

Connecting our Ethereum private blockchain and interacting with it.

Tools:

1. A private blockchain: Setup and provided by the university.
2. MetaMask: Wallet.
3. Remix Ethereum: Online Solidity compiler.

Outline

- We show how to connect to our private blockchain and interact with it.
- Steps:
 1. Install MetaMask. Create an account (i.e. an address and public-private key) via MetaMask.
 2. Send us your account address, so we can give you some Ether.
 3. Get familiar with Solidity and the Remix compiler:
 - Write smart contracts, debug and compile them online.
 4. Send/deploy the latest version of the contract to the blockchain and interact with the deployed contract.

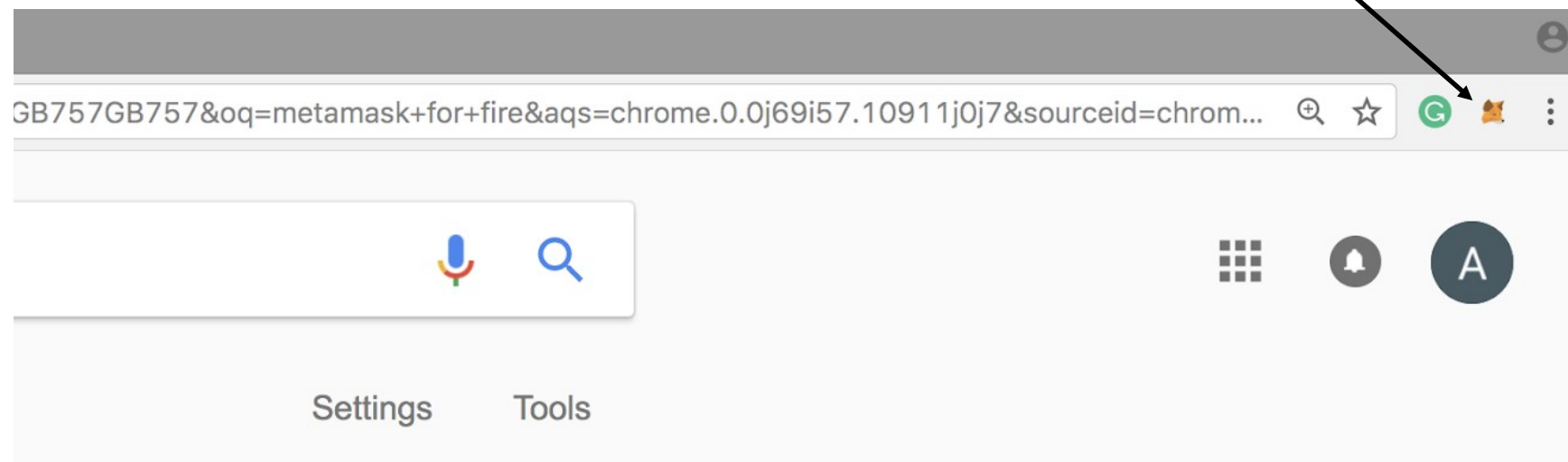
Step 1:

Install Metamask

- It is an extension for Firefox and Google Chrome.
- Allows us to create our public/private keys and connect to the blockchain.
- We recommend using MetaMask for Firefox or Chrome
 - Download it from:
<https://metamask.io/>
- Follow the instructions to install it.

Step1.1: Set Up an Account in MetaMask

- Click on the MetaMask icon on the top right side of your Firefox browser.



Step1.2:

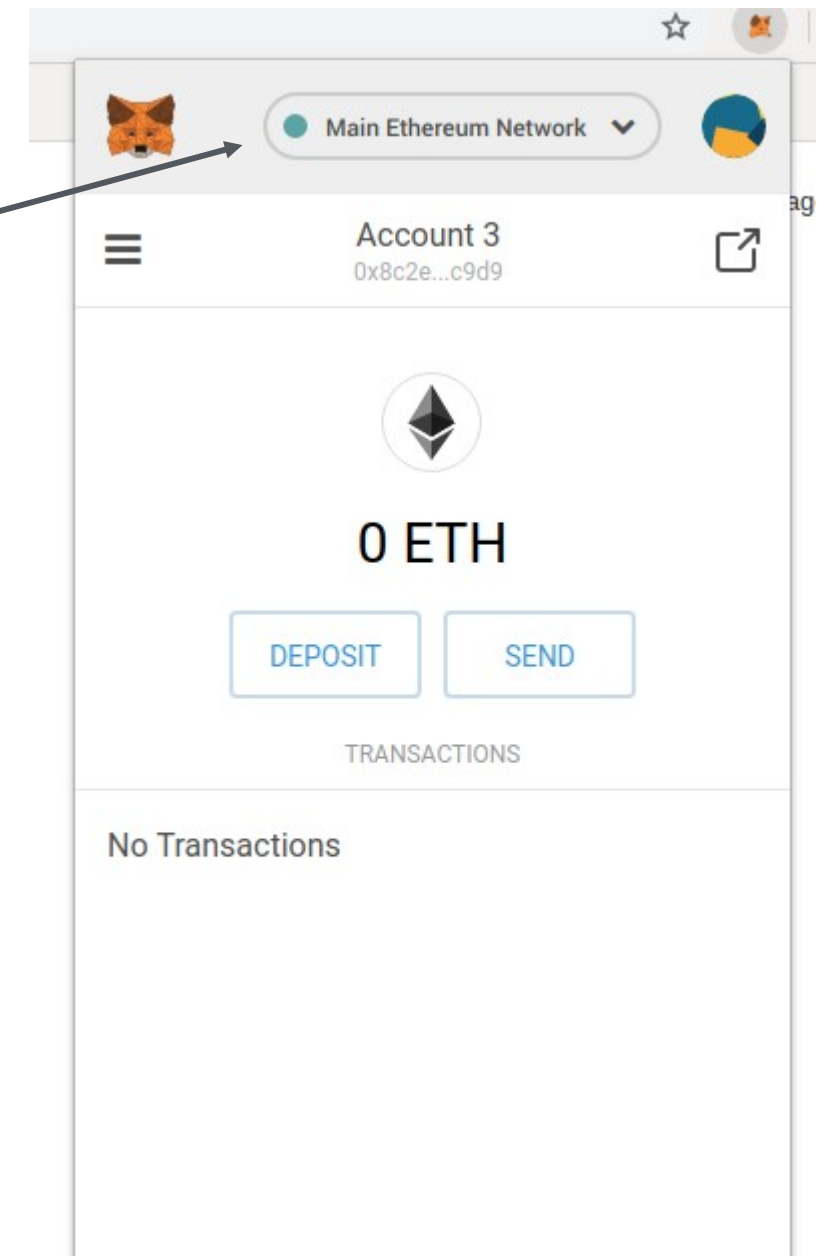
Create an Account in MetaMask

- Follow the instructions to create an account.
- After you provide a password, an account (i.e. an address, public and secret keys) will be created for you.
- **Store your seed:** you want it to restore your wallet in case you delete Metamask

Step1.3:

Connect MetaMask to the Private Blockchain

- 3.1. Click the MetaMask icon.
- 3.2. Click on the Network option
- 3.3. Click on “Custom RPC”



Step1.3:

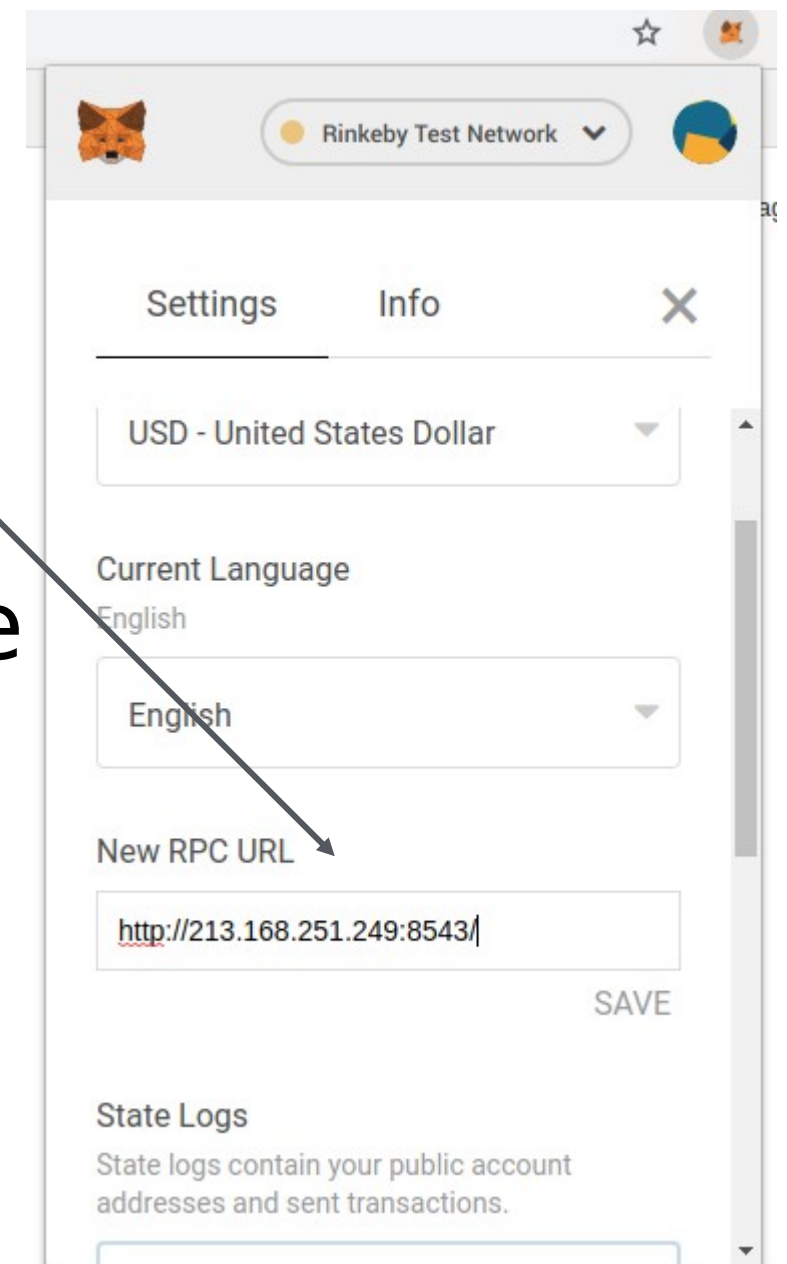
Connect MetaMask to the Private Blockchain

3.1. In the box on the top, insert the following link:

<http://213.168.251.249:8543/>

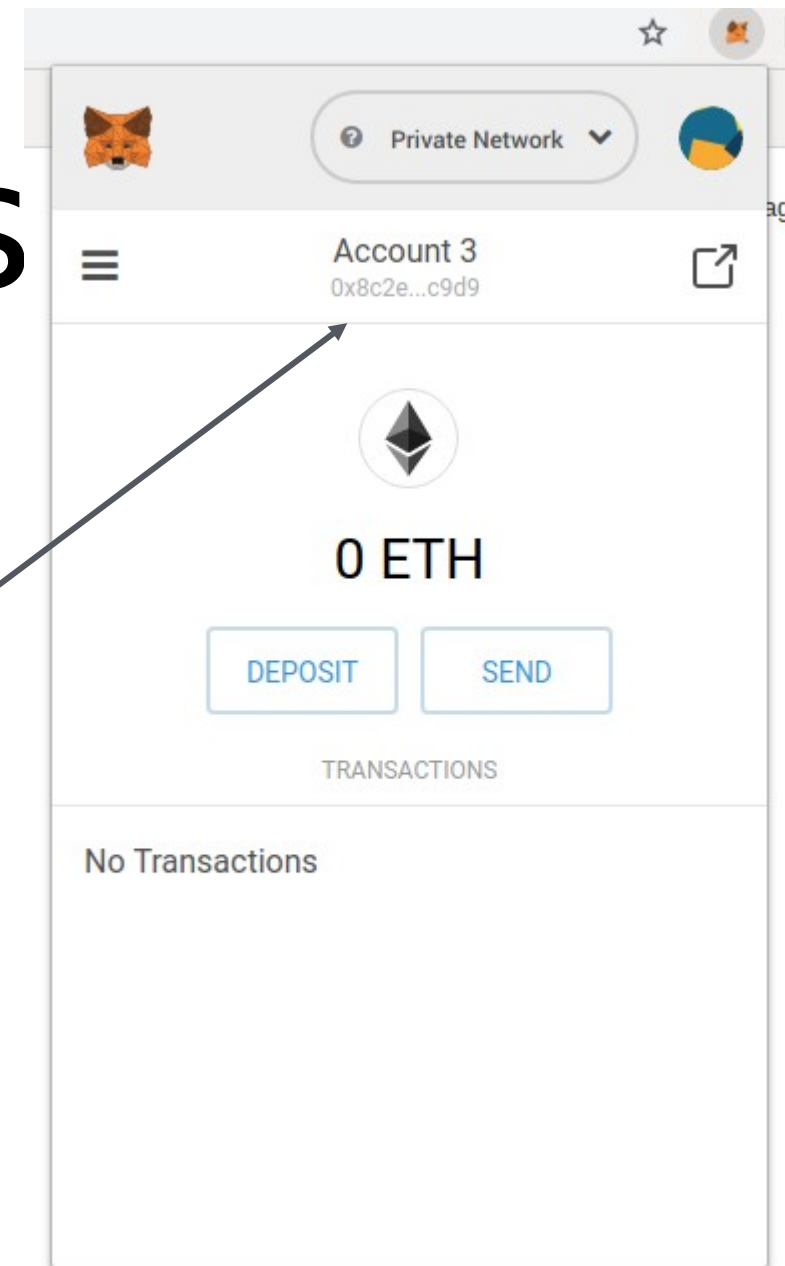
3.2. Click on Save to save it.

