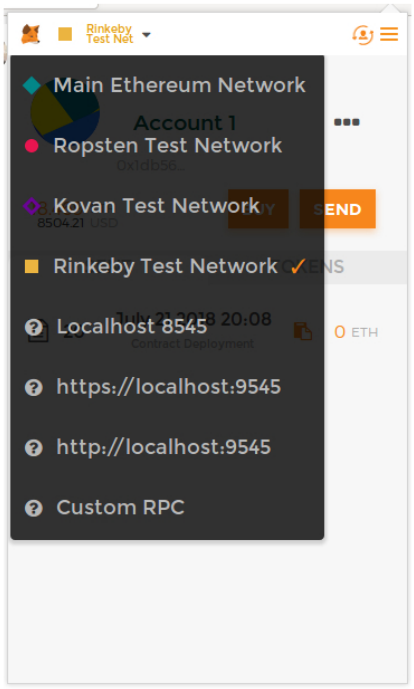
The Many Ways To Deploy Your Smart Contract To Rinkeby Network

arjuna sky kok Follow
Jul 22, 2018 · 9 min read

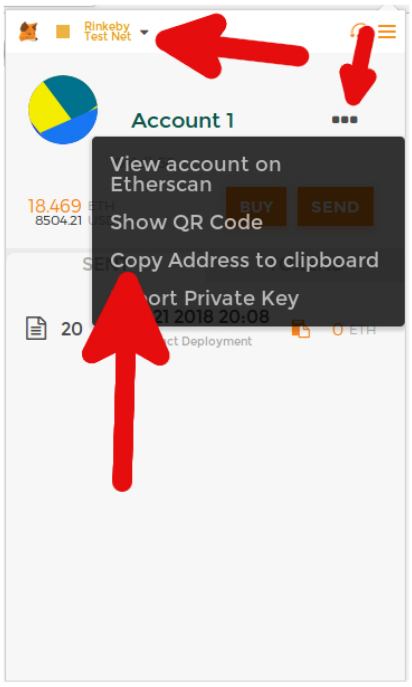
In order to understand this article, you need to understand how to create a smart contract, deploy it to test network and how to play with Metamask. In this article, I am going to show you how to deploy your smart contract to Rinkeby network. So what is Rinkeby network? It's like a staging network for Ethereum. While your test network solely lives in your computer, Rinkeby is a test network but lives globally. Maybe you can say it "lives" in cloud so you who lives in Jakarta can ask your friend who lives in Bogota to test your smart contract easily.

Upload your smart contract to Rinkeby network. Photo credit: henrique from Unsplash.

In local testing network, you get 100 ether from the get-go. In Rinkeby, you don't. You have to "beg" the ether. Go to https://faucet.rinkeby.io. Then create a tweet or public post (in Facebook or Google+) containing your public Ethereum address. Remember, in previous article, you create a new Ethereum public address by "creating an account" in Metamask.

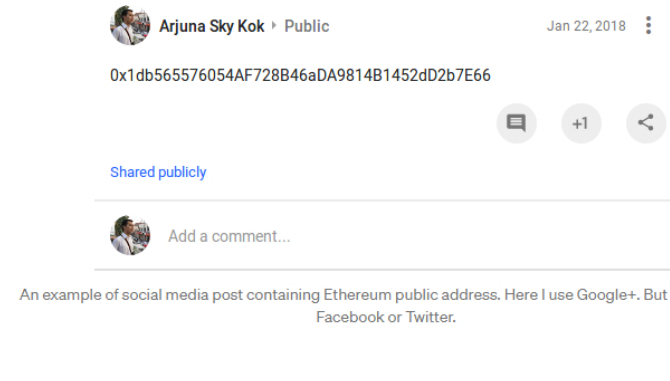


Go to Rinkeby Network in your Metamask software.

