Lab 06-07-2023

2.1 What is Ethereum?

Ethereum is a decentralized, open-source blockchain platform that enables the execution of smart contracts and the development of decentralized applications (dapps).

It was proposed by Vitalik Buterin in 2013 and launched in 2015.

Ethereum goes beyond a simple cryptocurrency and provides a programmable blockchain that allows developers to build and deploy their own applications on top of its platform.

2.2 How Ethereum Works:

Ethereum operates on a peer-to-peer network of computers called nodes. These nodes maintain a copy of the Ethereum blockchain and participate in the consensus mechanism to validate transactions and secure the network.

Ethereum uses a virtual machine called the *Ethereum Virtual Machine (EVM)* to execute smart contracts.

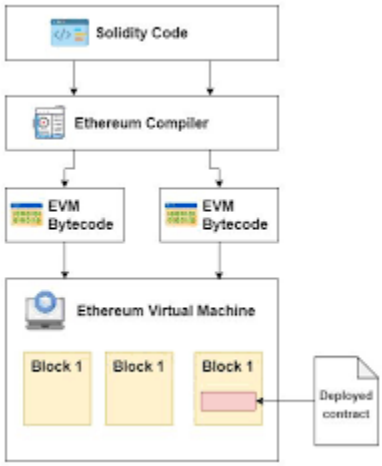
2.3 Ethereum Virtual Machine (EVM) and State Machines:

The Ethereum Virtual Machine (EVM) is a runtime environment that enables the execution of smart contracts on the Ethereum network.

It is a *Turing-complete virtual machine*, meaning it can execute any algorithm given enough time and resources.

The EVM operates using a stack-based bytecode language.

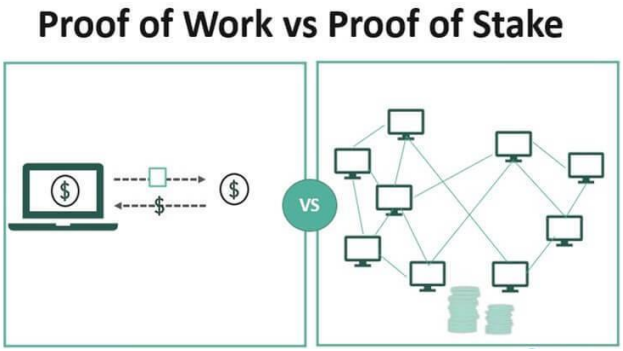
Ethereum's state machine represents the current state of the entire Ethereum network. It includes account balances, smart contract code, and storage. Each transaction and smart contract execution on the Ethereum network modifies the state, creating a new state for the network.



2.4 Proof of Stake vs. Proof of Work:

Ethereum is currently transitioning from a Proof of Work (PoW) consensus mechanism to a Proof of Stake (PoS) consensus mechanism. In PoW, miners compete to solve complex mathematical puzzles to validate transactions and create new blocks. This process requires significant computational power and energy consumption.

In contrast, PoS relies on validators who hold and "stake" their cryptocurrency to participate in block validation. Validators are chosen to create new blocks based on the amount of cryptocurrency they hold and are willing to "stake." PoS is considered more energy-efficient and scalable compared to PoW.



2.5 Turing Completeness:

Ethereum's EVM is Turing complete, which means it can perform any computation that can be expressed algorithmically. Turing completeness allows for the development and execution of complex smart contracts on the Ethereum network. It enables the creation of dapps with sophisticated functionality, including decentralized finance (DeFi) protocols, non-fungible tokens (NFTs), and more.

Turing completeness, however, also presents challenges in terms of security and efficiency. Developers must carefully consider the potential risks associated with executing complex computations on the Ethereum network.

Conclusion

Ethereum is a decentralized blockchain platform that supports the execution of smart contracts and the development of decentralized applications.

It operates using the EVM, which executes smart contracts and maintains the state of the Ethereum network. The transition from PoW to PoS consensus aims to improve efficiency, and Ethereum's Turing completeness allows for the development of complex applications.

3 Setup MetaMask Wallet

3.1 Introduction

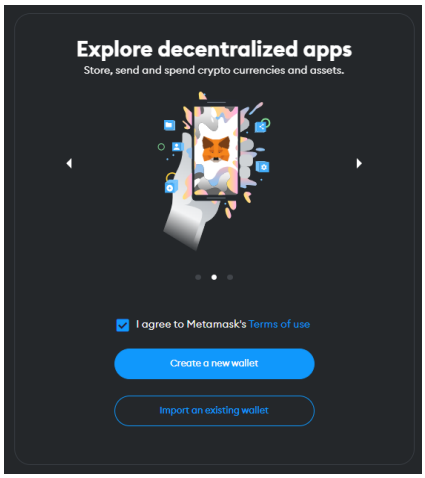
MetaMask is a popular cryptocurrency wallet that enables users to interact with the Ethereum blockchain and manage their Ethereum private keys. Here is a step-by-step guide on how to create a MetaMask wallet:

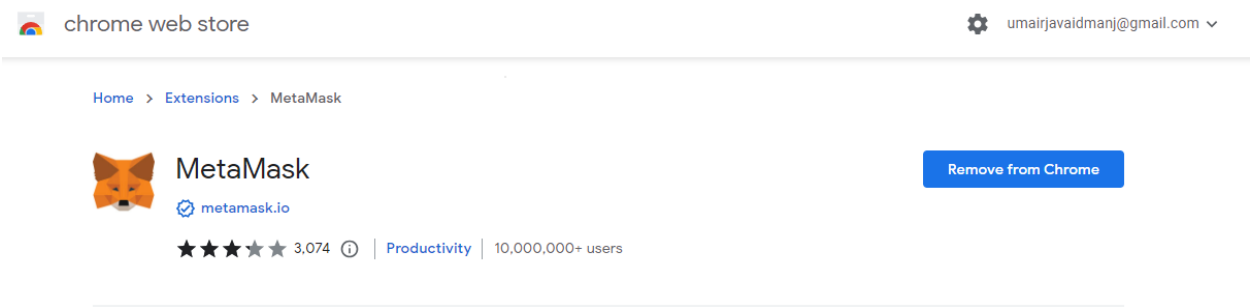
3.2 Steps Involved

Step 1: Download MetaMask Wallet

To begin, visit the official MetaMask website at https://metamask.io/.

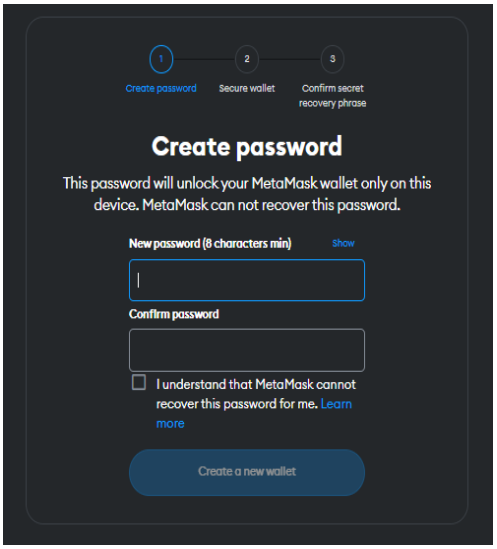
On the homepage, locate the "Download" button and click on it. Choose your preferred browser or mobile application and follow the installation instructions to install the **MetaMask extension**.





