# Lab 8: Bitcoin and Blockchain

## 1 Details

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

## 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.** 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.** Download and created 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:**

## Setting up your Ethereum wallet on Sepolia

The Sepolia network allows a user to test an Ethereum application, and using free Ether. Initially setup your MetaMask wallet. A document to outline how you set this up is [here](https://github.com/billbuchanan/appliedcrypto/blob/main/unit08_blockchain/lab/Metamask.pdf). 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](https://sepolia.etherscan.io/address/0xbb15b38e4ef6af154b89a2e57e03cd5cbd752233).
* 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;

}

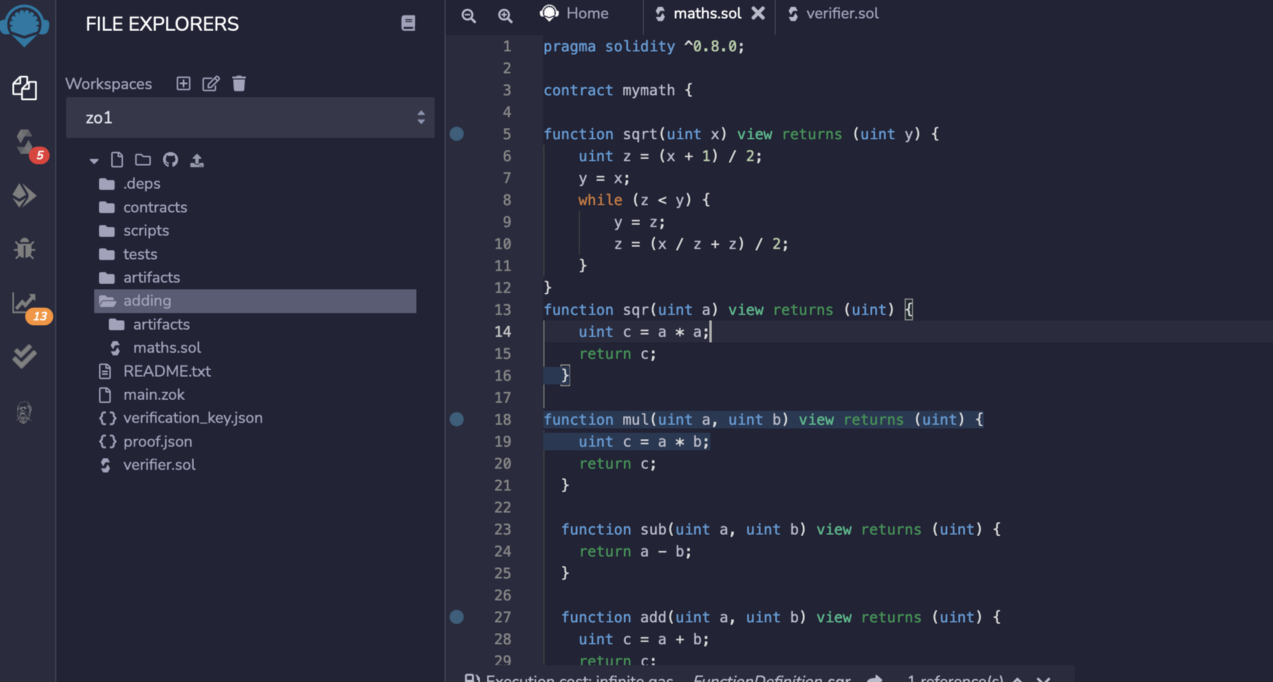
function add(uint a, uint b) public view returns (uint) {

uint c = a + b;

return c;