3.3. Press X to go to the main page



Step1.3: Your Address

- When you've successfully connected to the chain, this page will appear.



- This is your address; click on it to copy and send it to those who want to pay you.

Step 2:

Send us Your Account Address

- You need some Ether to send a transaction and interact with a smart contract.
- We have created a lot of Ether - you can also have some.
- Request some Ether by sending your account's address to this email address:

dimitris.karakostas@ed.ac.uk

Step 3:

Getting Familiar with Remix Ethereum: Online Solidity Compiler

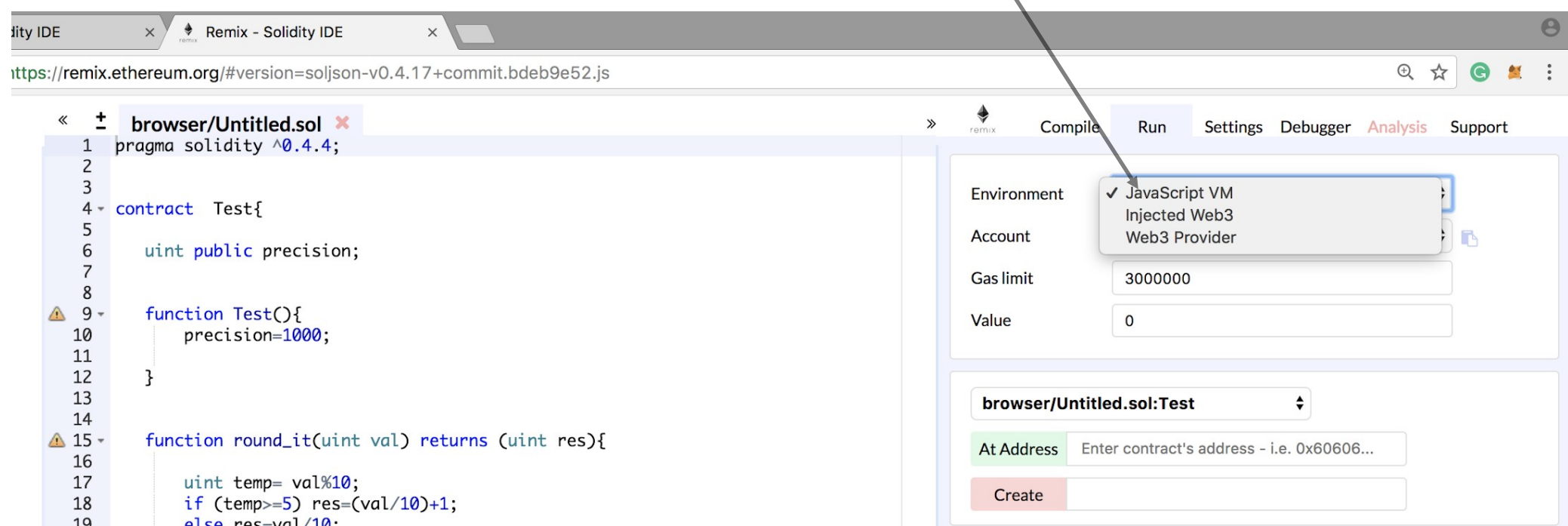
- You can write, debug, deploy (i.e. send to a blockchain) your smart contract via remix
Ethereum: remix.ethereum.org

(If you are using Chrome, it is possible that an error message appears when you load Remix – you can ignore it and then compile your smart contracts as usual, or use Mozilla Firefox instead)

- Also, you can interact with your deployed contract using Remix.

