

[Sandeep Panda](#) · [@sandeep](#) ·  

How to build your own Ethereum based ERC20 Token and launch an ICO in next 20 minutes

Update: I'm writing a book on Ethereum Smart Contracts. It's available for purchase on [LeanPub](#). If you are interested in learning more about Smart Contracts and building Decentralized apps, feel free to give this a try.

If you are looking for advanced stuff such as Presale, Public Sale, Discounts, goal, hard cap etc.. check out my [latest article](#).

Lately I have been digging into blockchain and decentralised apps to educate myself and improve my knowledge. To make the learning process fun I decided to build my own Ethereum based token and understand the process of launching an ICO (Initial Coin Offering).

This article aims to give you an overview of how smart contracts work in Ethereum by launching a simple demo ICO.

Make Hashnode your discussion hub for all things crypto. Check out our [crypto communities](#) and join the blockchain-related discussions.

Basics

Here are a few basic terms we are going to use in this article. If you are familiar with the following concepts, feel free to skip to the next section.

- **Ethereum based ERC20 Tokens:** In Ethereum tokens represent any tradable goods such as coins, loyalty points etc. You can create your own crypto-currencies based on Ethereum. Additionally the benefit of following ERC20 standard is that your tokens will be compatible with any other client or wallets that use the same standards.
- **Smart Contracts:** Smart Contracts are self executing code blocks deployed on the Ethereum blockchain. They contain data & code functions. Contracts make decisions, interact with other contracts, store data and transfer Ether (the unit of crypto-currency in the Ethereum blockchain) among users.
- **Solidity:** A language for writing smart contracts.

Submitted on

Monday, 18 December 2017 at 01:34 pm

Tagged in

[Blockchain](#) [Smart Contracts](#)

[Solidity](#) [Cryptocurrency](#)

[Ethereum](#)

Total views

46.5K

- **MetaMask/Mist/MEW Wallet:** A digital facility that holds your Ether and other Ethereum based tokens.

Now that you are aware of the basic terminologies used in this article let's get started.

Step 1: Code

Open your favourite text editor and paste the following code:

```
pragma solidity ^0.4.4;

contract Token {

    /// @return total amount of tokens
    function totalSupply() constant returns (uint256 supply) {}

    /// @param _owner The address from which the balance will be
    retrieved
    /// @return The balance
    function balanceOf(address _owner) constant returns (uint256
    balance) {}

    /// @notice send `_value` token to `_to` from `msg.sender`
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transfer(address _to, uint256 _value) returns (bool
    success) {}

    /// @notice send `_value` token to `_to` from `_from` on the
    condition it is approved by `_from`
    /// @param _from The address of the sender
    /// @param _to The address of the recipient
    /// @param _value The amount of token to be transferred
    /// @return Whether the transfer was successful or not
    function transferFrom(address _from, address _to, uint256
    _value) returns (bool success) {}

    /// @notice `msg.sender` approves `_addr` to spend `_value`
    tokens
    /// @param _spender The address of the account able to
    transfer the tokens
    /// @param _value The amount of wei to be approved for
    transfer
    /// @return Whether the approval was successful or not
    function approve(address _spender, uint256 _value) returns
    (bool success) {}

    /// @param _owner The address of the account owning tokens
    /// @param _spender The address of the account able to
    transfer the tokens
    /// @return Amount of remaining tokens allowed to spent
    function allowance(address _owner, address _spender) constant
    returns (uint256 remaining) {}

    event Transfer(address indexed _from, address indexed _to,
    uint256 _value);
    event Approval(address indexed _owner, address indexed
    _spender, uint256 _value);

}
```

```

contract StandardToken is Token {

    function transfer(address _to, uint256 _value) returns (bool
success) {
        //Default assumes totalSupply can't be over max (2^256 -
1).

        //If your token leaves out totalSupply and can issue more
tokens as time goes on, you need to check if it doesn't wrap.
        //Replace the if with this one instead.
        //if (balances[msg.sender] >= _value && balances[_to] +
_value > balances[_to]) {
            if (balances[msg.sender] >= _value && _value > 0) {
                balances[msg.sender] -= _value;
                balances[_to] += _value;
                Transfer(msg.sender, _to, _value);
                return true;
            } else { return false; }
        }

        function transferFrom(address _from, address _to, uint256
_value) returns (bool success) {
            //same as above. Replace this line with the following if
you want to protect against wrapping uints.
            //if (balances[_from] >= _value && allowed[_from]
[msg.sender] >= _value && balances[_to] + _value > balances[_to])
{
                if (balances[_from] >= _value && allowed[_from]
[msg.sender] >= _value && _value > 0) {
                    balances[_to] += _value;
                    balances[_from] -= _value;
                    allowed[_from][msg.sender] -= _value;
                    Transfer(_from, _to, _value);
                    return true;
                } else { return false; }
            }

            function balanceOf(address _owner) constant returns (uint256
balance) {
                return balances[_owner];
            }

            function approve(address _spender, uint256 _value) returns
(bool success) {
                allowed[msg.sender][_spender] = _value;
                Approval(msg.sender, _spender, _value);
                return true;
            }

            function allowance(address _owner, address _spender) constant
returns (uint256 remaining) {
                return allowed[_owner][_spender];
            }

```

```

mapping (address => uint256) balances;
mapping (address => mapping (address => uint256)) allowed;
uint256 public totalSupply;
}

```

contract HashnodeTestCoin is StandardToken { // CHANGE THIS.

