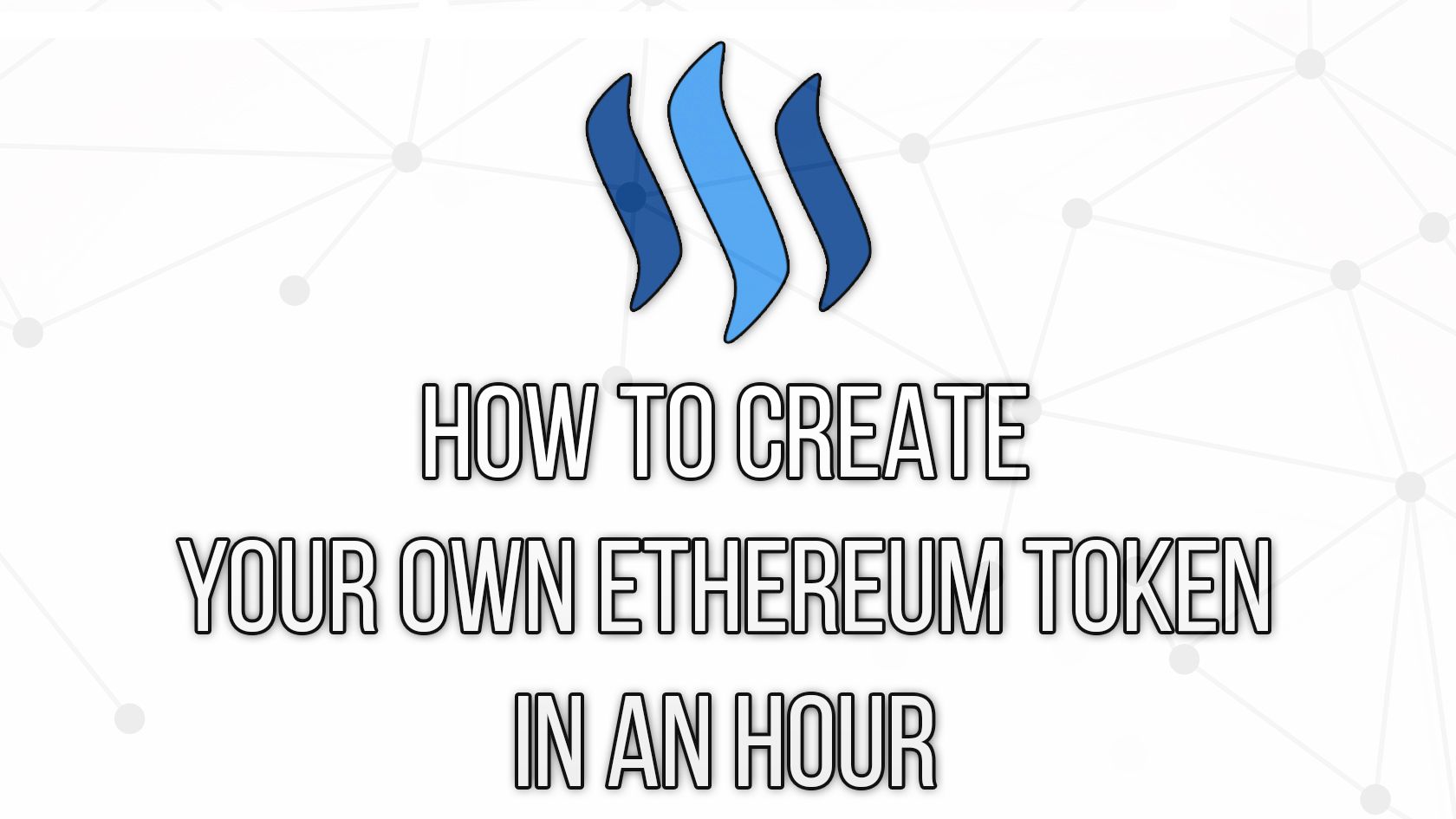
**How To Create Your Own Ethereum Token In An Hour (ERC20 + Verified)**



As part of my own education process, I wanted to create my own Ethereum token that would be viable to sell on an exchange. ICOs are all the rage these days, and I see them as a valuable avenue to raising funds for important projects in the world.

Thus [The Most Private Coin Ever](https://www.themostprivatecoinever.com/) was born.

Since I’ve launched the coin, I’ve gotten a lot of messages from people asking me how they can do the same. They’ve wanted to create their own token for fun, to learn, or for a business that they’re starting…

…but they don’t know where to start.