}}

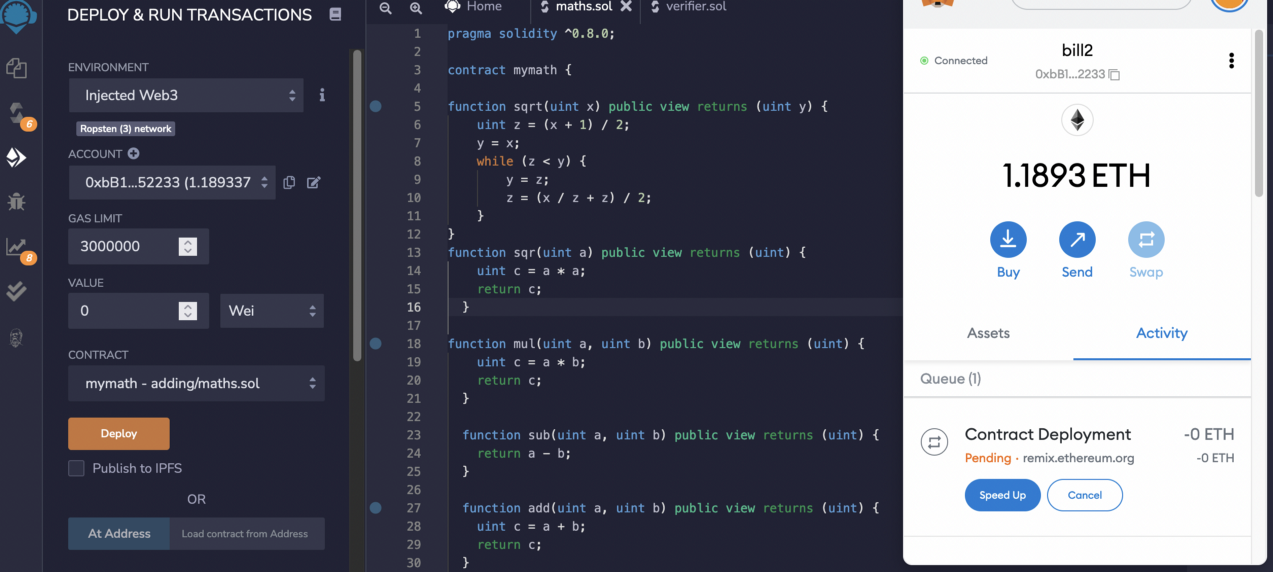
In this case, the "public" part makes sure we can see the output of the function, and the "view" part allows it to be stateless (and where we just have to receiver the value without the smart contact remember the state). On Ethereum we normally use the Solidity language to create a smart contract and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

[](https://camo.githubusercontent.com/3984d3518ad70445fd58753747ebfa7411319834ca4098b4db759a8eed27045a/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030312e706e67)

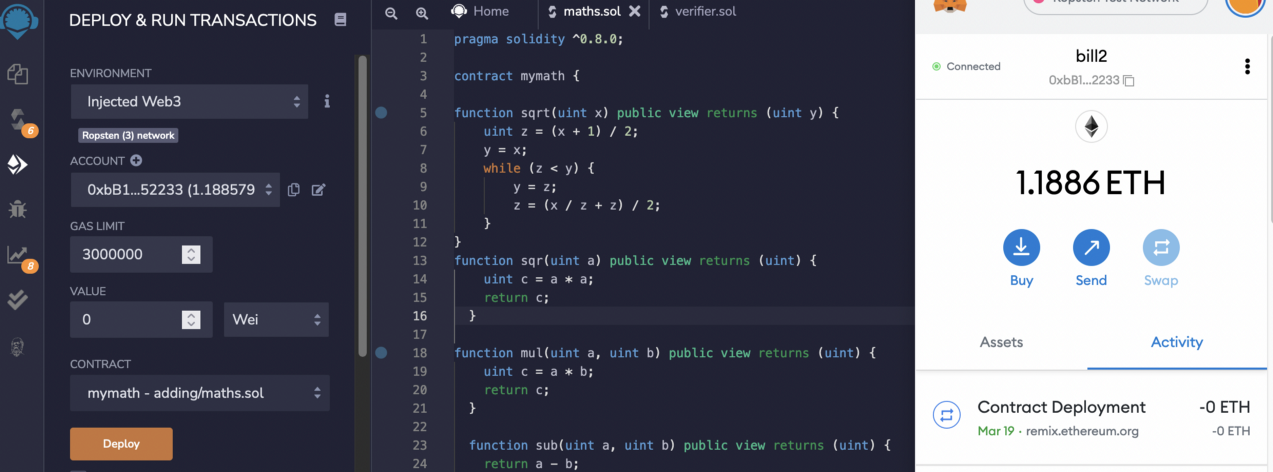
Once entered, we can then compile it with the Solidity compiler. It is important to take a note of the compiler version at this stage, as we will need this later:

[](https://camo.githubusercontent.com/3fba6812bdfa62277bea3c292eb283f25744a073dac728529368d33e54d8a391/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030322e706e67)