Update the contract name.

```

/* Public variables of the token */

/*
NOTE:
The following variables are OPTIONAL vanities. One does not
have to include them.

They allow one to customise the token contract & in no way
influences the core functionality.

Some wallets/interfaces might not even bother to look at this
information.
*/
string public name;                // Token Name
uint8 public decimals;            // How many decimals to
show. To be standard complicant keep it 18
string public symbol;             // An identifier: eg
SBX, XPR etc..
string public version = 'H1.0';
uint256 public unitsOneEthCanBuy; // How many units of
your coin can be bought by 1 ETH?
uint256 public totalEthInWei;     // WEI is the smallest
unit of ETH (the equivalent of cent in USD or satoshi in BTC).
We'll store the total ETH raised via our ICO here.
address public fundsWallet;       // Where should the
raised ETH go?

// This is a constructor function
// which means the following function name has to match the
contract name declared above
function HashnodeTestCoin() {
    balances[msg.sender] = 10000000000000000000;
// Give the creator all initial tokens. This is set to 1000 for
example. If you want your initial tokens to be X and your decimal
is 5, set this value to X * 100000. (CHANGE THIS)
    totalSupply = 10000000000000000000;
// Update total supply (1000 for example) (CHANGE THIS)
    name = "HashnodeTestCoin";
// Set the name for display purposes (CHANGE THIS)
    decimals = 18;
// Amount of decimals for display purposes (CHANGE THIS)
    symbol = "HTCN";
// Set the symbol for display purposes (CHANGE THIS)
    unitsOneEthCanBuy = 10;
// Set the price of your token for the ICO (CHANGE THIS)
    fundsWallet = msg.sender;
// The owner of the contract gets ETH

```

```

    }

    function() payable{
        totalEthInWei = totalEthInWei + msg.value;
        uint256 amount = msg.value * unitsOneEthCanBuy;
        if (balances[fundsWallet] < amount) {
            return;
        }

        balances[fundsWallet] = balances[fundsWallet] - amount;
        balances[msg.sender] = balances[msg.sender] + amount;

        Transfer(fundsWallet, msg.sender, amount); // Broadcast a
message to the blockchain

        //Transfer ether to fundsWallet
        fundsWallet.transfer(msg.value);
    }

    /* Approves and then calls the receiving contract */
    function approveAndCall(address _spender, uint256 _value,
bytes _extraData) returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);

        //call the receiveApproval function on the contract you
want to be notified. This crafts the function signature manually
so one doesn't have to include a contract in here just for this.
        //receiveApproval(address _from, uint256 _value, address
_tokenContract, bytes _extraData)
        //it is assumed that when does this that the call
*should* succeed, otherwise one would use vanilla approve
instead.

        if(!_spender.call(bytes4(bytes32(sha3("receiveApproval(address,uint256,address,bytes)"))
        msg.sender, _value, this, _extraData)) { throw; }

        return true;
    }
}

```

Based on the original source from [Token-Factory](#).

The above code uses Solidity language to build a simple ERC20 token. The code is well commented and is very easy to understand. Once you paste the code into your text editor, find the text: "CHANGE THIS". This is what you need to change based on the characteristics of your token. In the example above, I have named my token **HashnodeTestCoin (HTCN)**. The total supply is capped at 1000, but people can possess as little as 0.000000000000000001 because of 18 decimal places. Additionally, the owner of the contract (one who executes it) gets all the initial tokens. I have set the ICO price as following:

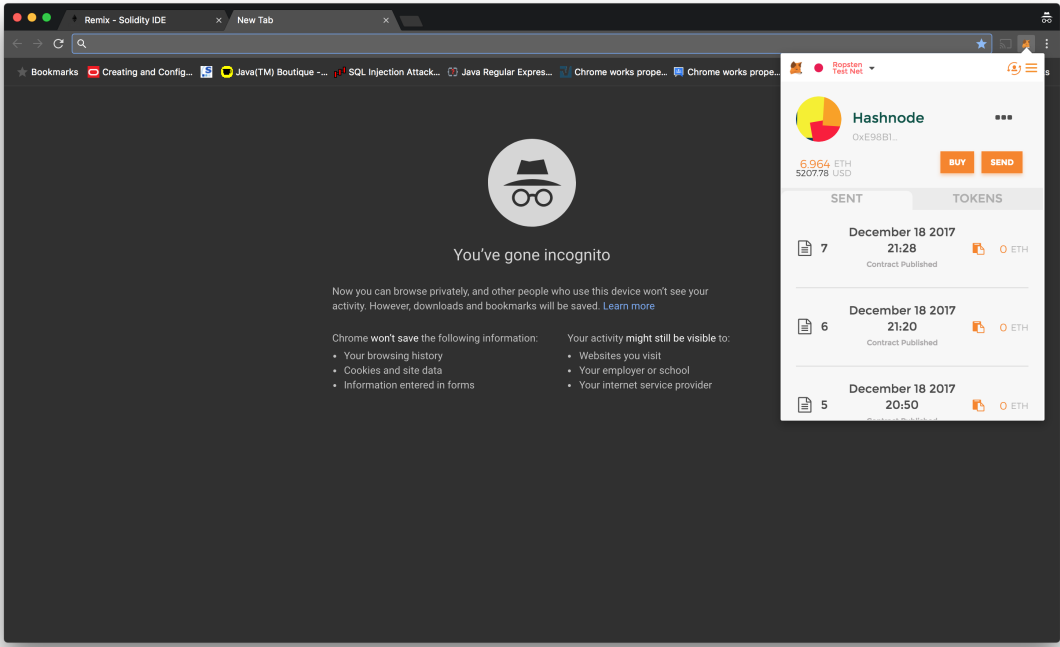
1 ETH = 10 HTCN

This means, if someone sends 1 ETH to this smart contract, they will get 10 HTCN units.

