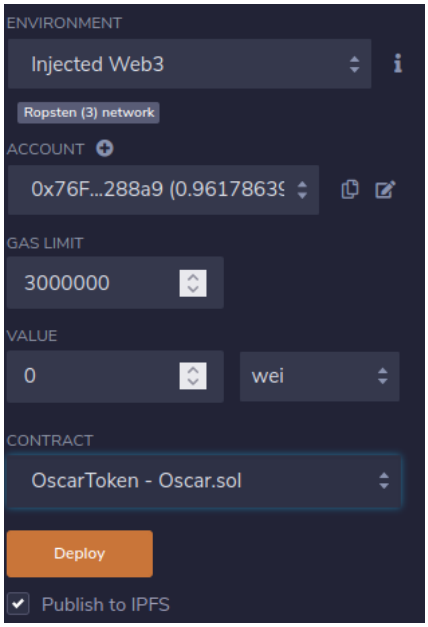
```
        return ERC20Interface(tokenAddress).transfer(owner, tokens);
    }
}
```

Compile this .sol file to get the ABI code

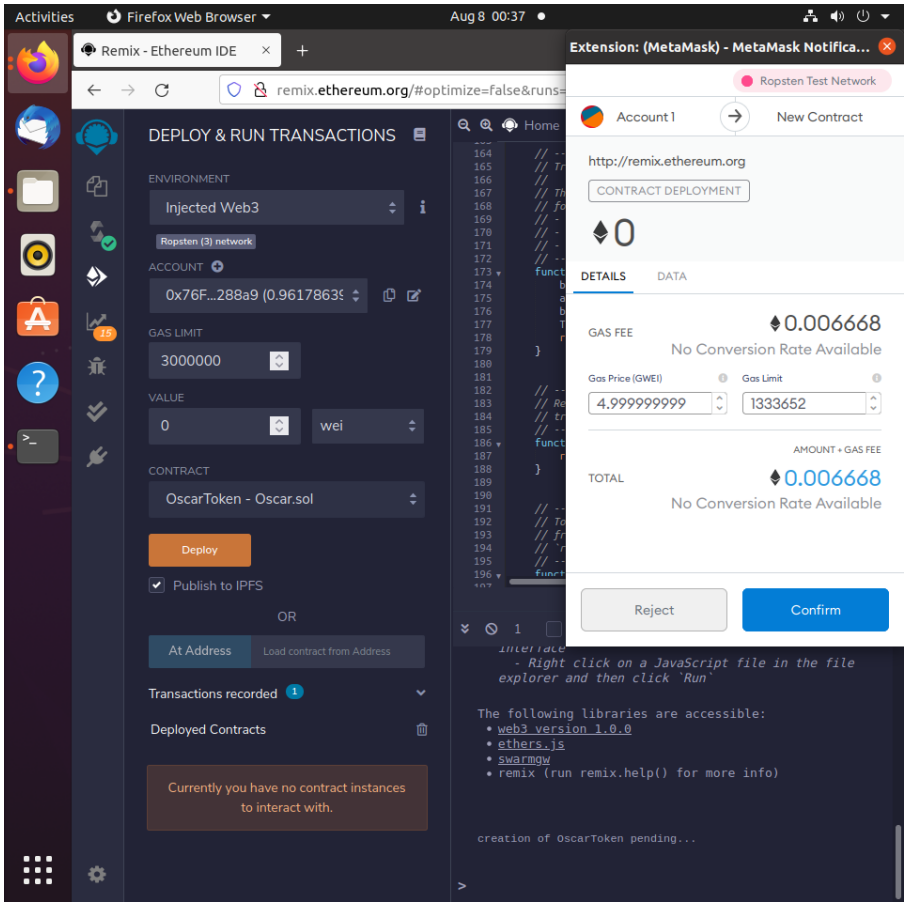


Login the Metamask and go to Deploy & run transactions, in Environment change to Injected Web3