Step 2: MetaMask Wallet Installation

Once the MetaMask extension is installed, click on its icon to launch it. On the welcome screen, click on "Get Started." At this point, you have two options: you can either import an existing wallet using the seed phrase or create a new one.

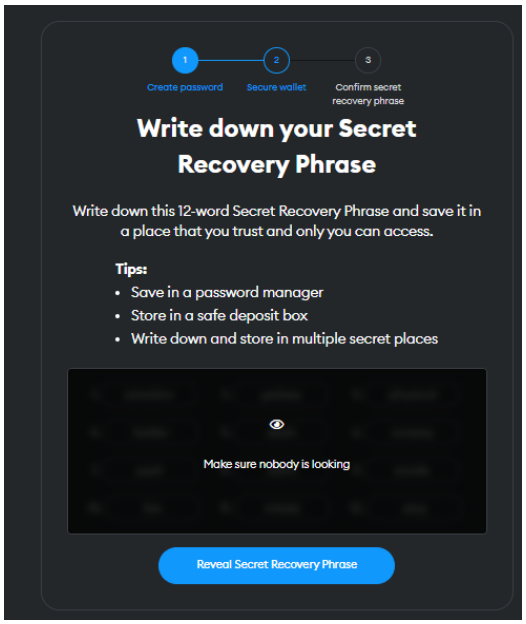


Step 3: Create a Strong Password

To ensure the security of your wallet, it is crucial to create a strong password. Choose a password that is unique, complex, and difficult to guess. Avoid using easily identifiable information and consider combining uppercase and lowercase letters, numbers, and special characters.

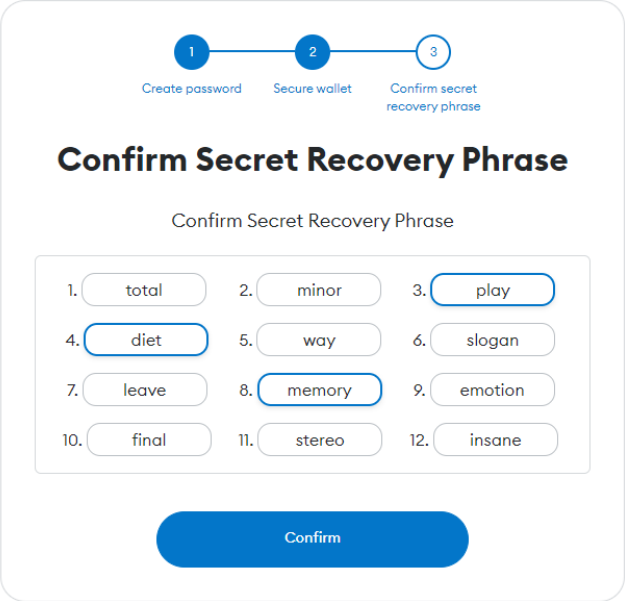
Step 4: Securely Store the Seed Phrase

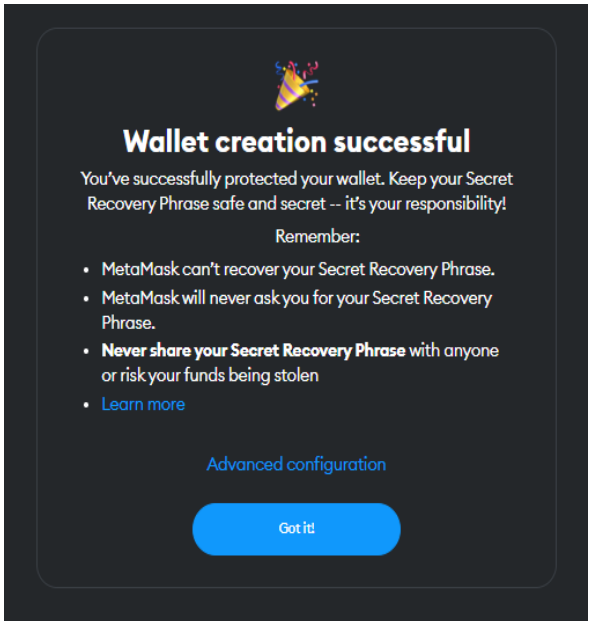
MetaMask requires you to store a seed phrase, also known as a Secret Recovery Phrase, in a safe place. This seed phrase is essential for recovering your funds in case of device failure or browser reset. Click on "**Click here to reveal secret words**" to display your seed phrase. We highly recommend writing down the 12-word phrase on a piece of paper and storing it in a secure location where only you have access. It is crucial to never share your seed phrase or private key with anyone or any website.



Step 5: Seed Phrase Confirmation

To ensure that you have correctly noted down your seed phrase, MetaMask will ask you to confirm it. On the following screen, click on each word in the order in which they were presented on the previous screen. Once you have selected all the words, click on "Confirm" to proceed.





Step 6: Congratulations!

Congratulations! You have successfully set up your MetaMask wallet. You can now access your wallet by clicking on the MetaMask icon located in the top-right corner of your preferred browser. From there, you can manage your Ethereum private keys, store Ether and other tokens, and interact with decentralized applications (dapps) on the Ethereum blockchain.

It is important to remember that **MetaMask does not store any personal information such as your email address or password**. You retain full control over your crypto-identity and assets. As a responsible user, always prioritize the security of your wallet by keeping your seed phrase and private key confidential and ensuring that your devices and browsers are adequately protected.

Fetch Some TestNet Crypto

4.1 Introduction

Testnet cryptocurrencies play a crucial role in the development and testing of decentralized applications (dapps) and smart contracts.

They provide a **simulated environment** that closely resembles the mainnet (production) blockchain but operates on a separate network specifically designed for testing purposes.

4.2 Importance

Here are some key points highlighting the importance of testnet crypto:

1) ***Experimentation:*** Testnet crypto allows developers and users to experiment with new features, functionalities, and smart contracts
2) ***Bug Testing and Security:*** By using testnet crypto, developers can **identify and fix any potential bugs or vulnerabilities** in their dapps or smart contracts before deploying them to the mainnet. This helps ensure the security and reliability of the applications.
3) ***Cost Savings:*** Testnet crypto is obtained for **free**, unlike the mainnet crypto, which has monetary value. This eliminates the need for developers and users to spend real funds while testing their applications.
4) ***Rapid Development***: Testnet crypto enables developers to iterate and deploy updates more quickly as they can test and validate their changes in a controlled environment.
5) ***Network Performance:*** Testnets allow developers to simulate various network conditions and test the scalability of their applications.

