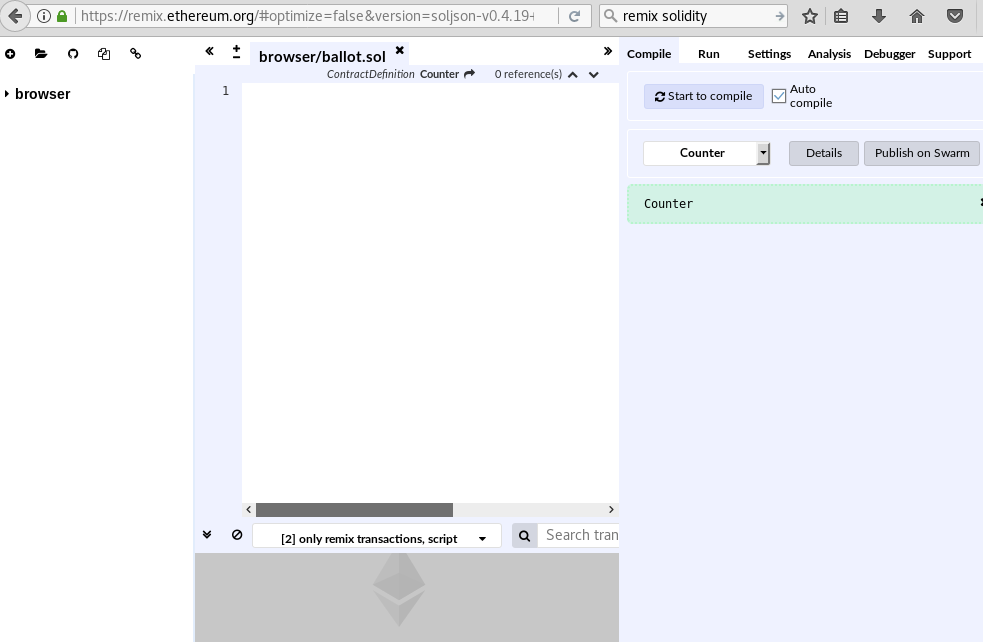
# Exercises: Build Smart Contract

The goal of this exercise is to build a simple counter smart contract. The contract will be written in **Solidity**. Before we get started, you must have **Ganache** installed, which will be the tool we will use to create a **private blockchain** that runs on our machine only. You will also need to download your own copy of **MyEtherWallet**. Throughout this exercise, we will launch a blockchain on our own machine, deploy a smart contract to it, and interact with the contract. This is very similar to the workflow of professional Ethereum smart contract developers.

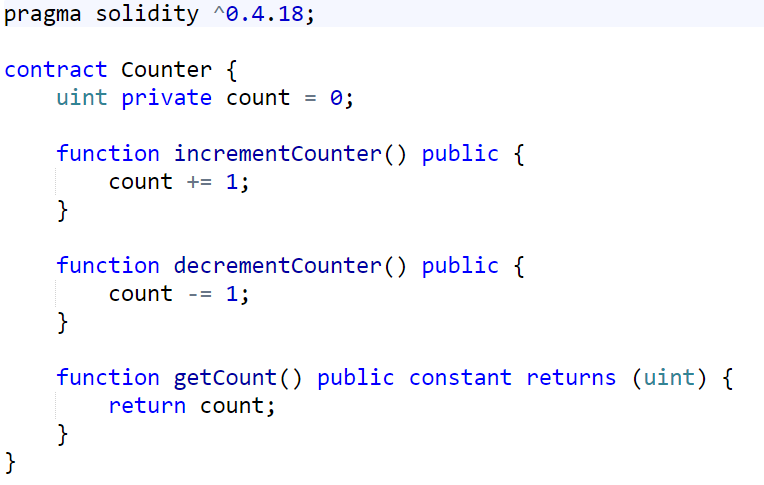
This exercise is based on this project: <https://medium.com/crypto-currently/build-your-first-smart-contract-fc36a8ff50ca>. Thanks to the original authors.

## Deploying a contract and playing with it

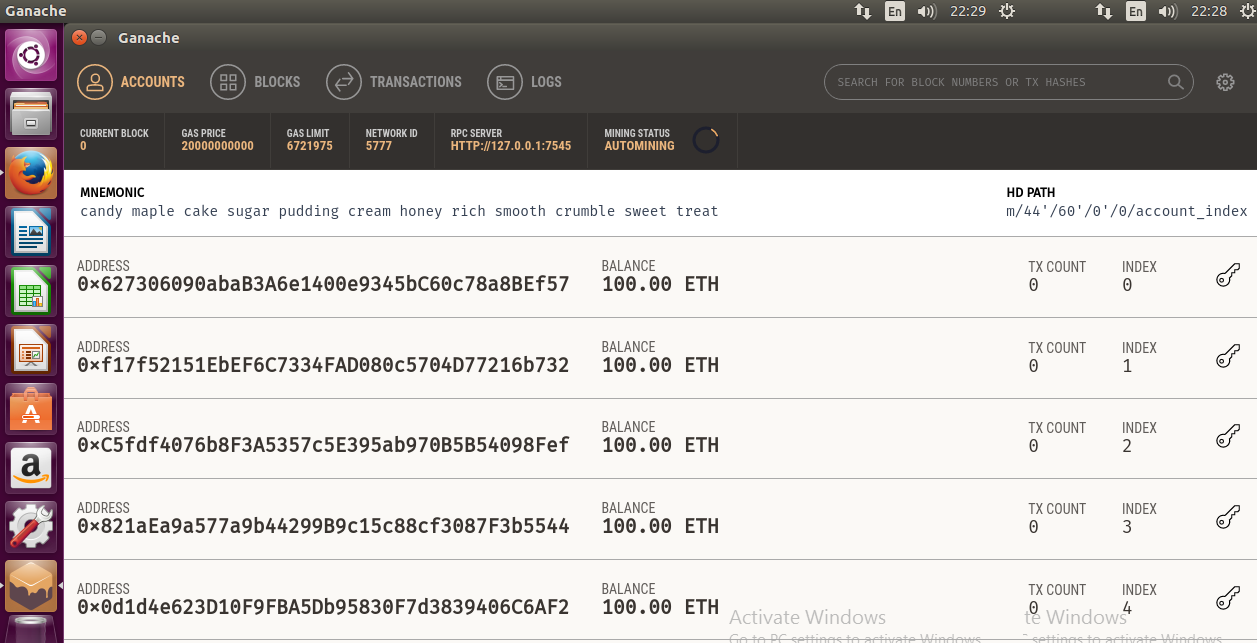
1. **Remix** is online compiler for Solidity. It is what we will use to write our **smart contract code**. Visit site <https://remix.ethereum.org/#optimize=false&version=soljson-v0.4.19+commit.c4cbbb05.js>