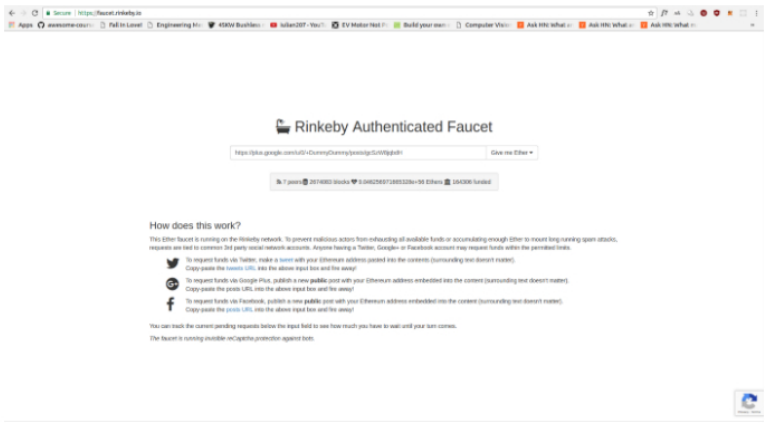


Once you are in Rinkeby network, you can get the public address easily.

Once you create a social media post containing your Ethereum public address, submit the link of your social media post to Rinkeby Authenticated Faucet form.



An example of social media post containing Ethereum public address. Here I use Google+. But you can use Facebook or Twitter.



Submit your public link of social media post in Rinkeby form.

Then depended on which option you choose, you could get 3 or 7.5 or 18.75 ethers while waiting for 8 hours or 1 day or 3 days.

Before we publish the smart contract to Rinkeby network, we need a smart contract. You can use any smart contract you know. Or if you are lazy, you could use mine.

```
pragma solidity ^0.4.19;

contract Auction {
  address public manager;
  address public seller;
  uint public latestBid;
  address public latestBidder;

  constructor() public {
    manager = msg.sender;
  }

  function auction(uint bid) public {
    latestBid = bid * 1 ether; //1000000000000000000;
    seller = msg.sender;
  }

  function bid() public payable {
    require(msg.value > latestBid);

    if (latestBidder != 0x0) {
      latestBidder.transfer(latestBid);
    }
    latestBidder = msg.sender;
    latestBid = msg.value;
  }

  function finishAuction() restricted public {
    seller.transfer(address(this).balance);
  }

  modifier restricted() {
    require(msg.sender == manager);
    _;
  }

  function kill() restricted public { selfdestruct(manager); }
```

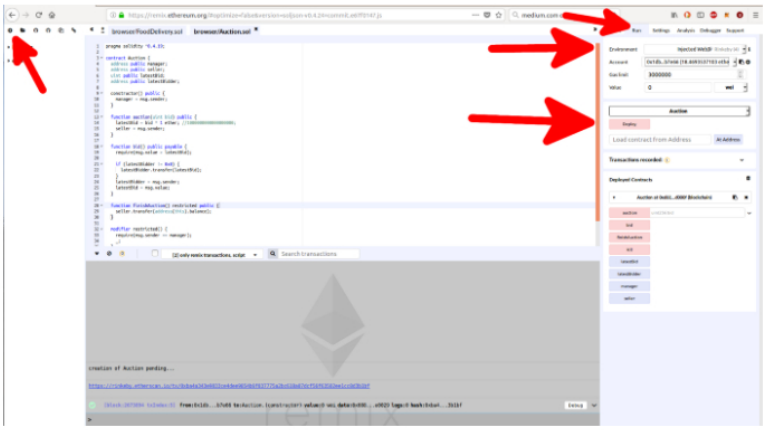
```
}
```

This is the smart contract that I developed in my previous article with a small addition: a method to kill (the method is named *kill*) the smart contract so you have a way to clean your smart contract if you don't need it anymore.

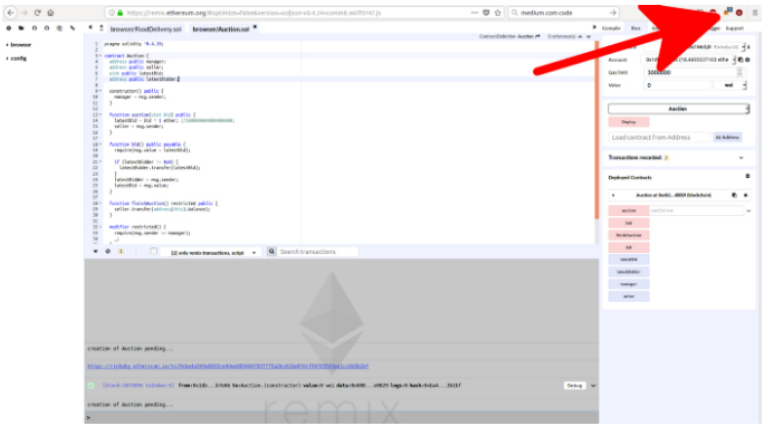
There are a couple of ways to upload your smart contract to Rinkeby network. We will discuss them one by one.