Step 2

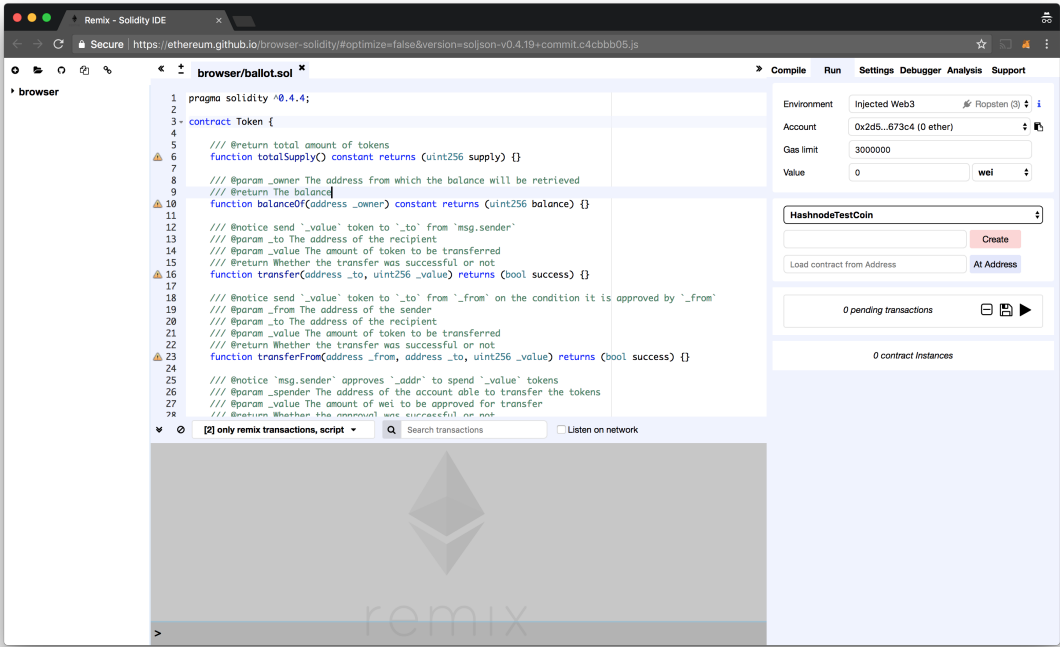
Download [MetaMask](#) chrome extension to generate a wallet. This is going to be the owner of the smart contract. Alternatively you can always use [Mist](#) or [My Ether Wallet](#). For the sake of simplicity let's use MetaMask extension in this project.

Once you download the extension, go ahead and create a new account protected by a password. Then choose "Ropsten TestNet" from the top left corner. Before we deploy the contract to Main Ethereum blockchain, we'll test it against TestNet and make sure everything works as expected. It looks something like this:

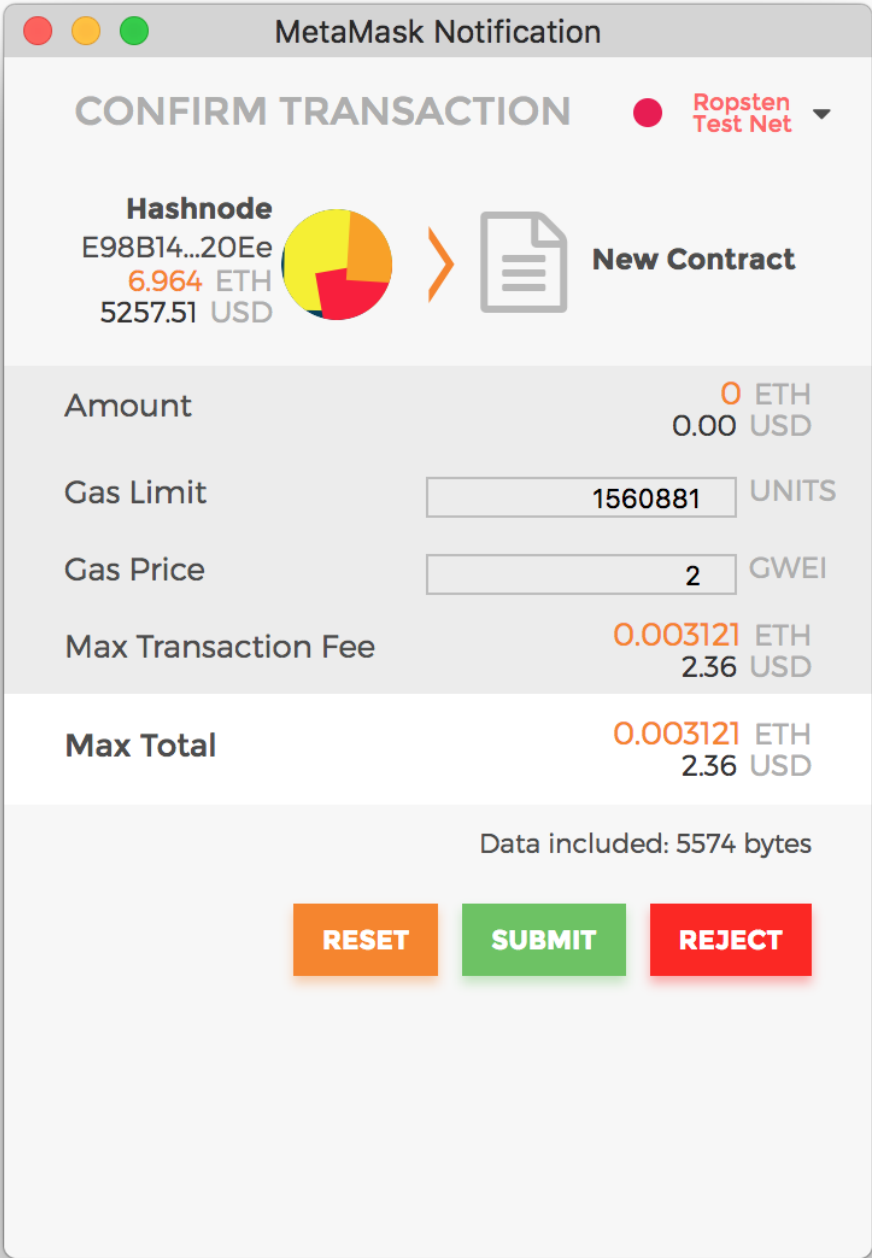


Now head over to [Remix IDE](#) (an online solidity compiler and debugger) and paste the code you just modified. Ignore any warnings you see. Next go to settings and uncheck "Enable Optimizations" if it's checked.

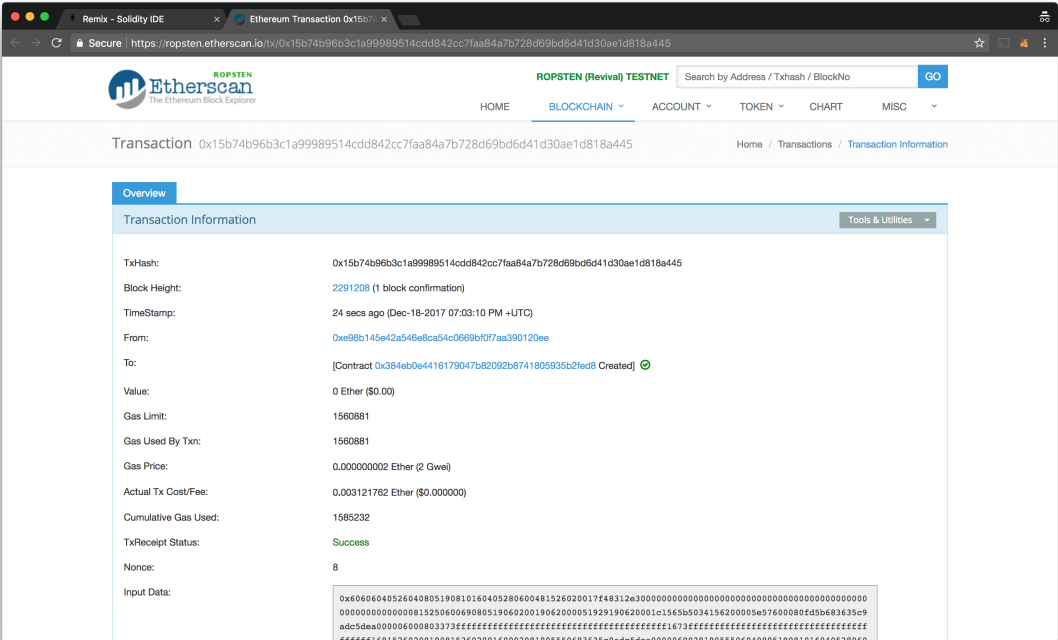
Now go to "Run" tab and click on **create** under <your token name>.



Once you hit create, MetaMask will prompt you to buy some test ether and submit the transaction. It looks something like this:



Just make sure you are on Ropsten TestNet and not on the MainNet and then hit Submit. Now open up MetaMask again and click on the first transaction. It'll take you to Etherscan where you can observe the ongoing transaction. It may take up to 30s to confirm the transaction. Once it's confirmed it looks like the following:

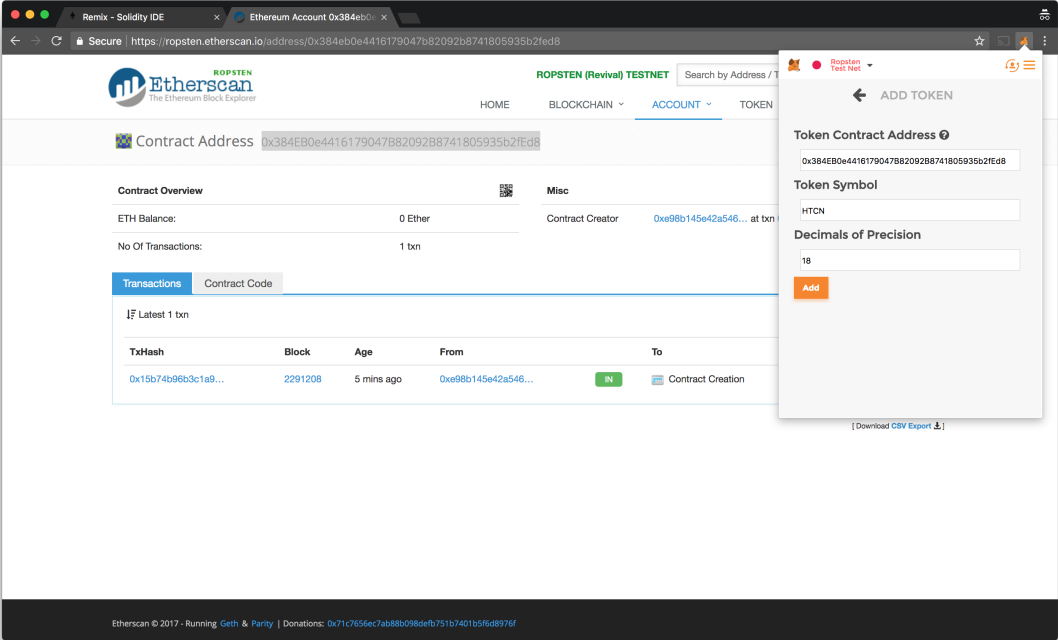


Viola!! You just deployed your contract. Note the **to** address in the above transaction page. That's your contract address.

Now it's time to verify if it actually works.

Step 3

Ideally if you've set up everything correctly you should receive all the initial tokens (1000 in my case) when you add it to your wallet. So, copy the contract address, go to MetaMask -> Add Token and paste the address. It looks like the following:

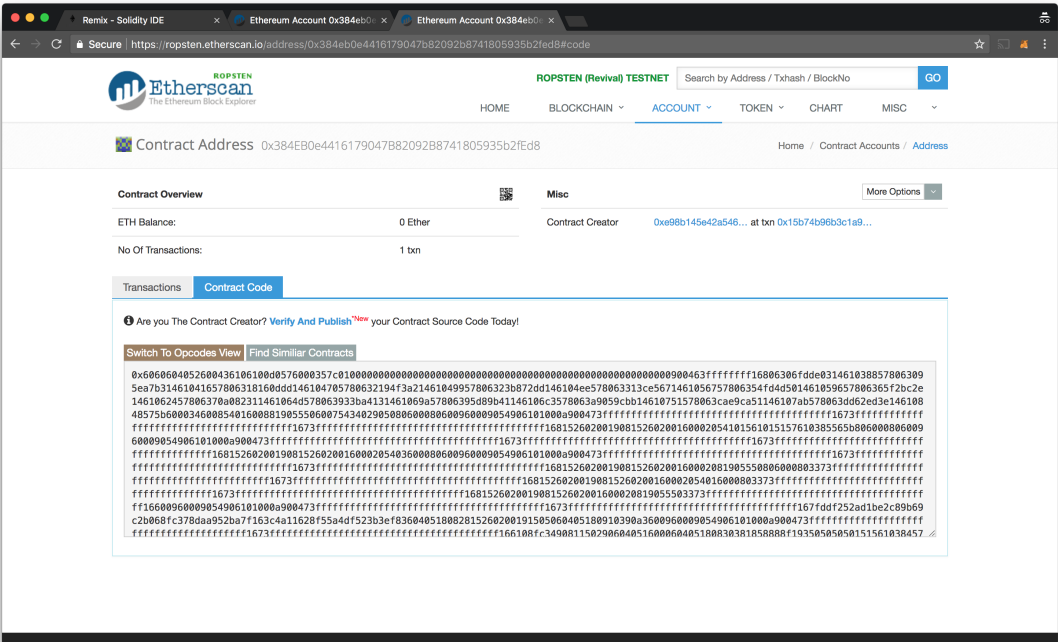


Hit **Add** and refresh MetaMask. You should now see all the initial supply (in my case it was 1000 HTCN). :)

Step 4

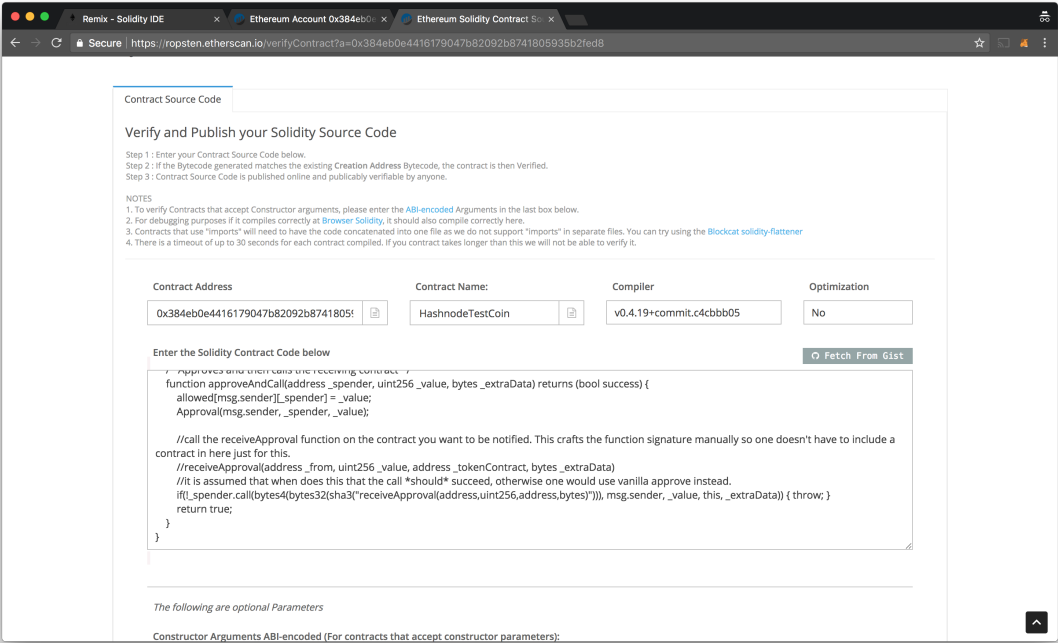
Now that everything works perfectly we just have to verify our smart contract so that everyone on the blockchain can read and understand it. It's always a good practice to verify since it helps establish trust.