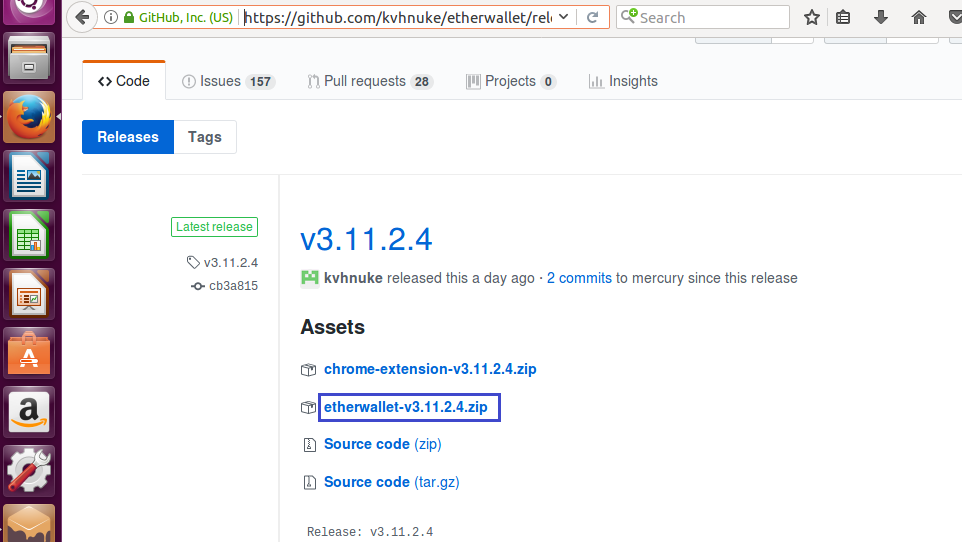
1. Write the following solidity code in Remix. Contract has one **variable** and three **functions**. The variable, **count**, is an unsigned integer that is private, which means that it cannot be accessed by anyone outside of the contract itself. The first function, **“incrementCounter()”**, changes, the value of **count** by incrementing its value. The second function, **“decrementCounter()”**, mutates the value of **count** by decrementing its value. In addition, the third function -**“getCount()”,** accesses **count** and returns its value. Function is **public**, which means whoever or whatever can call the function.



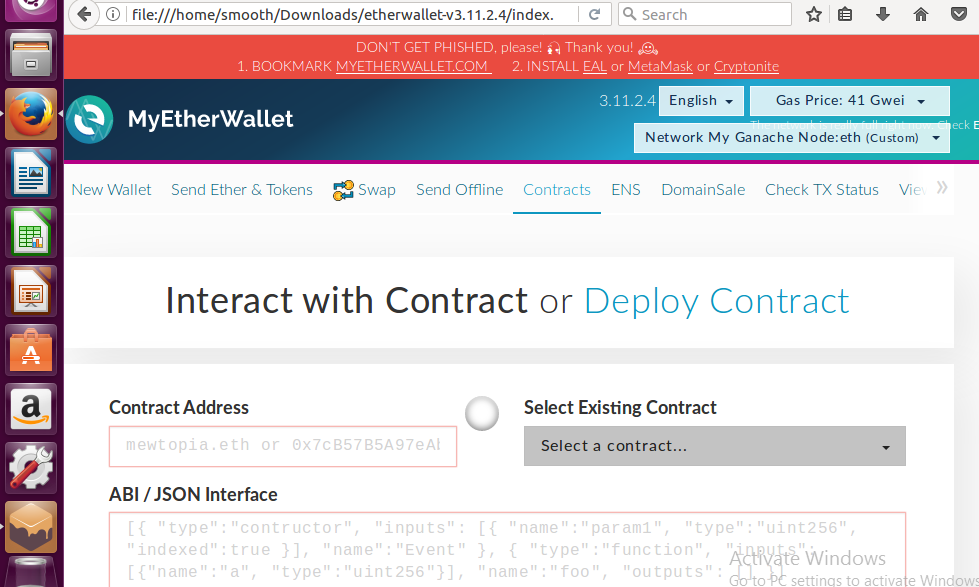
1. Now, open **Ganache** and you will see something like this.



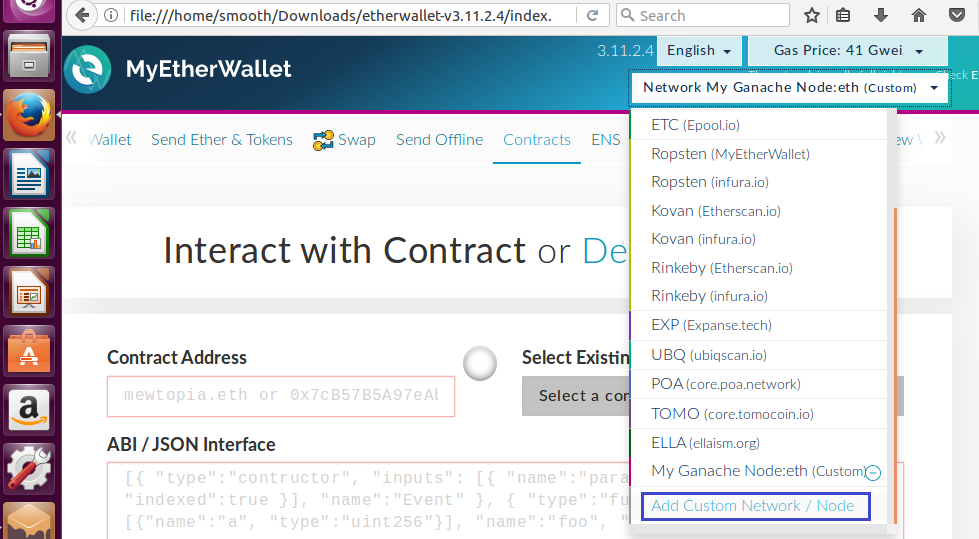
1. Go to <https://github.com/kvhnuke/etherwallet/releases/tag/v3.11.2.4> and download **MyEtherWallet.**



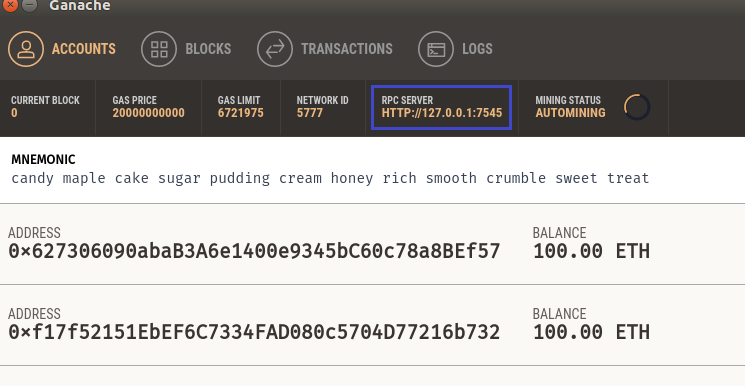
1. Now, unzip your **MyEtherWallet** download and open the folder. Then, open the **index.html** file in your browser to see the following screen.