The challenge with programming in Ethereum is that their aren’t a *ton* of resources out there for how to do things. It’s still pretty new, with limited documentation and Stack Overflow responses.

And so in this tutorial, I’m going to make it simple.

I’m going to show you how to create your own Ethereum Token in as little as one hour, so you can use it for your own projects.

This token will be a standard ERC20 token, meaning you’ll set a fixed amount to be created and won’t have any fancy rules. I'll also show you how to get it verified so that it's uber legit.

Let’s get started.

**Step 1: Decide what you want your token to be**

In order to create an ERC20 token, you need the following:

* The Token’s Name
* The Token’s Symbol
* The Token’s Decimal Places
* The Number of Tokens in Circulation

For [The Most Private Coin Ever](https://www.themostprivatecoinever.com/) , I chose:

* Name: The Most Private Coin Ever
* Symbol: ???
* Decimal Places: 0
* Amount of Tokens in Circulation: 100,000

The decimal places is where things can get tricky. Most tokens have 18 decimal places, meaning that you can have up to .0000000000000000001 tokens.

When creating the token, you’ll need to be aware of what decimal places you’d like and how it fits into the larger picture of your project.

For me, I wanted to keep it simple and wanted people to either have a token, or not. Nothing in between. So I chose 0. But you could choose 1 if you wanted people to have half a token, or 18 if you wanted it to be ‘standard’.

**Step 2: Code the Contract**

Here is a contract that you can essentially "Copy/Paste" to create your ERC20 token. Shoutout to [TokenFactory](https://github.com/ConsenSys/Token-Factory" \o "This link will take you away from steemit.com) for the source 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;

}

//name this contract whatever you'd like

contract ERC20Token is StandardToken {

function () {

//if ether is sent to this address, send it back.

throw;

}

/\* 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; //fancy name: eg Simon Bucks

uint8 public decimals; //How many decimals to show. ie. There could 1000 base units with 3 decimals. Meaning 0.980 SBX = 980 base units. It's like comparing 1 wei to 1 ether.

string public symbol; //An identifier: eg SBX

string public version = 'H1.0'; //human 0.1 standard. Just an arbitrary versioning scheme.

//

// CHANGE THESE VALUES FOR YOUR TOKEN

//

//make sure this function name matches the contract name above. So if you're token is called TutorialToken, make sure the //contract name above is also TutorialToken instead of ERC20Token

function ERC20Token(

) {

balances[msg.sender] = NUMBER\_OF\_TOKENS\_HERE; // Give the creator all initial tokens (100000 for example)

totalSupply = NUMBER\_OF\_TOKENS\_HERE; // Update total supply (100000 for example)

name = "NAME OF YOUR TOKEN HERE"; // Set the name for display purposes

decimals = 0; // Amount of decimals for display purposes

symbol = "SYM"; // Set the symbol for display purposes

}

/\* 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;

}

}

Throw this into your favorite text editor ([Sublime](https://www.sublimetext.com/) is my personal favorite).

You’re going to want to replace everything in the area where it says “CHANGE THESE VARIABLES FOR YOUR TOKEN”.

So that means, change:

* The token name
* The token symbol (I'd go no more than 4 characters here)
* The token decimal places
* How much you as the owner want to start off with
* The amount of tokens in circulation (to keep things basic, make this is the same amount as the owner supply)

Some things to keep in mind:

1. The supply that you set for the token is correlated to the amount of decimal places that you set.

For example, if you want a token that has 0 decimal places to have 100 tokens, then the supply would be 100.

But if you have a token with 18 decimal places and you want 100 of them, then the supply would be 100000000000000000000 (18 zeros added to the amount).

1. You set the amount of tokens you receive as the creator of the contract.

That’s what this line of code is:

balances[msg.sender] = NUMBER\_OF\_TOKENS\_HERE;

Whatever you set here will be sent to the ETH wallet of wherever you deploy the contract. We’ll get to that in a few.

But for now just set this equal to the supply so that you receive all the tokens. If you’re wanting to be more advanced with the token logic, you could set it with different rules, like different founders of your projects would receive different amounts or something like that.

Once you have all the variables in, it’s now time to deploy it to the blockchain and test it.

**Step 3: Test The Token on The TestNet**

Next we’re going to deploy the contract to the Test Net to see if it works. It sucks to deploy a contract to the MainNet, pay for it, and then watch it fail.

I *may* have done that while creating my own token….so heed the warning.

