Deploy ERC20 Contract

The original and more readable Markdown version of this report can be found at the following link: https://github.com/chutrunganh/Blockchain-and-Applications-IT4527E/tree/master/Lab_02

> 💡 Tip
>
> You can conduct this lab using the Remix IDE for fast startup with zero configuration, or run it locally using Hardhat for a more realistic development environment. This will require using WSL and some additional configuration steps.

Details about the ERC20 standard can be found at: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/. Explained in simple terms, it is a contract that contains 9 basic functions:

Function	Type	Purpose	Description & Example
`name()`	`view`	Return token full name	Returns the token's **full name** (e.g., `"USD Coin"`). 📌 Used in wallets and explorers.
`symbol()`	`view`	Return token ticker symbol	Returns the **symbol** (like stock tickers). 📌 Example: `"USDC"`, `"DAI"`, `"UNI"`.
`decimals()`	`view`	Decimal precision	Defines how many **decimal places** the token supports. 📌 If `decimals = 18`, then `1 token = 10^18` units.
`totalSupply()`	`view`	Provides total supply of tokens	Returns how many tokens exist in total. 🔶 *Ex:* If a token has 1 million total tokens, `totalSupply()` returns `1,000,000`.
`balanceOf(address _owner)`	`view`	Check balance of an account	Returns how many tokens a given address owns. 🔶 *Ex:*

			`balanceOf(0xAbC...)` might return `1000` tokens.
`transfer(address _to, uint256 _value)`	`public`	Directly send tokens	Transfers `_value` tokens from **msg.sender** to `_to`. 🔶 *Ex:* Alice sends 100 tokens to Bob → `transfer(Bob, 100)`
`transferFrom(address _from, address _to, uint256 _value)`	`public`	Transfer on behalf of another	Allows an **approved address** (like a contract or user) to transfer tokens from `_from` to `_to`. 🔶 *Ex:* Uniswap calls `transferFrom(Alice, Pool, 300)` to take tokens after Alice approved it.
`approve(address _spender, uint256 _value)`	`public`	Authorize spending	Allows `_spender` to spend up to `_value` tokens from **your** account. Used for delegated spending. 🔶 *Ex:* Alice approves Uniswap to spend 500 tokens → `approve(Uniswap, 500)`
`allowance(address _owner, address _spender)`	`view`	Check remaining approved tokens	Shows how many tokens `_spender` is still allowed to spend from `_owner`. 🔶 *Ex:* `allowance(Alice, Uniswap)` returns `200`, meaning Uniswap can still spend 200 more tokens from Alice.

And two events:

Event	**Type**	**Purpose**	**Description & Example**
`Transfer`	`event`	Log token transfer	Emitted when tokens are transferred (`transfer()` or `transferFrom()`). Used by wallets and block explorers to track transactions.

`Approval`	`event`	Log approval of allowance	Emitted when an `approve()` call is made. Helps dApps and wallets show which contracts have permission to spend tokens.

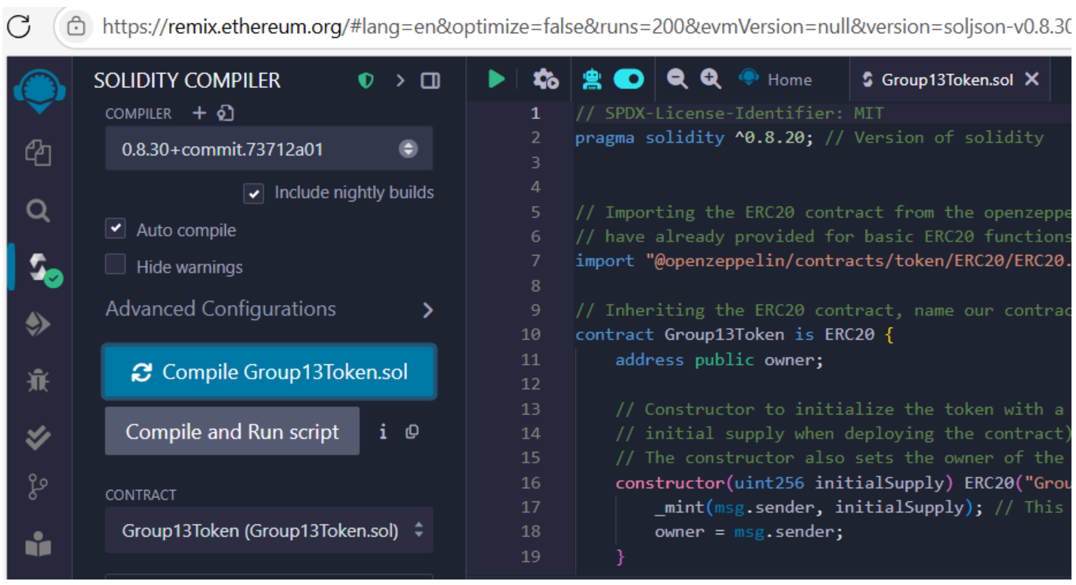
That is enough for the theory, now we will write the contract code. Let's try to write the code and deploy on Remix IDE VM first:

Overview of the project structure:

```
Lab_02/
├── contracts/
│   ├── Group13Token.sol          # ERC-20 token definition contract
│   └── Group13TokenSale.sol      # Token sale contract with tiered
│
├── scripts/ (only needed if run locally)
│   └── deploy.js                 # Script to deploy the contracts
│
├── test/
│   └── Group13TokenSale_test.js  # Unit tests for the token sale co
│
├── hardhat.config.js             # Hardhat configuration file (only
├── README.md
└── Requirement.md                # Requirements for the Lab 02
```

1. Token definition contract

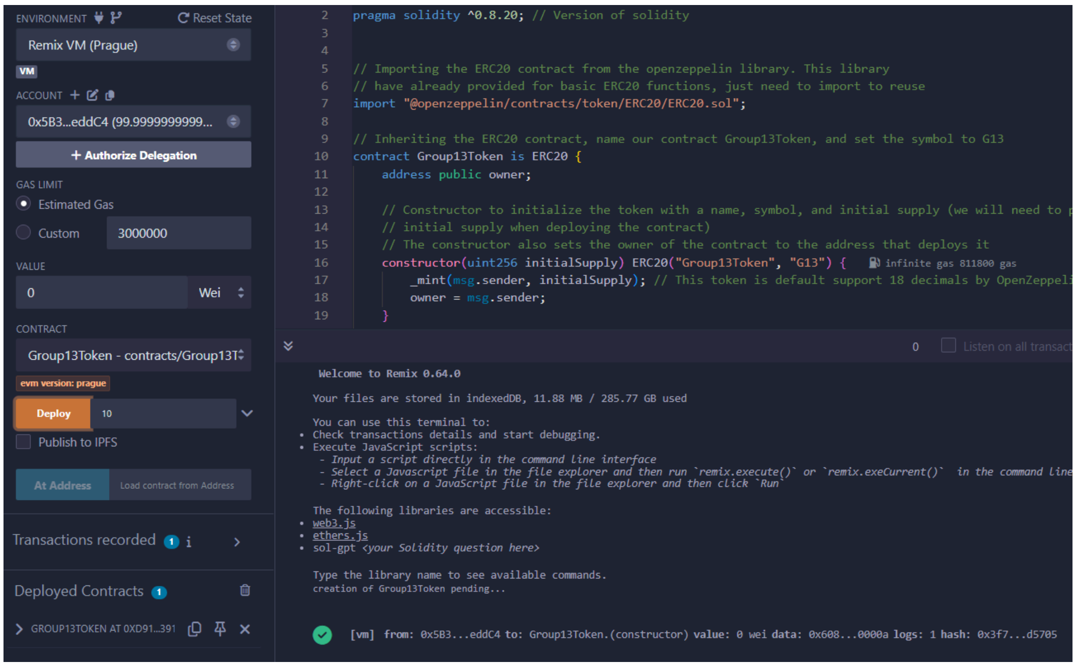
Start with the contract that defines the ERC20 token: (the source code can be found in this file):



```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20; // Version of solidity


// Importing the ERC20 contract from the openzeppe
// have already provided for basic ERC20 functions
import "@openzeppelin/contracts/token/ERC20/ERC20.

// Inheriting the ERC20 contract, name our contrac
contract Group13Token is ERC20 {
    address public owner;

    // Constructor to initialize the token with a
    // initial supply when deploying the contract)
    // The constructor also sets the owner of the
    constructor(uint256 initialSupply) ERC20("Grou
        _mint(msg.sender, initialSupply); // This
        owner = msg.sender;
    }
```

Code compiled successfully

Now deploy this token contract to the Remix IDE VM. Beside the `Deploy` button, we need to provide initial values for the constructor parameters, as in our code, it would be the `initialSupply` variable. We will init with, let's say, 10 tokens.