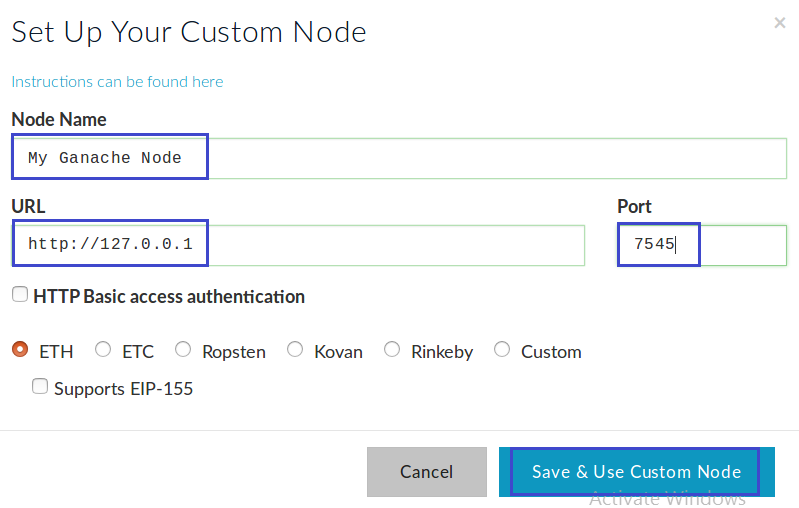
1. From dropdown menu in the top right corner change the **Ethereum network** to connect to. By default, it connects to the Ethereum (ETH) main network (mainnet). We want to change this to **[Add Custom Network/Node]**.



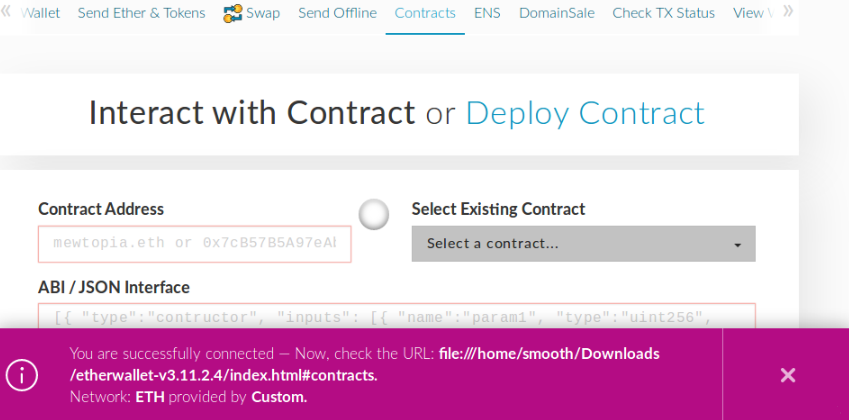
1. Now you must take the **RPC Server information** from **Ganache.**



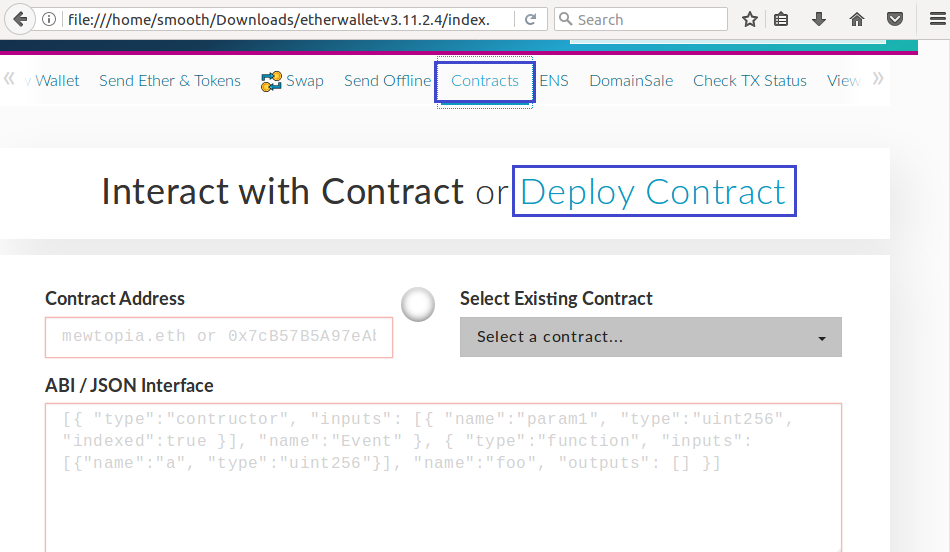
1. And fill in the information:



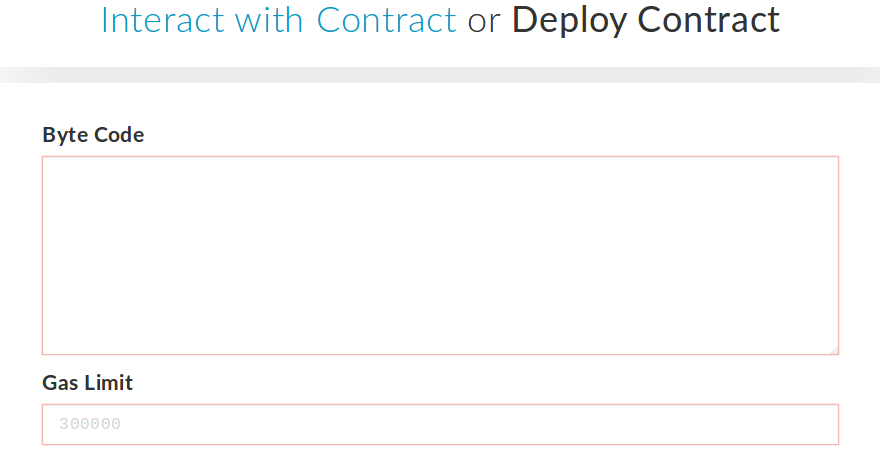
1. **MyEtherWallet** is now connected to your self-hosted blockchain through **Ganache**. You must see something like this:



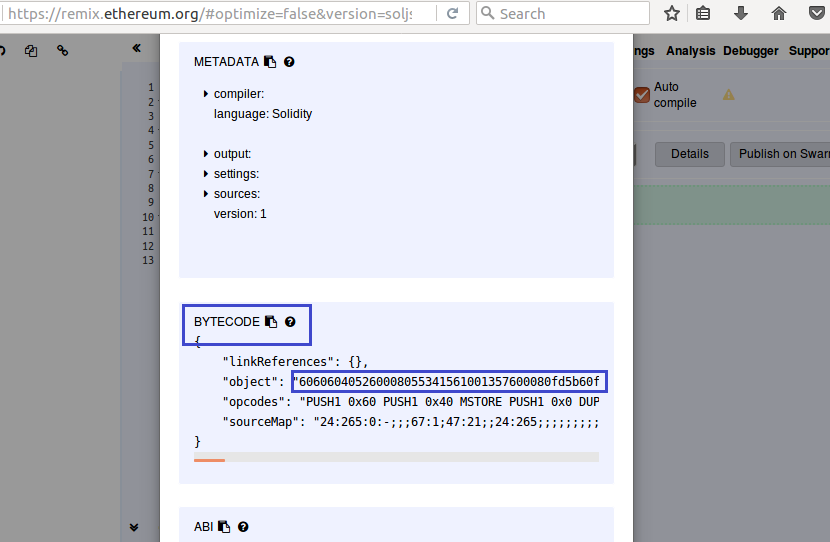
1. Now, use **MyEtherWallet** to upload **Counter** smart contract to our blockchain. Click **[Contracts]** in MyEtherWallet’s top navigation bar and select **[Deploy Contract]**.



1. Now, MyEtherWallet asks us for the contract’s **byte code**.



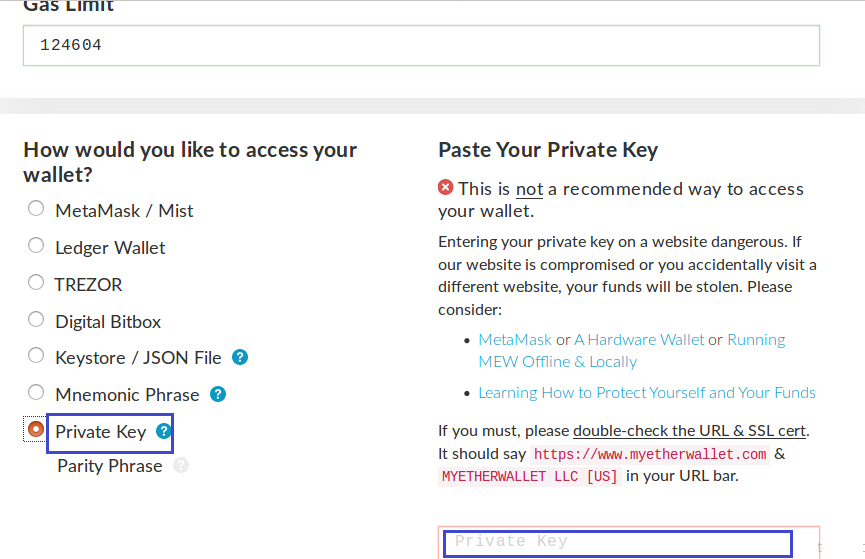
1. You can find the **byte code** in **Remix IDE.** Go back to Remix and click the **[Details]** button
2. Copy **byte code** number. This will help you to connect with local blockchain.



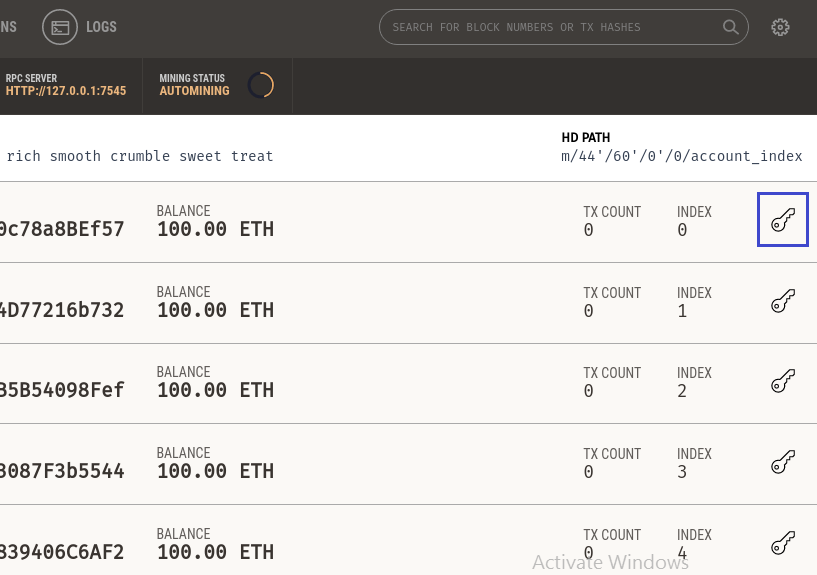
1. Now go back to **MyEtherWallet** and paste the **byte code** into the dialog box. You will see the **Gas Limit** calculated automatically.



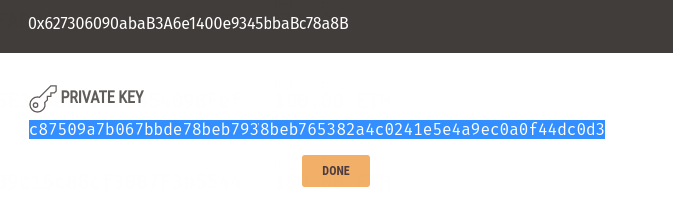
1. Now you must import an **account** to upload the contract with it. Select **[Private Key].** Here in local Ganache blockchain, with test accounts, this is secure, but in the real world, you must be very careful where you import your private keys.