4.3 How to obtain TestNet Crypto

Now, let's discuss the steps to obtain testnet crypto:

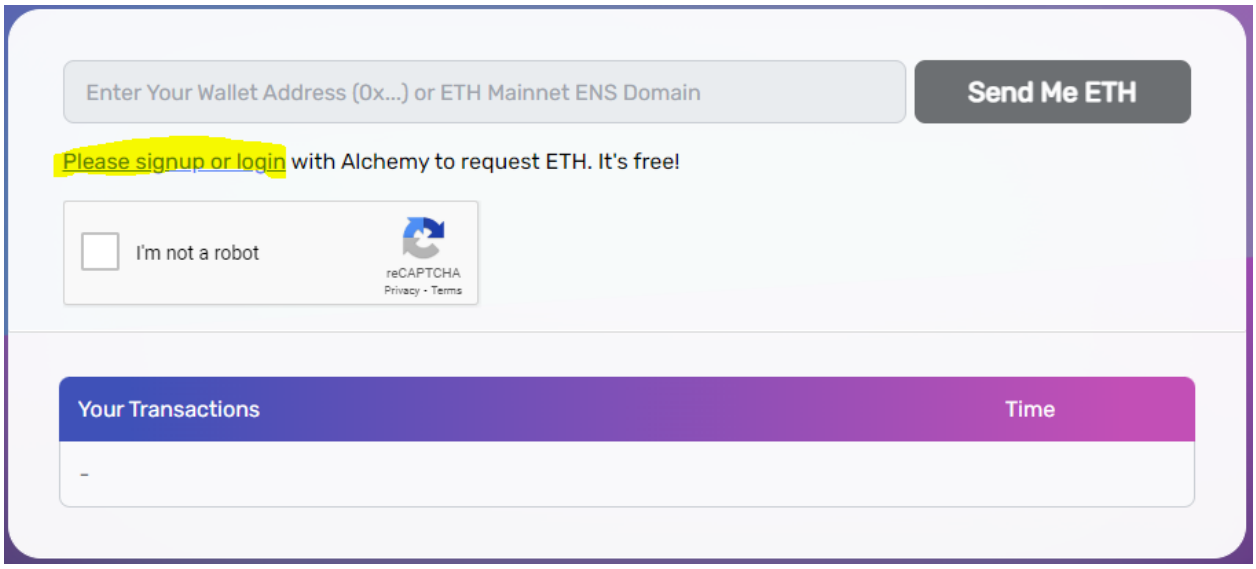
Step 1: Select a Testnet

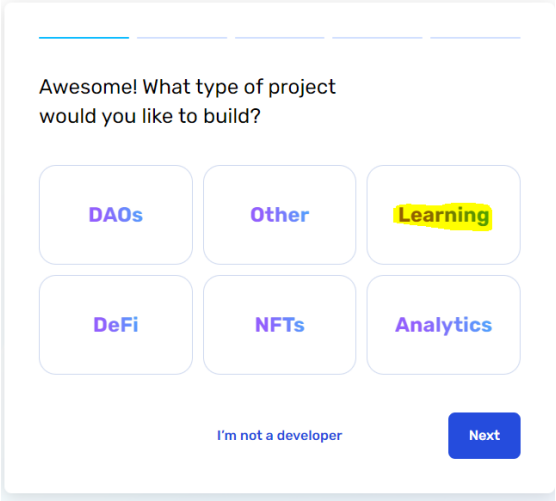
Choose the testnet that aligns with your development needs. Popular Ethereum testnets include Ropsten, Rinkeby, Kovan, and Goerli. Each testnet has its own characteristics and methods to obtain testnet crypto.

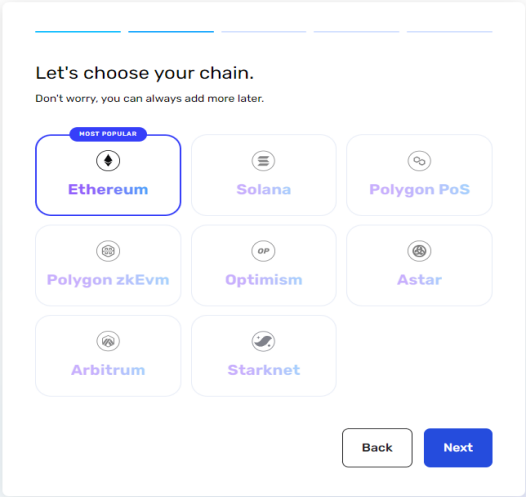
Step 2: Testnet Faucets

Testnet faucets are web platforms that distribute testnet crypto to users. Visit a reliable testnet faucet website, such as https://sepoliafaucet.com/, signup using your email and phone number.

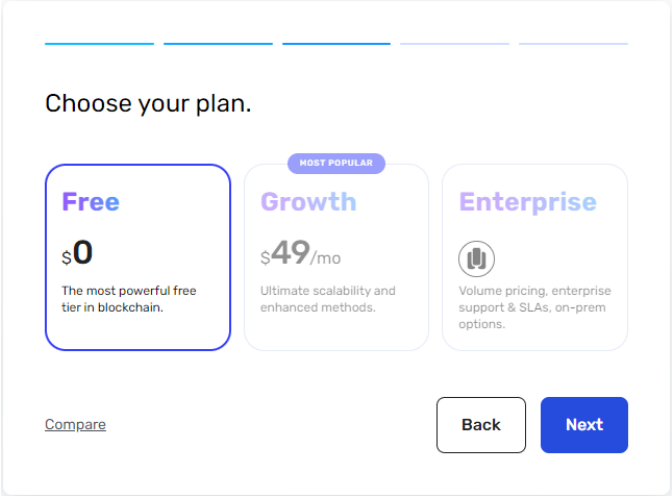
Follow their instructions to request testnet crypto for your chosen network.

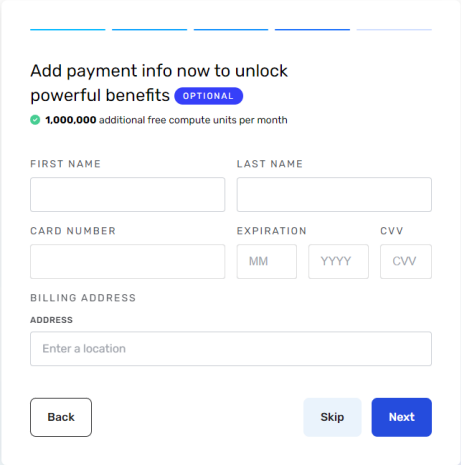






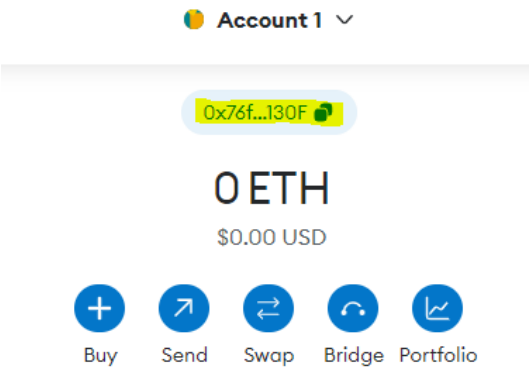
Choose the free payment plan and **skip** the options for financial details.

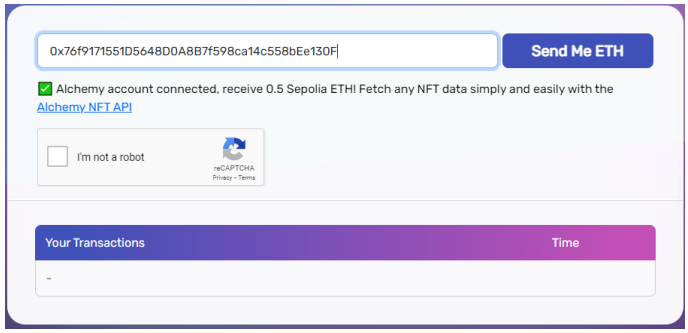




Step 3: Provide Wallet Address

Most testnet faucets require you to provide your wallet address to send the testnet crypto. Ensure you have a compatible wallet for the selected testnet (e.g., MetaMask), and copy the address from your wallet to the faucet website.



