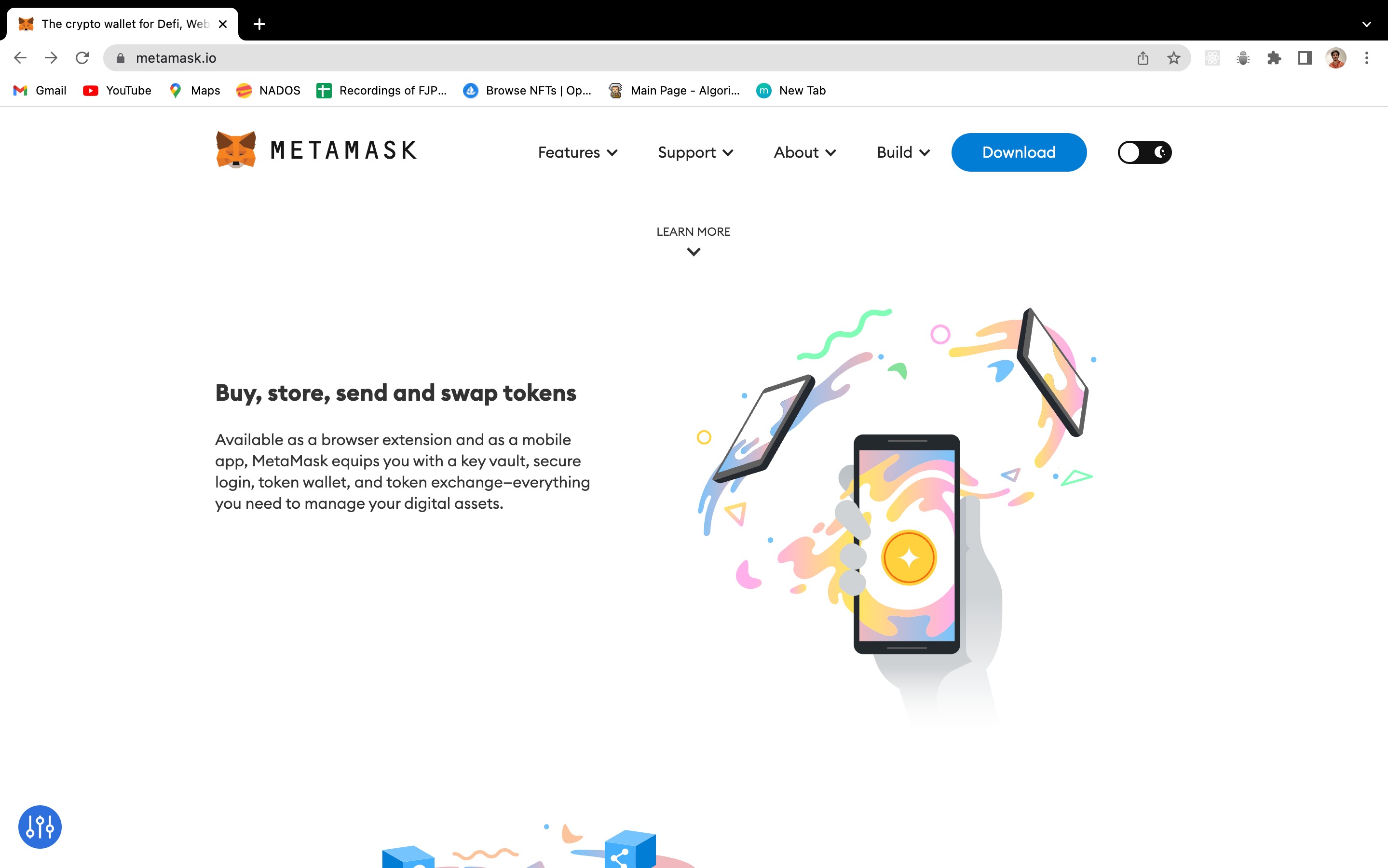
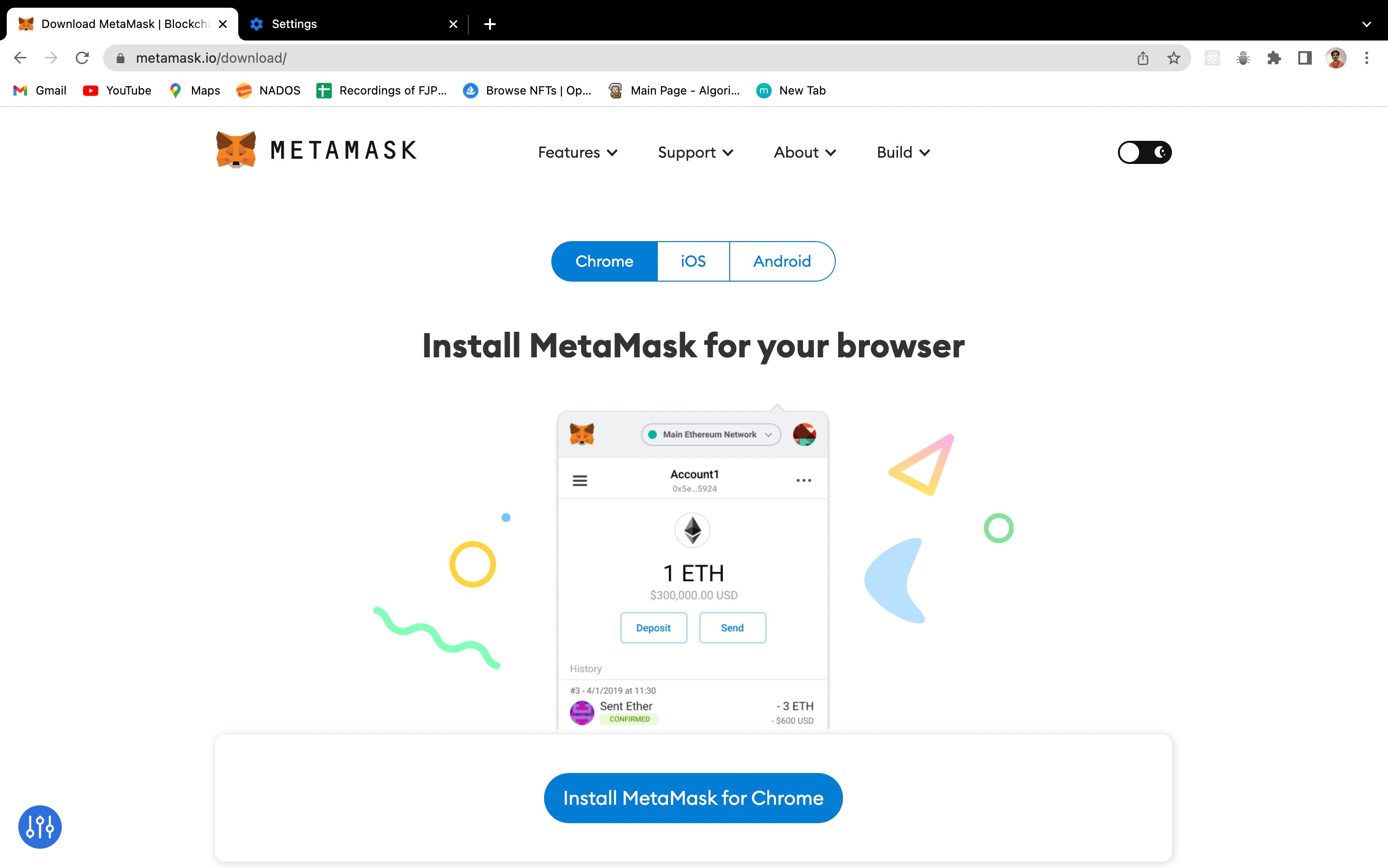
# Installation of Metamask

Step 