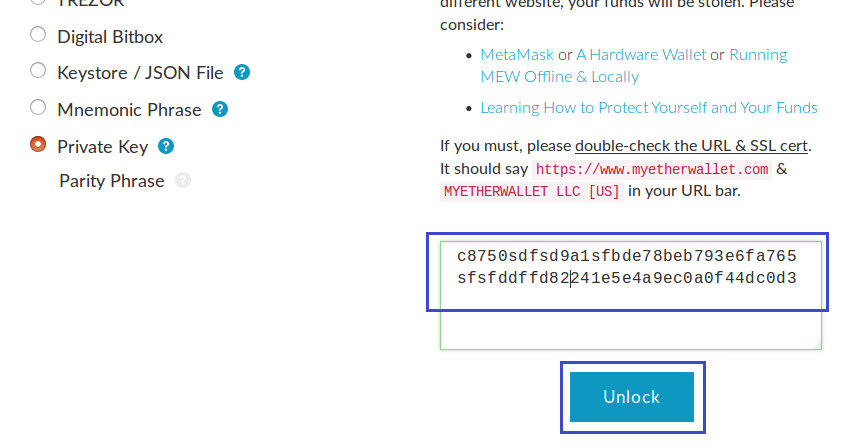
1. Now go to Ganache. Here you have 5 addresses that can use to interact with our private blockchain. Click the key icon for any of the addresses.



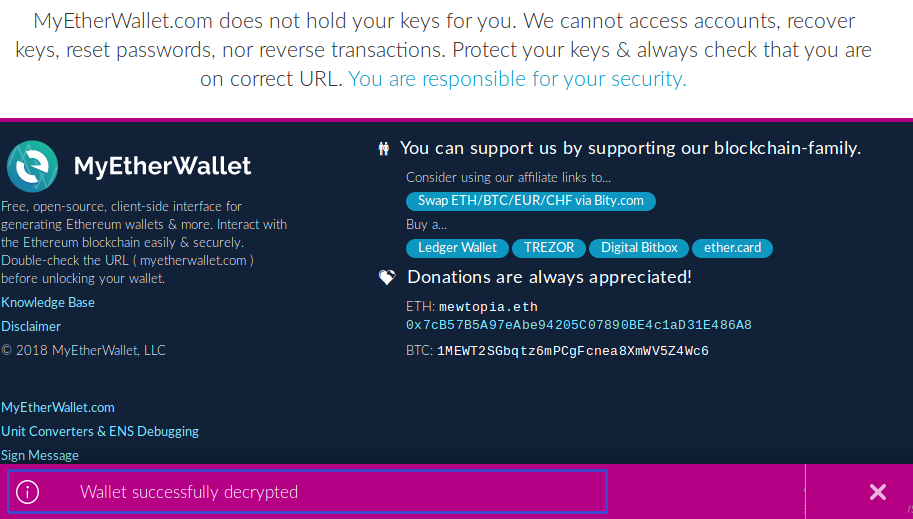
1. Copy the **Private Key**.



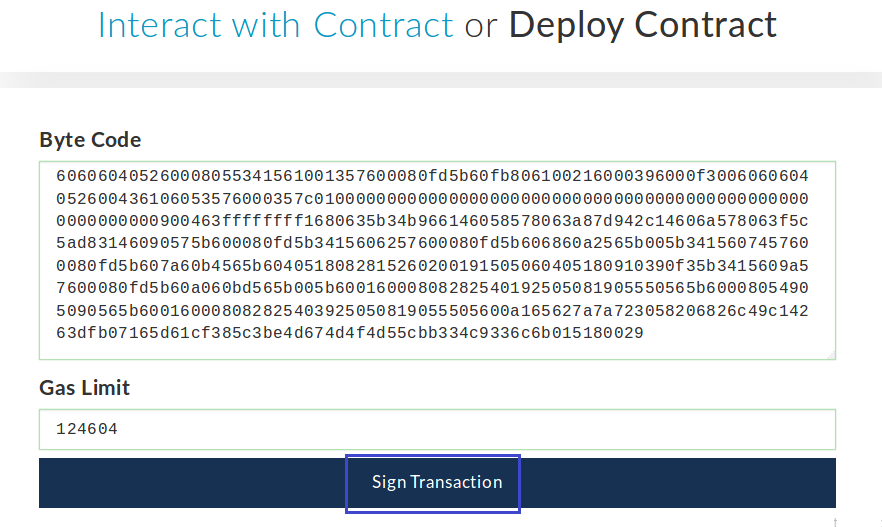
1. Paste the private key into **MyEtherWallet**. Click **[Unlock]**.



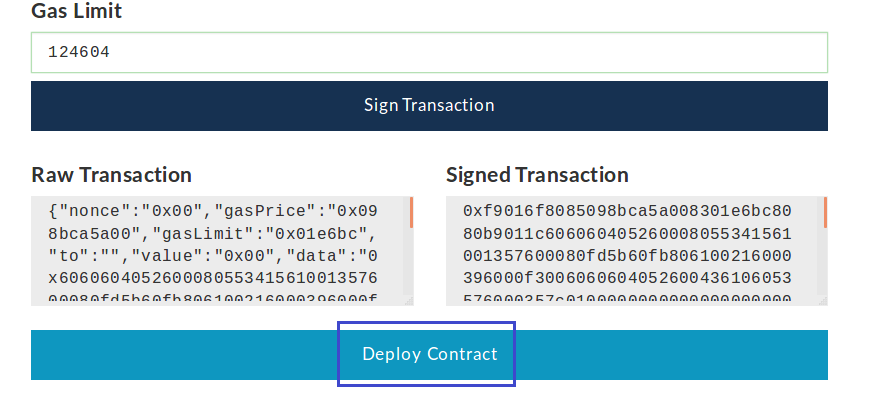
1. You must see the message **“Wallet successfully decrypted”**.



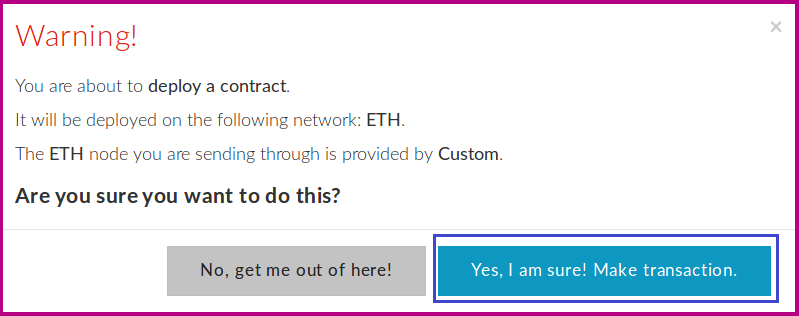
1. Now you must **sign** the transaction.



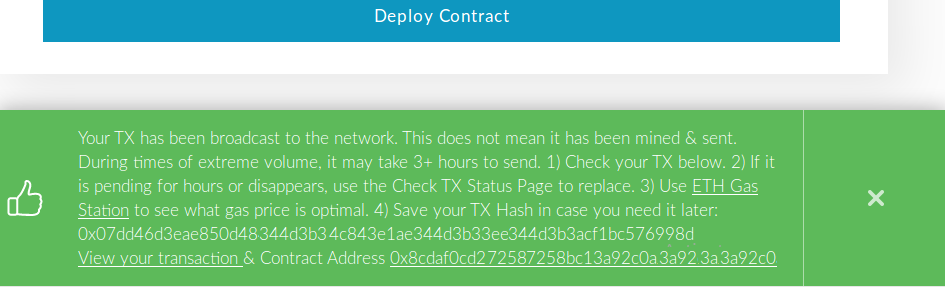
1. Then **deploy** the contract.