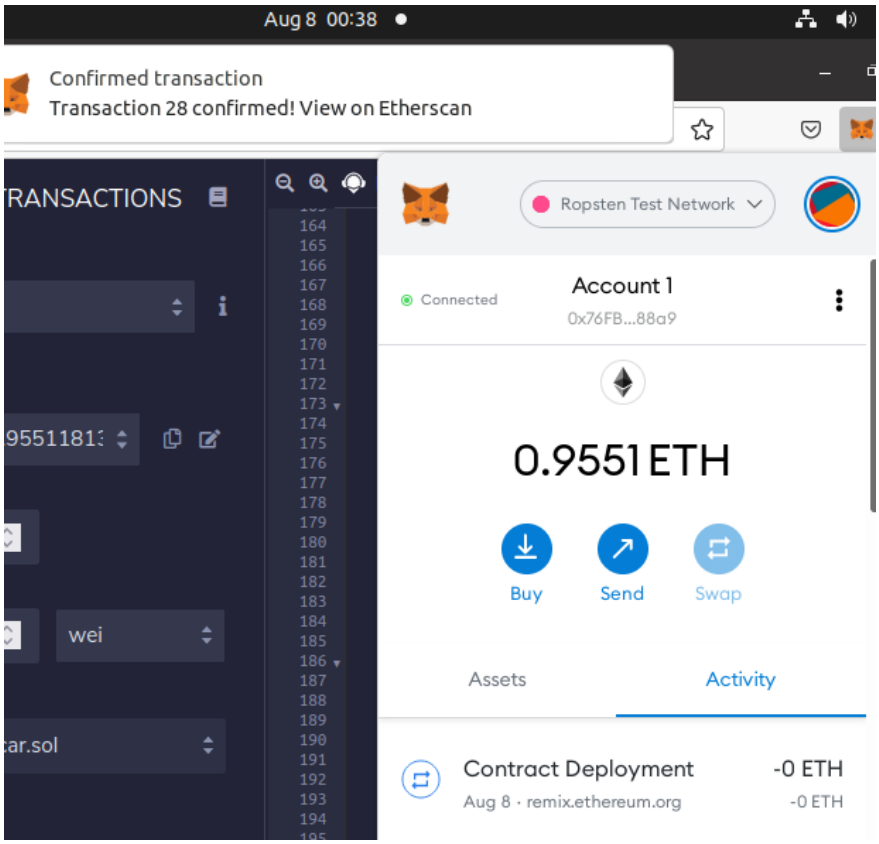
Choose the “OscarToken” and click Deploy



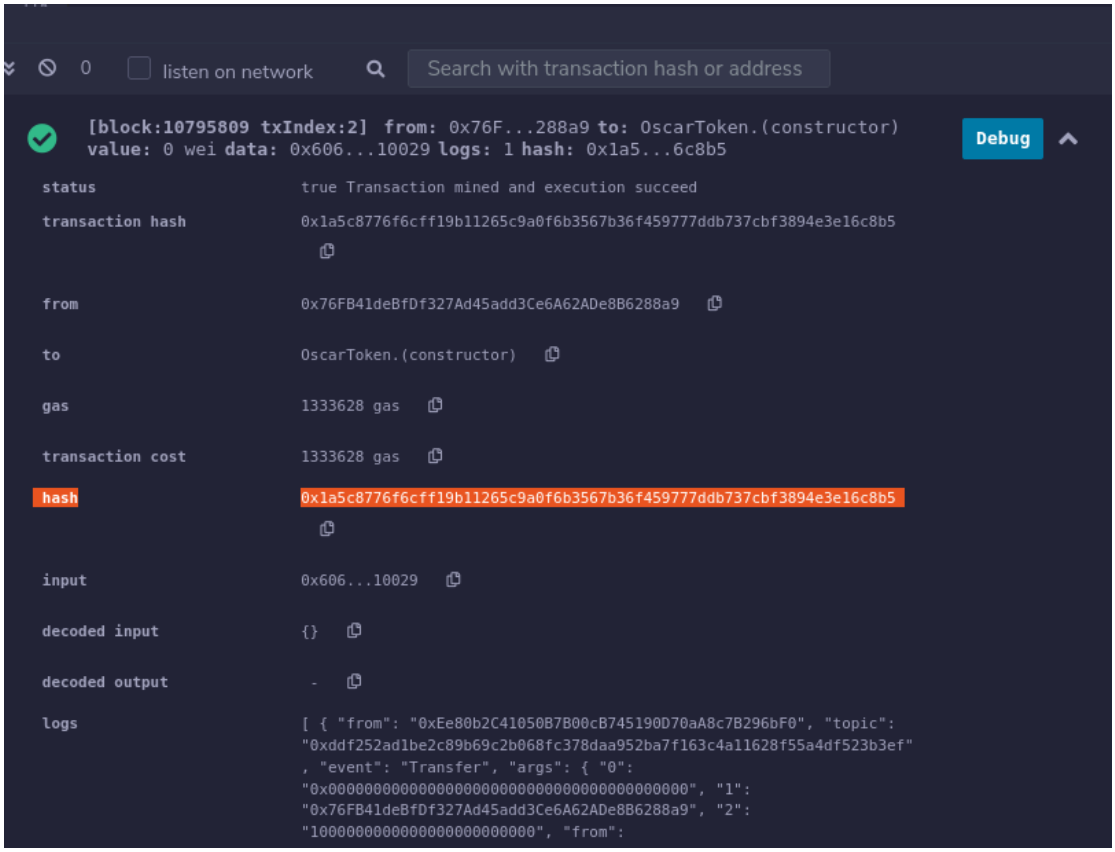
The MetaMask will pop up, we click the confirm