Metamask

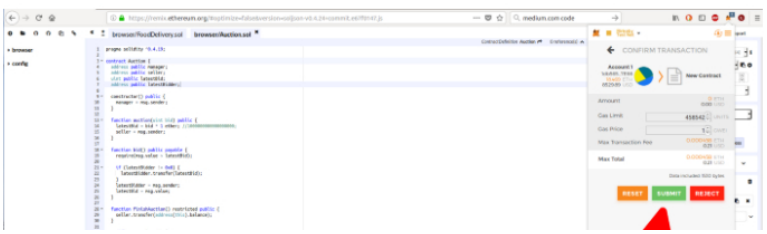
Open https://remix.ethereum.org, create a new file by clicking + button on the top left position. Name it *Auction.sol*. Paste the smart contract you want to deploy. Then click *Run* tab, then choose *Injected Web3* environment. It will automatically use your Metamask account (make sure you have logged in your Metamask first). Then click "*Deploy*" button. Finally click Metamask icon on your browser to approve the launching of your smart contract to Rinkeby network.

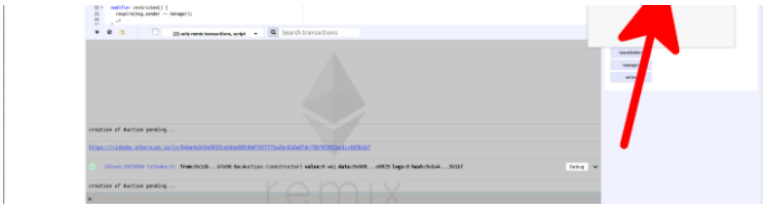


Deploy your smart contract in Remix.

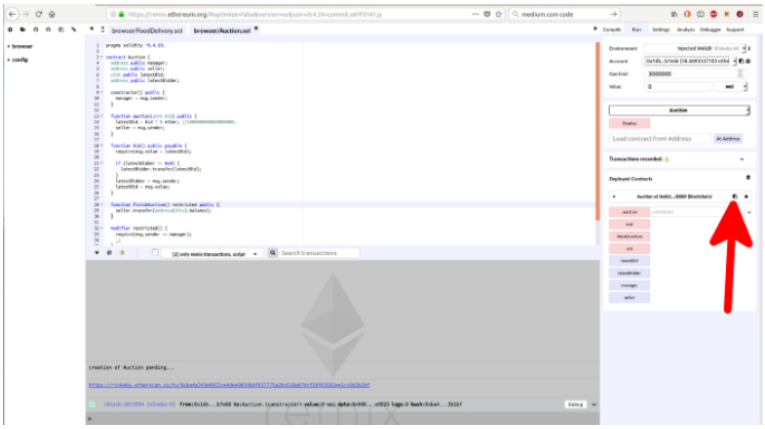


Click Metamask icon after clicking "Deploy" button.



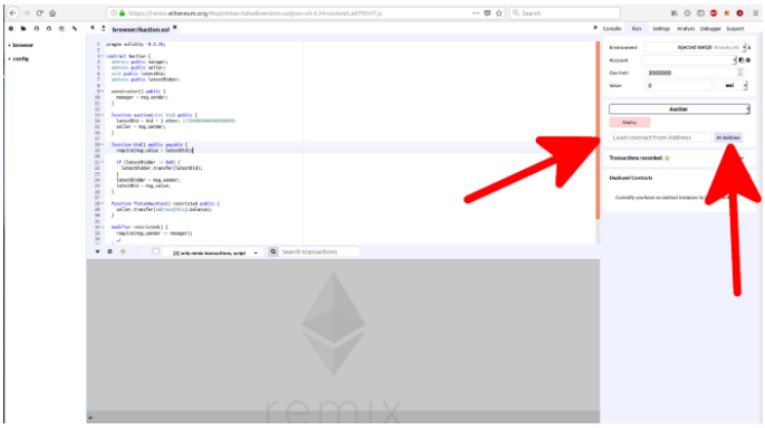


Approve the launching of your smart contract.



You can get your smart contract address by clicking that “small copy” paste icon.

You can let people from all over the world play with your smart contract by giving this smart contract address. Here’s how to do it in case your friend in Bogota wants to play with your smart contract. She can open Remix in her browser then put your smart contract address in form with label “*Load contract from Address*” then click “*At Address*” button.