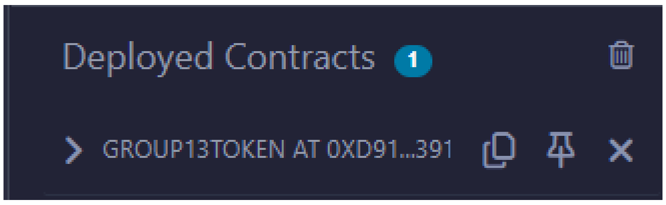


```
pragma solidity ^0.8.20; // Version of solidity


// Importing the ERC20 contract from the openzeppelin library. This library
// have already provided for basic ERC20 functions, just need to import to reuse
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

// Inheriting the ERC20 contract, name our contract Group13Token, and set the symbol to G13
contract Group13Token is ERC20 {
    address public owner;

    // Constructor to initialize the token with a name, symbol, and initial supply (we will need to p
    // initial supply when deploying the contract)
    // The constructor also sets the owner of the contract to the address that deploys it
    constructor(uint256 initialSupply) ERC20("Group13Token", "G13") {    infinite gas 811800 gas
        _mint(msg.sender, initialSupply); // This token is default support 18 decimals by OpenZeppeli
        owner = msg.sender;
    }
```

```
 Welcome to Remix 0.64.0

Your files are stored in indexedDB, 11.88 MB / 285.77 GB used

You can use this terminal to:
• Check transactions details and start debugging.
• Execute JavaScript scripts:
  - Input a script directly in the command line interface
  - Select a Javascript file in the file explorer and then run `remix.execute()` or `remix.exeCurrent()`  in the command line
  - Right-click on a JavaScript file in the file explorer and then click `Run`

The following libraries are accessible:
• web3.js
• ethers.js
• sol-gpt <your Solidity question here>

Type the library name to see available commands.
creation of Group13Token pending...

[vm] from: 0x5B3...eddC4 to: Group13Token.(constructor) value: 0 wei data: 0x608...0000a logs: 1 hash: 0x3f7...d5705
```

Deploy the contract with initial supply, make sure to choose the right contract to deploy, in this case it is `Group13Token`. The account that clicks `Deploy` will be the owner of the contract and will have all the tokens in its balance, in this case the owner is 0x5B3...eddC4

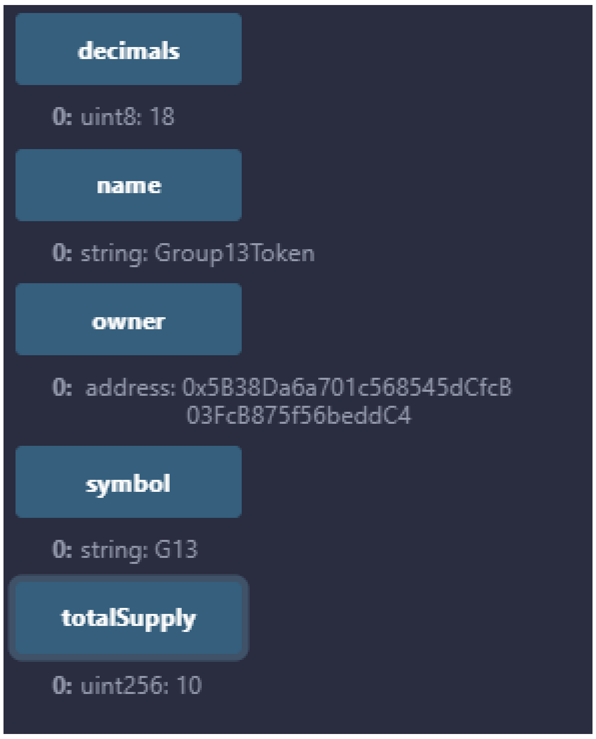
After deploying the contract, we will see the contract address in the `Deployed Contracts` section. Pay attention to this contract address since this is also the address of this token and we will need to use this in later steps, especially when we create the token sale contract.



The contract address is 0xD91...391

We can now test some functions/view variables of the token contract, such as:

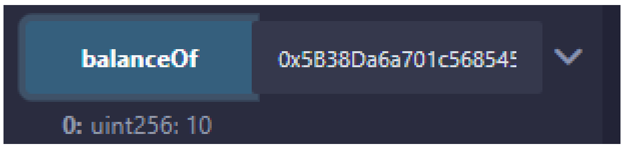
- Check the name, decimals, symbol, owner, totalSupply:



```
0: uint8: 18
0: string: Group13Token
0: address: 0x5B38Da6a701c568545dCfcB
            03FcB875f56beddC4
0: string: G13
0: uint256: 10
```

`decimals` is 18 by default of OpenZeppelin ERC20, name is Group13Token, `symbol` is G13 and `totalSupply` is 10 as we set in the constructor.

- Check balanceOf function:

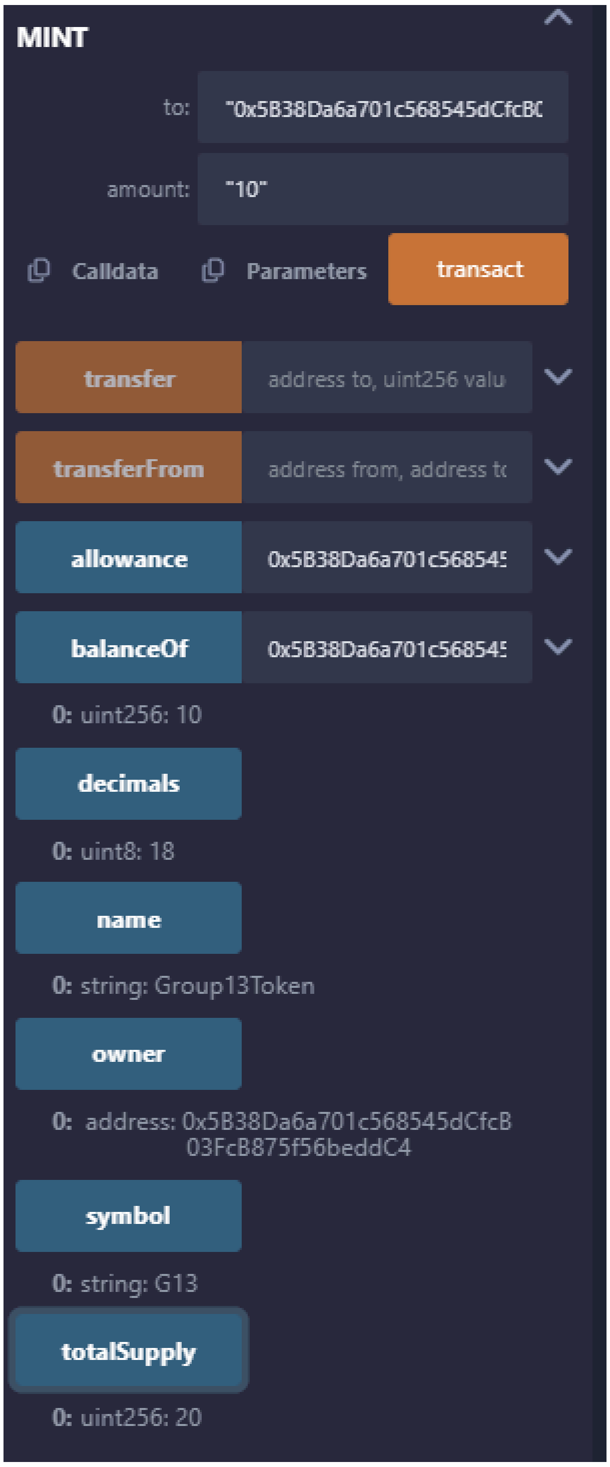


```
0: uint256: 10
```

As we mentioned before, the owner of the contract will have all the tokens in its balance, so the balance of the owner should be equal to the total supply, which is 10 tokens in this case.