1 - Go to the site https://metamask.io/ and click on the “Download” button in the menu bar

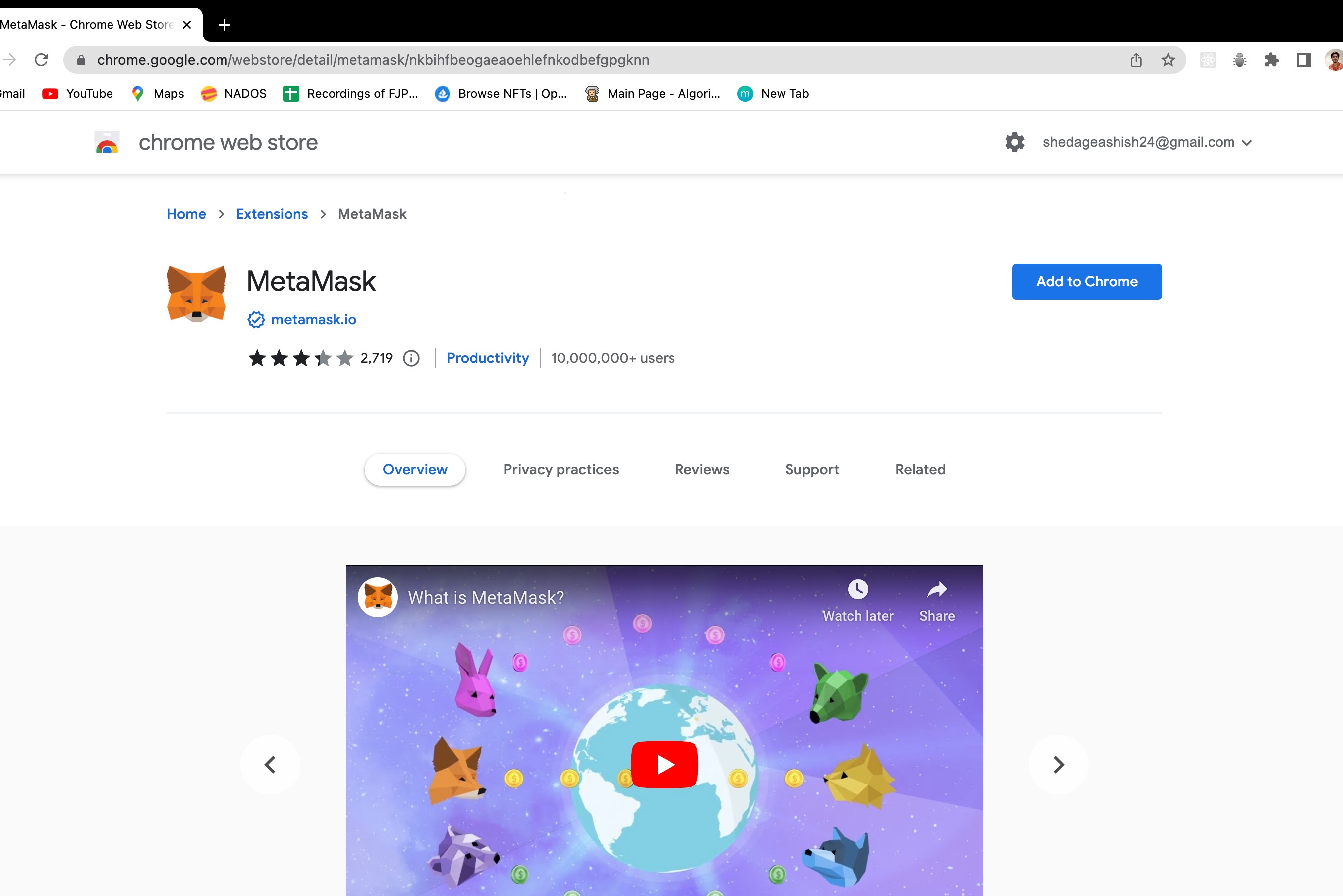


Step 2 - Click on the “Install metamask for chrome” you will be directed to the chrome web store