First, if you don’t have it already, download [MetaMask](https://metamask.io/" \o "This link will take you away from steemit.com). They have an easy-to-use interface to test this.

Once you’ve installed MetaMask, make sure that you’re logged in and setup on the Ropsten test network. If you click in the top left where it says ‘Main Ethereum Network’ you can change it to Ropsten.

To confirm, the top of your MetaMask window should look like this:

**This Wallet is going to be the ‘Owner’ of the contract, so don’t lose this wallet! If you’d like it not to be Metamask, you can use Mist or MyEtherWallet to create contracts as well. I recommend using MM for simplicity, and you can always export your private key to MyEtherWallet for usage later.**

Now head to the [Solidity Remix Compiler](https://ethereum.github.io/browser-solidity/) - it’s an online compiler that allows you to publish the Smart Contract straight to the blockchain.

Copy/Paste the source of the contract you just modified into the main window. It'll look something like this:



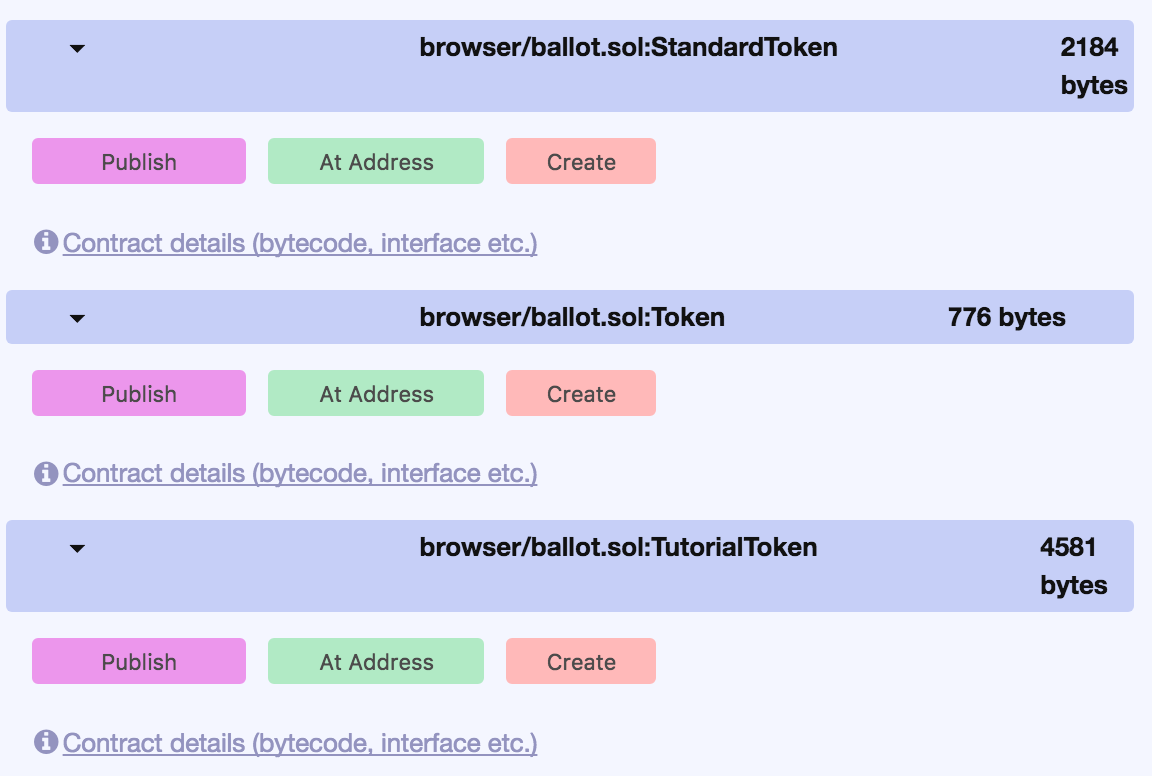
Now, go to settings on the right and select the latest release compiler version (NOT nightly), as well as unchecking ‘Enable Optimization’.

So it should look something like this:

Keep note of the current Solidity version in the compiler. We’ll need that later to verify the contract source.

Now go back to the Contract tab and hit ‘Create’ under the name of the Token function that you created.

So you’d hit ‘Create’ under ‘TutorialToken’.