- Check the mint function:

Try to mint more tokens using the mint function with parameter `(owner_address, 10)` to create 10 more tokens for the owner.



to: "0x5B38Da6a701c568545dCfcB0
amount: "10"

transfer: address to, uint256 valu
transferFrom: address from, address to
allowance: 0x5B38Da6a701c56854
balanceOf: 0x5B38Da6a701c56854

```
0: uint256: 10
```

decimals

```
0: uint8: 18
```

name

```
0: string: Group13Token
```

owner

```
0: address: 0x5B38Da6a701c568545dCfcB
            03FcB875f56beddC4
```

symbol

```
0: string: G13
```

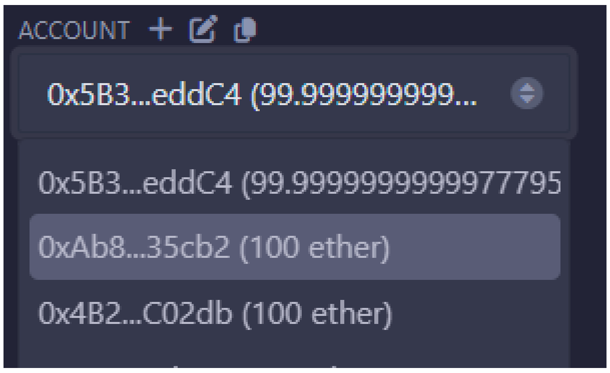
totalSupply

```
0: uint256: 20
```

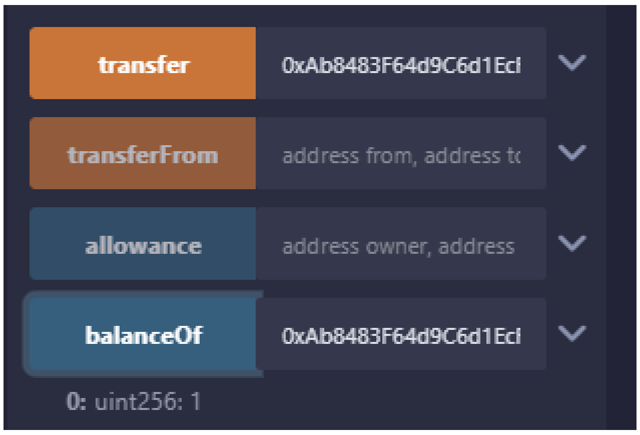
Notice that when you re-run the `totalSupply` *function, it should return 20 tokens as image shows*

- Check the transfer function:

Switch to a second address and copy its address, it acts as the receiver of the transfer, in this case: `0xAb8...35cb2`.

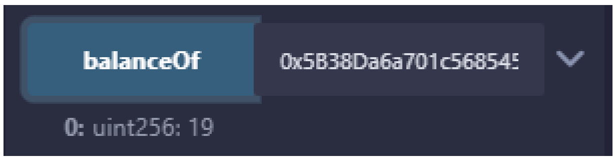

ACCOUNT
0x5B3...eddC4 (99.999999999...
0x5B3...eddC4 (99.9999999999977795
0xAb8...35cb2 (100 ether)
0x4B2...C02db (100 ether)

Copy this receiver address. Then switch back to the owner account (the one that deployed the contract) and use the `transfer` function with parameters `(receiver_address, 1)` to send 1 token to the receiver address.



Transfer 1 token to the receiver address `0xAb8...35cb2`

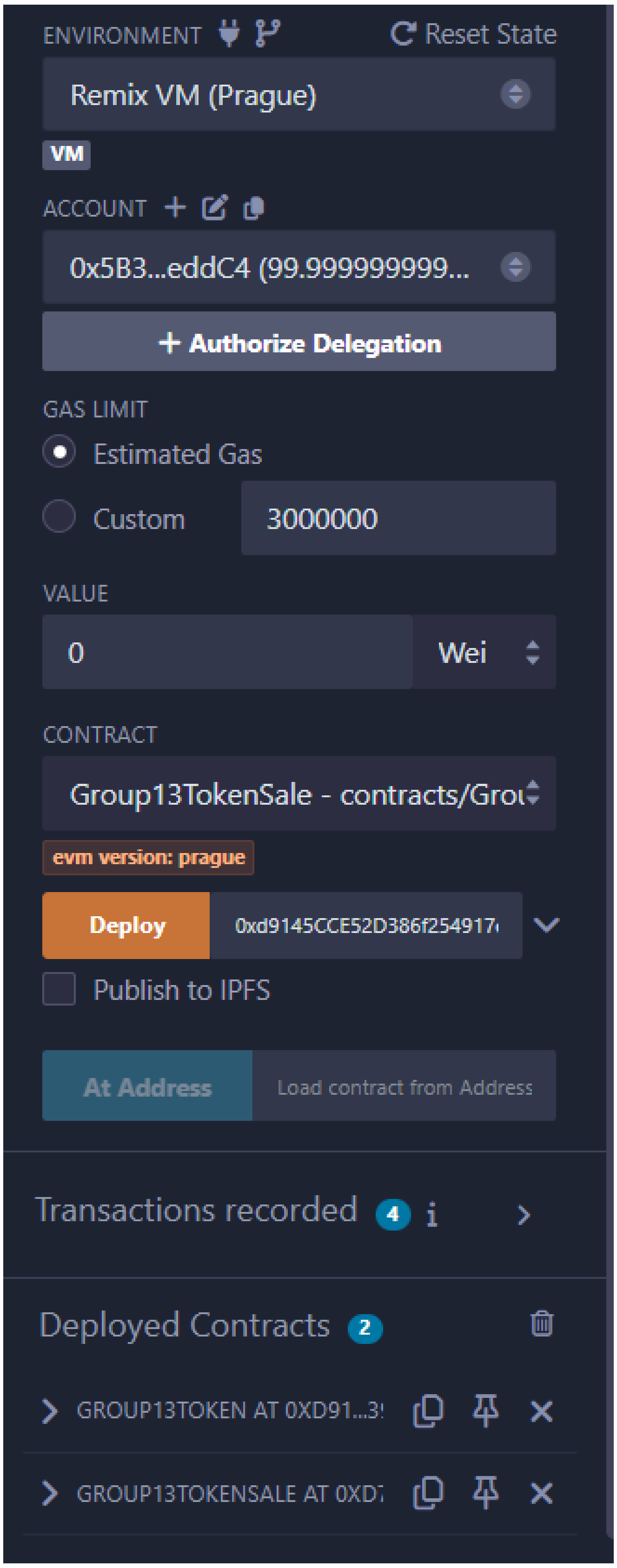
See that the balance of the receiver address is now 1 token, and then check the balance of the owner, it should be 19 tokens now.



