Lab 8: Bitcoin and Blockchain

1 Details

Aim: To provide a foundation in understanding in Bitcoin and Blockchain.

2 Activities

L1.1 Using blockchain.info, find the details of the genesis block:

Date created:

Reward:

Number of transactions:

Size of block:

Which account received the mining reward for the genesis block (last four digits):

How many USD does the original miner have in the account they used for the first genesis record:

When did the genesis block creator stop trading?

L1.2 Using blockchain.info, determine the following

Total bitcoins in circulation:

Most recent hash block (last four hex digits):

Block reward per block:

Difficulty:

Average time between blocks:

Market capitalisation (USD):

24 hr price (USD):

24hr transactions (USD):

Hash rate:

Last successful miner:

Maximum block size:

Balance for 1GbVUSW5WJmRCpaCJ4hanUny77oDaWW4to:

L1.3 Download and create the Python file defined on this page:

https://asecuritysite.com/encryption/bit

Now run the Python file, and compare the results in L.1.2.

Total bitcoins in circulation:

Most recent hash block (last four hex digits):

Block reward per block:

Difficulty:

Average time between blocks:

Market capitalisation (USD):

24 hr price (USD):

24hr transactions (USD):

Hash rate:

Balance for 1GbVUSW5WJmRCpaCJ4hanUny77oDaWW4to:

C Setting up your Ethereum wallet on Sopolea

Demo: https://www.youtube.com/watch?v=xeaDE8wgVVQ

The Sopolea network allows a user to test an Ethereum application and using free Ether. Initially, set up your MetaMask wallet. A document to outline how you set this up is here. Once you have set it up, answer the following:

- What is your public ID (just define the first four hex values)?
- Find out someone else's public ID, and send them 0.001 Ether. If you are doing the lab on your own, send it to Bill (ID: 0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233).

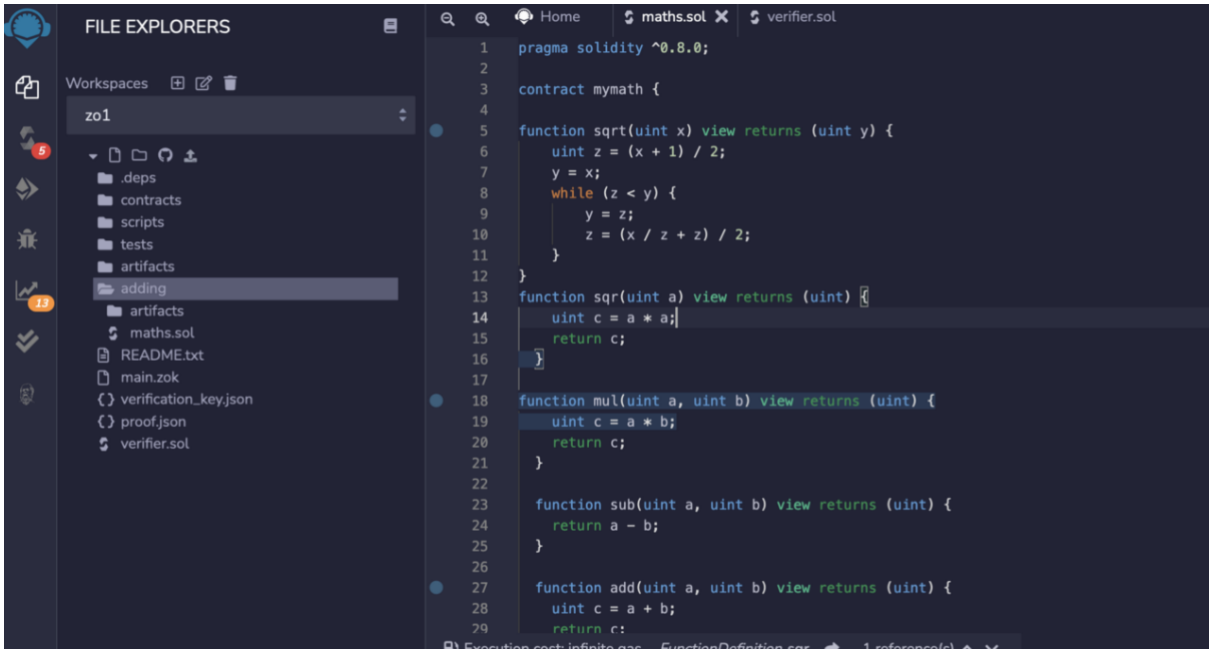
- Can you see the transaction on the Ethereum network? An example of a wallet is here.
- Can you see your transaction on the Ethereum network for the person you send it to?
- What was the transaction fee for the transfer? If you were using the main Ethereum network, how much would the transaction cost in Dollars?
- Ask someone to send you 0.001 Ether. Did you receive it? If you are doing the lab on your own, ask your lab tutor to send you 0.001 Ether.

Creating a Smart Contract in Ethereum

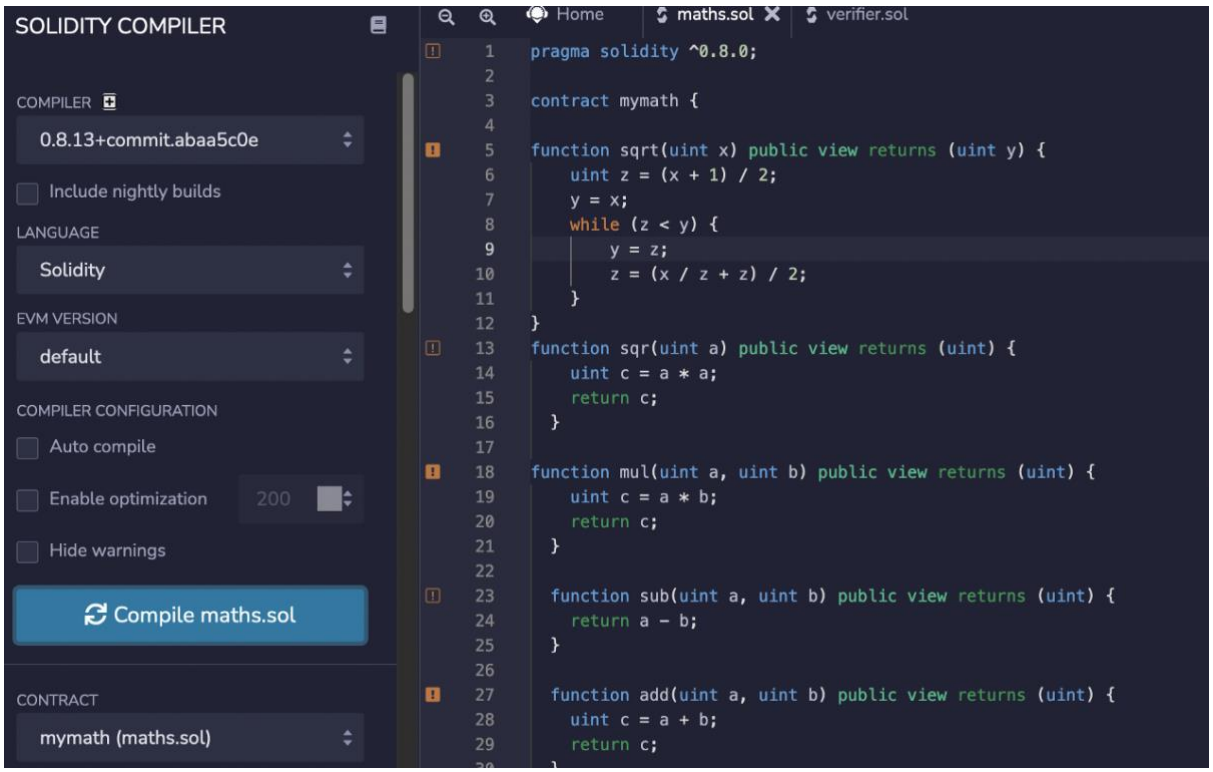
So, let's write a bit of code that does some simple maths. In the following we will implement sqrt(), sqr(), mul(), sub(), and add():

```
pragma solidity ^0.8.0;
contract mymath {function sqrt(uint x) public view returns (uint y) {
    uint z = (x + 1) / 2;
    y = x;
    while (z < y) {
        y = z;
        z = (x / z + z) / 2;
    }
}
function sqr(uint a) public view returns (uint) {
    uint c = a * a;
    return c;
  }
function mul(uint a, uint b) public view returns (uint) {
    uint c = a * b;
    return c;
  }
function sub(uint a, uint b) public view returns (uint) {
    return a - b;
  }
function add(uint a, uint b) public view returns (uint) {
    uint c = a + b;
    return c;
}}
```