Load smart contract from the address.

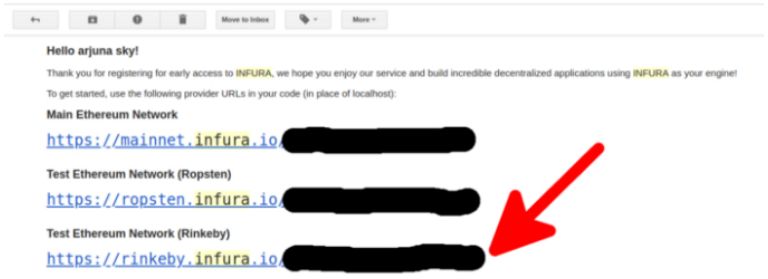
This is not the only way. You can play with the smart contract programmatically by other ways, such as using geth client, Truffle console and using code (Solidity). I am just giving you the easy way to test the smart contract that has been deployed.

Truffle With Infura

In this way, you are going to deploy the smart contract via Infura with Truffle framework. Basically Infura is someone’s Ethereum node with API. Register for free.

Infura website.

You'll get the API key when they send you an email.



Your API key from Infura.

Now develop your smart contract using Truffle framework.

Go to empty directory.

```
truffle init
```

Paste the smart contract into *contracts/Auction.sol*. Then compile it.

```
truffle compile
```

Install *truffle-hdwallet-provider* module.

```
npm install --save truffle-hdwallet-provider
```

Create deploying contract file in *migrations/2_deploy_contracts.js*.

```
var Auction = artifacts.require("Auction");

module.exports = function(deployer) {
  deployer.deploy(Auction);
};
```

Edit *truffle.js*.

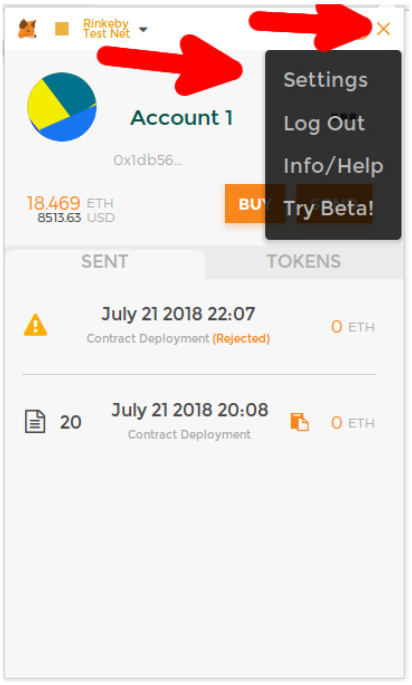
```
var HDWalletProvider = require("truffle-hdwallet-provider");

var mnemonic = "aquaman superman batman shazam flash wonderwoman
```

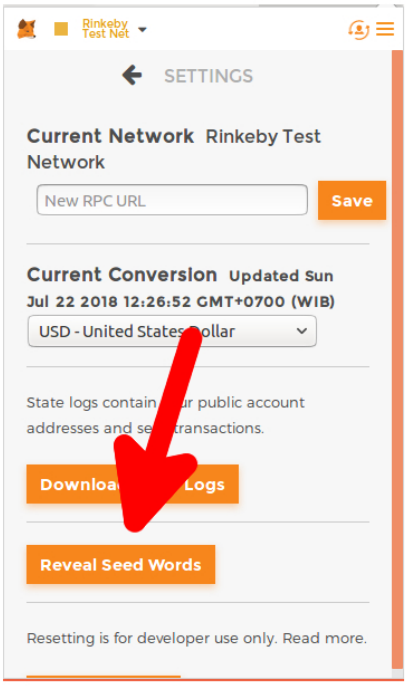
```
greenlantern ironman captainamerica hulk blackwidow thanos";

module.exports = {
  // See <http://truffleframework.com/docs/advanced/configuration>
  // to customize your Truffle configuration!
  networks: {
    rinkeby: {
      provider: function() {
        return new HDWalletProvider(mnemonic,
"https://rinkeby.infura.io/thisistheapikey");
      },
      network_id: 1
    }
  }
};
```