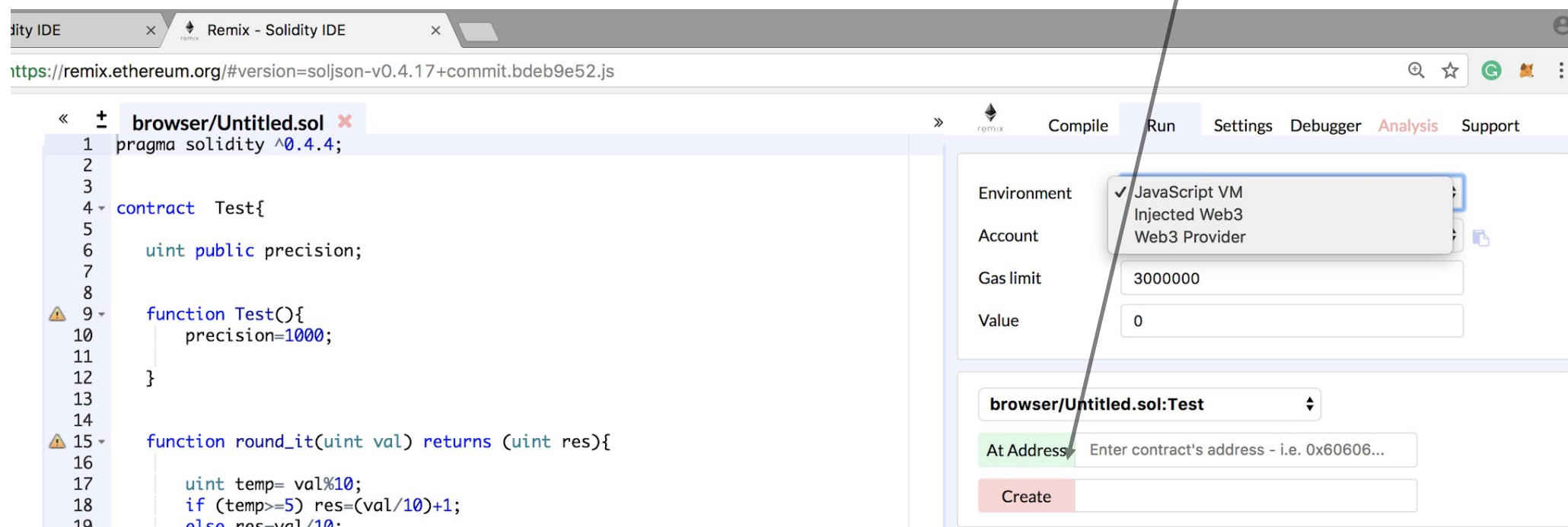
Step 3: Getting familiar with Remix Ethereum: Online Solidity Compiler

- Before you deploy your smart contract to the **private chain**, run and debug it online.
 - In the case where you want to run it online, you should set environment to: **JavaScript VM**.



Step 3: Getting familiar with Remix Ethereum: Online Solidity Compiler

- To compile your smart contract, click on **Create** button.
- After compiling the contract, remix creates a user interface for the functions you defined in the contract and you can pass parameters to it.



Step 4.1: Deploying Smart Contract to the Private Chain Configurations

- First, you need to connect Metamask to the blockchain, as we described in the earlier slides.
- In remix, set the environment to: **Injected Web3**.

The screenshot displays the Remix IDE interface. On the left, a code editor shows a Solidity smart contract named 'Test' with the following code:

```
1 pragma solidity ^0.4.4;  
2  
3  
4 contract Test{  
5     uint public precision;  
6  
7  
8     function Test(){  
9         precision=1000;  
10    }  
11  
12  
13  
14  
15 function round_it(uint val) returns (uint res){  
16  
17     uint temp= val%10;  
18     if (temp>=5) res=(val/10)+1;  
19 }
```