2. Token Sale Contract

Okay, after successfully creating our token, the next step is to implement the token sale contract, ensuring it adheres to the constraints specified in the Requirement_Lab02.md:

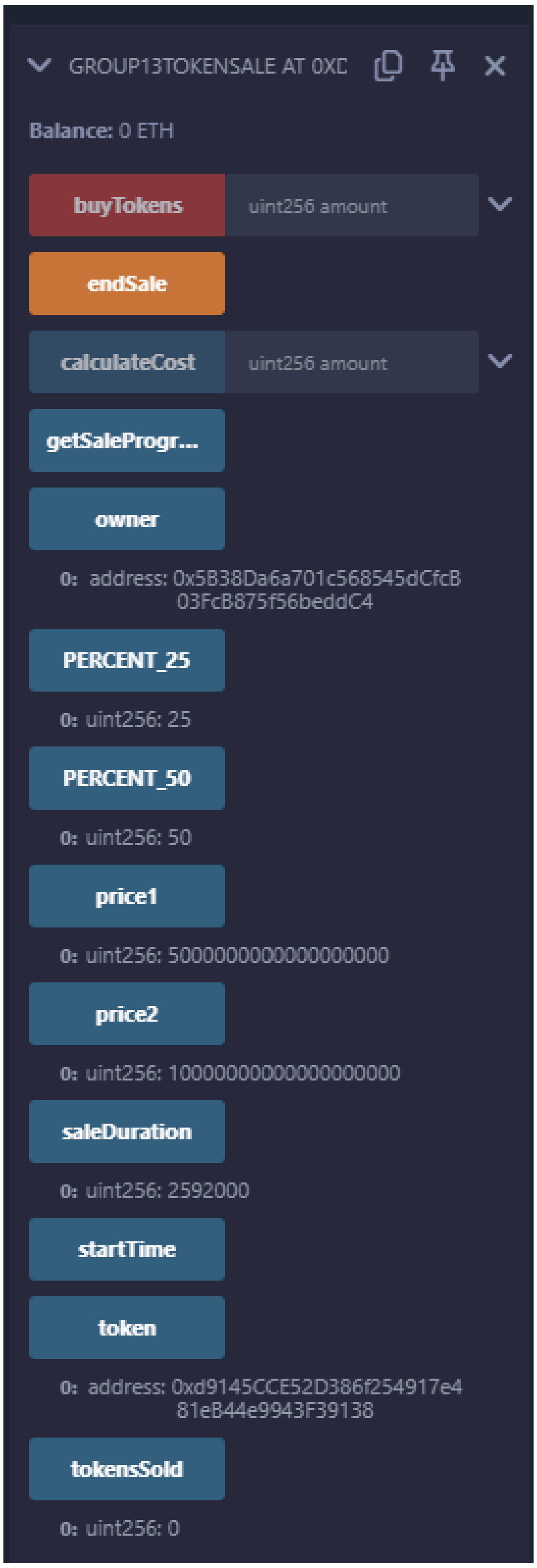
The source code for the token sale contract can be found in this file. To deploy this, click on the file, choose the `Group13TokenSale` contract in Remix IDE. This contract requires one parameter in the constructor, which is the address of the token contract we just deployed, copy the address of the token contract from the previous step and paste it into the constructor parameter field.


ENVIRONMENT
Reset State
Remix VM (Prague)
VM
ACCOUNT
0x5B3...eddC4 (99.999999999...
+ Authorize Delegation
GAS LIMIT
Estimated Gas
Custom 3000000
VALUE
0 Wei
CONTRACT
Group13TokenSale - contracts/Grou
evm version: prague
Deploy 0xd9145CCE52D386f254917
Publish to IPFS
At Address Load contract from Address
Transactions recorded 4
Deployed Contracts 2
GROUP13TOKEN AT 0XD91...3!
GROUP13TOKENSALE AT 0XD7

Copy token address contract as parameter to the constructor of the token sale contract

See that we now have two contracts in the `Deployed Contracts` section, one is the token contract and the other is the token sale contract.

Check some variables of this contract:

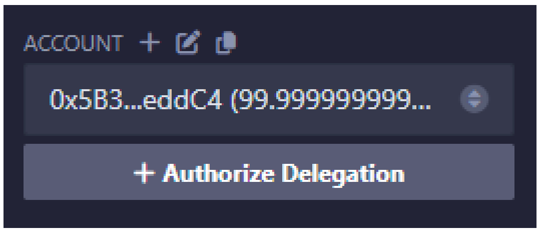


```
0: address: 0x5B38Da6a701c568545dCfcB
            03FcB875f56beddC4
0: uint256: 25
0: uint256: 50
0: uint256: 5000000000000000000
0: uint256: 10000000000000000000
0: uint256: 2592000
0: address: 0xd9145CCE52D386f254917e4
            81eB44e9943F39138
0: uint256: 0
```

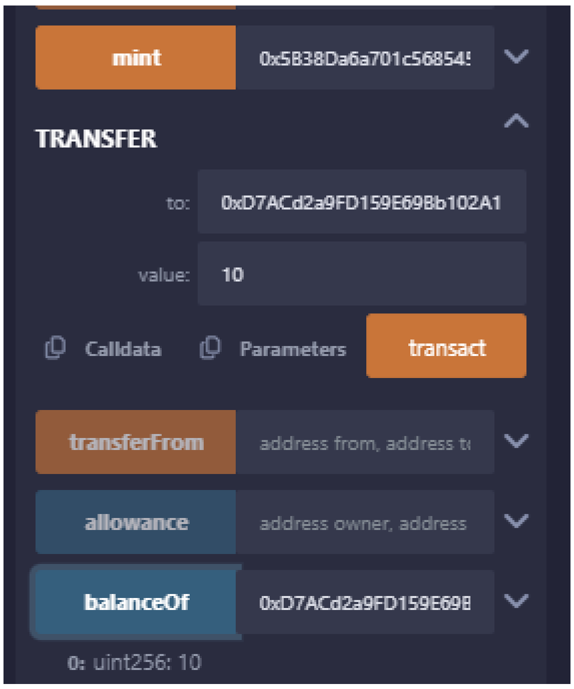
This constract has custom buyTokens and endSale functions that we implemented

In the code, we specify the `price1 = 5 ether`, with ether keyword just a unit conversion to wei. Solidity uses wei as its base, so the price will be displayed as 5 * 10^18 wei, which as you see in the image, is 5000000000000000000.

Now switch back to the owner account (the one that deployed the token contract) and transfer some tokens to the token sale contract address. This is necessary so that the token sale contract can sell those tokens to other users.

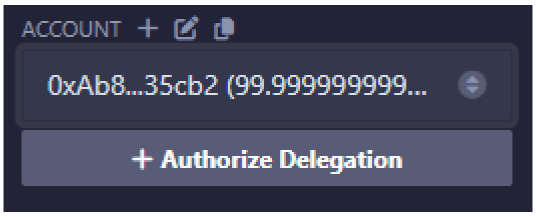


Copy the address of the token sale contract from the `Deployed Contracts` section, and then use the `transfer` function of the token contract to transfer 10 tokens to the token sale contract address.



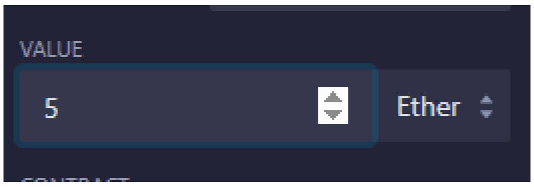
Transfer 10 tokens from token contract to the token sale contract address

Check the balance of the token sale contract address, it should be 10 tokens now. From now on, users can buy tokens from this contract. Let's switch to a different account that is not the owner of the token sale contract, for example, `0xAb8...` that we used before.

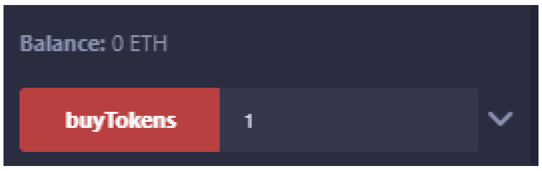


Then use the `buyTokens` function to buy tokens from the token sale contract. This function requires one parameter, which is the amount of tokens you want to buy. The price is 5 ether for 1 token (for the first 25%), 10 ether for the next 25% as mentioned in the sale contract.

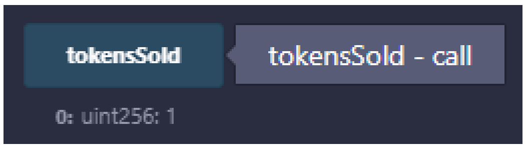
Change the value section to 5 ether to send 5 ether to include 5 eth in the transaction.