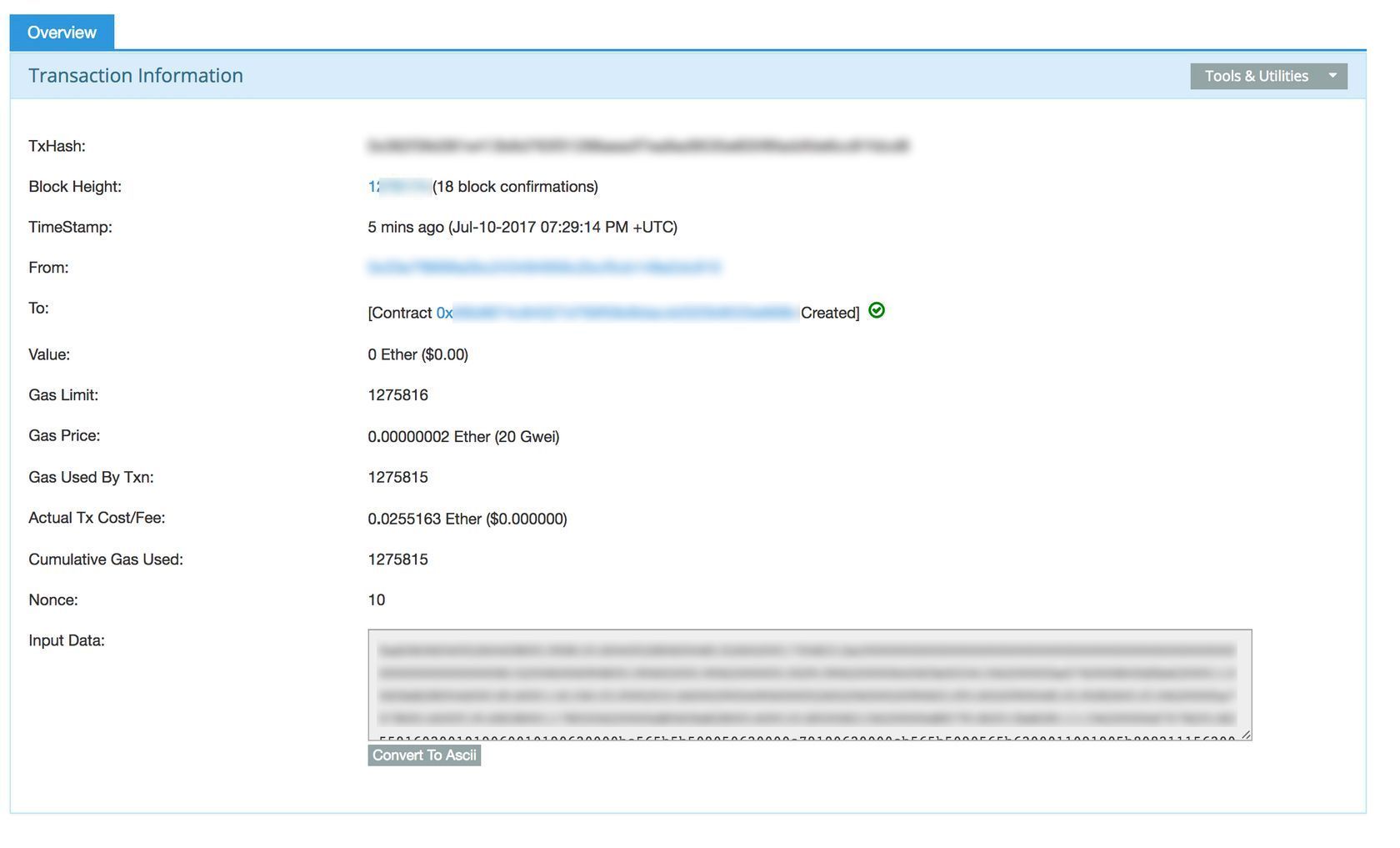


What will happen is MetaMask will pop up asking you to hit ‘Submit’ to pay for the transaction.

Remember, this is the Ropsten test net, so it’s not real Ether. You can double check to make sure you’re on the test network in MetaMask before you hit submit just to be sure.

When you hit Submit, it’ll say ‘Contract Pending’ in MetaMask. When it’s ready, click the ‘Date’ and it’ll bring up the transaction in EtherScan. Like this:

If you click the date, it’ll bring up the following screen:



If this process worked, it’s now time to verify the source code.

If not, you’ll want to go back to the code and modify the source to get it to work.

I can’t say exactly what that’d look like, but this is where having a “programmers mindset” comes in. A lot of time there’s unexpected bugs that can’t be predicted until you just do it.