Now go to your [contract address](#) and click on Contract Code tab.

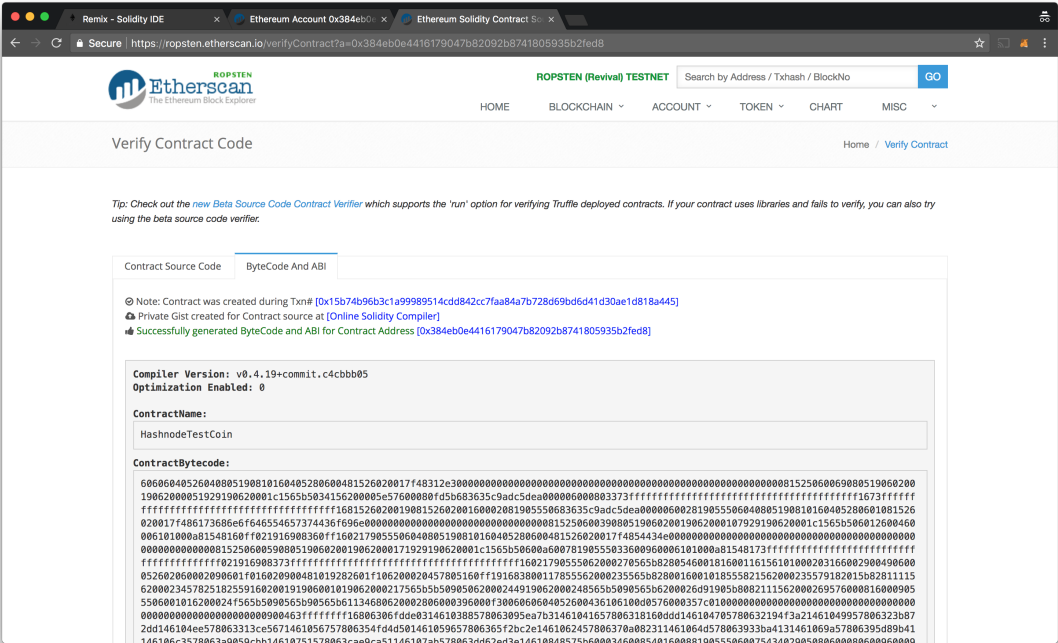


.

Now click on "verify and publish" link. Once you are taken to the new page, fill up the details such as compiler version, Enable Optimizations etc and paste the solidity source we compiled in the first step.



Make sure the compiler version you choose matches the one you compiled your code against in the first step. Now hit "verify and publish". If successful, it'll generate bytecode and ABI as following:



Congrats! Now anyone can visit your contract address and read the source.

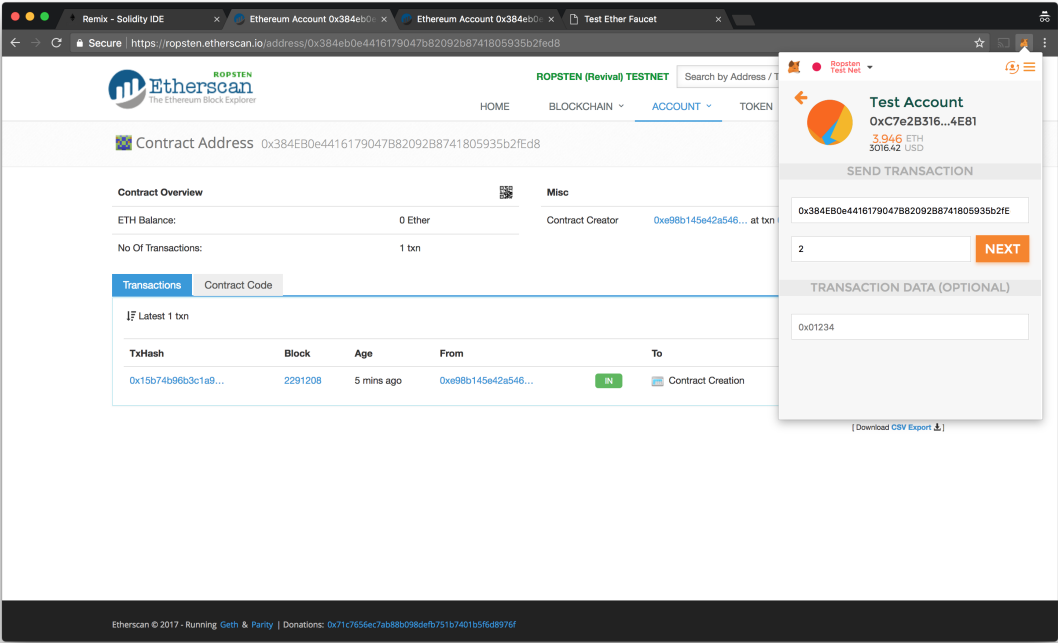
Step 5

To deploy your contract to production, you just need to switch *TestNet* to *MainNet* on MetaMask (located at top left corner) and repeat step 2 to 4. Please be aware that you will have to spend **real Ether** over there to deploy your contract. So, don't deploy the contract unless you are fully ready (Contracts are immutable and can't be updated once deployed). We'll keep using TestNet in this tutorial.