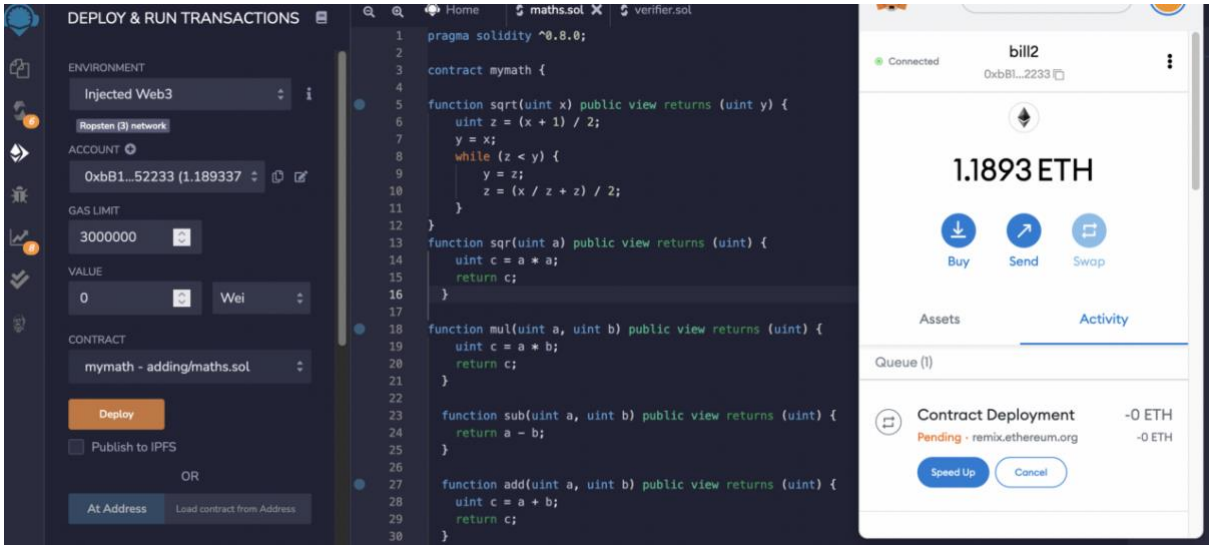
In this case, the "public" part makes sure we can see the output of the function, and the "view" part allows it to be stateless (and where we just have to receiver the value without the smart contact remember the state). On Ethereum we normally use the Solidity language to create a smart contract and then compile it into the byte code required for the ledger. First, can we start by entering the Solidity code into Remix [here]:



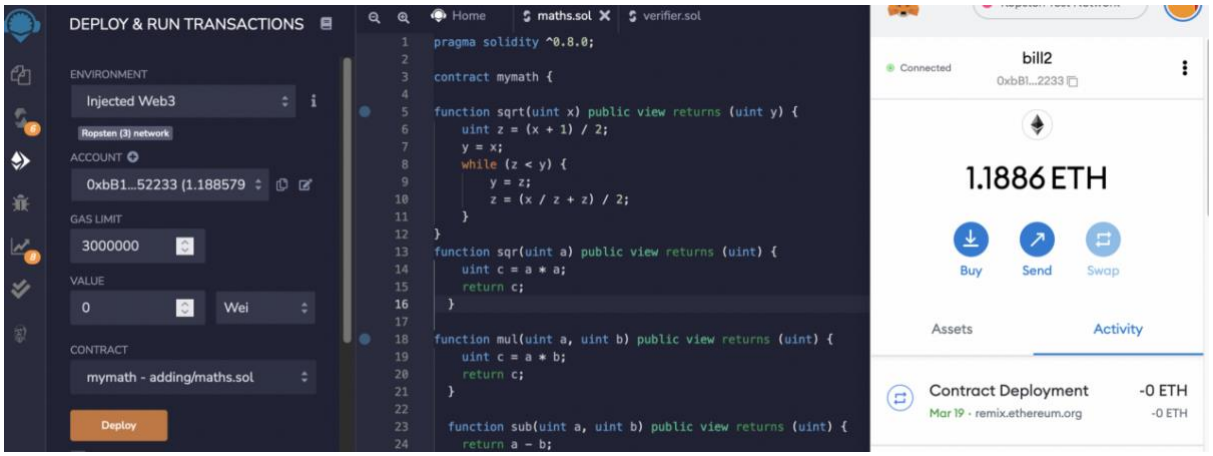
Once entered, we can then compile it with the Solidity compiler. It is important to take a note of the compiler version at this stage, as we will need this later:



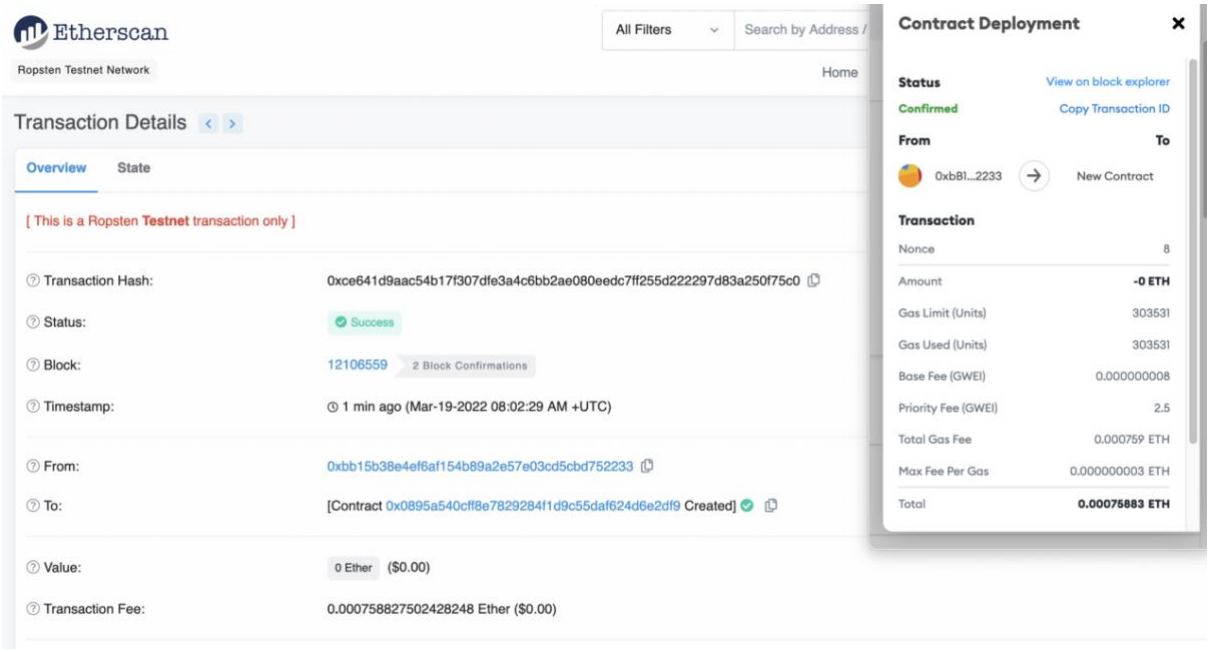
Once compiled we can then deploy the smart contract to a test network (Ropsten). For this , we need to connect our MetaMask wallet:



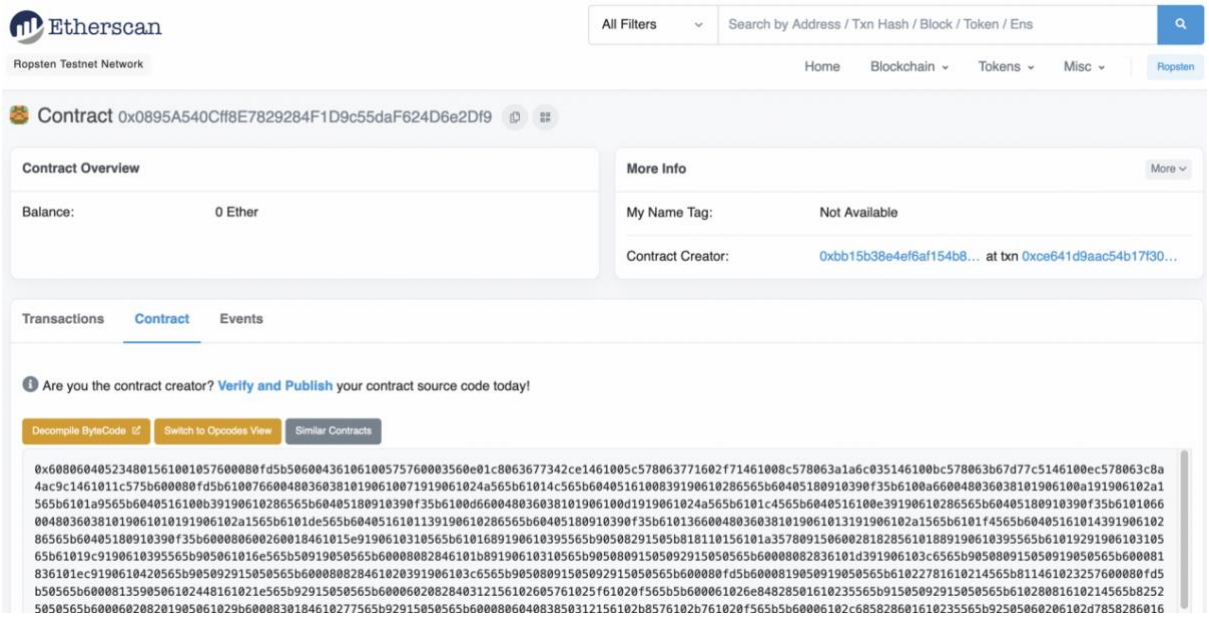
Once it has been deployed, we can see our wallet identifies the deployed contract:



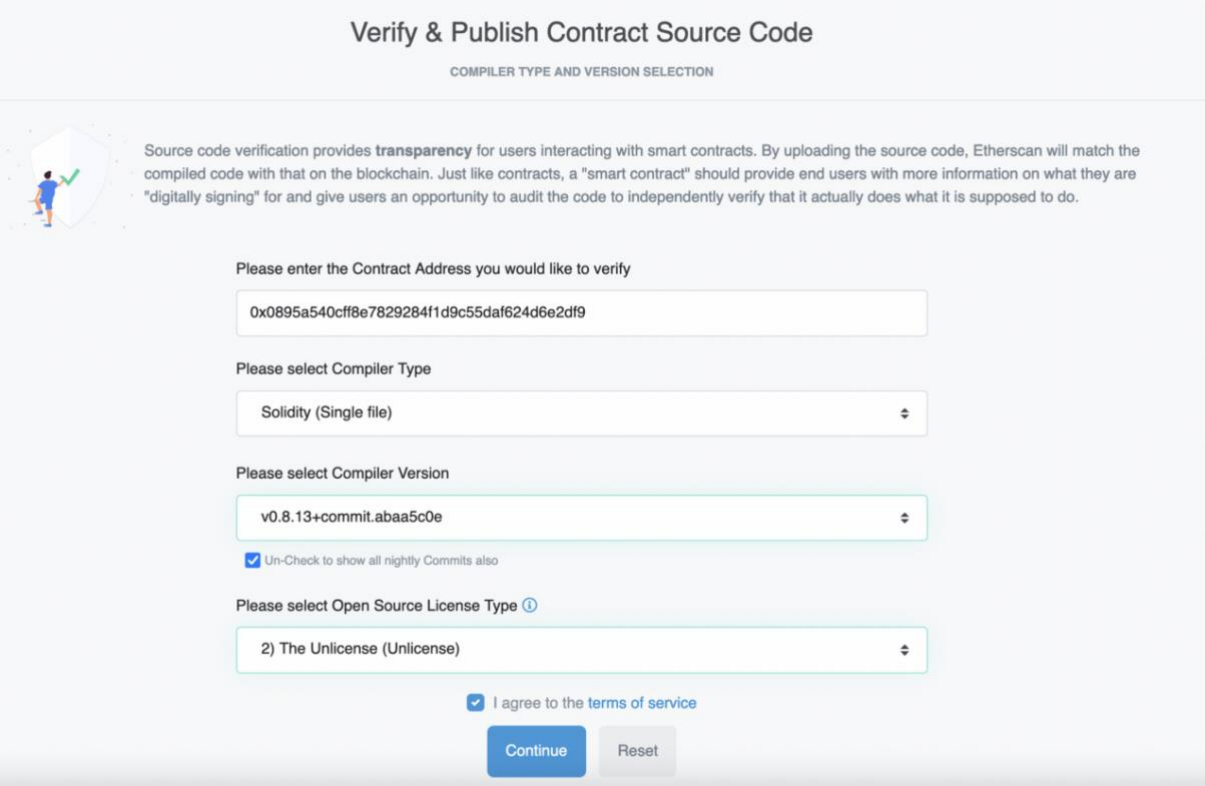
And clicking through gives us the address of the contract, and then viewing it on the explorer, we can see the transaction:



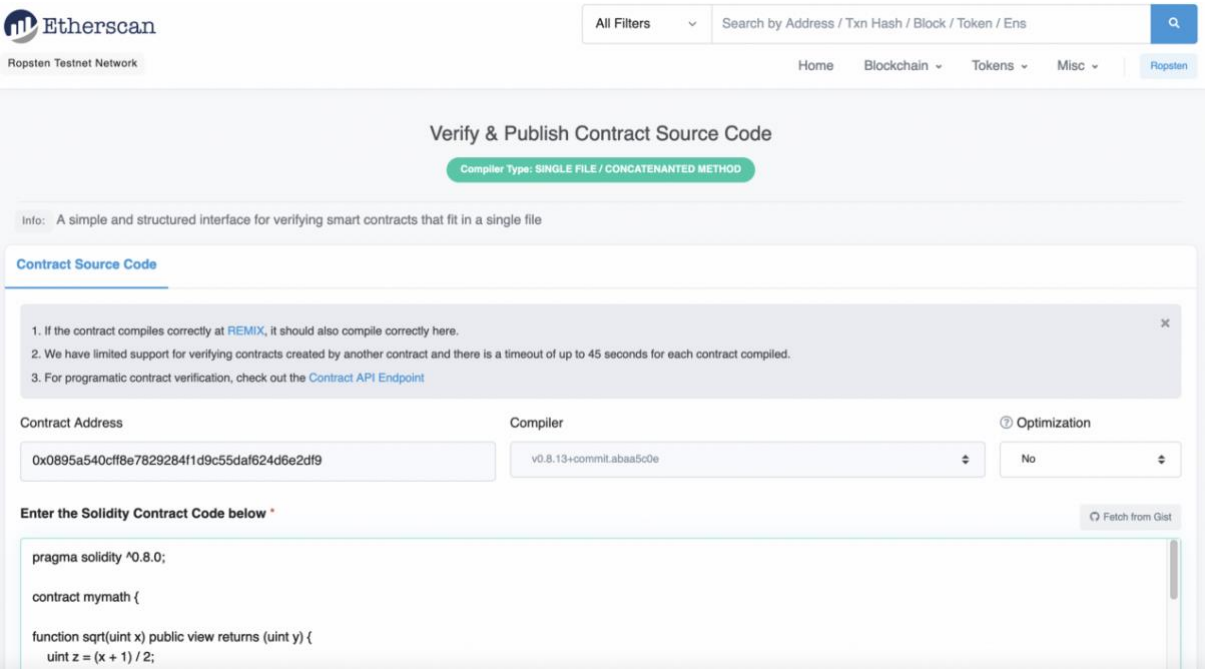
The address here is "0x0895..", so we can view the smart contract from: here. We now need to verify and publish the contact, with click on "Verify and Publish":



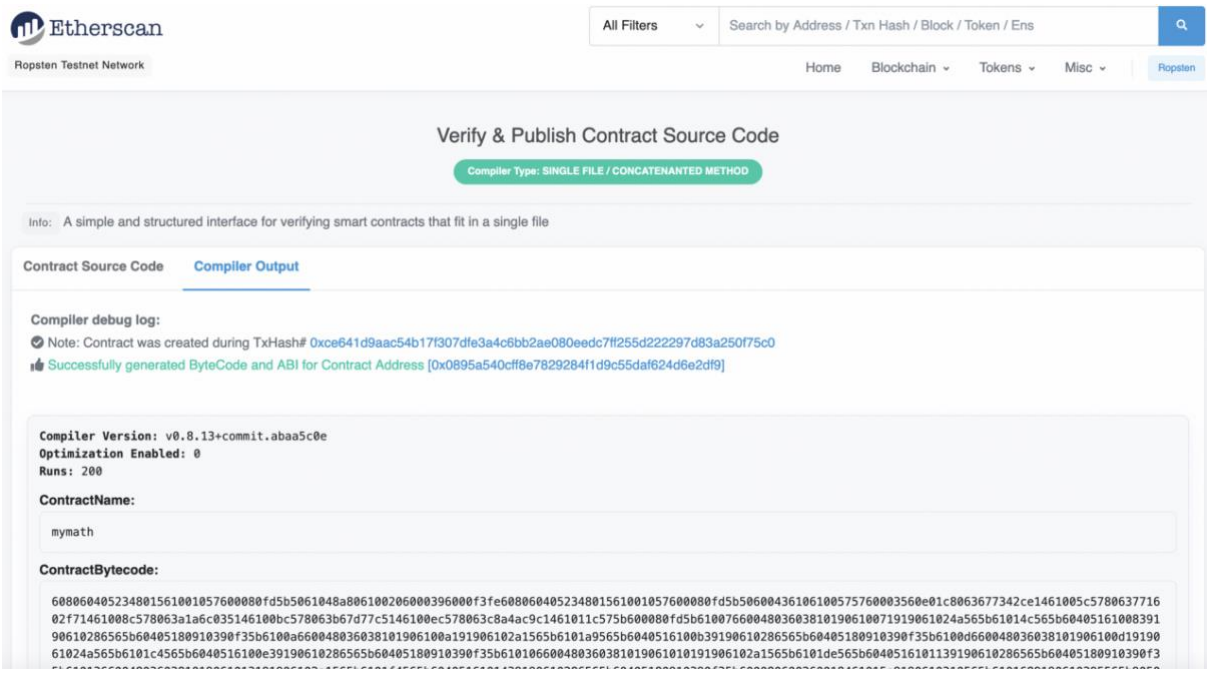
After this, we can define the Compiler Version and the licence



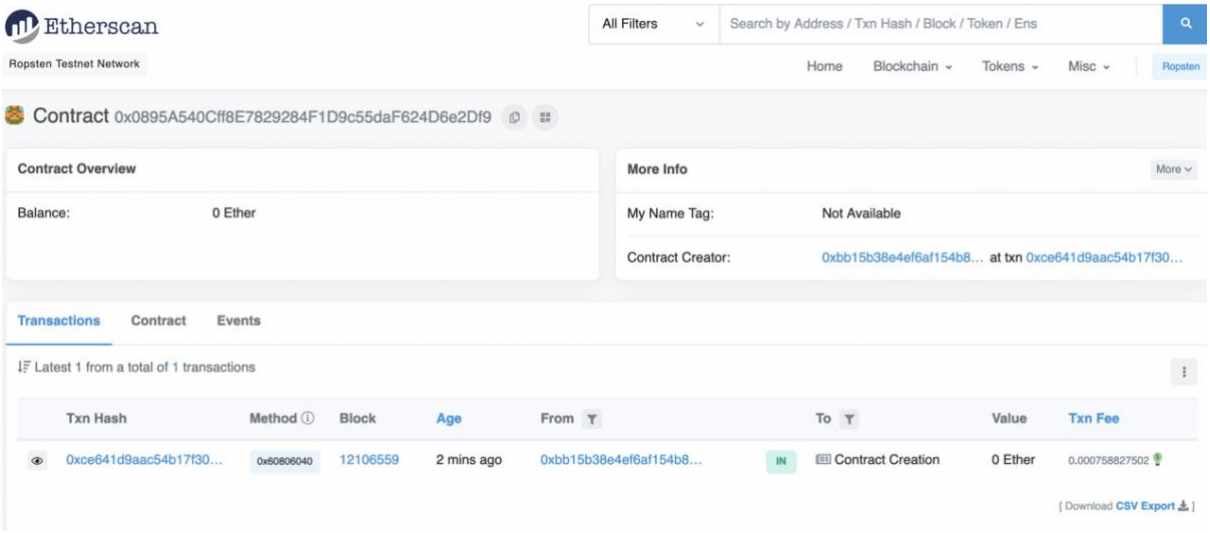
We then need to add your code for it to be checked:



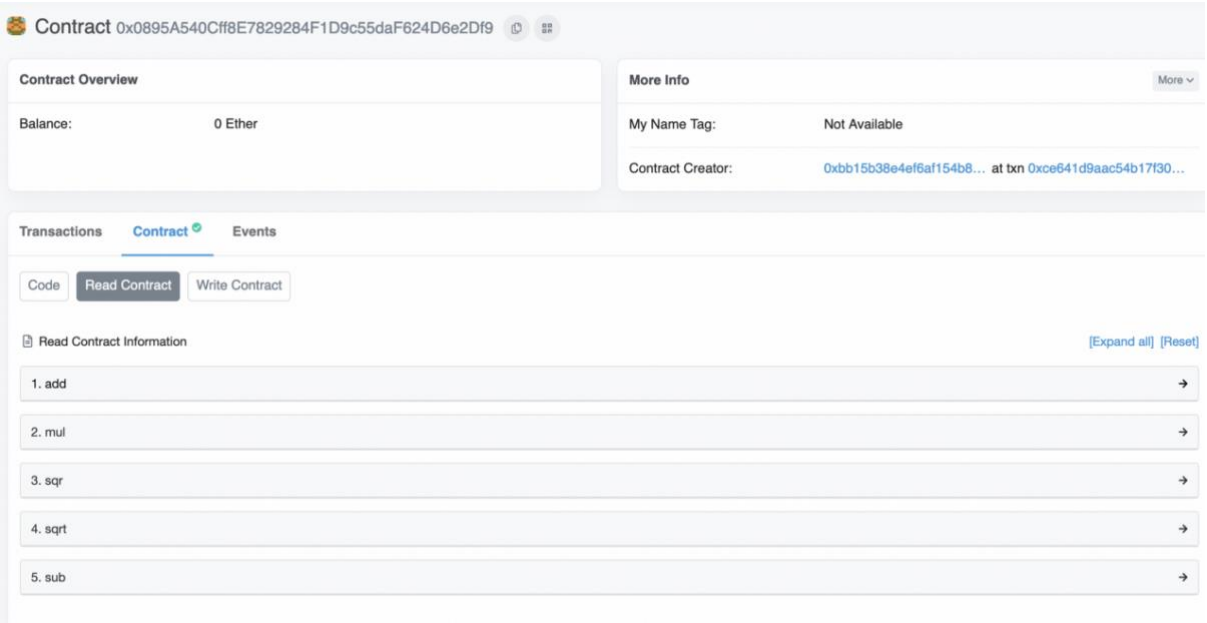
It takes around 30 seconds, but, eventually, we should have our code accepted:



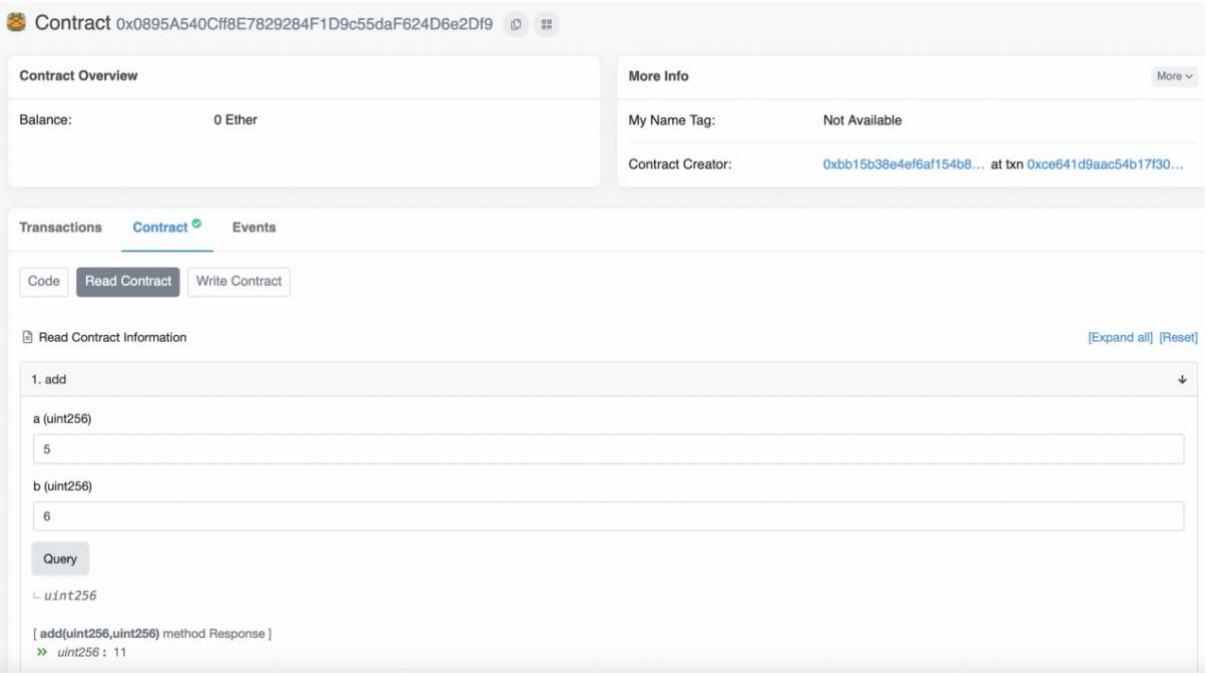
We now have the contract published to the Ropsten test network:



Next, by selected the Contract tab, and can view the read parameters. The exposed functions are add(), mul(), sqr(), sqrt() and sub():



To test, we can just enter the variables for a given function, and get a result:



Creating ERC-20 tokens

Within the Ethereum blockchain, we can record transactions and run smart contracts. These things allow us to run DApps (decentralized applications) and which can support the running of the infrastructure in return for some payment (Ether). A DApp can also create tokens for new currencies, shares in a company or to prove the ownership of an asset. ERC-20 is a standard format for a Fungible Token and which can support the sharing, transfer and storage of tokens. These tokens are supported by the whole of the Ethereum infrastructure and can be easily traded. They support a number of mandatory functions:

- totalSupply. This function is the total number of ERC-20 tokens that have been created.
- balanceOf. This function identifies the number of tokens that a given address has in its account.
- transfer. This function supports the transfer of tokens to a defined user address.
- transferFrom. This function supports a user to transfer tokens to another user.
- approve. This function checks that a transaction is valid, based on the supply of token.
- allowance. This function checks if a user has enough funds in their account for a transaction.

There are also a number of options:

- Token Name. This is the name that the token will be defined as.
- Symbol. This is the symbol that the token will use.
- Decimal. This is the number of decimal places to be used for any transactions.

Now we you create your own token. If you are Bob Smith, then call your token "BobSmithToken", and your currency will be "BobSmith".

So, let's create a token named "ENUToken" (change the name to your name), and use the tutorial sample from here. First, we open up https://remix.ethereum.org/, and enter the following Solidy contract:

```
pragma solidity ^0.4.24;

// ----------------------------------------------------------------------
----
// 'ENU Token' token contract
//
// Deployed to : 0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233
// Symbol      : ENUToken
// Name        : ENU Token
// Total supply: 100000000
// Decimals    : 18

// Based on https://github.com/bitfwdcommunity/Issue-your-own-ERC20-
token/tree/master/contracts


// ----------------------------------------------------------------------
----
// Safe maths
// ----------------------------------------------------------------------
----
contract SafeMath {
    function safeAdd(uint a, uint b) public pure returns (uint c) {
        c = a + b;
        require(c >= a);
    }
    function safeSub(uint a, uint b) public pure returns (uint c) {
        require(b <= a);
        c = a - b;
    }
    function safeMul(uint a, uint b) public pure returns (uint c) {
```

```
        c = a * b;
        require(a == 0 || c / a == b);
    }
    function safeDiv(uint a, uint b) public 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
//
// Borrowed from MiniMeToken
// ----------------------------------------------------------------------
----
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);

    constructor() 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);
        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;
        newOwner = address(0);
    }
}


// ----------------------------------------------------------------------
----
// ERC20 Token, with the addition of symbol, name and decimals and assisted
// token transfers
// ----------------------------------------------------------------------
----
contract <b style="color:red">BillToken</b> is ERC20Interface, Owned,
SafeMath {
    string public symbol;
    string public  name;
    uint8 public decimals;
    uint public _totalSupply;

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


    // ----------------------------------------------------------------------
----
    // Constructor
    // ----------------------------------------------------------------------
----
    constructor() public {
        symbol = "ENUToken";
        name = "ENU Token";
        decimals = 18;
        _totalSupply = 1000000000000000000000000000;
        balances[0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233] =
_totalSupply;
        emit Transfer(address(0), <b
style="color:red">0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233</b>,
_totalSupply);
    }


    // ----------------------------------------------------------------------
----
    // Total supply
    // ----------------------------------------------------------------------
----
    function totalSupply() public constant returns (uint) {
```