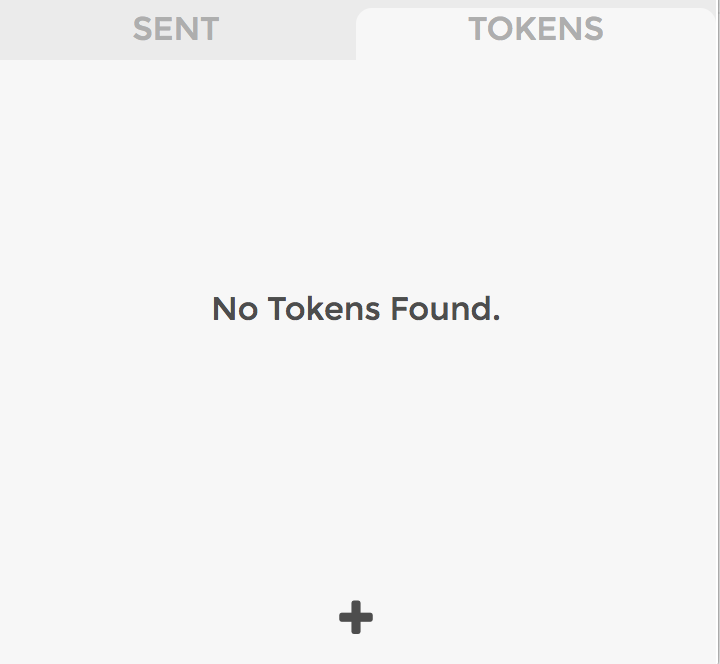
**Step 3.5. Watch The Custom Token**

Now let’s see if it actually created the tokens and sent them to me.

Copy the Contract Address that’s listed in the Transaction information (see screenshot above).

In this case, for me, it’s 0x5xxxxxxxxxxxxxxxx.

I’m going to add that to MetaMask ‘Tokens’ tab:

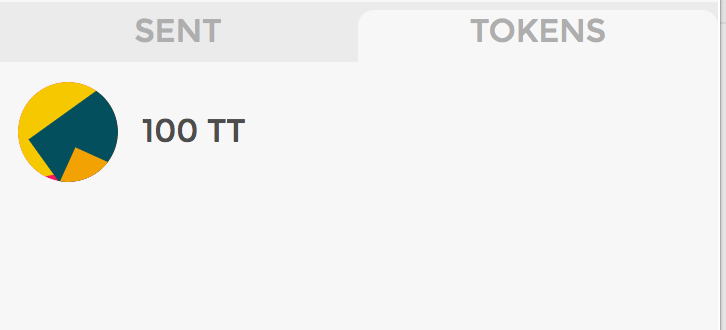


When I click the “+” button, I can paste it in there and it’ll automatically insert the information of the token, like so:

Let’s hit ‘Add’.

Sweet! It says I have the 100 tokens that I created - it worked!

Now I can send those tokens, or sell them on a market if I decide to do so.



Boo ya!