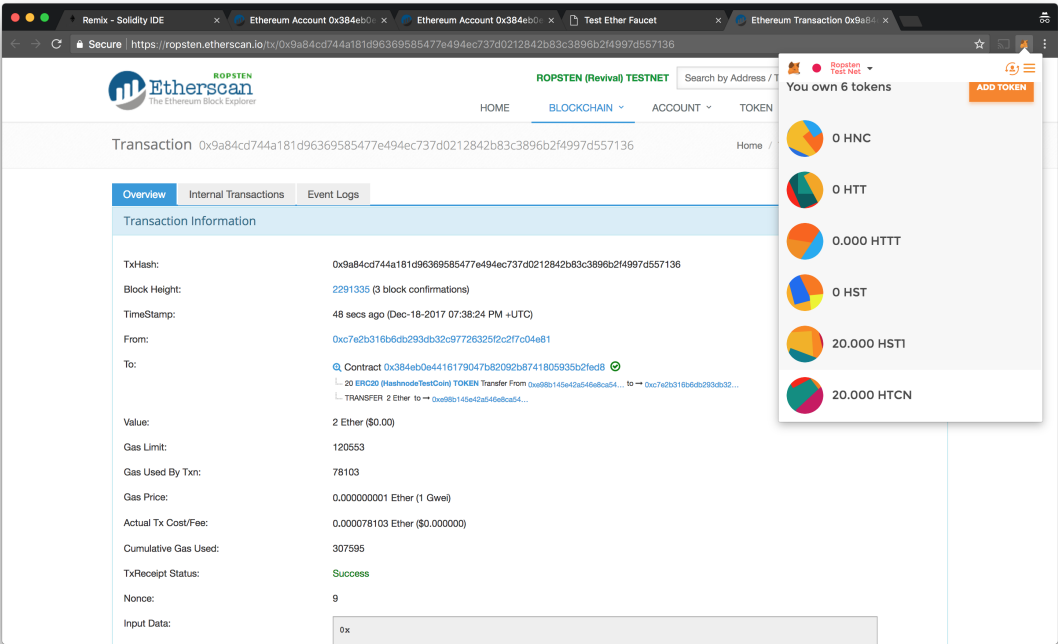
Buying tokens with Ether

As a part of the ICO, your users will buy tokens from you by paying ETH. Remember we set the price as **1 ETH = 10 HTCN** while deploying the contract? So, if a user wants to buy 10 HTCNs through your ICO, they have to pay 1 ETH. Let's test this out.

Go to MetaMask, create a new account and load it with some Test Ether. Once the account is loaded, click "Send" and fill up your contract address. In the amount field, enter 2 (ETH).



Next, send 2 ETH to the contract address and wait for the transaction to be confirmed. Refresh MetaMask and check your tokens after a few seconds. The new test account should have got 20 HTCNs (or something different depending on your config) and the contract owner (you) should have 980 (or something similar) tokens.



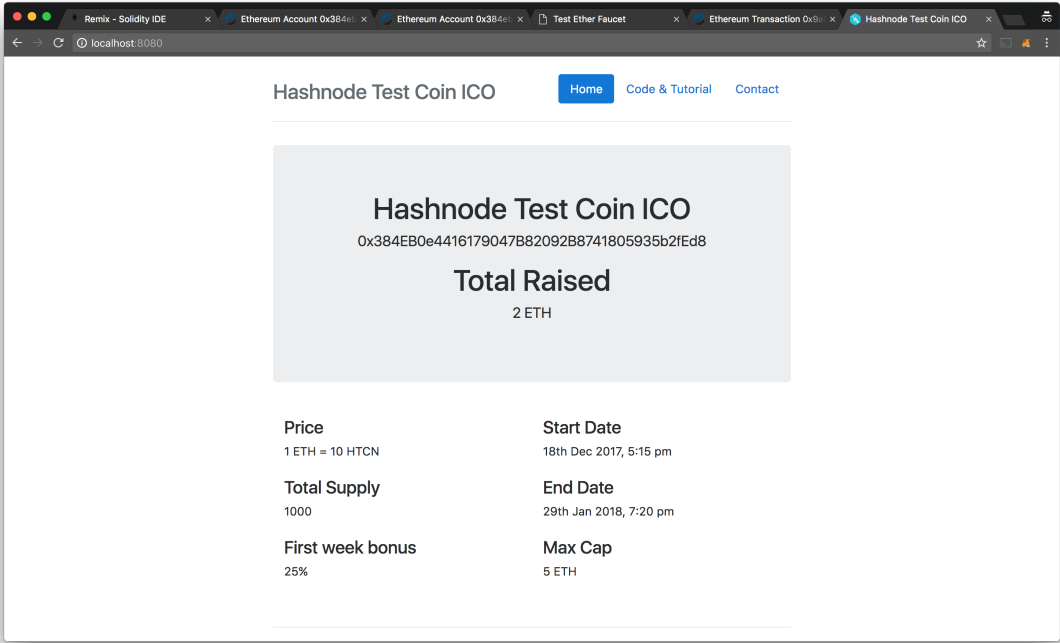
Additionally, you should have received 2 ETH.