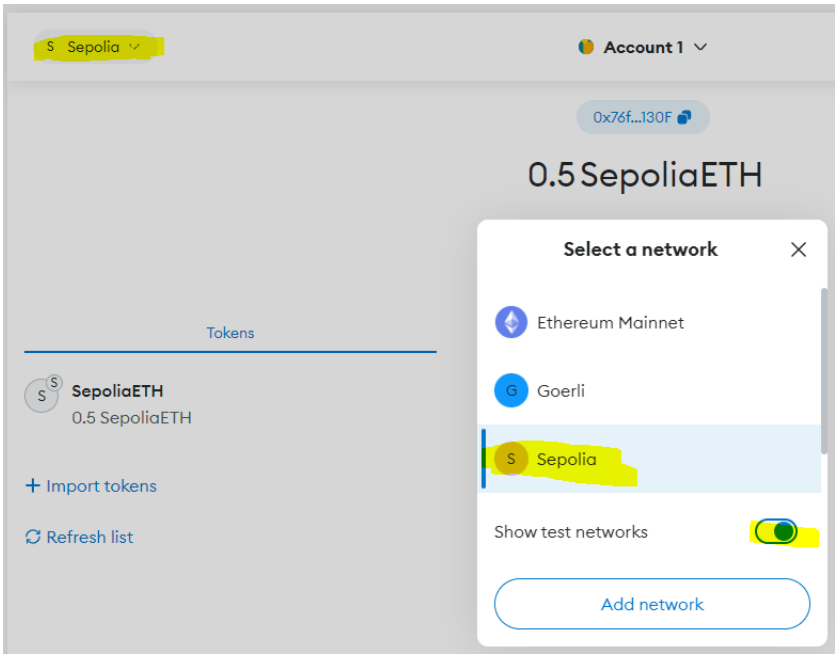


Step 4: Complete Verification (If Required)

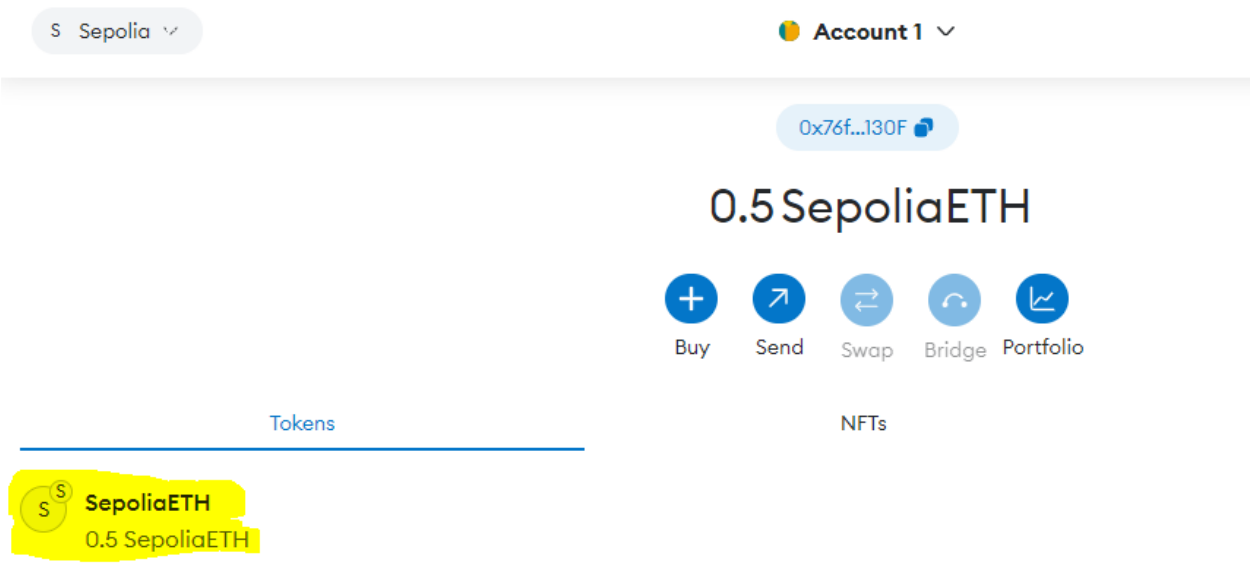
Some testnet faucets may require additional verification steps to prevent abuse. Follow their instructions, such as solving captchas or completing simple tasks, to proceed with the testnet crypto request.

Step 5: Receive Testnet Crypto

Once you have completed the necessary steps, the testnet faucet will process your request and send the testnet crypto to the provided wallet address. The time required for the testnet crypto to arrive may vary depending on the specific testnet and faucet used.



By following these steps, you can obtain testnet crypto and begin testing and developing your applications in a secure and cost-effective environment.



Remember, testnet crypto has no *monetary value* and cannot be used on the mainnet. It is crucial to differentiate between testnet and mainnet crypto to avoid any confusion or unintentional financial transactions.

5 Creating own ERC-20 Token

5.1 What are ERC-20 Tokens?

ERC20

ERC20 tokens are a widely adopted standard for creating and managing tokens on the Ethereum blockchain.

They play a significant role in the decentralized finance (DeFi) ecosystem and enable the creation of various digital assets, including cryptocurrencies, utility tokens, and security tokens. Here's a brief note on the importance of ERC20 tokens and how to create them using Remix IDE:

5.2 Importance of ERC20 Tokens:

1) ***Interoperability:*** The ERC20 standard ensures interoperability, meaning ERC20 tokens can be seamlessly integrated with other platforms, wallets, and exchanges that support the standard. This compatibility promotes liquidity and widespread adoption of tokens.
2) ***Smart Contract Integration:*** ERC20 tokens are implemented as smart contracts, allowing developers to leverage the full potential of Ethereum's programmable blockchain. Smart contracts enable self-executing, trustless agreements, ensuring transparent and secure token transfers.
3) ***Token Standards and Features:*** ERC20 defines a set of common functions and events for tokens, including transferring tokens, checking balances, and approving token allowances. These standardized features simplify token development, enhance user experience, and facilitate the integration of tokens into various applications.

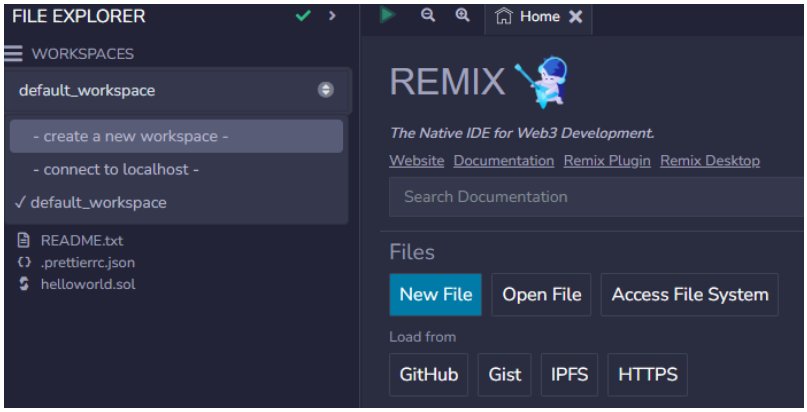
5.3 Creating an ERC20 Token using Remix IDE:

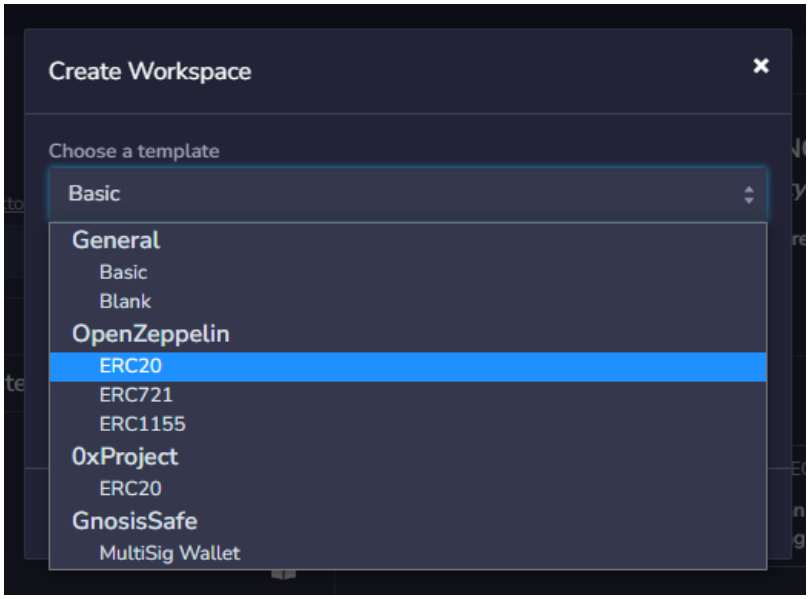
Step 1: Set Up Remix IDE

Open your web browser and navigate to Remix IDE at https://remix.ethereum.org/. Ensure that you have the latest version of the web browser and a stable internet connection.

Step 2: Create a New WorkSpace

In the File Explorer, create a new workspace as following:

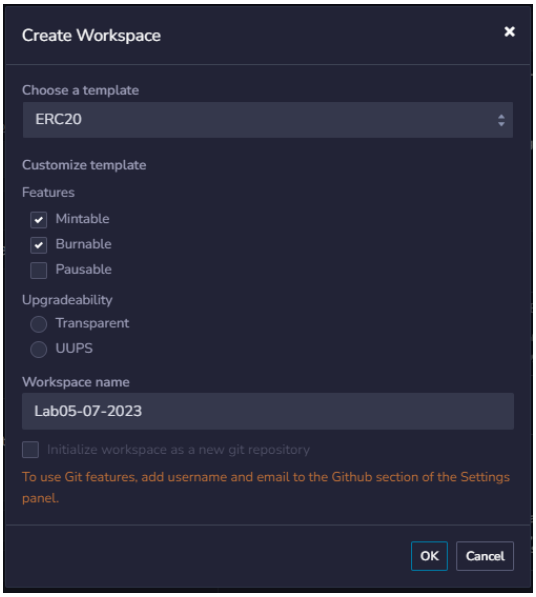




Step 3: Setting Up Properties

A window appears in which you have to

1. Name the workspace
2. Select the template (ERC-20)
3. Select features (Mintable and Burnable)



Step 4: Locate the Code

Locate your file with .sol extension **under artifacts in the contracts directory**. The code looks like this

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.9;


import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";

import "@openzeppelin/contracts/access/Ownable.sol";


contract MyToken is ERC20, ERC20Burnable, Ownable {

    constructor() ERC20("MyToken", "MTK") {}


    function mint(address to, uint256 amount) public onlyOwner {

        _mint(to, amount);