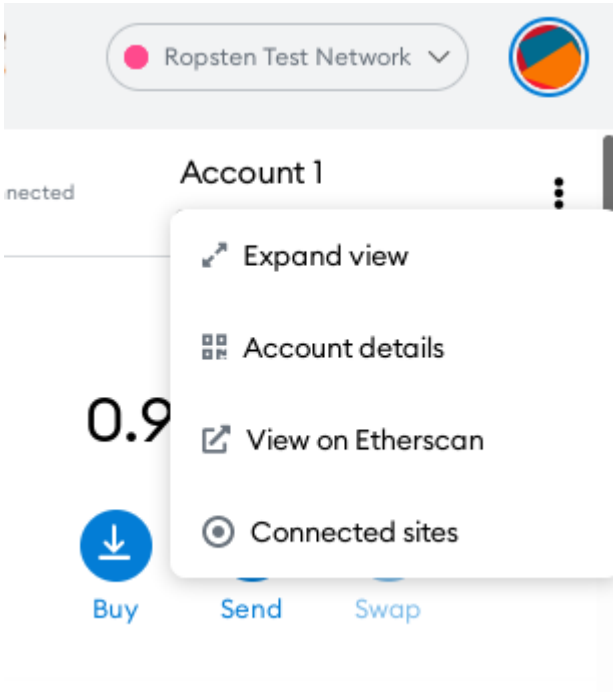
Transaction finished



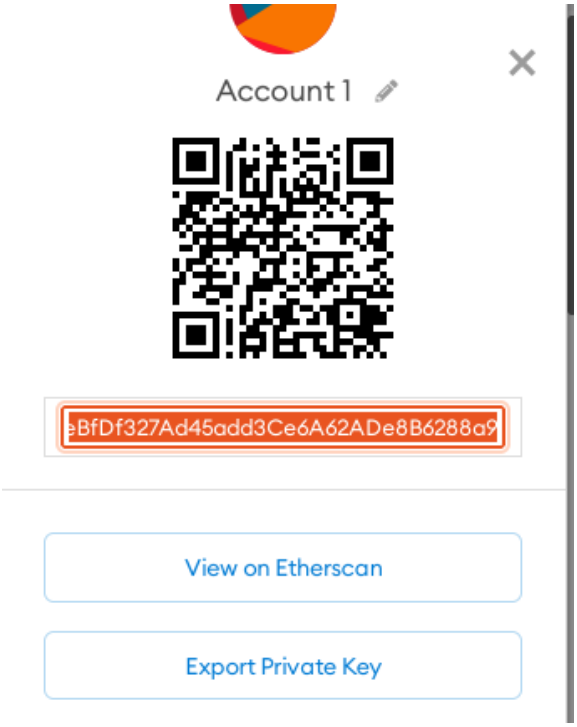
In the console find and copy the Hash



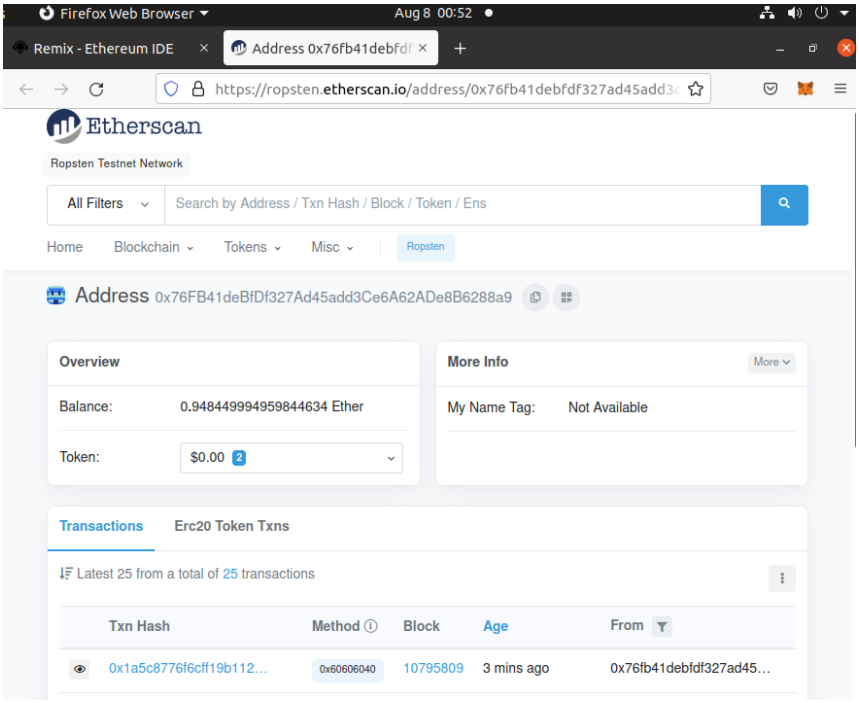
Go to MetaMask and click the Account Details



Click View on Etherson open the https://ropsten.etherscan.io/ link



In this page, in the Transactions, click the Hash(Same as the Hash copied in REMIX console before)



Now, can see my “1,000,000 Oscar Token” created