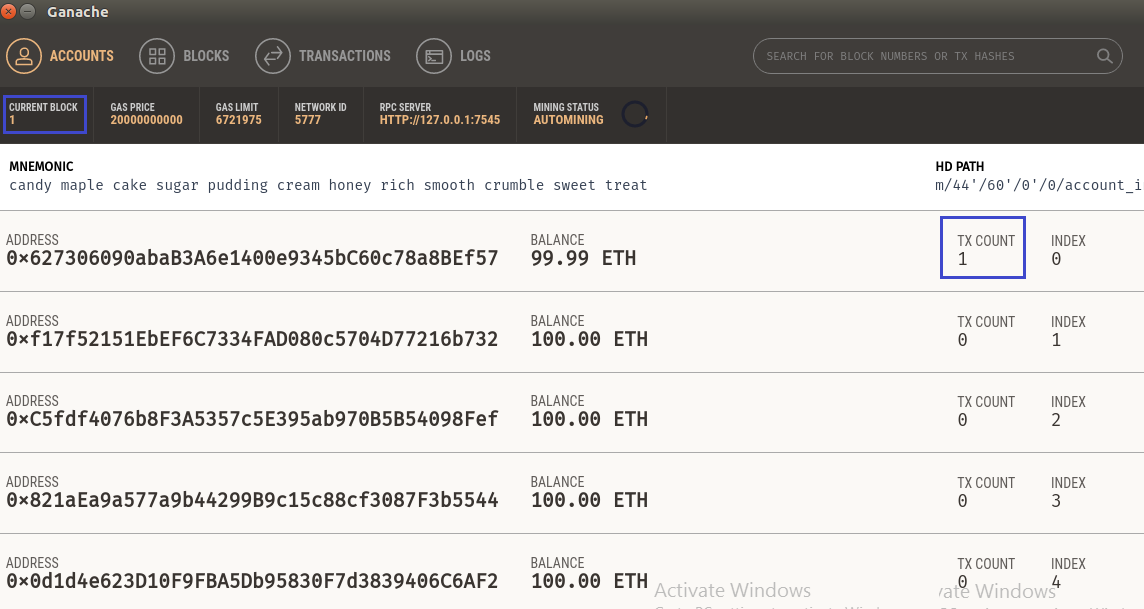
1. You will see the warning message. Answer **[Yes…]**.



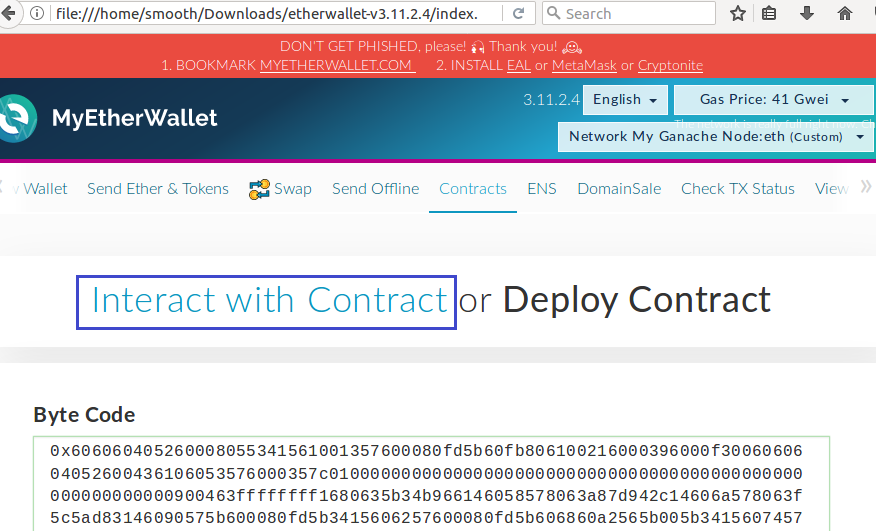
1. If the transaction occurred successfully, you will see the green message.



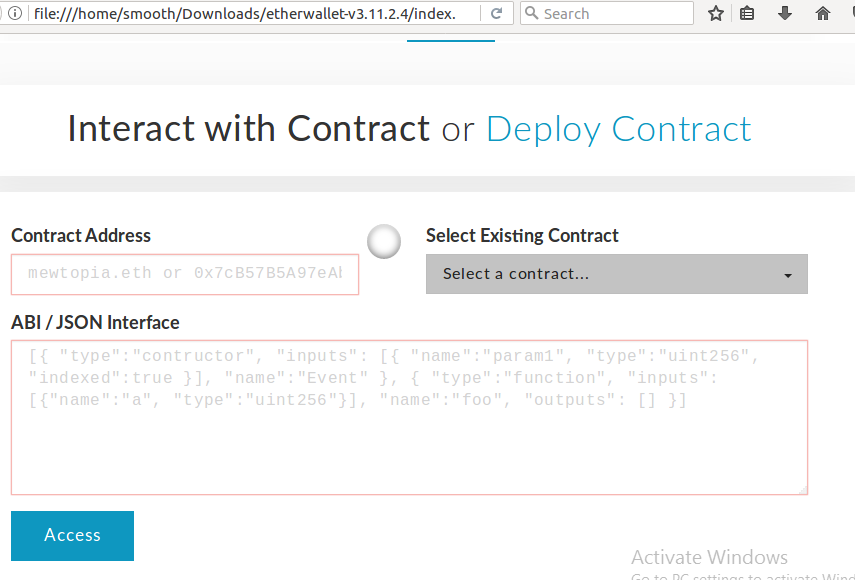
1. In the same time, Ganache will increment its “**Current Block**” value and the **transaction count** of the account that we used to deploy the contract also increment.