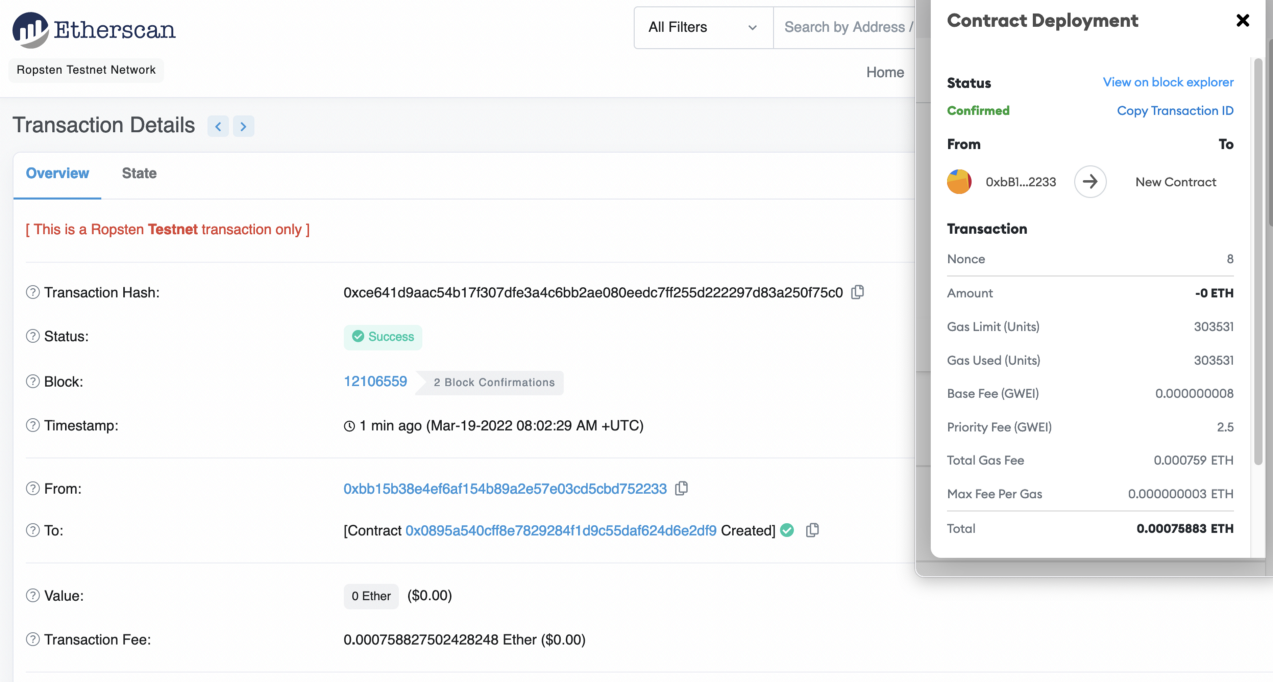
Once compiled we can then deploy the smart contract to a test network (Sepolia). For this, we need to connect our Metamask wallet:

[](https://camo.githubusercontent.com/d05ff4f5868040cc8199ce0712ed5285c44254858f77273112c93ccd43e138c9/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030332e706e67)