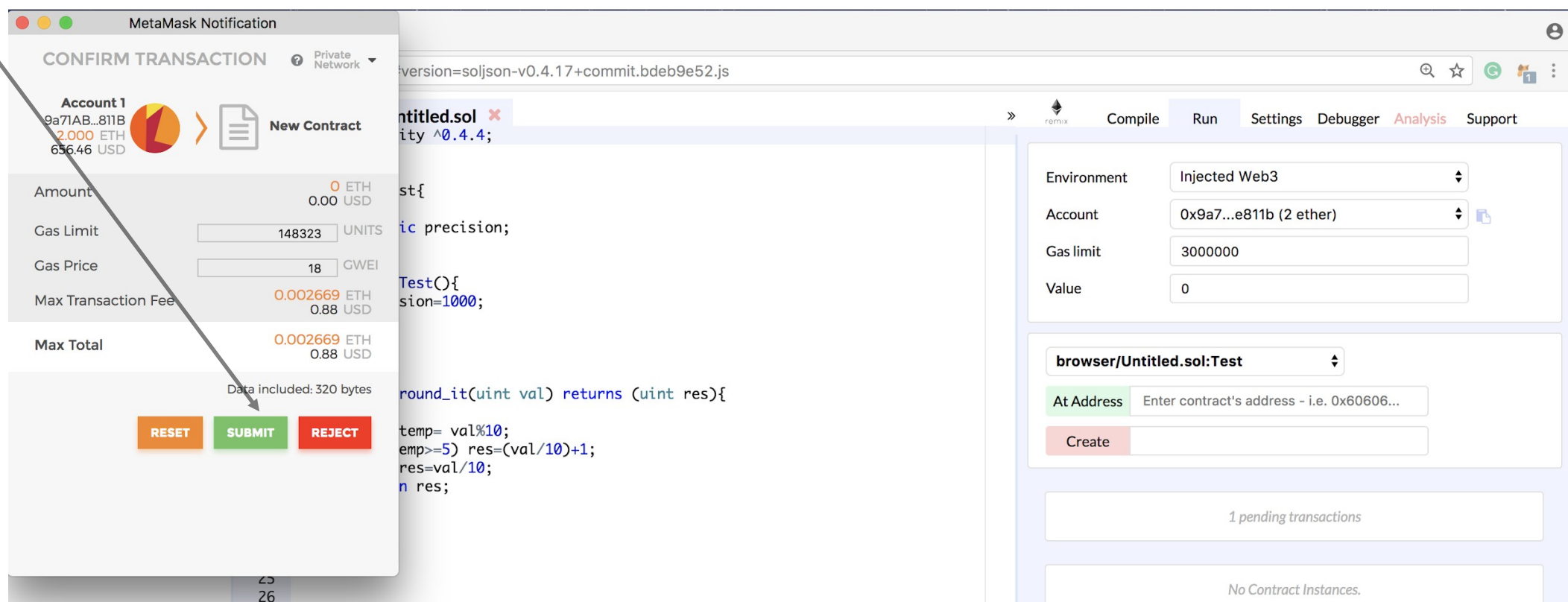
On the right, the 'Run' tab is active, showing deployment configuration. The 'Environment' dropdown menu is open, with 'Injected Web3' selected. Below this, the 'Account' field shows '0x9a7...e811b (0 ether)'. The 'Gas limit' is set to '3000000' and the 'Value' is '0'. At the bottom, the 'Create' button is visible next to a field for the contract address.

Step 4.2:

Deploying Smart Contract to the Private Chain

Deploying a Contract to the Blockchain

- Click on **Create** button.
- Next, MetaMask page will appear and by clicking on **submit**, you send your contract to the blockchain.