    }

}
```

Change your Token name and ID present in the constructor.

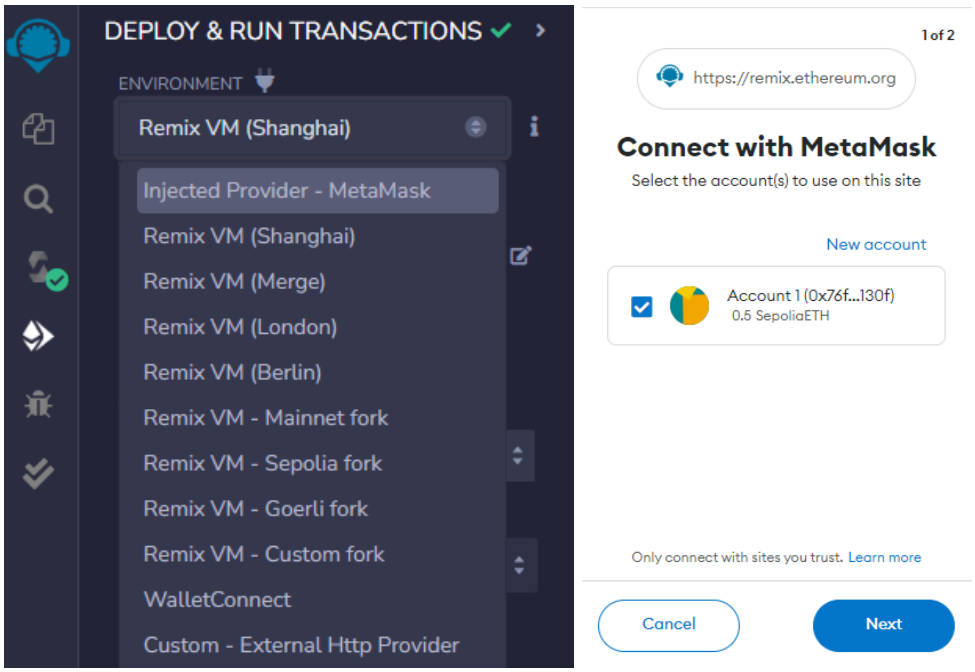
```
contract MyToken is ERC20, ERC20Burnable, Ownable {
    constructor() ERC20("Gilgitclass", "GCL") {}    infinite gas 1408000 gas
```

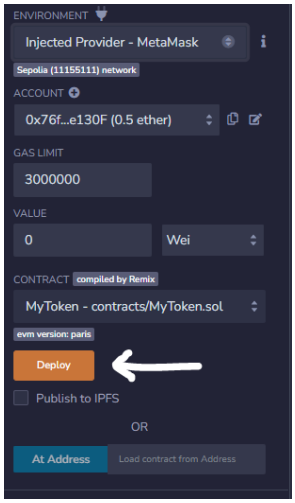
Step 5: Compile the Contract

In Remix IDE, go to the "Solidity Compiler" tab in the right sidebar. Select the version of Solidity you used in your contract from the dropdown. Click on the "Compile MyToken.sol" button to compile the contract.

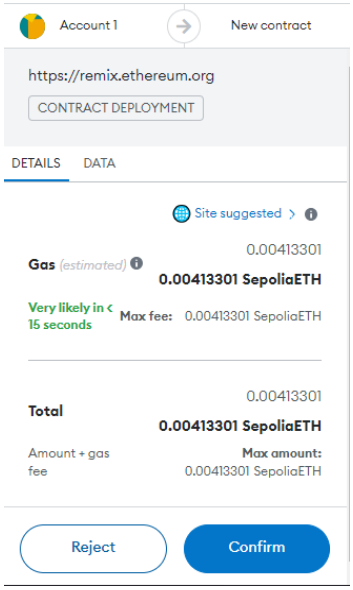
Step 6: Deploy and Test the Token

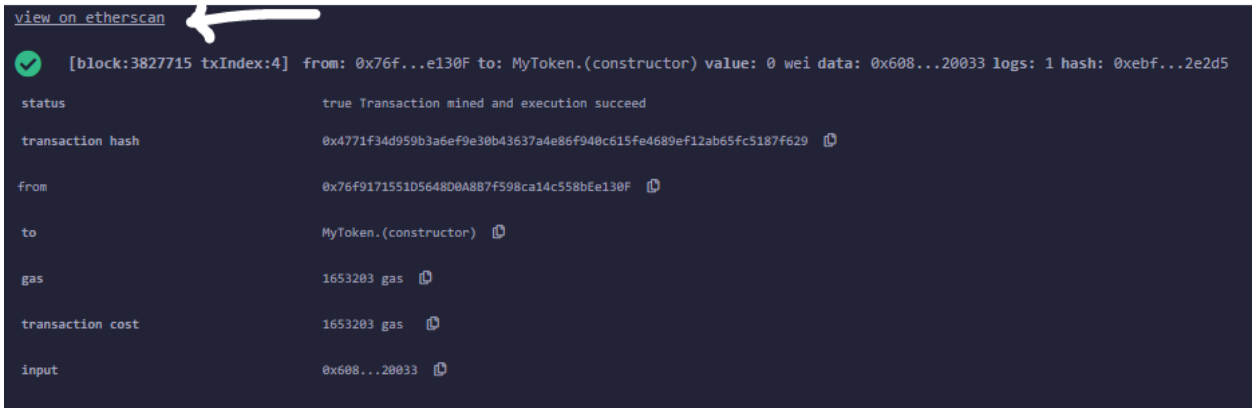
Switch to the "**Deploy & Run Transactions**" tab in the right sidebar. Under "**Environment**," select "**Injected Web3**" to connect Remix with your Ethereum wallet.

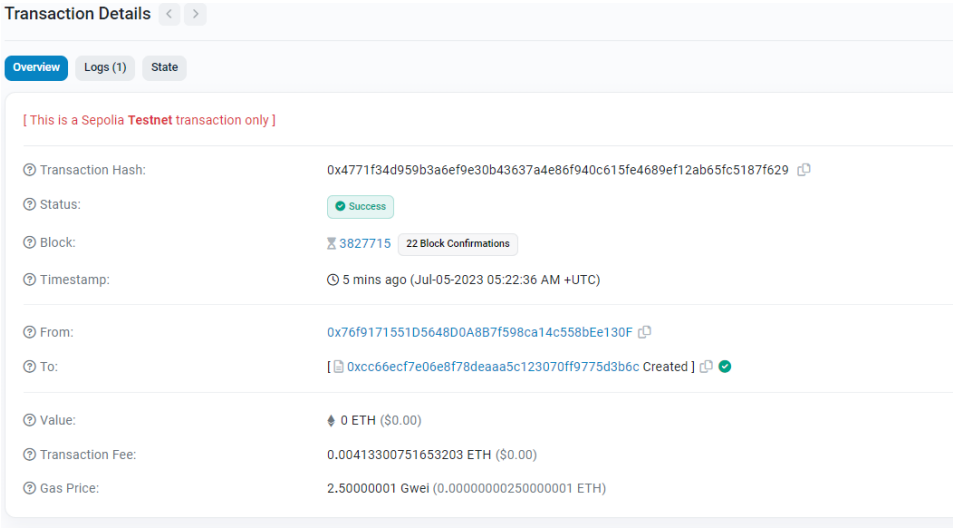




Click on the "Deploy" button to deploy your token contract to the Ethereum network. Once deployed, you can test your token by interacting with its functions through the Remix interface.







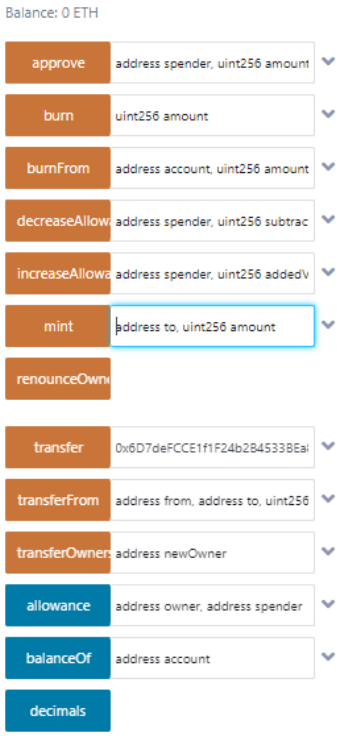
Congratulations! You have successfully created an ERC20 token using Remix IDE. Customize your token contract by modifying the parameters and functions according to your specific requirements.

Please note that thorough testing and auditing are essential before deploying your token to the mainnet.

Additionally, consider security measures and follow best practices to protect your token and its users.

5.4 Functions Involved

On successful deployment, under Deployed Contracts, you would have following functions



1) ***'approve:*** Allows an account to approve the transfer of tokens from their account to another account.
2) ***'burn*:** Permanently removes tokens from the caller's account, reducing the total supply.
3) ***'burnFrom':*** Allows a designated spender to burn tokens from a specified account.
4) ***'decreaseAllowance':*** Reduces the spender's allowance to spend tokens on behalf of the owner.
5) ***'increaseAllowance':*** Increases the spender's allowance to spend tokens on behalf of the owner.
6) ***'mint':*** Creates new tokens and adds them to the total supply, typically used by the token contract owner.
7) ***'renounceOwner':*** Allows the current owner to renounce their ownership rights, making the token contract ownerless.
8) ***'transfer':*** Moves a specified amount of tokens from the sender's account to another account.
9) ***'transferFrom'***: Moves a specified amount of tokens from one account to another, allowed by the token owner.
10) ***'transferOwner':*** Transfers ownership of the token contract to another account.
11) ***'allowance':*** Retrieves the amount of tokens that the spender is allowed to spend on behalf of the owner.
12) ***'balanceOf':*** Retrieves the token balance of a specified account.
13) ***'decimals':*** Retrieves the number of decimal places used for token values.

These functions provide various functionalities for managing and interacting with the ERC20 token, such as transferring tokens, managing allowances, adjusting token supply, and changing ownership.

6 Ethereum, Ether and Wei

Here's a brief explanation of the **difference between Ethereum, Ether, and Wei**:

6.1 Ethereum

- Ethereum is a decentralized, open-source *blockchain platform* that enables the creation and execution of smart contracts.
- Ethereum allows developers to build *decentralized applications (dapps)* and offers a programmable blockchain that supports a wide range of applications beyond cryptocurrencies.

6.2 Ether (ETH)

- Ether is the *native cryptocurrency* of the Ethereum blockchain.
- It serves as the fuel for running smart contracts and powering decentralized applications on the Ethereum network.
- Ether is used to pay for transaction fees, computational services, and participate in activities such as staking, voting, and governance within the Ethereum ecosystem.

6.3 Wei

- Wei is the *smallest and indivisible unit* of Ether.
- It is the base denomination used to represent the value of Ether in the Ethereum network.
- *One Ether (ETH) is equal to 10^18 Wei.* **Wei is named after Wei Dai, a computer scientist** known for his contributions to cryptography and blockchain technology.

To summarize, Ethereum is the blockchain platform that facilitates the execution of smart contracts, Ether (ETH) is the cryptocurrency used within the Ethereum ecosystem for various purposes, and Wei is the smallest unit of Ether used to denote its value and facilitate precise calculations within the network.

It's important to note that while **Ether is commonly used as a store of value and medium of exchange**, **Ethereum's primary function is to provide a decentralized platform for executing smart contracts** and building decentralized applications.

7 Gas and Gas Price

Here's a brief explanation of the difference between gas and gas price in the context of the Ethereum network:

7.1 Gas

In the Ethereum network, gas refers to a unit of measurement that quantifies the computational effort required to execute operations within a smart contract or transaction.

Each operation in Ethereum consumes a specific amount of gas, and the cumulative gas cost determines the total fees associated with the execution.

Gas is used to manage resources, prevent spam, and maintain network stability. It ensures that each operation on the Ethereum network has a specific cost associated with it, which helps prioritize and incentivize efficient use of computational resources.

7.2 Gas Price

Gas price, on the other hand, represents the amount of Ether (ETH) users are willing to pay for each unit of gas when executing a transaction or interacting with a smart contract. Gas price is denoted in terms of Gwei, where 1 Gwei is equal to 0.000000001 ETH.

The gas price determines the financial cost of executing a transaction or smart contract operation. A higher gas price incentivizes miners to prioritize a transaction or operation, as it increases the potential rewards for including it in a block.

Conversely, a lower gas price may result in slower execution or even rejection if the network is congested.

Users can set the gas price for their transactions based on their desired trade-off between transaction speed and cost. Higher gas prices result in faster processing times, while lower gas prices offer cost savings but may take longer to be included in a block.

In ***summary***, gas is a unit of measurement that quantifies computational effort within the Ethereum network, while gas price represents the amount of Ether a user is willing to pay for each unit of gas during transaction execution. Understanding gas and gas prices is essential for managing transaction costs and ensuring timely execution on the Ethereum network.

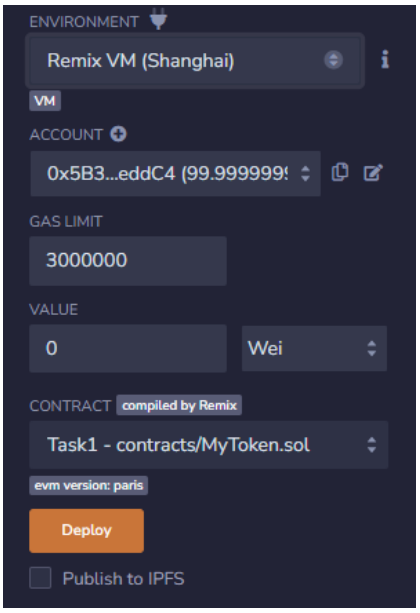
8 Lab Exercises

8.1 Ether Units Conversion

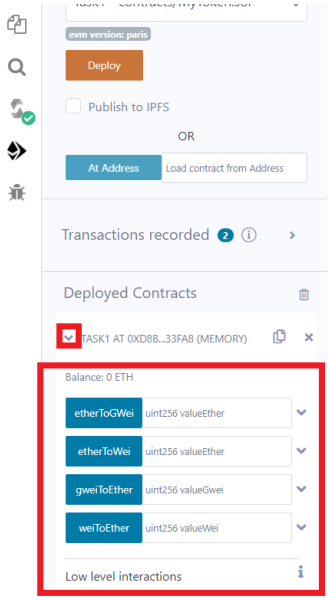
Deploy the following smart contract.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

contract Task1 {
    function etherToWei(uint valueEther) public pure returns (uint) {
        return valueEther * (10**18);
    }

    function weiToEther(uint valueWei) public pure returns (uint) {
        return valueWei / (10**18);
    }

    function etherToGWei(uint valueEther) public pure returns (uint) {
        return valueEther * (10**9);
    }

    function gweiToEther(uint valueGwei) public pure returns (uint) {
        return valueGwei / (10**9);
    }
}
```