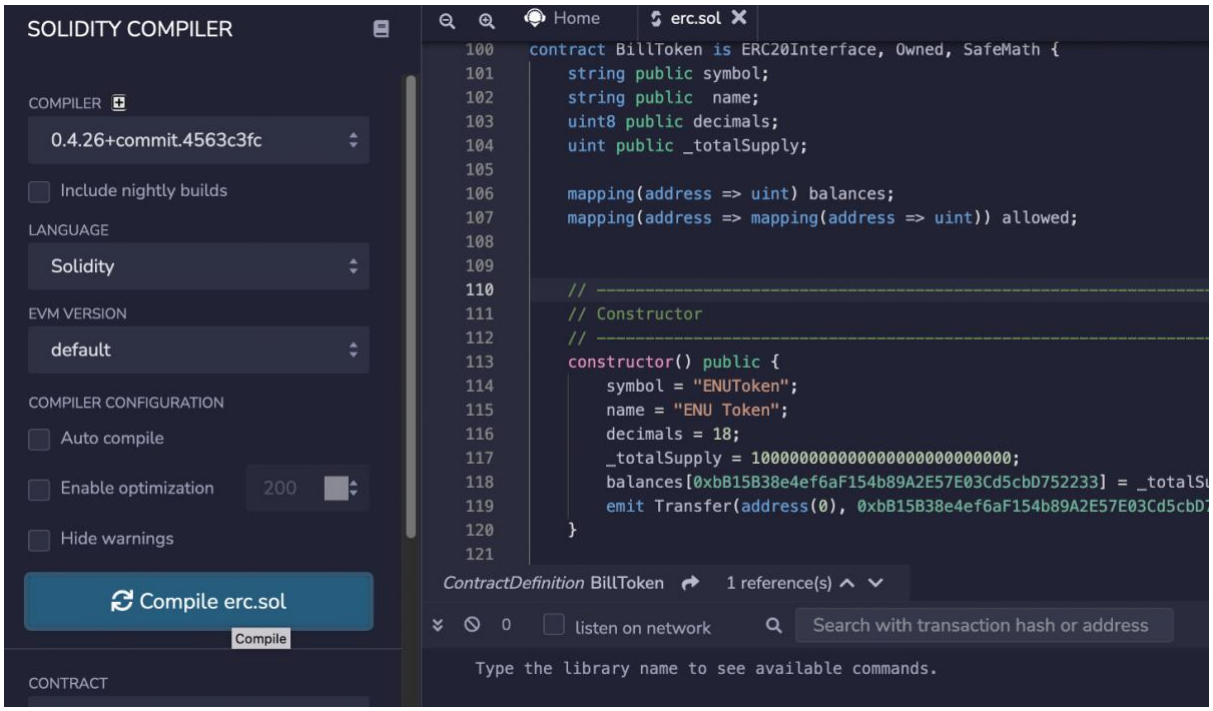
```
        return _totalSupply  - balances[address(0)];
    }


    // ------------------------------------------------------------------------
----
    // 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] = safeSub(balances[msg.sender], tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit 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;
        emit 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] = safeSub(balances[from], tokens);
        allowed[from][msg.sender] = safeSub(allowed[from][msg.sender],
tokens);
        balances[to] = safeAdd(balances[to], tokens);
        emit 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;
        emit 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);
    }
}
```

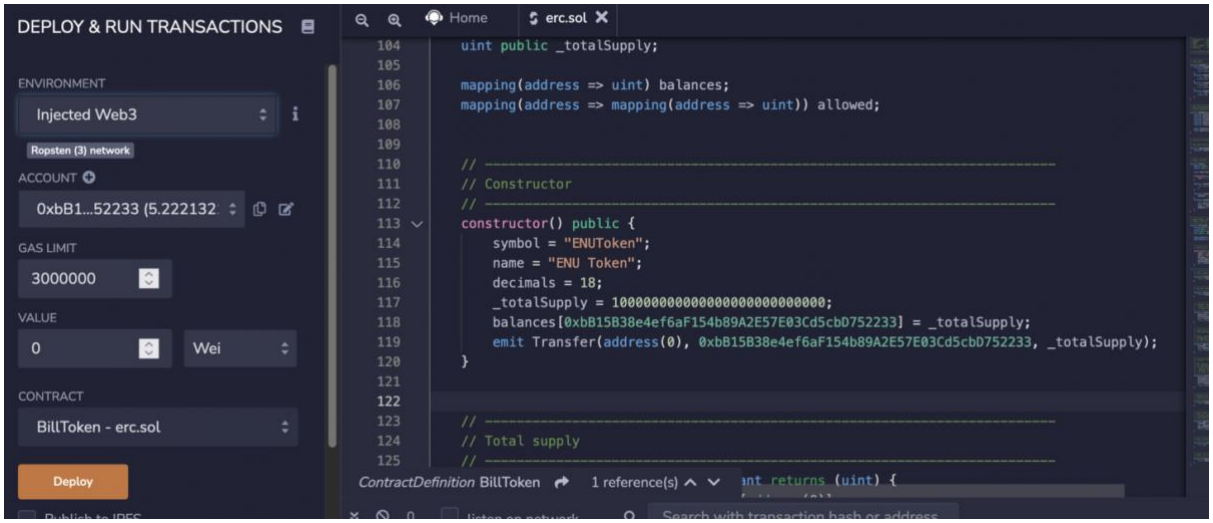
When you create your own contract, make sure you change the public constructor() with: the **symbol**, the **name**, and the **wallet ID**. You are the owner of the token, so you need to enter the public ID of your wallet for two hex values given next:

```
    constructor() public {
    symbol = "ENUToken";
    name = "ENU Token";
    decimals = 18;
    _totalSupply = 100000000000000000000000000;
    balances[0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233] = _totalSupply;
    emit Transfer(address(0), 0xbB15B38e4ef6aF154b89A2E57E03Cd5cbD752233,
_totalSupply);
    }
```

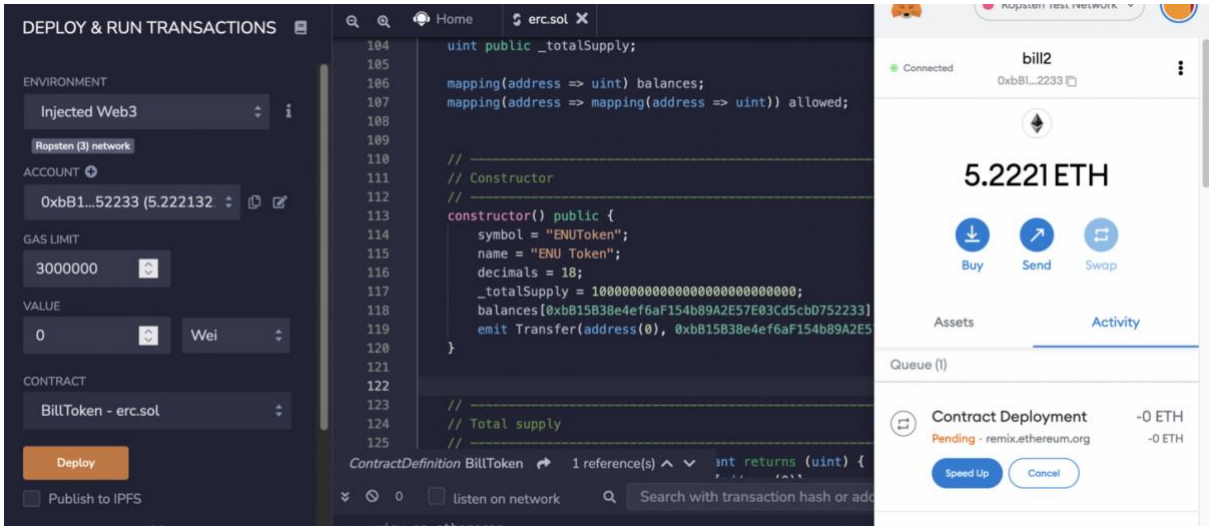
The wallet ID is the public ID of your wallet in MetaMask. Now we compile:



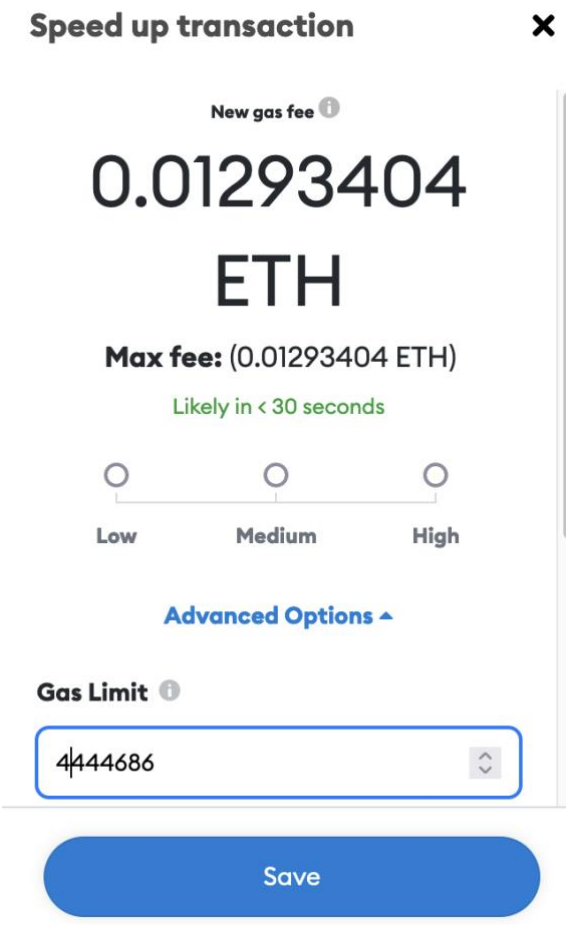
Next, we will deploy to the Ropsten test network:



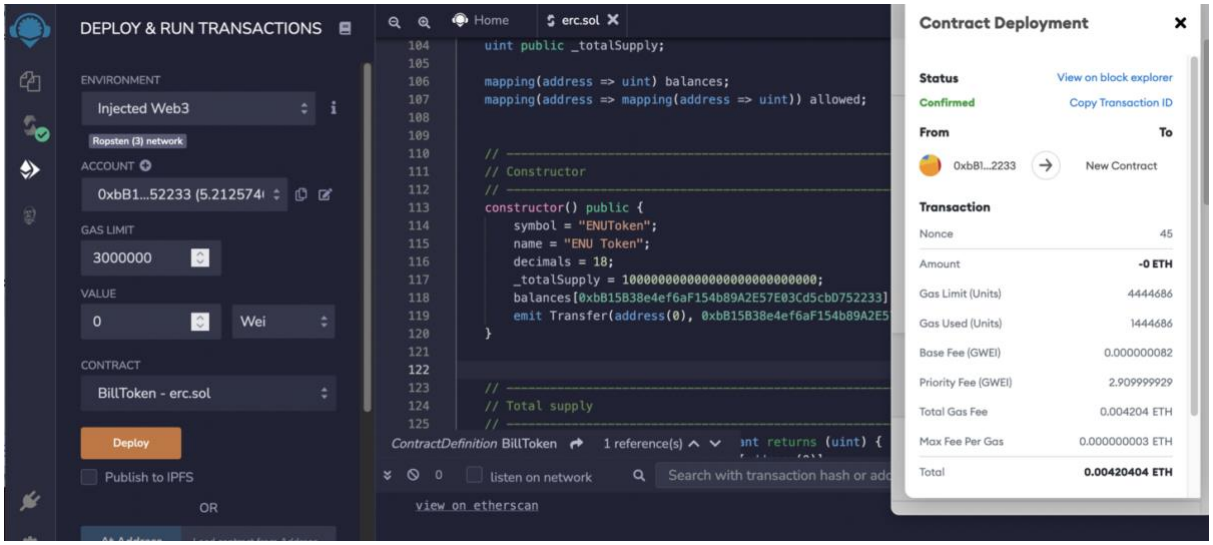
After this, our contract will be shown as being pending deployment:



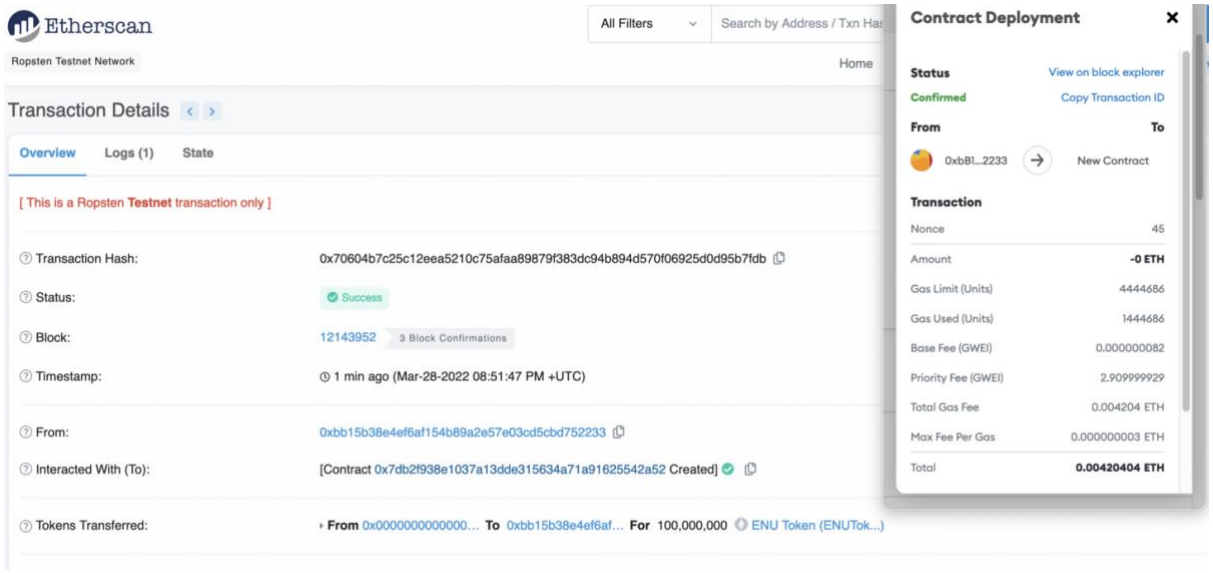
It will take 10–15 minutes to deploy, but it can be speeded up by increasing the gas limit:



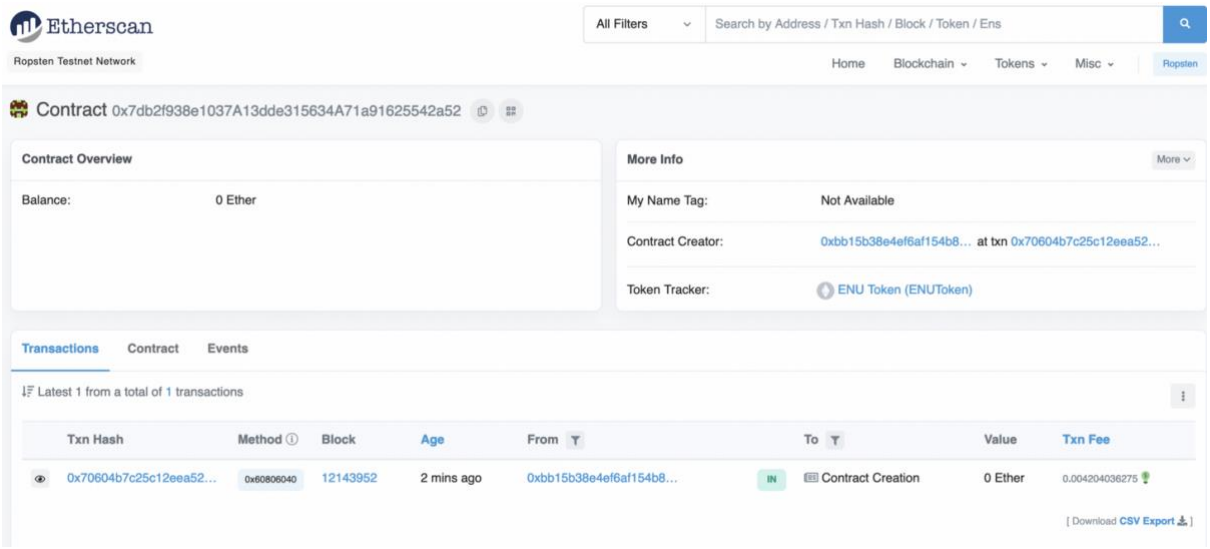
Once deployed, we can view the contract details:



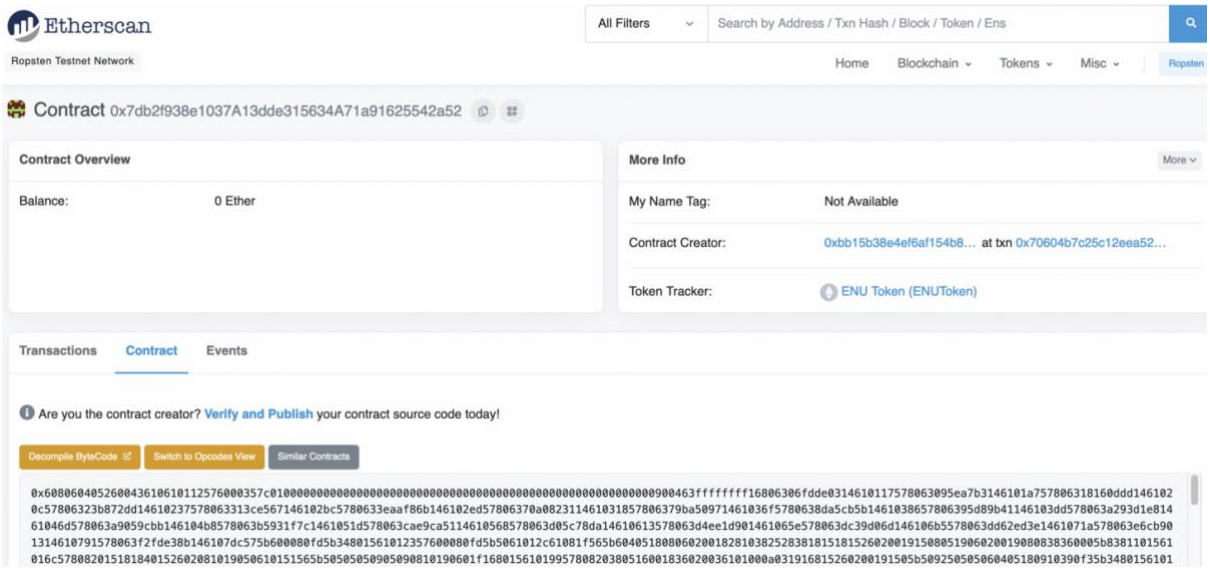
And can then view the transaction for the contact [here]:



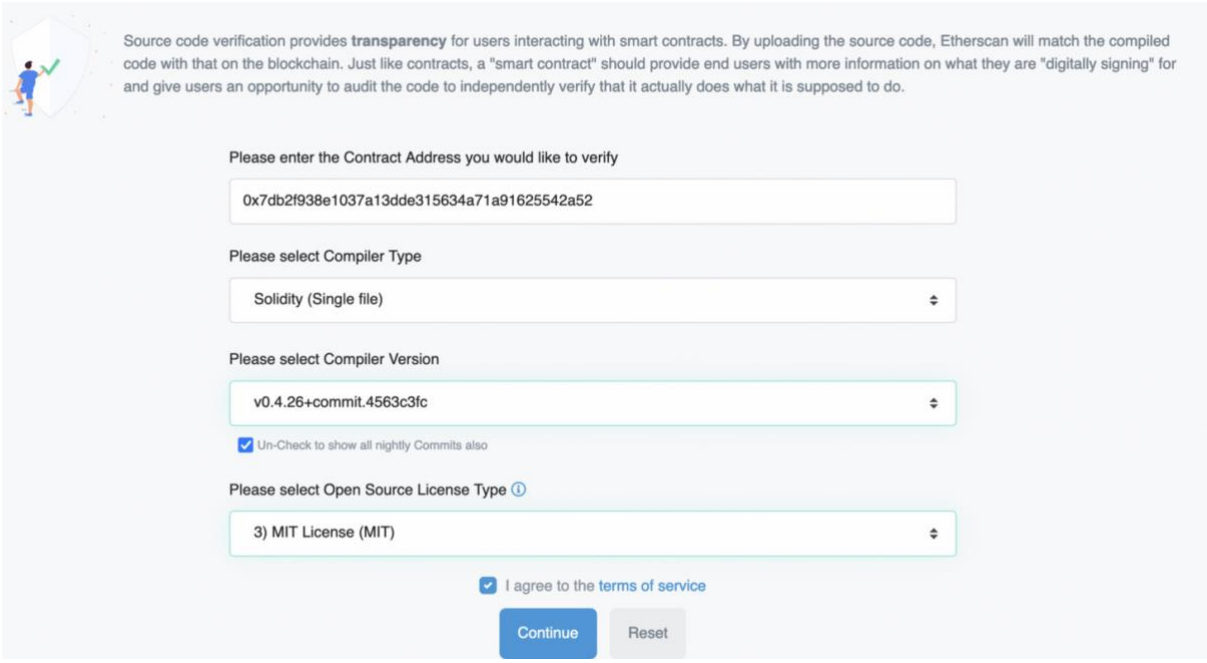
And then view the contact here]:



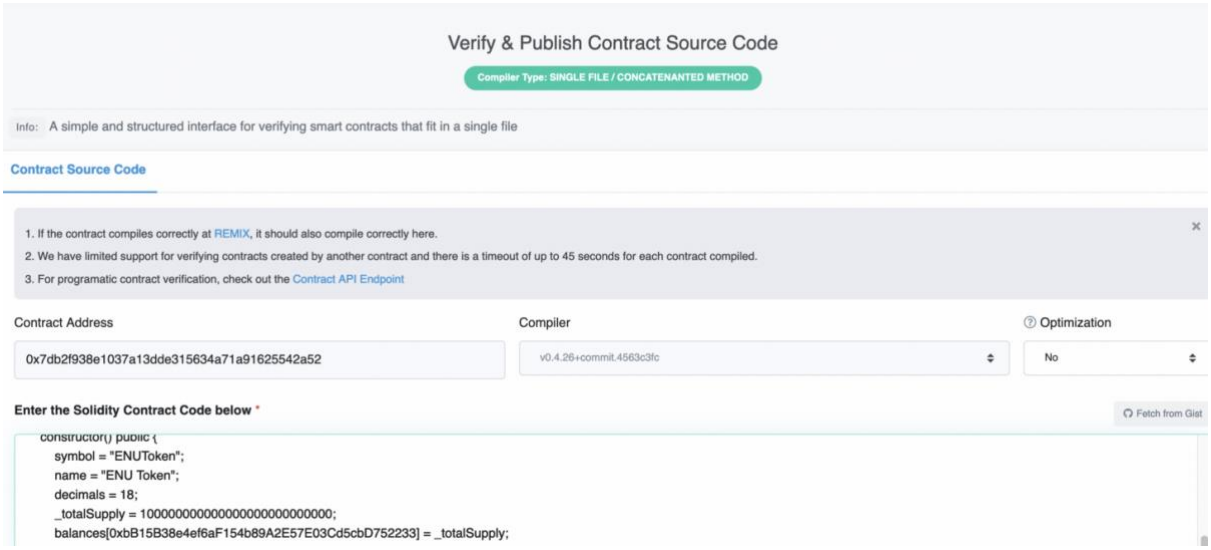
Next, we select the Contract tab:



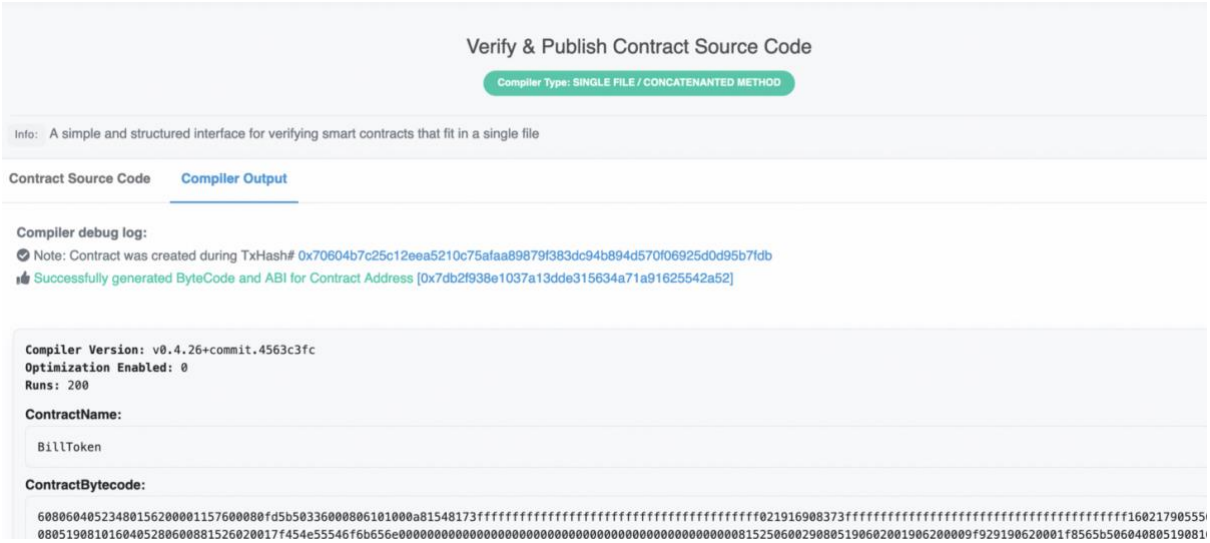
And then select "Verify and Publish" and enter the details of the compiler version (v0.4.26):



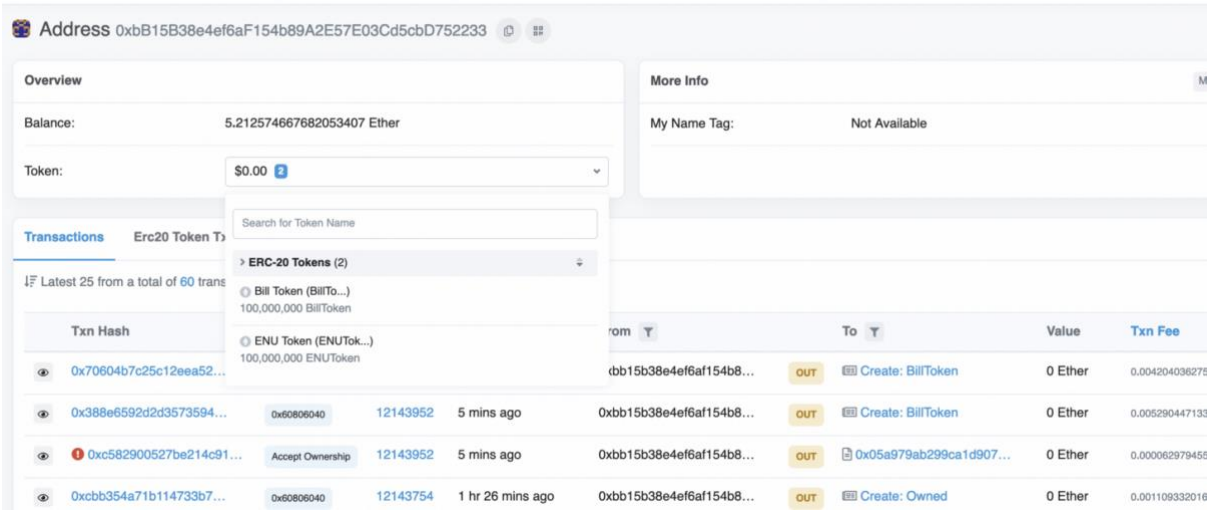
We then need to copy-and-pasete the contract code into the Source Code text box:



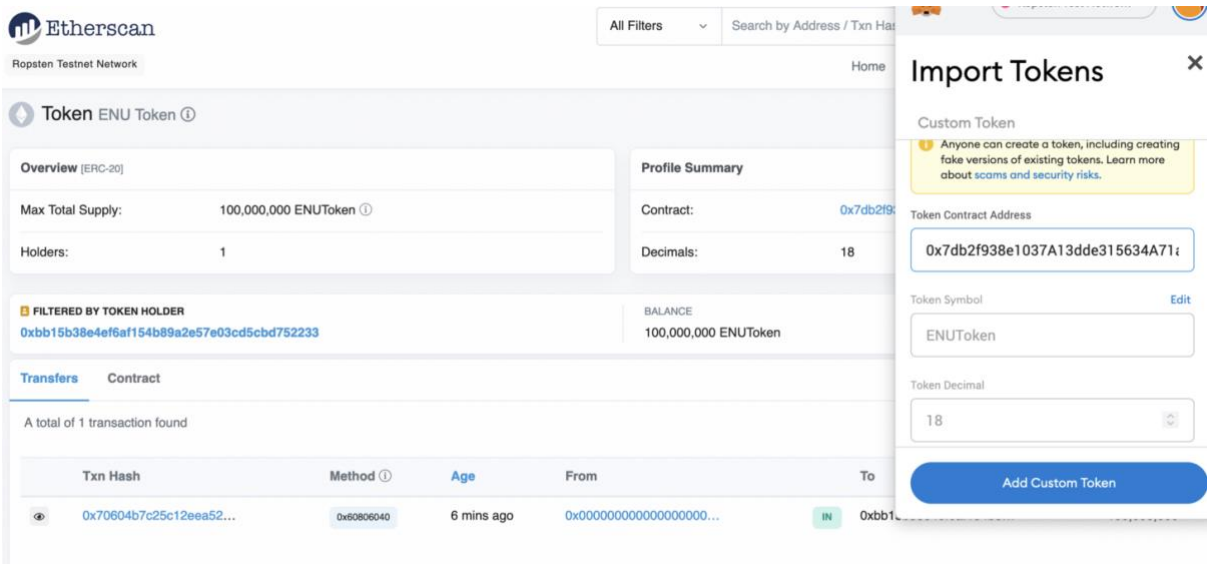
After less than 45 seconds, the contract will be approved:



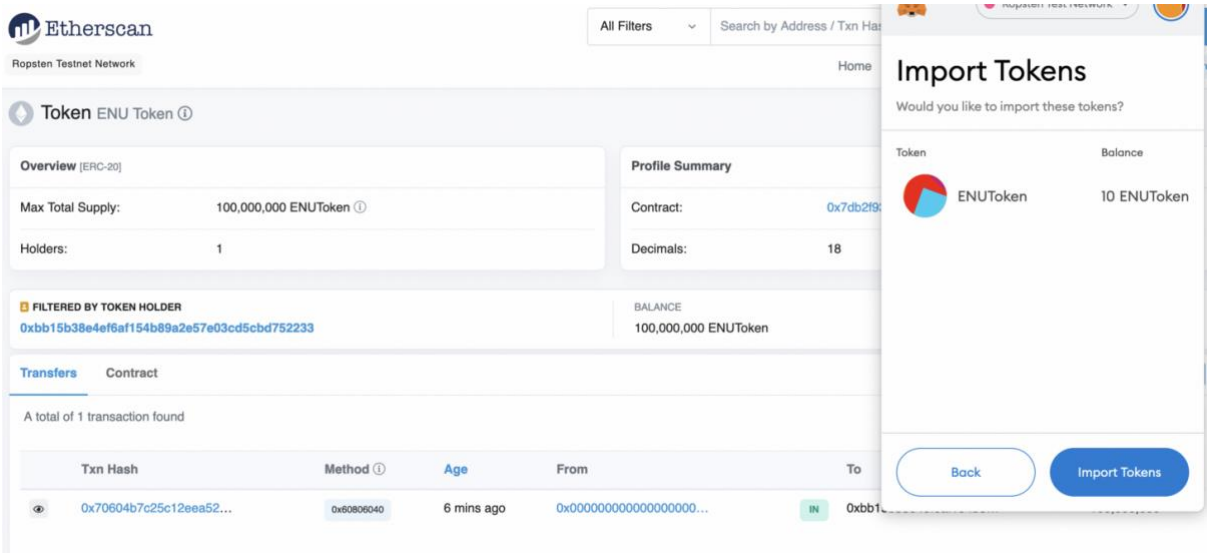
When the contact is run there is a constructor to transfer the tokens to the wallet we have defined (and who will be the owner of the token). We can now go back to the wallet which is specified, to see if the tokens have been transferred:



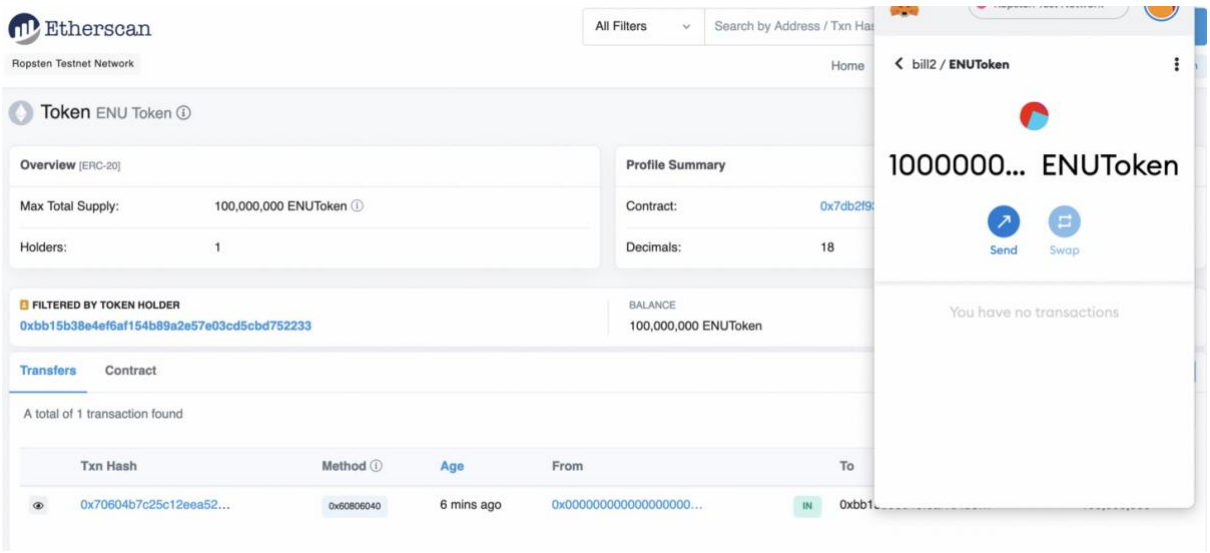
Next, we can transfer the tokens into our wallet, by defining the contract address:



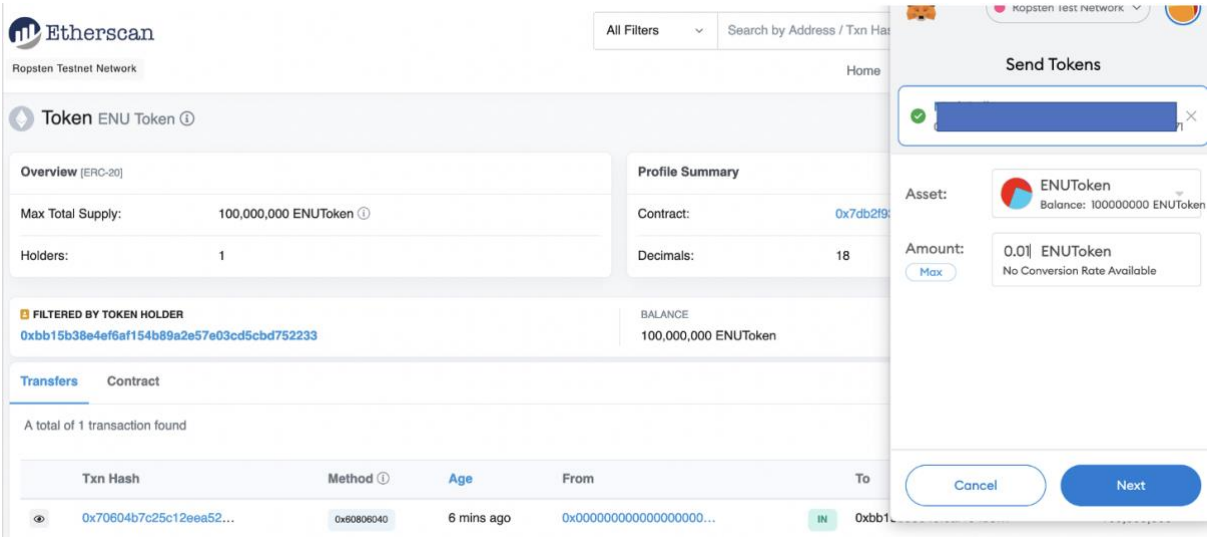
We will now have our new tokens in the wallet:



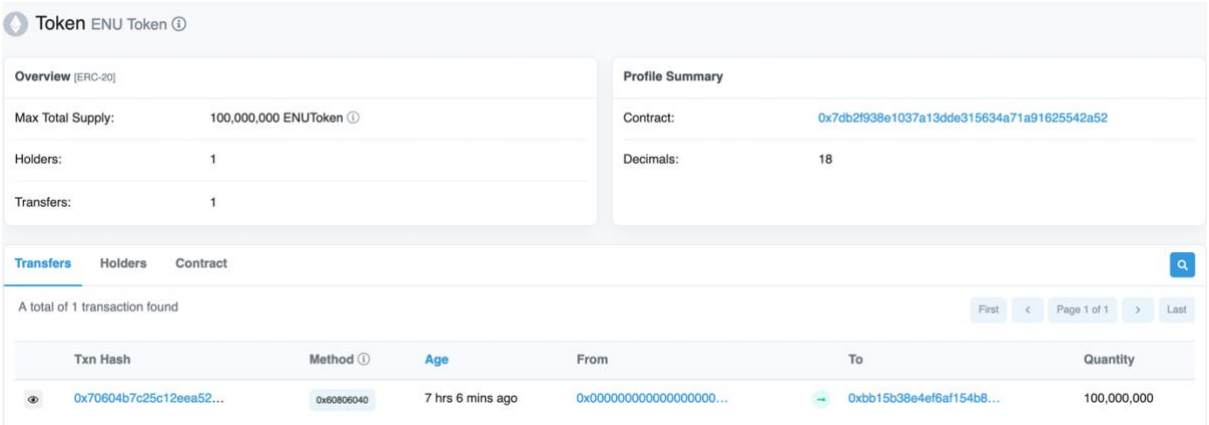
And with:



We can now transfer the cryptocurrency to another wallet:



We can view the ENUToken: here]:



Now answer the following:

- Do you see the tokens in your wallet?
- Now send 0.1 of your token to someone else's wallet. If you want, you can send to your tutor's wallet. Bill's wallet is 0xbb15b38e4ef6af154b89a2e57e03cd5cbd752233
- Did they receive the token?