Then buy 1 token:

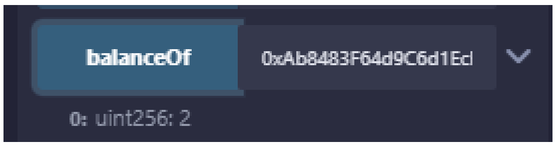


Check the variable tokensSold, it should be 1 token now:



```
0: uint256: 1
```

Then use the `balanceOf` function of the token contract to check the balance of the account that just bought the token, it should be adding one more token to the balance of the account:

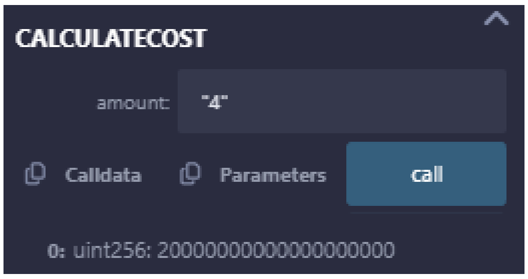


```
0: uint256: 2
```

Previously when we tested the transfer function of the token contract, we used the owner account to transfer 1 token to this second account, and now we have just bought 1 token, so the balance of this account should be 2 tokens.

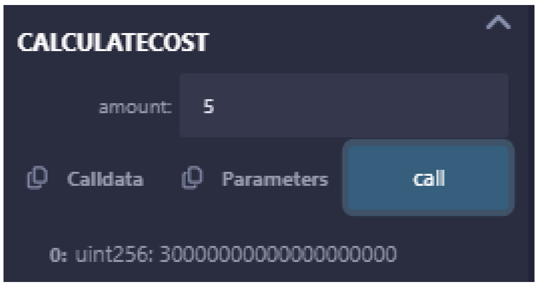
Our sale contract also includes a `calculateCost` function that can be used to calculate the cost of buying a certain amount of tokens. The total amount of tokens as we created in the token contract step is 20 tokens, so:

- Buying less than 25% of 20, let's say 4 tokens, the cost should be 4 * 5 ether = 20 ether:



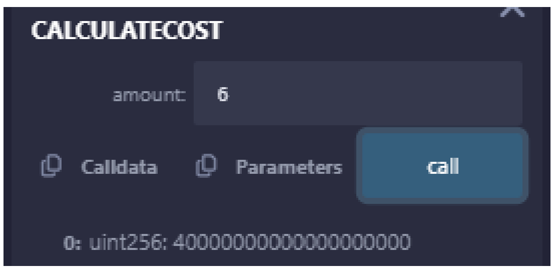
```
0: uint256: 20000000000000000000
```

- From 25%, let's say 5 tokens, the cost should be 4 * 5 ether + 1 * 10 ether = 30 ether:



```
0: uint256: 30000000000000000000
```

- Or 6 tokens, the cost should be 4 * 5 ether + 2 * 10 ether = 40 ether:

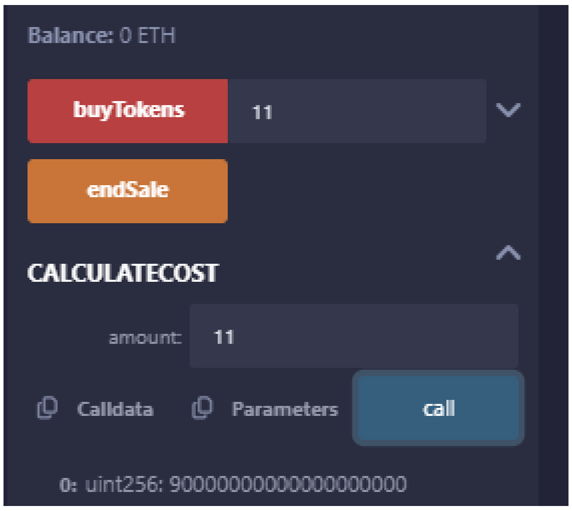


```
0: uint256: 40000000000000000000
```

Now let's test for other constraints in the requirement:

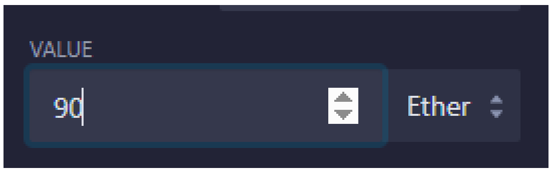
- Cannot buy more than 50% of the total supply:

Switch to the account not owner, try to buy 11 tokens, the price will be 4 * 5 ether + 7 * 10 ether = 90 ether:

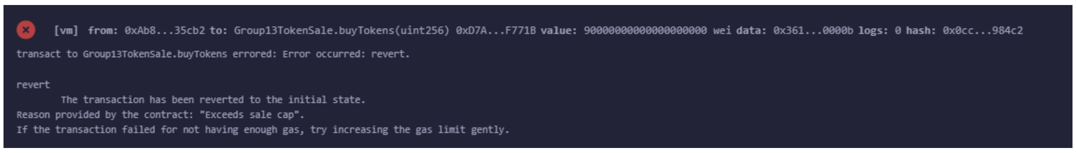


```
0: uint256: 90000000000000000000
```

Enter enough ether:



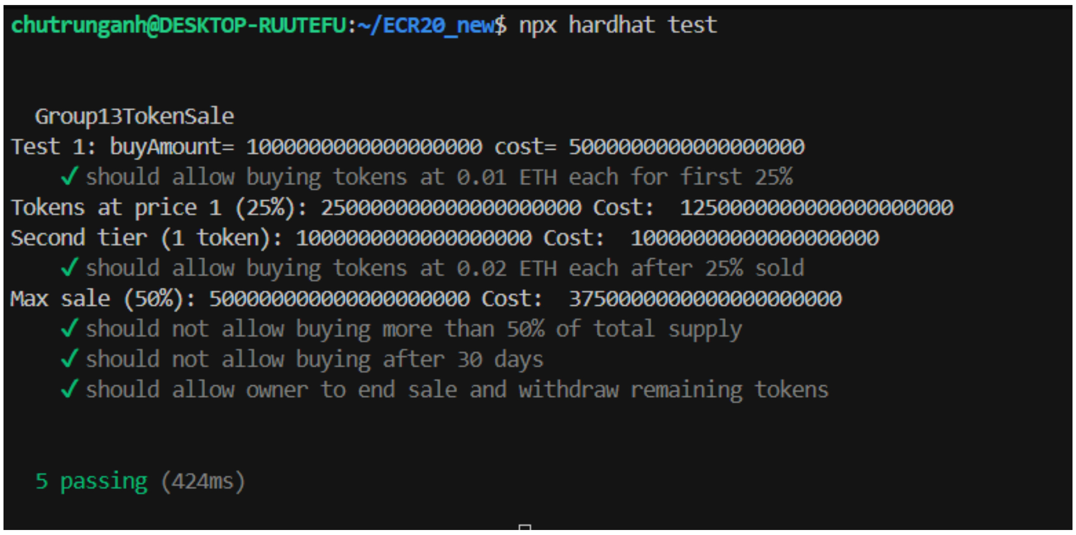
This should return an error like this indicating that the amount of tokens you are trying to buy is more than 50% of the total supply:



```
[vm] from: 0xAb8...35cb2 to: Group13TokenSale.buyTokens(uint256) 0xD7A...F771B value: 90000000000000000000 wei data: 0x361...0000b logs: 0 hash: 0x0cc...984c2
transact to Group13TokenSale.buyTokens errored: Error occurred: revert.

revert
	The transaction has been reverted to the initial state.
Reason provided by the contract: "Exceeds sale cap".
If the transaction failed for not having enough gas, try increasing the gas limit gently.
```

For a more comprehensive, automated testing, see our unit test file here.