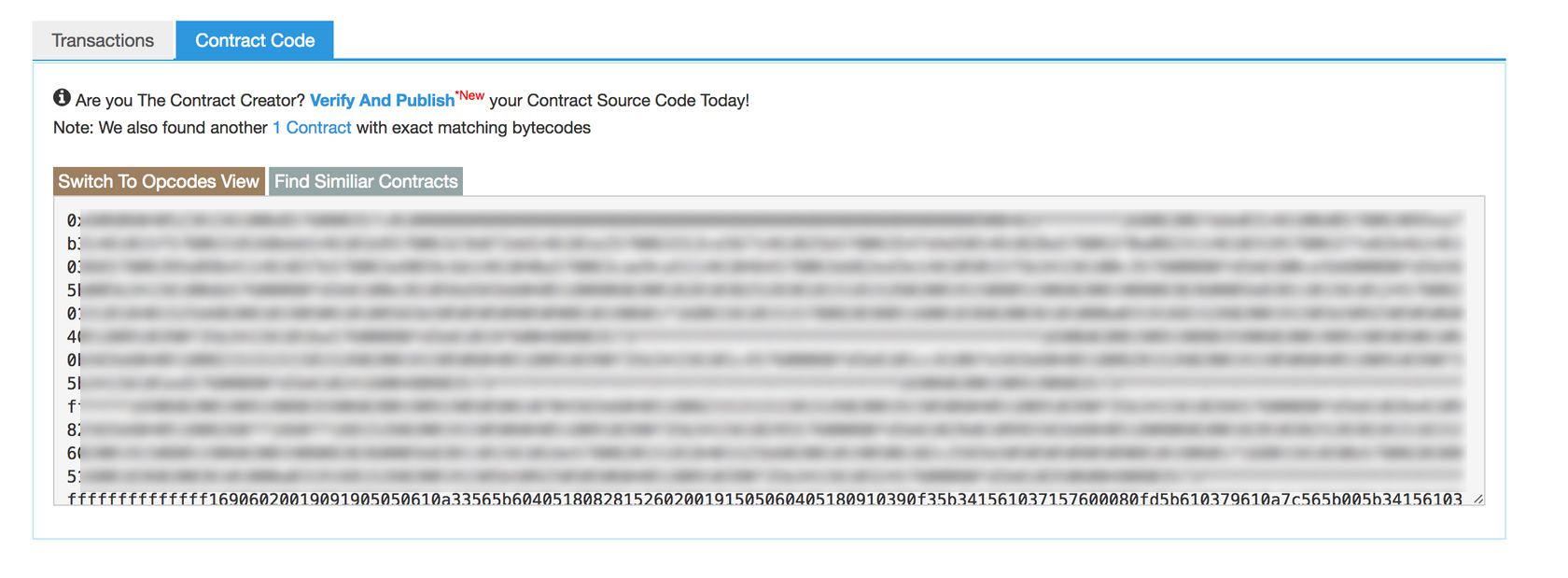
**Step 4. Verify the Source Code**

This is going to be important for various reasons, mainly verifying the validity of your token to the public.

It doesn’t technically matter, and your token will still be usable if you don’t do it, but it’s good practice to verify it so people know that you’re not doing anything shady.

When you’re on the Transaction screen from the previous step, click where it says [Contract xxxxxx Created] in the To: field. That’s the Contract we just published.

Then click the ‘Contract Code’ tab.



Hit ‘Verify and Publish’. That’ll bring you to this screen:

Here’s where you NEED to have the right settings.

The Contract Address will already be filled in.

For Contract Name, make sure to put the name of the function that you modified for your custom token. The default in the code I gave you is ERC20Token, but if you renamed it, make sure you put that.

I renamed mine ‘TutorialToken’ for this tutorial, so I put that.

For Compiler, select the SAME compiler you used in the Solidity Compiler. Otherwise you will not be able to verify your source code!

Make sure Optimization is disabled as well.

Then Copy/Paste the code from the compiler right into the Contract Code field (the big one).

Hit submit.

If you did the steps right, you’ll get something like this:

  
That means it was verified. Woot!

If you go to the Contract address page, you’ll see that ‘Contract Source’ says Yes and says that it’s Verified. Like this:

If not, double check your steps, or make a comment in this thread and we’ll see what went wrong.

**Step 5. Get it on The Main Net**

Now that everything’s working, it’s time to deploy it on the MainNet and let other people use it.

This part is simple.

Just do steps 3 & 4, but instead of being connected to the Ropsten Test Network, you want to be connected to the MainNet. Make sure your MetaMask account is in Mainnet mode.



You’ll need to fund your contract with real Ether to do this. This cost me ~$30 USD when I did this for [The Most Private Coin Ever](https://www.themostprivatecoinever.com/)

**Step 6. Get it Verified on Etherscan**

This step is not required, but it adds to the validity of your token if you get it verified by them.

You can see how my token is verified by how it has the logo and it doesn't say "UNVERIFIED" next to it.

[Click here to see what I mean](https://etherscan.io/token/0x1a645debd700890f1bc93626078d89e260bd09ce)

To do this you’ll need to go to the [Etherscan Contact Us Page](https://etherscan.io/contactus" \o "This link will take you away from steemit.com)) and send them an e-mail with the following information:

1. Ensure that you token contract source code has been verified

2. Provide us with your Official Site URL:

3. Contract Address:

4. Link to download a 28x28png icon logo:

So yes, you'll need to have a website with a stated purpose of the token.

They’ll get back to you and let you know if they’ll verify it or not. Remember, Etherscan is a centralized service so it's very possible that they will not verify your token if they don't deem it worthy.

**Congrats! & Next Steps**

You now have a token on the ETH MainNet that other people can use. It's now primed to be sent to others and received.

To buy and sell, you'll need to get your token listed on an exchange. But that's a tutorial for another day :)

For now, enjoy your newly created (and personal) ERC20 & Verified Ethereum Token!!