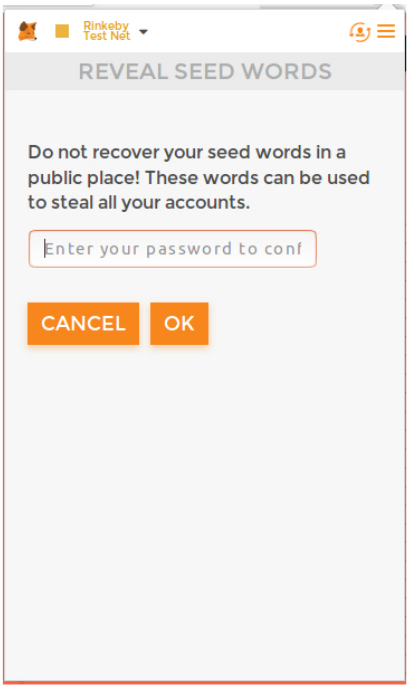
Change "*thisistheapikey*" with your API key from Infura. And change your mnemonic also. How do you get your mnemonic? You can get it from Metamask as shown in these illustrations.



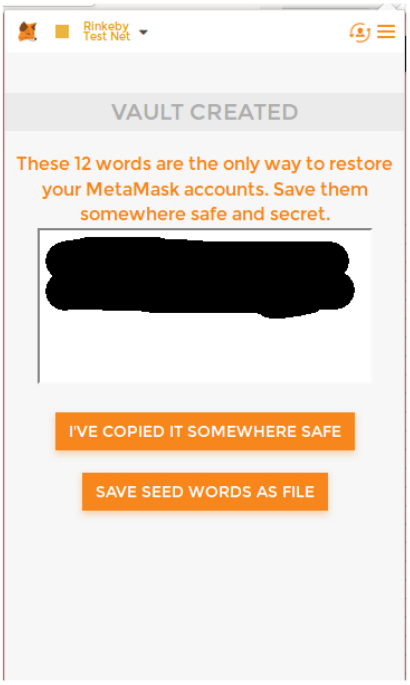
Click Settings.



Click “Reveal Seed Words” button.



Submit password.



Your seed words.

Then run the migration.

```
truffle migrate -f 2 --network rinkeby
```

“-f 2” means you run the migration of *migrations/2_deploy_contracts.js* only. You don’t run *migrations/1_initial_migration.js*. Omit if if you want to run all migrations.

You will get the output like this:

```
Running migration: 2_deploy_contracts.js
 Replacing Auction…
 … 0xe473f0879bc0cbec1e9b72cf881e07a148d733753b20d9795d549a7903b5d39d
 Auction: 0x85db00a8660b4c006bcdeb0d221f2e6c3860c66d
Saving artifacts…
```

The address of the smart contract is the one after "Auction:" word. It is 0x85db00a8660b4c006bcdeb0d221f2e6c3860c66d.

The execution of the command will hang. Don't worry. You can Ctrl+C after the address is shown to you.

web3.js With Infura

If you don't/can't use Truffle framework, here's how you deploy your smart contract to Rinkeby network using Solidity compiler and Web3 library.

Inside a new directory, optionally initialize it as Node.js project. It is optional. You don't have to do it but it is recommended.

```
npm init
```

Then install the necessary tools (the Solidity compiler, web3 and truffle hdwallet provider).

```
npm install --save web3@1.0.0-beta.34 truffle-hdwallet-provider solc
```

For *web3* we have to use the beta version because the last stable version is too primitive. *solc* module is the Solidity compiler. If you use Truffle, you don't need to install this module manually. *web3* module is the library which is responsible for deploying the smart contract.

Put your smart contract, *Auction.sol* in *contracts/Auction.sol*. Create a file to compile this smart contract. Name it *compile.js* and place it in root directory of your project directory. It should be in the same directory as *contracts* directory.

```
//compile.js
const path = require('path');
const fs = require('fs');
const solc = require('solc');

const auctionPath = path.resolve(__dirname, 'contracts',
'Auction.sol');
const source = fs.readFileSync(auctionPath, 'utf8');

module.exports = solc.compile(source, 1).contracts[':Auction'];
```

Then create a file to deploy the smart contract. Name it *deploy.js* and place it in root directory of your project directory. It should be in the same directory as *contracts* directory.

```
//deploy.js
const HDWalletProvider = require('truffle-hdwallet-provider')
```

```
const web3 = require('web3');
const { interface, bytecode } = require('./compile');

const provider = new HDWalletProvider(
  'aquaman superman batman shazam flash wonderwoman greenlantern
ironman captainamerica hulk blackwidow thanos',
  'https://rinkeby.infura.io/thisistheapikey'
);

const web3 = new Web3(provider);

const deploy = async () => {
     const accounts = await web3.eth.getAccounts();

console.log('Attempting to deploy from account', accounts[0]);

const result = await new web3.eth.Contract(JSON.parse(interface))
 .deploy({ data: '0x' + bytecode})
 .send({ gas: '1000000', from: accounts[0] });

console.log('Contract deployed to', result.options.address);
};

deploy();
```

This is based on the code from Stephen Grinder. Credit to him. I modified it a little bit so it works with the latest version of *truffle-hdwallet-provider* module.