Step 01: After Compiling, set the Environment as Remix VM (Shanghai) in Deploy and run tab, and then click on Deploy.



Step 02: Under Deployed Contracts, click on small drag down button as highlighted and locate the highlighted conversion functions.



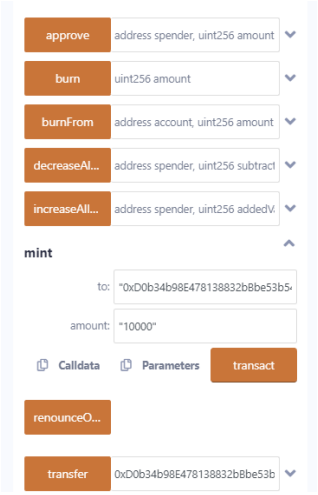
Step 03: Using functions,

1. Convert 2 Ether to Gwei.
2. Convert 2 Ether to Wei.
3. Convert 40000000000 Gwei to Ether.
4. Convert 40000000000000000000 Wei to Ether.

9 Lab Tasks

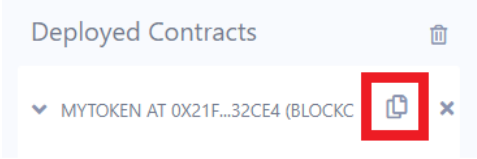
9.1 ERC-20 Token:

1. Create ERC-20 token with your Name and Enrollment ID
2. Use the mint() function and Mint 10,000 tokens on your metamask wallet.

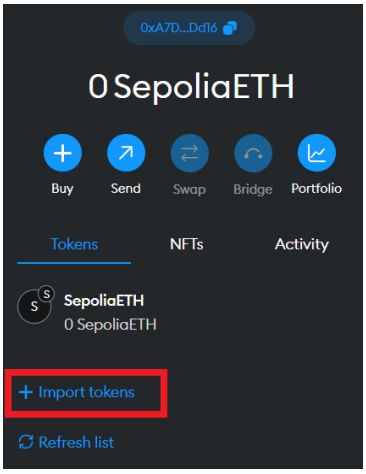


3. Transfer 5000 tokens on your metamask by first configuring the metamask as following

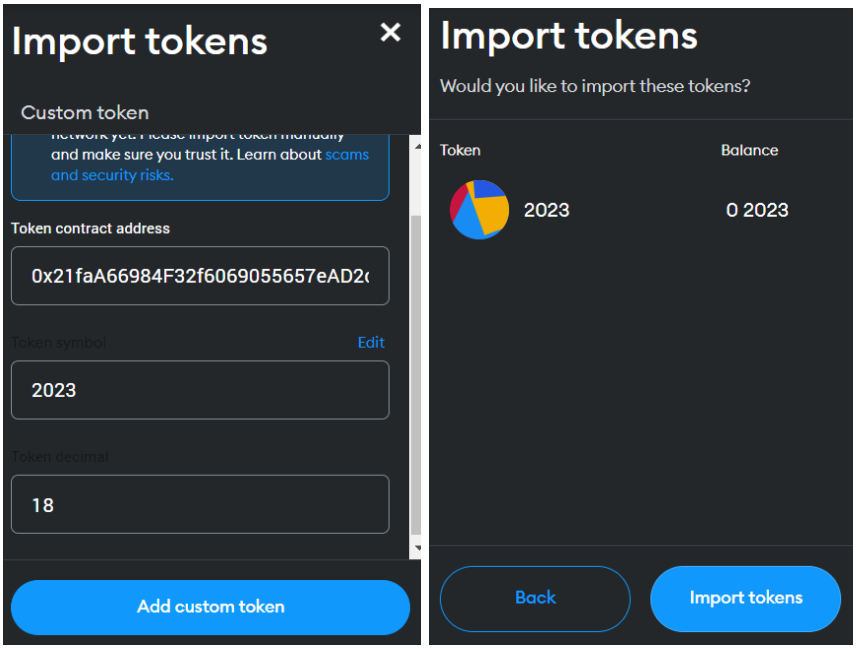
Step 01 Copy the Contract Address as following



Step 02 Go to MetaMask and click on '+ **Import Tokens**'.