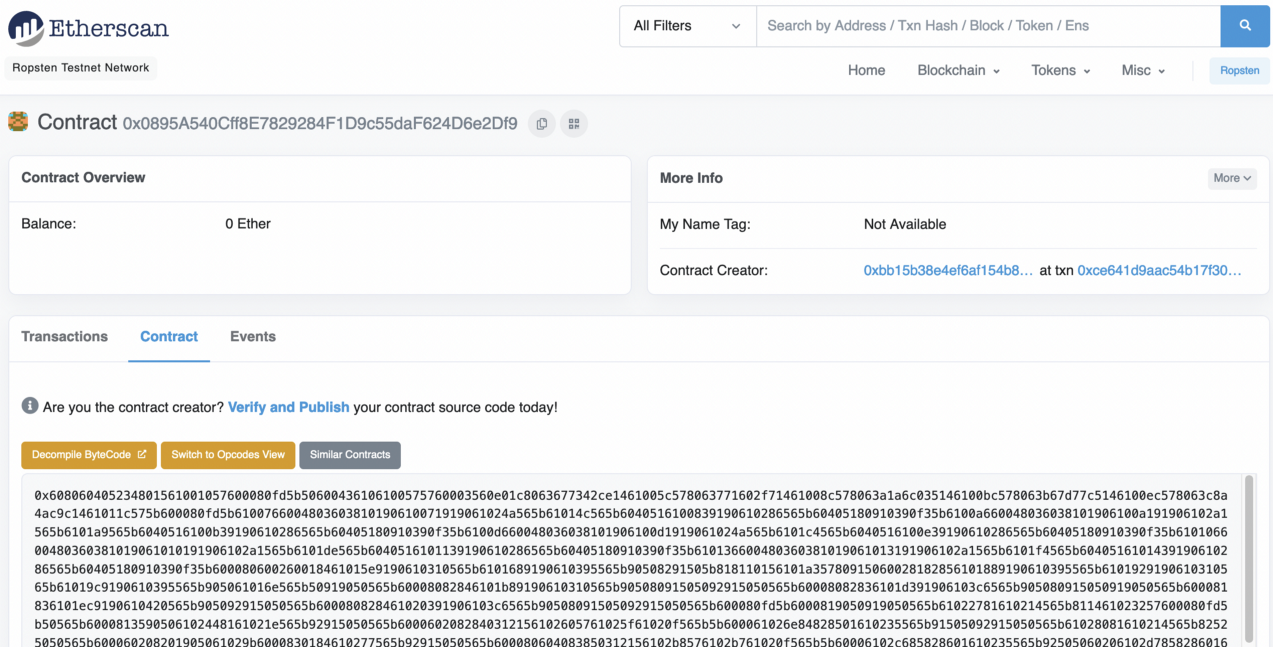
Once it has been deployed, we can see our wallet identifies the deployed contract:

[](https://camo.githubusercontent.com/5f986d3f7b49c1face4e4f65a8a38ca8075b5e16be50f06a15c72118019a8baf/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030342e706e67)

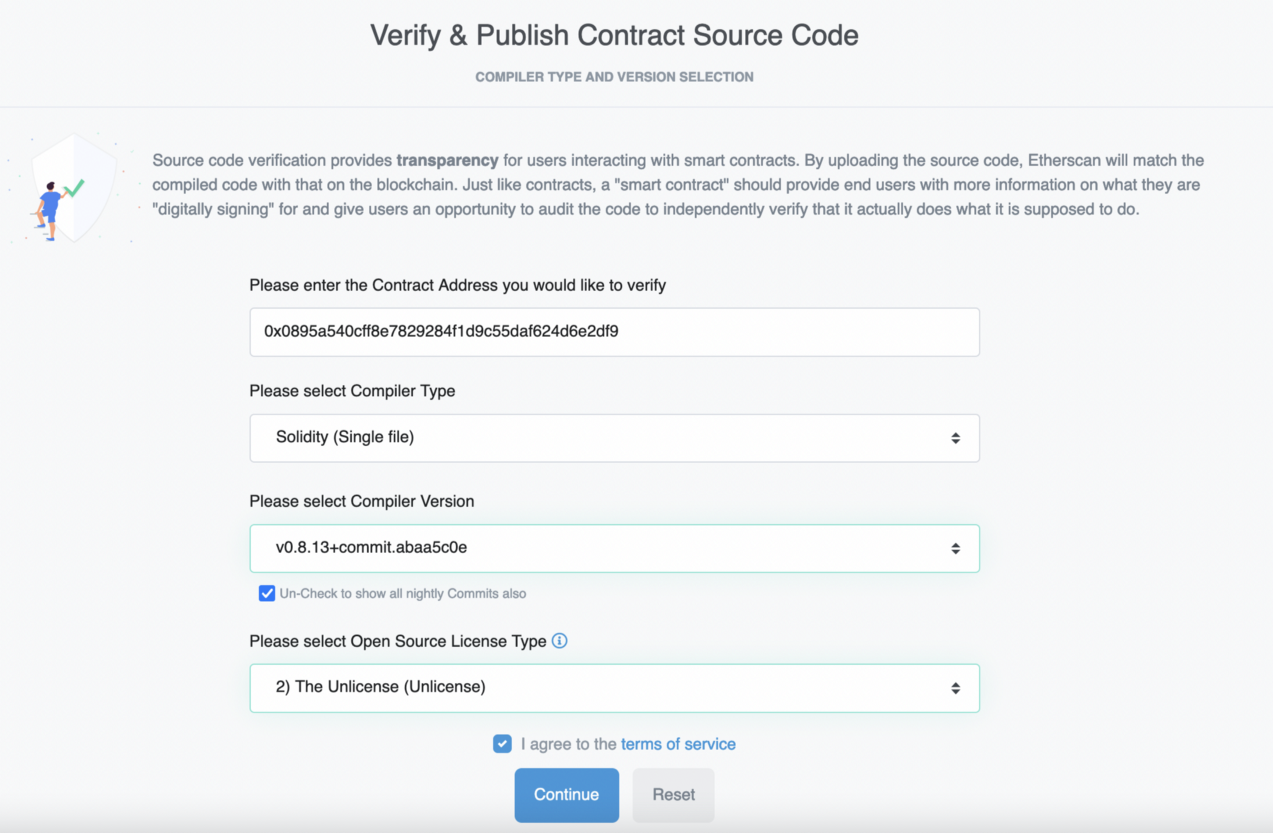
And clicking through gives us the address of the contract, and then viewing it on the explorer, we can see the transaction:

[](https://camo.githubusercontent.com/b7cfd3f713a3e30d7fb54562e8bff47a277459549b4f8a3c5e4bea2f4ea3a5a7/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030352e706e67)

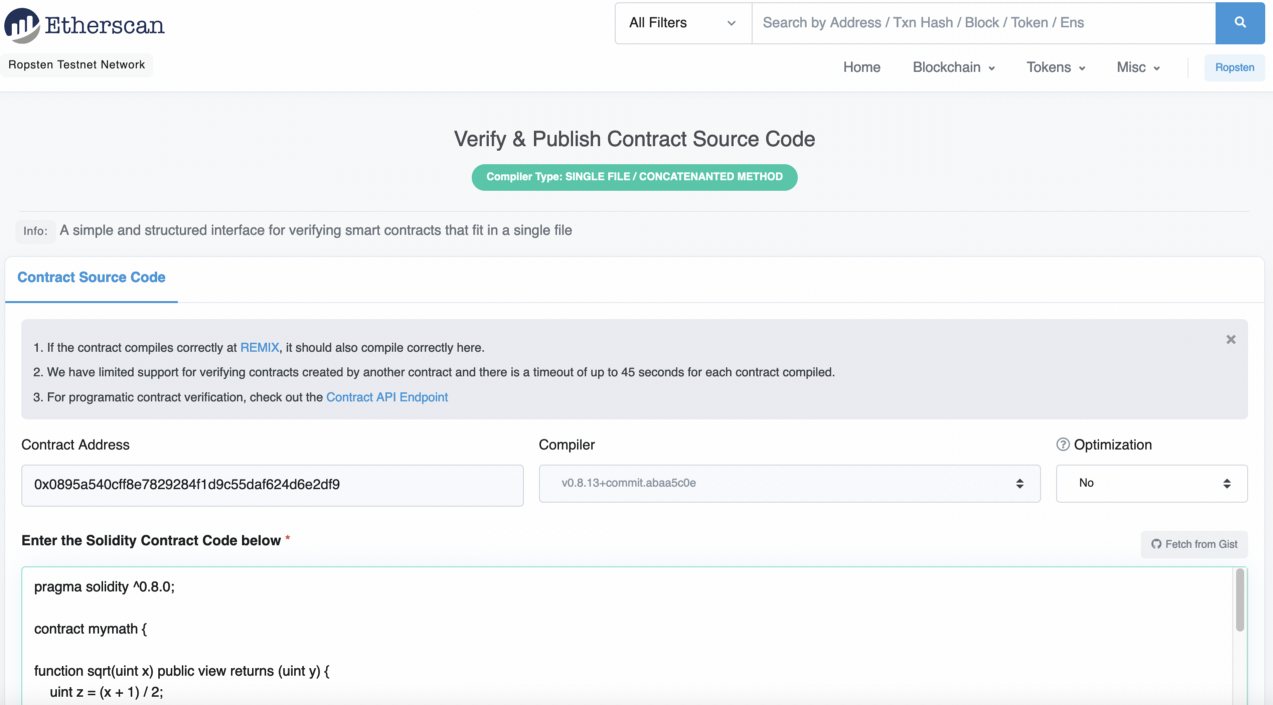
The address here is “0x0895..”, so we can view the smart contract from: [here](https://ropsten.etherscan.io/address/0x0895a540cff8e7829284f1d9c55daf624d6e2df9). We now need to verify and publish the contact, with click on “Verify and Publish”:

[](https://camo.githubusercontent.com/e19fd31fb737bba9415de968b44746cf27be0c5f51dd579daa36bdb129b97a2f/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030362e706e67)

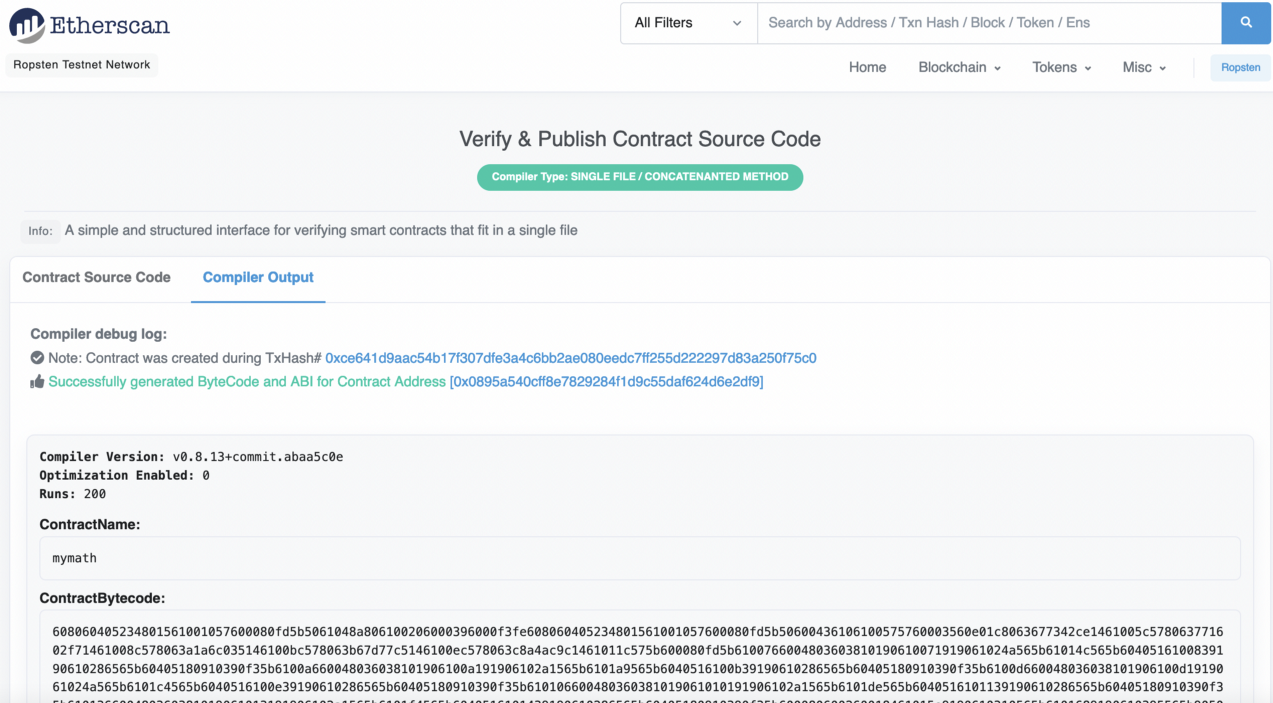
After this, we can define the Compiler Version and the licence

[](https://camo.githubusercontent.com/39d21df34d7a37cbfe0203a010a02854520831b7a46c5f0e08f572a2c5bfbbdc/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030372e706e67)