The expected output of the test should be like this:



```
chutrunganh@DESKTOP-RUUTEFU:~/ECR20_new$ npx hardhat test


  Group13TokenSale
Test 1: buyAmount= 1000000000000000000 cost= 5000000000000000000
    ✓ should allow buying tokens at 0.01 ETH each for first 25%
Tokens at price 1 (25%): 250000000000000000000 Cost:  1250000000000000000000
Second tier (1 token): 1000000000000000000 Cost:  10000000000000000000
    ✓ should allow buying tokens at 0.02 ETH each after 25% sold
Max sale (50%): 500000000000000000000 Cost:  3750000000000000000000
    ✓ should not allow buying more than 50% of total supply
    ✓ should not allow buying after 30 days
    ✓ should allow owner to end sale and withdraw remaining tokens


  5 passing (424ms)
```

Deploy to Sepolia Testnet

Before deploying, modify the sale contract a little bit, comment out these two lines:

```
uint256 public price1 = 5 ether;  // Price per token for the first 2
uint256 public price2 = 10 ether; // Price per token for the remain
```

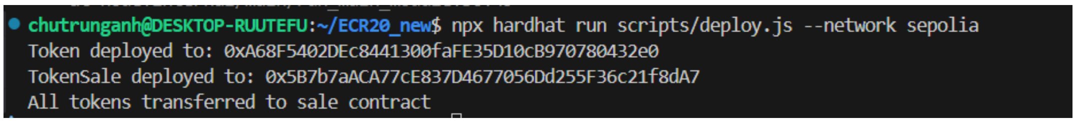
The requirement specifies that first 25% is 5 ether and next 25% is 10 ether, that is okay to deploy on local. However, I want to deploy these two contracts to the Sepolia testnet, I will change the ratio a little bit, since Sepolia testnet only gives you 0.05 ETH a day for testing. Therefore, we do not have the ETH to buy with 5 ether, 10 ether as current ratio, let's change it to 0.0005 ether and 0.001 ether. Uncomment these two lines:

```
uint256 public price1 = 0.0005 ether;  // Price per token for the fi
uint256 public price2 = 0.001 ether; // Price per token for the rema
```

To deploy you need the API key ...., place it in the env file

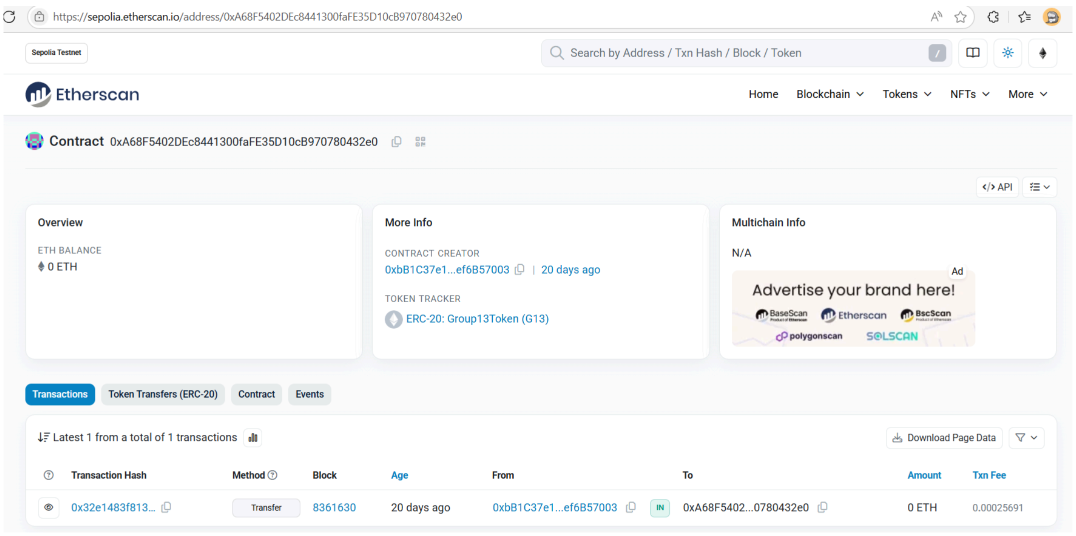
Now we can deploy the contract to the Sepolia testnet with command: (check the deployment script at the deploy.js file)

```
npx hardhat run scripts/deploy.js --network sepolia
```



```
chutrunganh@DESKTOP-RUUTEFU:~/ECR20_new$ npx hardhat run scripts/deploy.js --network sepolia
Token deployed to: 0xA68F5402DEc8441300faFE35D10cB970780432e0
TokenSale deployed to: 0x5B7b7aACA77cE837D4677056Dd255F36c21f8dA7
All tokens transferred to sale contract
```

Then copy these two addresses to https://sepolia.etherscan.io/. The result should be like this:



https://sepolia.etherscan.io/address/0xA68F5402DEc8441300faFE35D10cB970780432e0

Sepolia Testnet

Etherscan

Contract 0xA68F5402DEc8441300faFE35D10cB970780432e0

Overview
ETH BALANCE
0 ETH

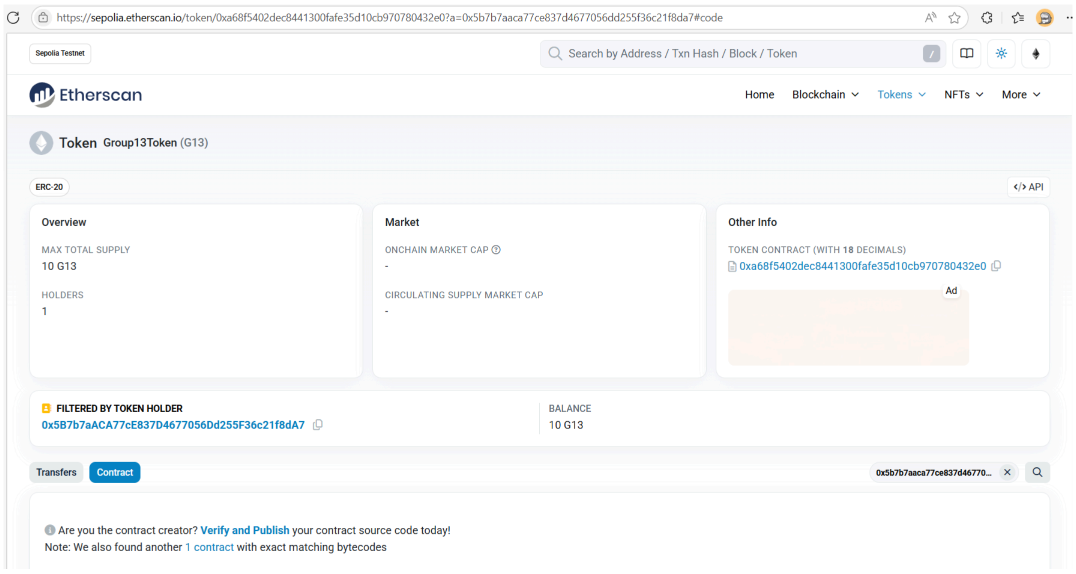
More Info
CONTRACT CREATOR
0xbB1C37e1...ef6B57003 | 20 days ago
TOKEN TRACKER
ERC-20: Group13Token (G13)

Multichain Info
N/A

Latest 1 from a total of 1 transactions

	Transaction Hash	Method	Block	Age	From		To	Amount	Txn Fee
	0x32e1483f813...	Transfer	8361630	20 days ago	0xbB1C37e1...ef6B57003	IN	0xA68F5402...0780432e0	0 ETH	0.00025691

Token contract



https://sepolia.etherscan.io/token/0xa68f5402dec8441300fafe35d10cb970780432e0?a=0x5b7b7aaca77ce837d4677056dd255f36c21f8da7#code

Sepolia Testnet

Etherscan

Token Group13Token (G13)

ERC-20

Overview
MAX TOTAL SUPPLY
10 G13
HOLDERS
1

Market
ONCHAIN MARKET CAP
-
CIRCULATING SUPPLY MARKET CAP
-

Other Info
TOKEN CONTRACT (WITH **18** DECIMALS)
0xa68f5402dec8441300fafe35d10cb970780432e0