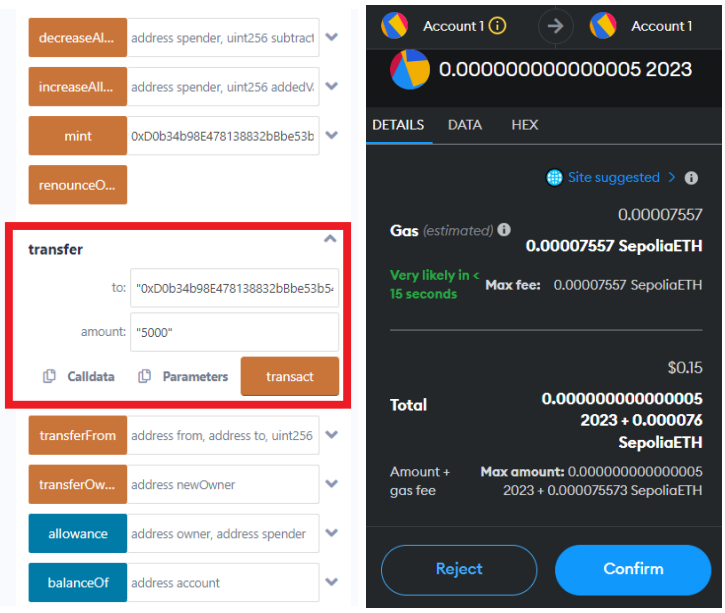


Step 03 Paste your Contract Address. Your Token Symbol and decimals would automatically be detected, and then click on **"Add Custom Token".**

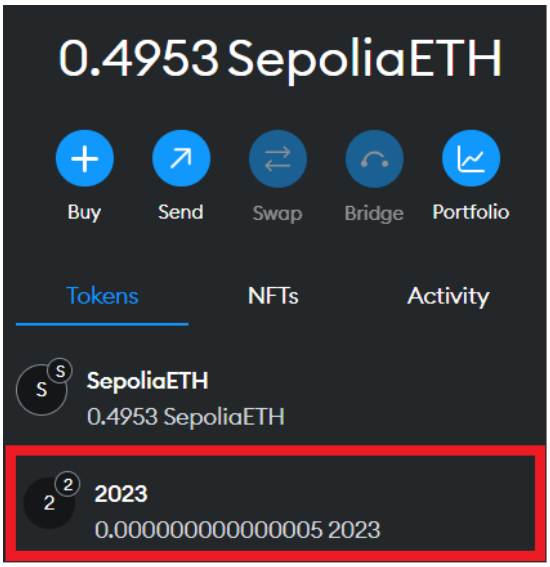


Finally click on **"Import Tokens"** and your token would be ready for transfer.

Step 04 Use the transfer() function and transfer 10,000 tokens on your metamask wallet.



Wait for transaction and then check your metamask wallet to check whether you have successfully transferred the tokens.



9.2 **Gas Activity:** Measure and record gas consumed.

Here's a simple contract named **GasConsumer** that increments a state variable based on the number of iterations passed as an argument.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.5.0 <0.9.0;

contract GasConsumer {
    uint256 public count;

    function consumeGas(uint256 iterations) public {
        for(uint256 i = 0; i < iterations; i++) {
            count++;
        }
    }
}
```

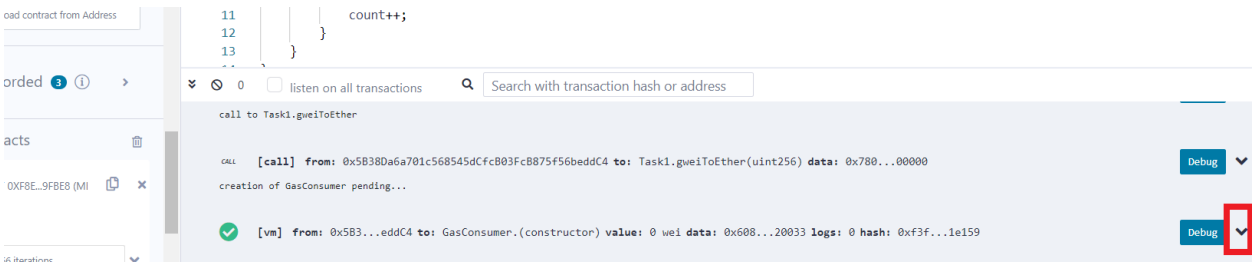
……

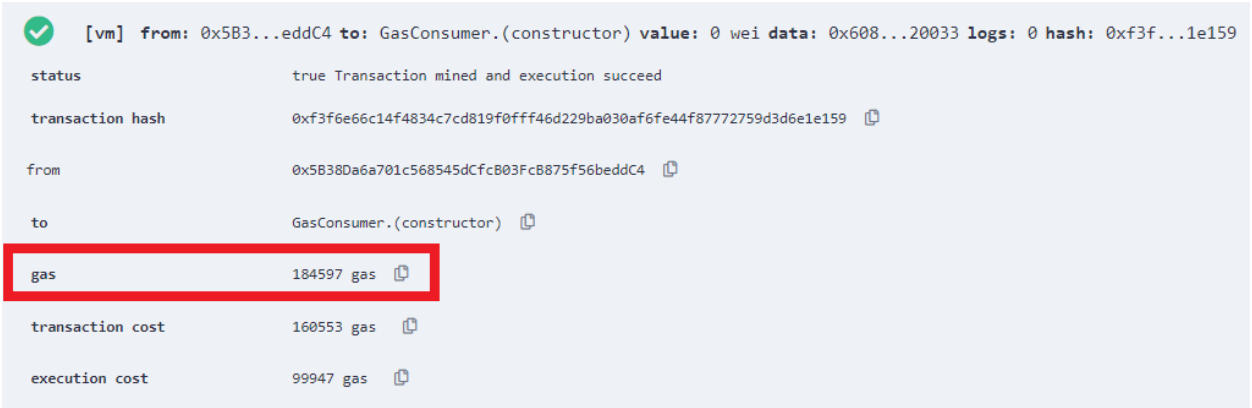
Measure the gas for

1) Deploying the contract.
2) Making 10 iterations.
3) Making 100 iterations.

Step 1 Compile and deploy the above code.

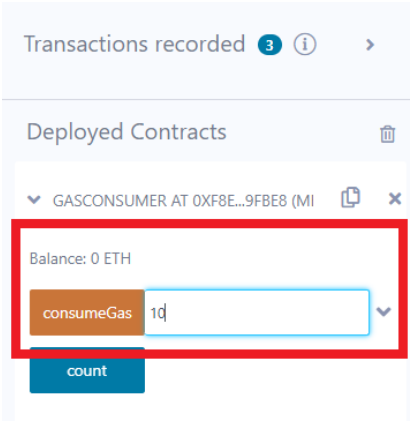
Step 2 Click on the drag down button as highlighted and observe the ***gas.***





Step 3 Under Deployed Contracts, locate the ***consumeGas*** function and then first enter 10, click on ***consumeGas*** and record the gas by re-performing Step 02

Do the same for 100 iterations.



10 Fun Activity

Try to have some free NFTs collection in your MetaMask Wallet via

https://opensea.io/collection/freenfts-collection

For your knowledge, NFT named Merged got sold for US$91.8M.

➔ For complete documentation, visit

1. https://remix-ide.readthedocs.io/en/latest/
2. https://github.com/ethereumbook/ethereumbook