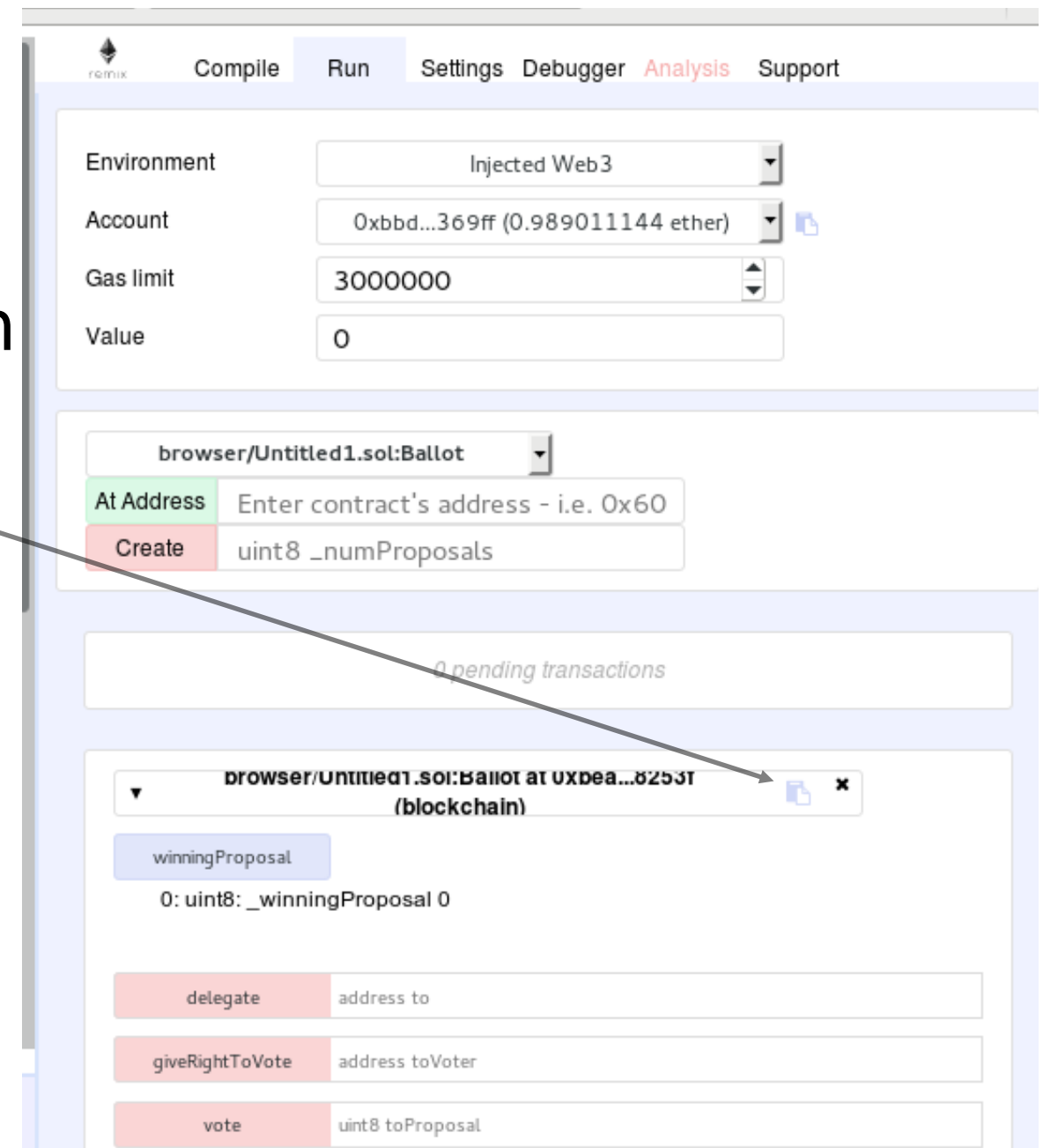


Step 4.3:

Deploying Smart Contract to the Private Chain

Saving the Deployed Contract's Address

- When, your contract is successfully submitted/deployed, remix provides the contract **address** on the blockchain.
- You can copy the address from here.
- You need the **contract code** and the **address** next time you want to interact with your deployed contract.



Step 4.3:

Deploying Smart Contract to the Private Chain

Interacting with a Deployed Contract

1. Log in to MetaMask and connect to the blockchain (as previously explained)
2. In remix, set the environment to: **Injected Web3**.

3. In remix, insert the contract **code**, insert the deployed contract's address and click on: **At Address**.

The screenshot displays the Remix IDE interface. On the left, the Solidity code for a 'Ballot' contract is visible, including a pragma statement for Solidity 0.4.0, a 'contract' definition, and functions for creating a ballot, giving rights to vote, and delegating votes. A variable '_numProposals' is defined as a 'uint8' with one reference. On the right, the sidebar contains several sections: 'Environment' set to 'Injected Web3', 'Account' showing '0xbbd...369ff (0.989011144 ether)', 'Gas limit' set to '3000000', and 'Value' set to '0'. Below this, a dropdown shows 'browser/Untitled1.sol:Ballot'. The 'At Address' button is highlighted, with a tooltip indicating it is used to enter the contract's address (e.g., '0x60'). The 'Create' button is also visible, with a tooltip showing it takes 'uint8 _numProposals' as an argument. At the bottom, there is a section for '0 pending transactions' and a table of functions: 'winningProposal' (returns '0: uint8: _winningProposal 0'), 'delegate' (takes 'address to'), 'giveRightToVote' (takes 'address toVoter'), and 'vote' (takes 'uint8 toProposal').