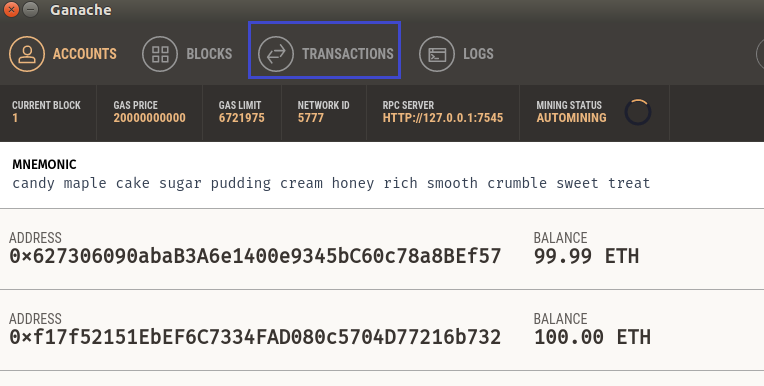
1. Contract is now **uploaded** to our **blockchain**. To **interact** with it by **incrementing** and **decrementing** the counter, we can go back to MyEtherWallet and select **[Interact with Contract]**.



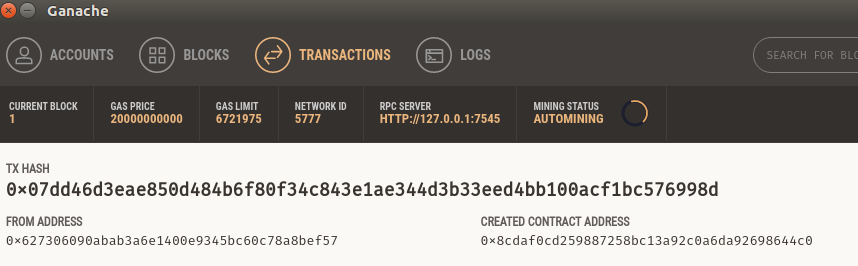
1. MyEtherWallet now asks for the **address** at which our newly deployed contract **resides** and the **Application Binary Interface** (ABI) of our contract.



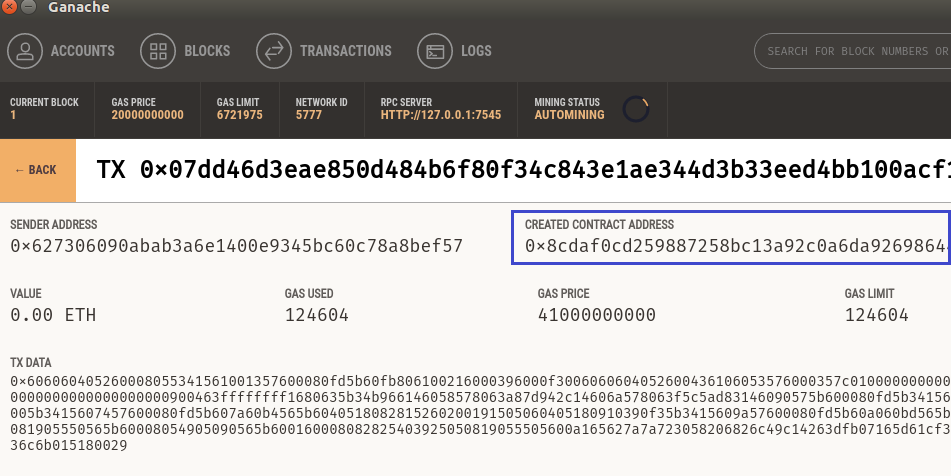
1. To find the contract address, we can go back to **Ganache** and view our **transactions log**.



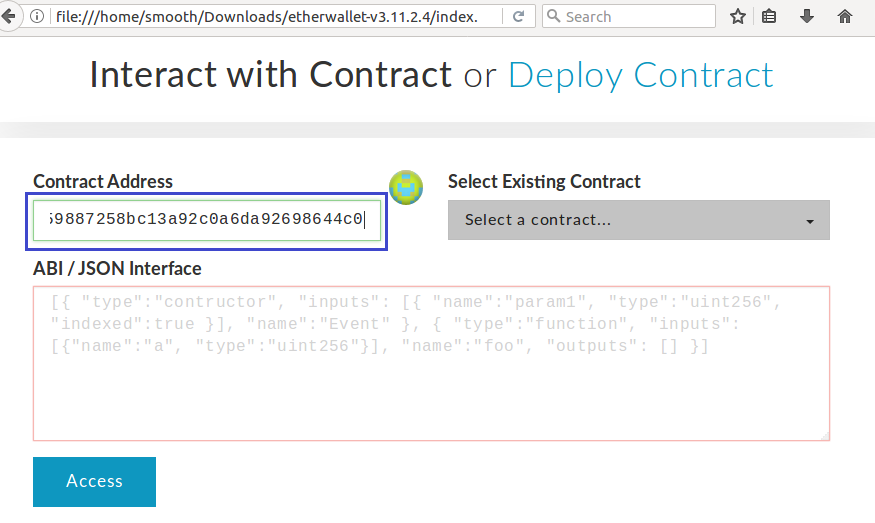
1. This page shows us the transaction that we had created earlier when we deployed our contract. As you can see, Ganache tells us the **address** we had used to **deploy** the contract, the **address** of the **contract on our blockchain**, and more information about the transaction. Let’s **click** the transaction.



1. **Copy** the created contract address**.**



1. Then **paste** it into MyEtherWallet.



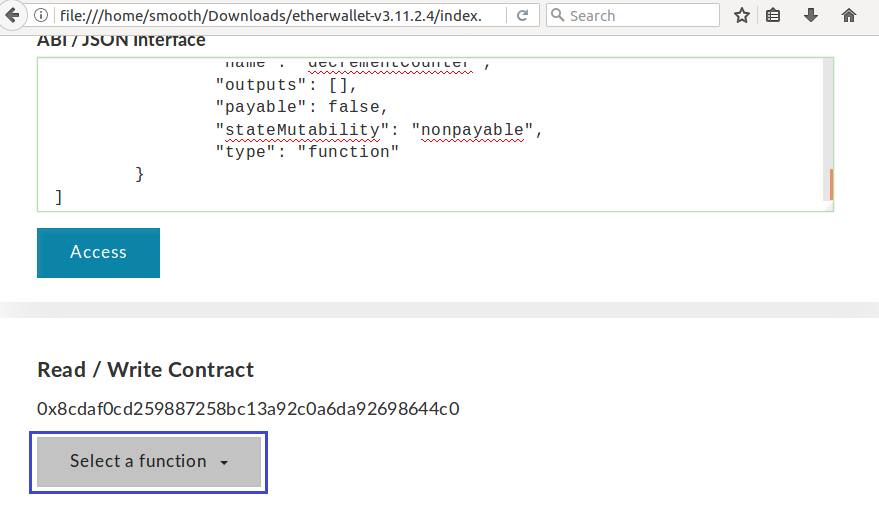
1. All that is left is that we need is the ABI. This is what tells MyEtherWallet how to **interact** with our contract. To get it, we will go back to **Remix** and click the clipboard icon next to “**ABI**” to copy it.