FILTERED BY TOKEN HOLDER
0x5B7b7aACA77cE837D4677056Dd255F36c21f8dA7

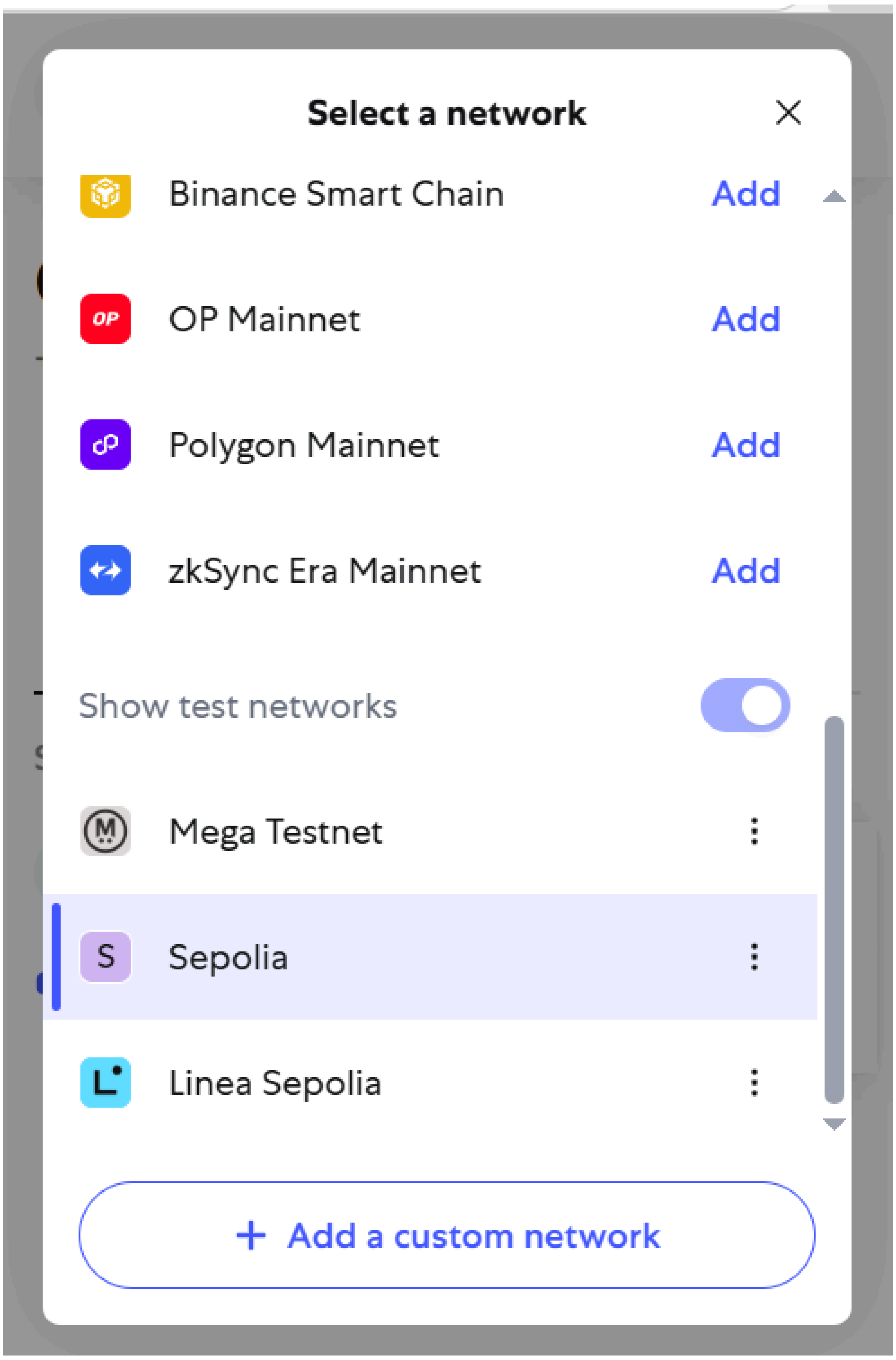
BALANCE
10 G13

Are you the contract creator? **Verify and Publish** your contract source code today!
Note: We also found another 1 contract with exact matching bytecodes

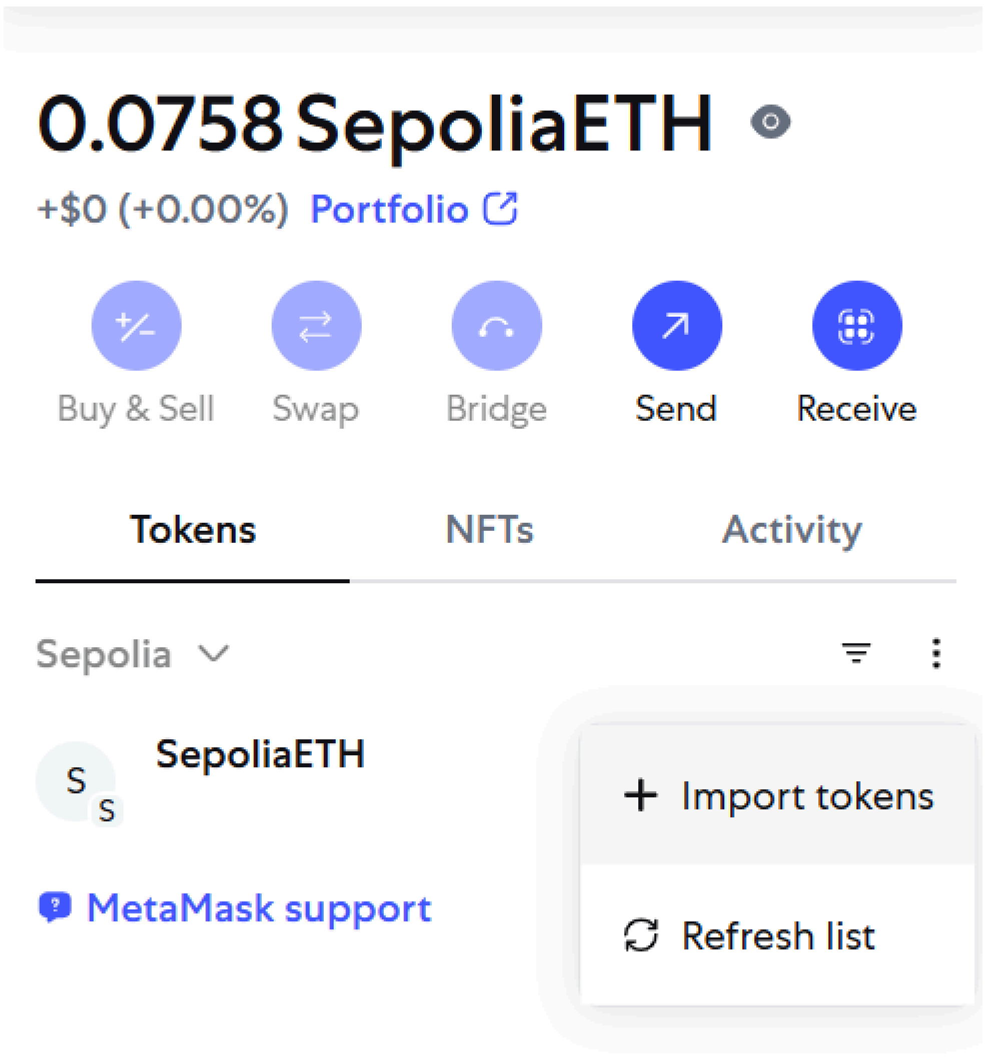
Search by Address / Txn Hash / Block / Token
Home
Blockchain
Tokens
NFTs
More
API
Ad
Transfers
Contract
0x5b7b7aaca77ce837d46770...