When you compile the code using *solc* module, you'll get the interface and the bytecode. Then we connect to the provider with *web3*. From this provider, *web3* get the accounts. Finally, the *web3* has the method to deploy the smart contract.

Execute it like this.

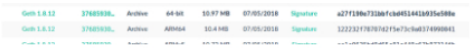
```
node deploy.js
```

This command also hangs. Once you get the smart contract address, just Ctrl+C it. It is fine. No problem.

Geth

This is the hardest method to deploy the smart contract to Rinkeby network. Install geth software from Ethereum website.
https://geth.ethereum.org/downloads/

Geth is a software from Ethereum website. It can be used for mining to get ethers. But here we use it to deploy the smart contract. Download it and install it. The instructions vary between operating systems you use.



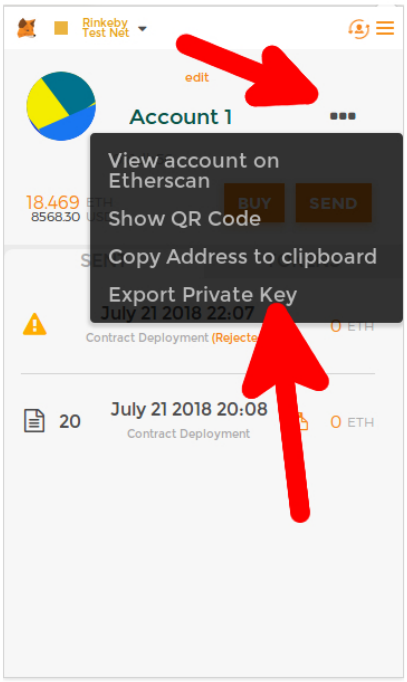
Geth website.

Then launch geth like this.

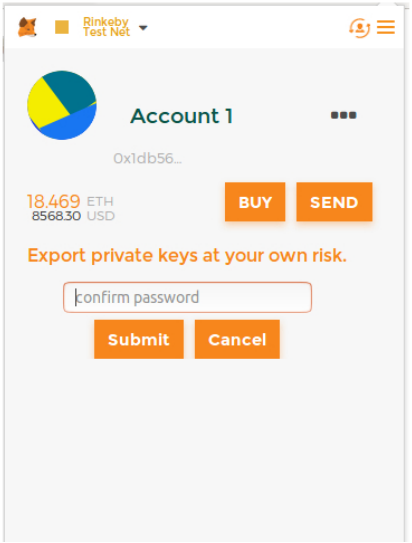
```
geth --rinkeby --datadir /opt/data/ethereumdata
```

Geth has many functionalities. When you run it like this, it means, it syncs the node fully to the rinkeby network and put the node data into */opt/data/ethereumdata* directory. You do this before deploying the smart contract to Rinkeby network. It takes a while before you get a fully sync node. In my case, one night is the length of time needed.

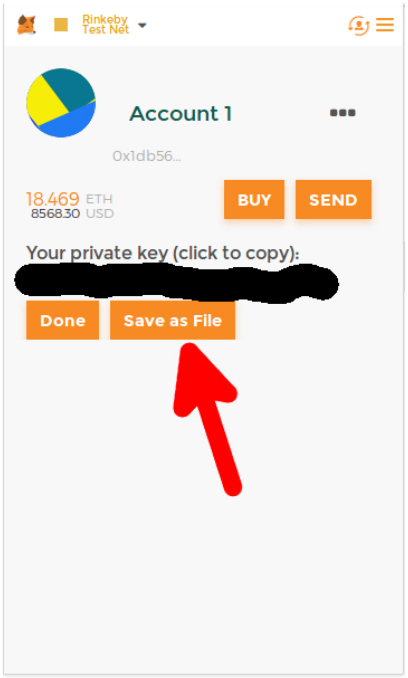
Then you need to import your account that you created in Metamask into geth software. In this case, you don't use mnemonic. You use private key. Export private key from Metamask.