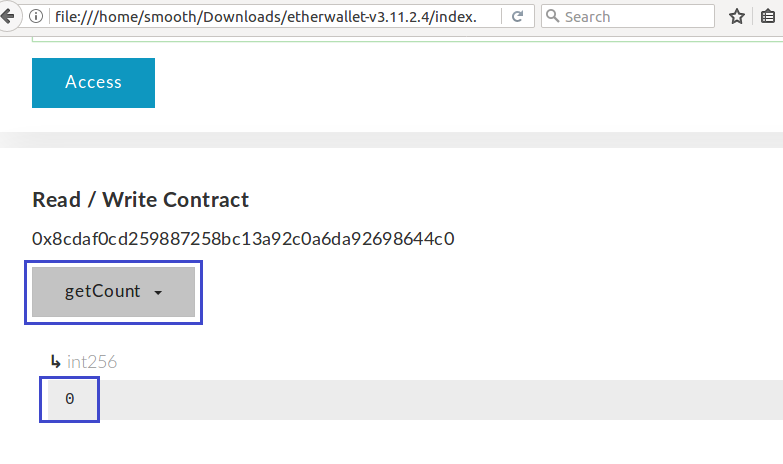
1. Now we can go back to **MyEtherWallet**, paste the ABI into its text box, and click the **[Access]** button.



1. Beneath the **[Access]** button, it will appear a section called **“Read / Write Contract”**. With it, we can **interact** with our contract by clicking the **[Select a function]** dropdown.



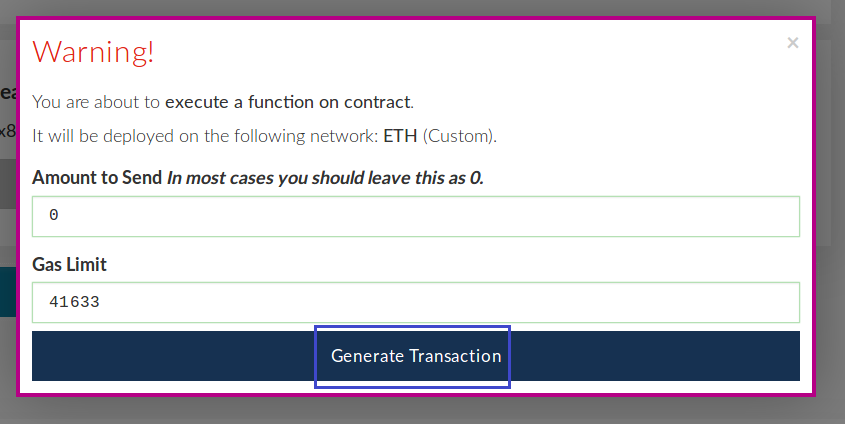
1. In our code, we set **count**‘s initial **value** to **0**. To confirm that the contract is working properly, let’s call the **“getCount()”** function.



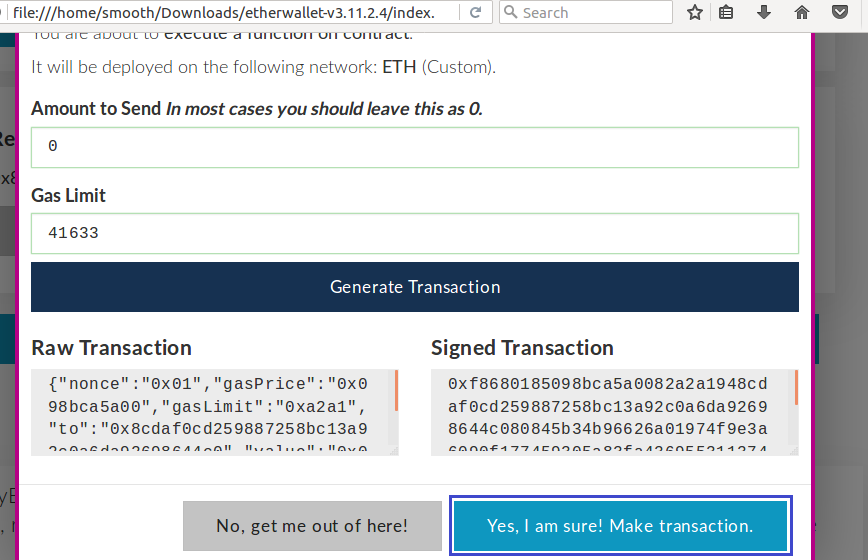
1. You made it! Our contract returned **0** when getting the value of count before changing it. However, we also have two other functions, **“incrementCounter()”** and **“decrementCounter()”**. Let’s call **“incrementCounter()”** to test it. We will do this by selecting the function dropdown again, selecting **incrementCounter**, and creating a new transaction.



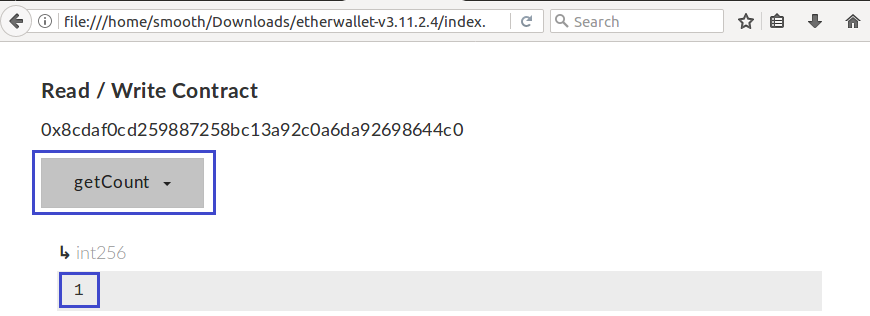
1. You will see warning message. Click **[Generate Transaction]**.



1. Here **MyEtherWallet** shows you the **Raw Transaction** and the hashed **Signed Transaction**. Click **[Yes, I am sure! Make Transaction!]**.



1. This just **incremented** the value of **count**s. Now we can call **“getCount()”** again to confirm whether the value actually changed.



1. Congratulation! Count is now equal to 1! So, our **“incrementCount()”** function works.

## Decrement Count

1. Test **“decrementCount()”** of the contract we deployed in the task above, yourself.
2. What happens when you decrement the counter when it is 0? Think for a solution, deploy the new contract and test it.

# What to Submit?

Create a **zip file** (e.g. your-username-**build-smart-contract**-exercise.zip) holding the screenshots with your experiments. Make screenshots of **MyEtherWallet** results.

Submit your **zip** file as **homework** at the course Web site.