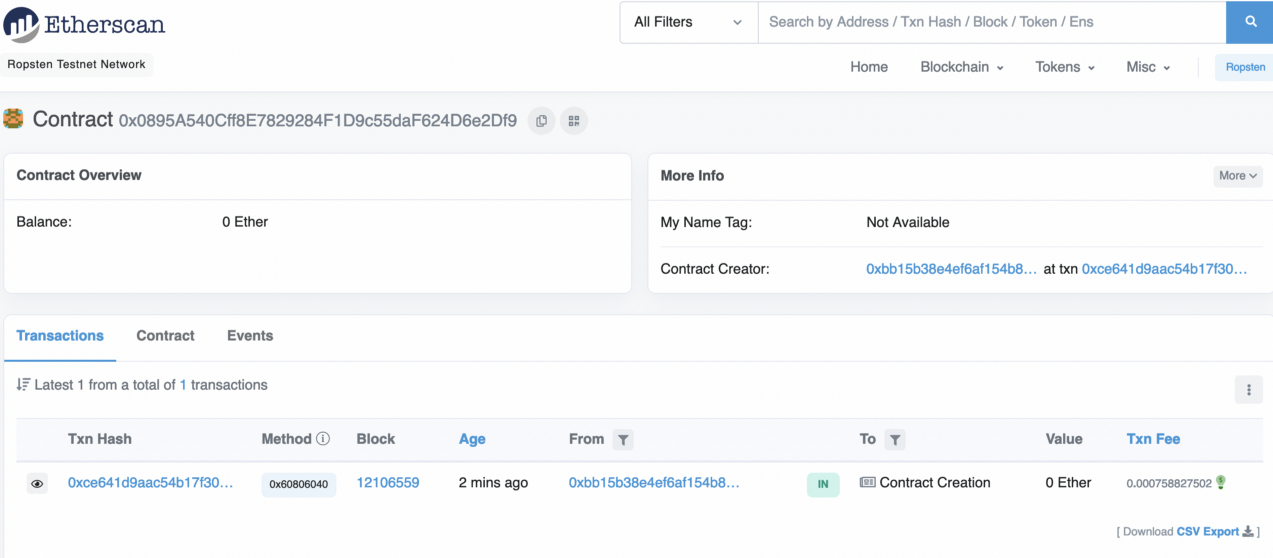
We then need to add your code for it to be checked:

[](https://camo.githubusercontent.com/70345e373f90166213e19bc1264cfd2ffb012ab1c8269cf63babdacd437bf7d2/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030382e706e67)

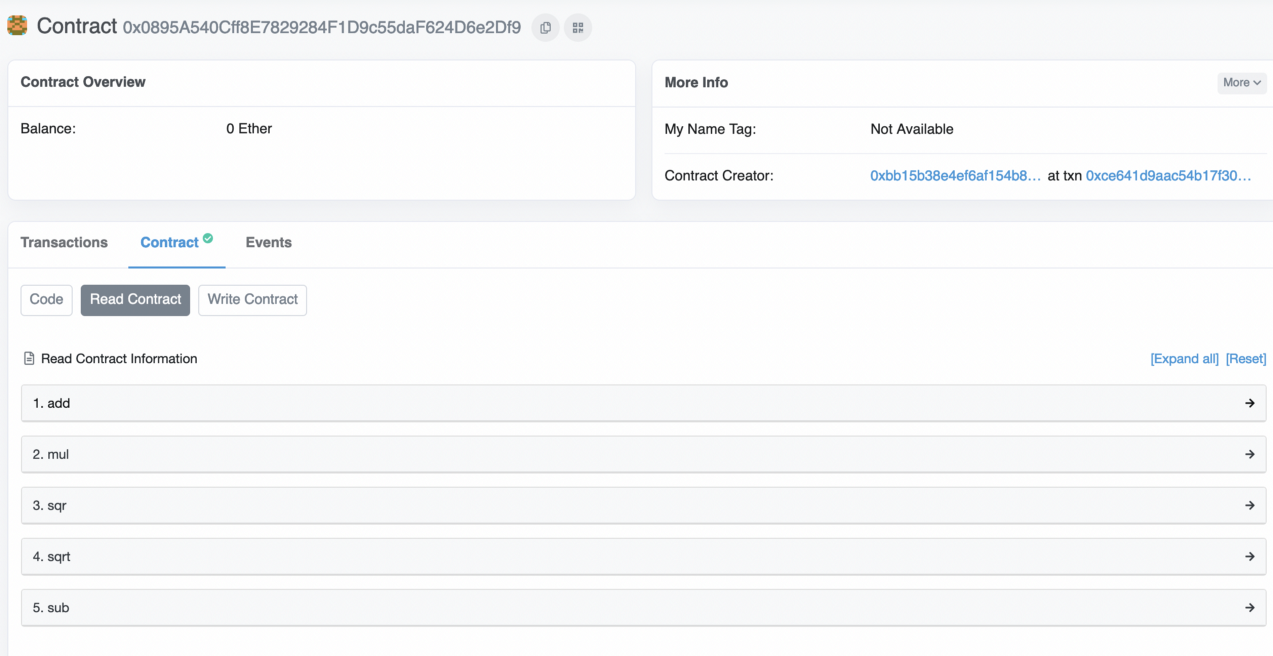
It takes around 30 seconds, but, eventually, we should have our code accepted:

[](https://camo.githubusercontent.com/5145e214fe7234427c05357567b0d130ffeea83dbfce6c9ff2396ec6e9845e9d/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683030392e706e67)

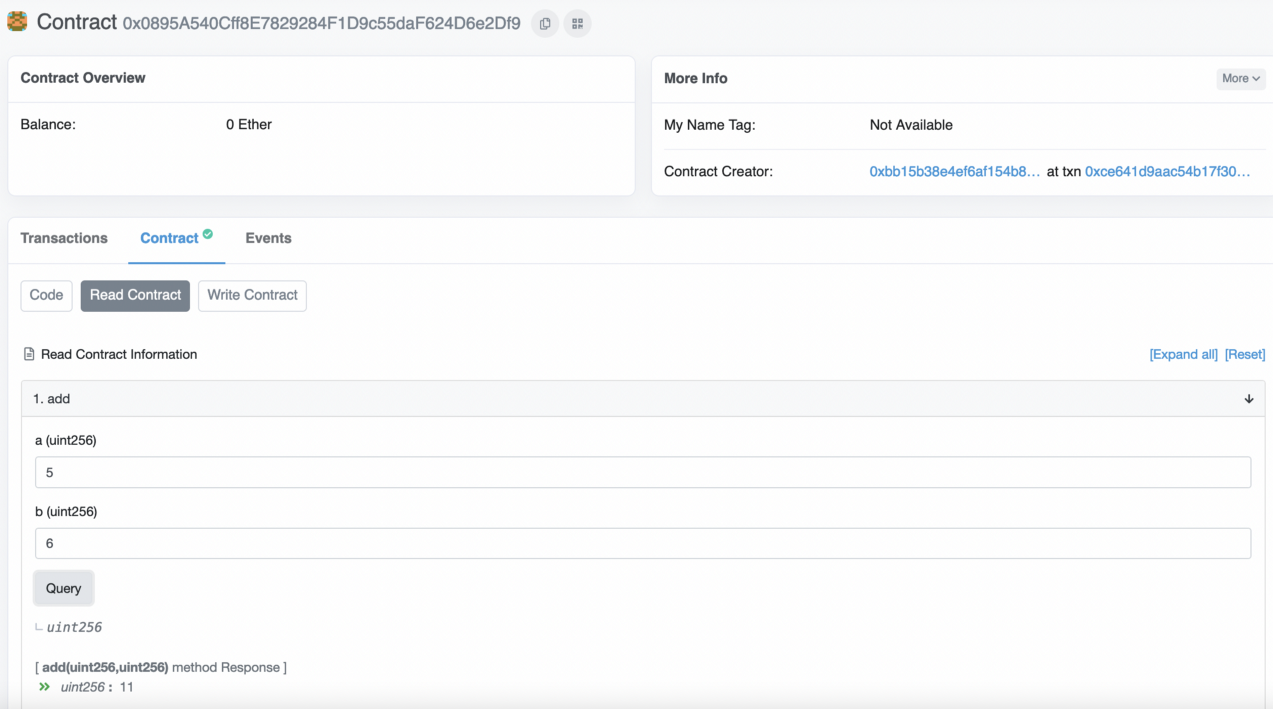
We now have the contract published to the Ropsten test network:

[](https://camo.githubusercontent.com/e0cb4ce8fb9db0e81f61c3d86b29d1e1babff5f606d3e65bba69447786536f07/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683031302e706e67)

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

[](https://camo.githubusercontent.com/7950565596c1c2fc25e500313dcf9f6f8fa56dc8b133ab0380792d5c8d1a3c9c/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683031312e706e67)

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

[](https://camo.githubusercontent.com/fdcc8e1859d911fddc5e875b26d07f721dd41a0df6b49bbcacd491d642307b42/68747470733a2f2f617365637572697479736974652e636f6d2f7075626c69632f6574683031322e706e67)

Note:You can get Ether for our wallet [here](https://faucet.metamask.io/)