Congrats on the success!

Launching an ICO page

To display the amount of ETH raised by our project, we'll use a JavaScript library called [Web3.js](#) on our website.

Head over to [Hashnode Test Coin ICO](#) and check out the code in the last `<script>` tag. It's fairly simple and just uses Web3.js and ABI obtained from [your contract](#).



Congrats if you have made this far! Do keep in mind that real production grade contracts and ICOs take significant efforts and are supported by thorough tests. While this tutorial gives you an overview of writing contracts for ICOs, it's never meant for production level deployment. Don't deploy this on MainNet without proper testing!

Thanks for reading this article. If you have any questions, please feel free to comment below.

Have blockchain-related questions? Check out [these crypto communities on Hashnode](#) and start a discussion.

29

Follow (13)

Share

Save

Your response

Write your comment...

37 responses

Popular

Recent

Sai Kishore Komanduri · @saiki ·

3m

I'll invest in *this* ICO! 😊 Go HTCN!



3

Reply (1)

Share

Save

Anwar Shaikh

11d

Hey Sundeep, Not able to buy your book mentioned above as it says payment declined. I am pretty sure I have money in my account. Also the web3.js is not displaying the ethers earned. Can you please have a tutorial on web3.js to learn the front end as well.

 [Upvote](#)

 [Your reply](#)

Reply to this...

 [Ukognes · @UKognes](#) 3m

Thank so much for this great write up.

I have a question regarding the web3.js ico announcement page.

It seems that the js only runs if you have metamask installed on the browser.

Is there a way to display the "total raised" for all visitors regardless of which extensions they have running?

Perhaps I need to understand it better. Is there a write up anywhere you'd recommend?


Thanks!

 3

 [Reply](#) (1)



 [Share](#)

 [Save](#)

 [Sandeep Panda](#) 2m

Sorry about the late response. You can try [Ethers-js](#) and fetch the balance. In fact, I should have used that in the article instead of relying on web3. Thanks for pointing out.

 [Upvote](#)

 [Your reply](#)

Reply to this...

 [Sebastian · @sebdn](#) 2m

Great job! I have a problem however:

I have set that unitsOneEthCanBuy = 1000;

In MetaMask (Eth net): when I send 1 ETH I got 10.000 of mycoin..
when I send 0.001 ETH I got 0.010 - why is that?


In MyEtherWallet: when I send 0.001 ETH I got 0.1 mycoin...

Why there are differences? Moreover, when I deploy the SAME contract in test net, and test it, it works properly... There are no differences in codes, only difference is that one is on real eth net, and the other in test net.

 2


 [Reply \(7\)](#)



 [Share](#)

 [Save](#)


[Show all replies](#)




Kallejo

21d

I have the same problem, all ok but if i send 0.05 eth yo contract, this contract dont send me mys tokens... Any solution?




[Upvote](#)



Your reply.


Reply to this...





Gautam Lakum · [@gautamlakum](#)


3m


Thanks [Sandeep Panda](#) for posting this. It seems a long article, could't read fully. I will surely go through it.


 1

 [Reply \(1\)](#)



 [Share](#)


 [Save](#)




Eshwaran Veerabahu

3m

Not able to switch to Ropsten TestNet on the meta mask prompt.




[Upvote](#)



Your reply.

Reply to this...



rendy setyawan (The Rens) · [@JustR](#)

2m

Why I get error like this :

"creation of RENToken errored: Gas required exceeds block gas limit: 30000000. An important gas estimation might also be the sign of a problem in the contract code. Please check loops and be sure you did not sent value to a non payable function (that's also the reason of strong gas estimation). "

whats wrong ? I follow all tutorial but I get thats :(

1

Reply (2)

...

Share

Save

Show all replies

Wei Huang

20d

hi. Great article. I have question. On your final ICO webpage, there is starting and ending date, plus bonus for the first week buyer, do you have code example for that? Great thanks!

Upvote

Your reply

Reply to this...

Show more responses

The Dev Community

Free, friendly and inclusive platform for developers

Enter your email address

Sign up / Sign in

Or connect with

GitHub

Google

Facebook

LinkedIn




Blockchain Applications and Potential Across Industries

Blockchain is one of the most discussed buzzwords among tech entrepreneurs of today. I was

Blockchain easily explained..

Hey folks! I have written an easy explanation for Blockchain. Its actually a story of an ancient community who traded like a Blockchain but ...

[searching blockchain on google, I found that blog...](#)

 1

 4

[Blockchain for the React developer](#)

[Blockchain is about building decentralized apps. Apps and data which are controlled by a community rather than a single organization. Ethere...](#)

 4

[Playing around with simple block-chain principles](#)

[Introduction The preface can be skipped; it's just my reduced opinion about the whole hype of the real world application as _virtual money_.](#)

...

 8