Choose "Export Private Key" menu.



Submit password before exporting your private key.



Save it as a file.

Name your private key file *privatekeyfile.txt*. Import it to geth software like this.

```
geth --rinkeby account import privatekeyfile.txt --datadir
/opt/data/ethereumdata
```

You'll be asked to set the password to protect this account in geth software.

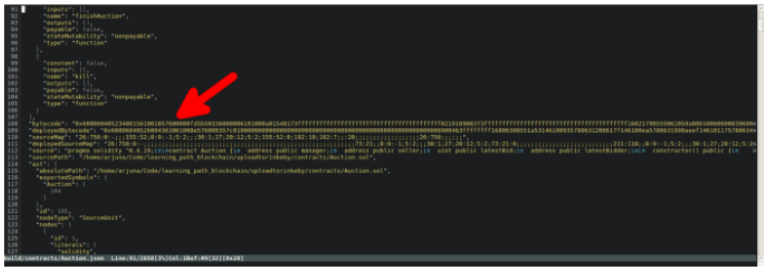
Then you connect to geth software using its console.

```
geth --rinkeby --verbosity 0 console --datadir /opt/data/ethereumdata
```

In its console (in geth console, **not** in your operating system console), you set the bytecode of your smart contract to the variable. Let's name the variable *bytecode*.

```
bytecode = "your_smart_contract_bytecode"
```

So how do you get your smart contract bytecode? It depends on how you compile your smart contract. For example, if you compile using "*truffle compile*", then your bytecode is located inside *build/contracts/Auction.json*. Open the file. Find the JSON key "bytecode". It points to the bytecode of your smart contract.



Your smart contract bytecode.

In the same geth console, unlock your account.

```
personal.unlockAccount(eth.accounts[0], "password")
```

Replace "*password*" with your password when you imported the account from Metamask. Then upload your smart contract to Rinkeby network.

```
tx = eth.sendTransaction({from: eth.accounts[0], data: bytecode, gas:
500e3})
```

Then check whether it has been uploaded or not.

```
web3.eth.getTransactionReceipt(tx)
```

If it has been deployed, you will get the output like this.

```
{
 blockHash:
"0xfed7dcbd5e8c68e17bff9f42cd30d95588674497ae719a04fd6a2ff219bb001d",
 blockNumber: 2534930,
 contractAddress: "0xbd3ffb07250634ba413e782002e8f880155007c8",
 cumulativeGasUsed: 1071323,
 from: "0x1db565576054af728b46ada9814b1452dd2b7e66",
 gasUsed: 458542,
 logs: [],
 logsBloom: "0x00000...",
 status: "0x1",
 to: null,
 transactionHash:
"0x1a341c613c2f03a9bba32be3c8652b2d5a1e93f612308978bbff77ce05ab02c7",
 transactionIndex: 4
}
```

Your contract address is pointed by JSON key "*contractAddress*".

Remember when you want to deploy smart contract via geth software, you must sync the node first. Whether you can deploy the smart contract to Rinkeby network if you only sync lightly or not, I am not sure. For now, just sync the node fully.

Ethereum Metamask Geth Solidity Tutorial

438 2

WRITTEN BY

arjuna sky kok

Follow

I do 3 things for my livelihood: I build mobile apps and web apps for clients, I write about blockchain and cryptocurrency, and I give tech corporate training.

Coinmonks

Follow

Coinmonks is a non-profit Crypto educational publication. Follow us on Twitter @coinmonks Our other project—https://coincodecap.com

More From Medium

How Is Blockchain Making a Great Difference in Mobile App Security?
Sophia Martin in DataSeries

1ConfValue—a simple PoW confirmation rule-of-thumb
JP Thor in Coinmonks

SIX Network unveils the newest blockchain-based image verification app "snap" to help the world...
SIX Network in SIX.network

Why Chainge?
Chookz

ChainX to release 2+1 testnet by the end of May
ChainX

IOST x Binance Korea | AMA With CTO Terrence Wang On IOST Plans and Roadmap!
IOS Foundation in IOST

Why HODL DHV?
Team DeHive in DeHive

A Blockchain Ecosystem to put on your radar Pt. 1!
Fernando Trouw in Caribbean Blockchain Network