Sale contract

Try to import this token in the metamask wallet (use the first address shown in the image above, which is token contract). Choose Sepolia testnet in the metamask wallet, then click on `Import tokens`, paste the token address, it should be like this:



Click on `Import tokens`, then paste the address of G13Token contract:


0.0758 SepoliaETH

+$0 (+0.00%) Portfolio

Buy & Sell Swap Bridge Send Receive

Tokens NFTs Activity

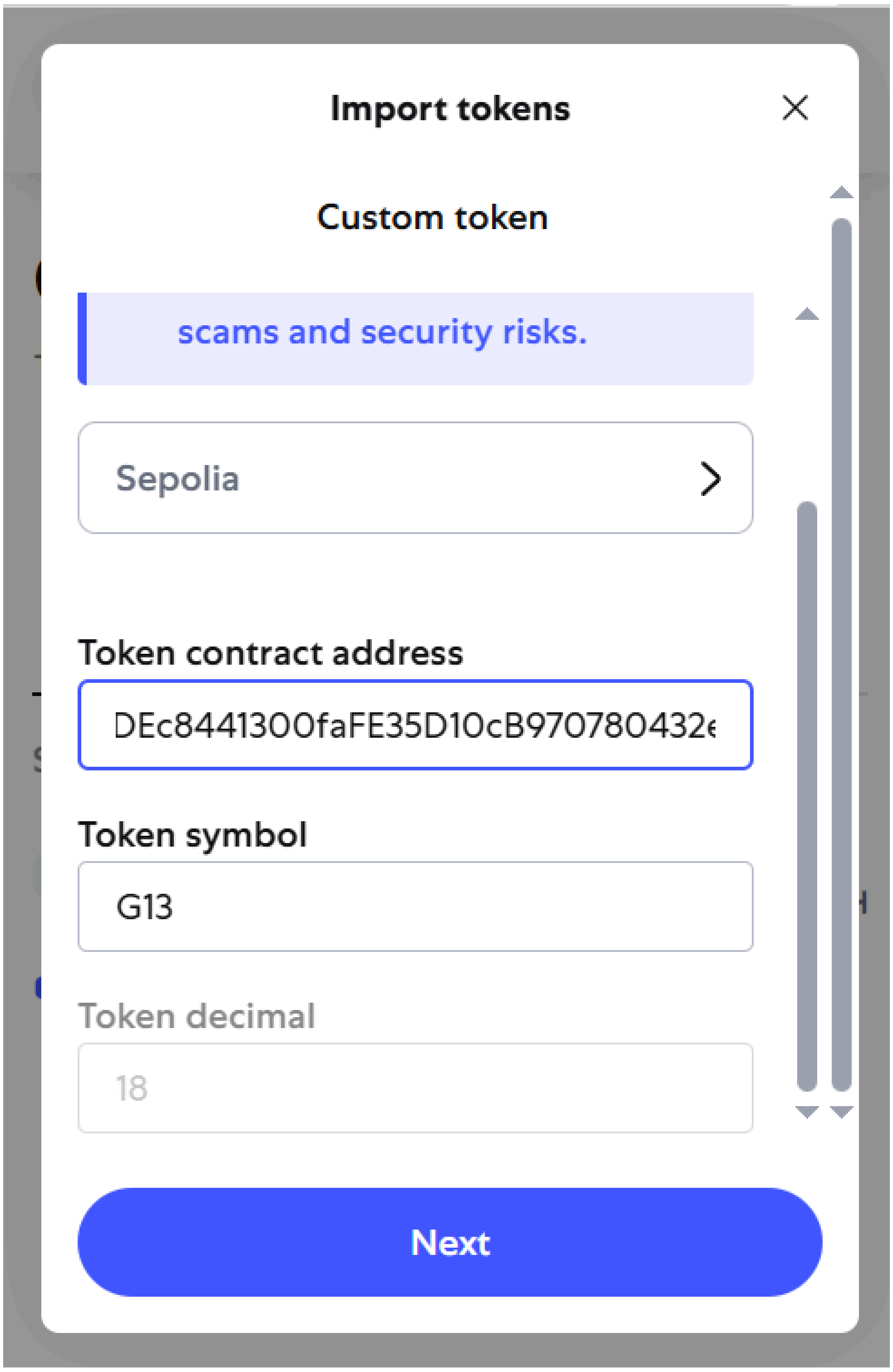
Sepolia

SepoliaETH

MetaMask support

+ Import tokens

Refresh list



Import tokens

Custom token

scams and security risks.

Sepolia

Token contract address

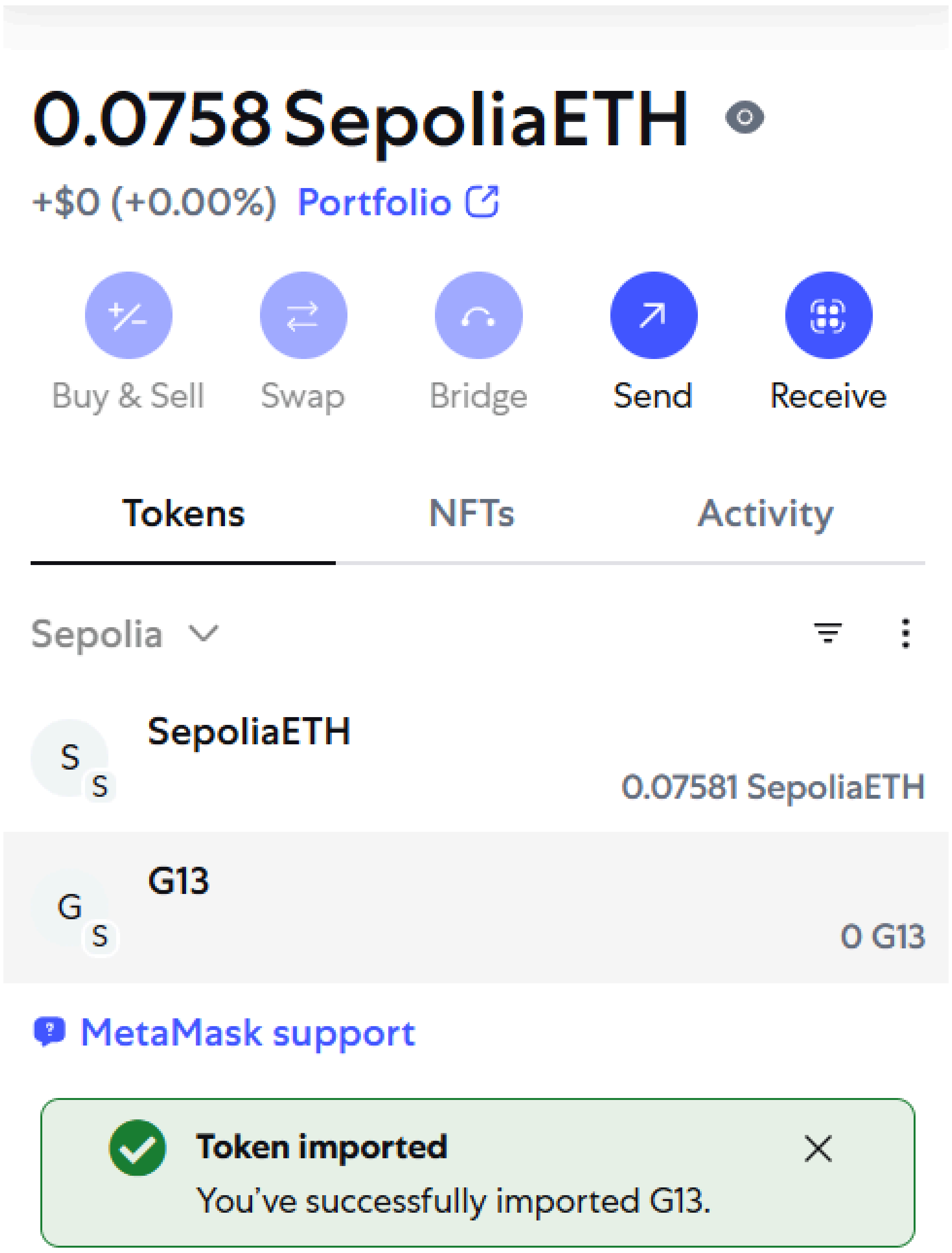
DEc8441300faFE35D10cB970780432

Token symbol

G13

Token decimal

18


0.0758 SepoliaETH

+$0 (+0.00%) Portfolio

SepoliaETH 0.07581 SepoliaETH

G13 0 G13

Token imported
You've successfully imported G13.

Currently still meet issue with verifying the contract on Sepolia etherscan, maybe try this again later when I have more time....