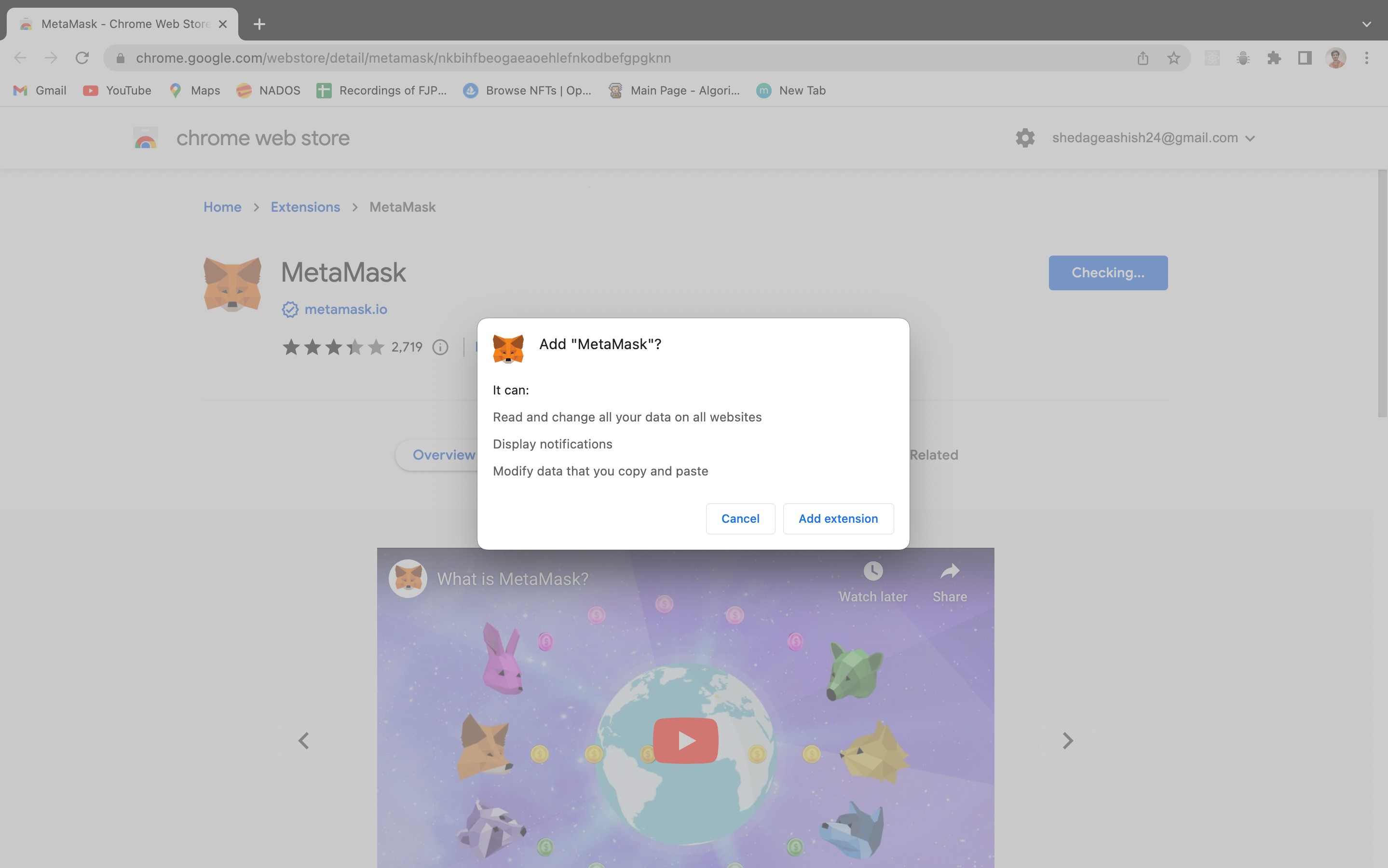


After clicked on “Download”

Step 3 - Click “Add to Chrome”

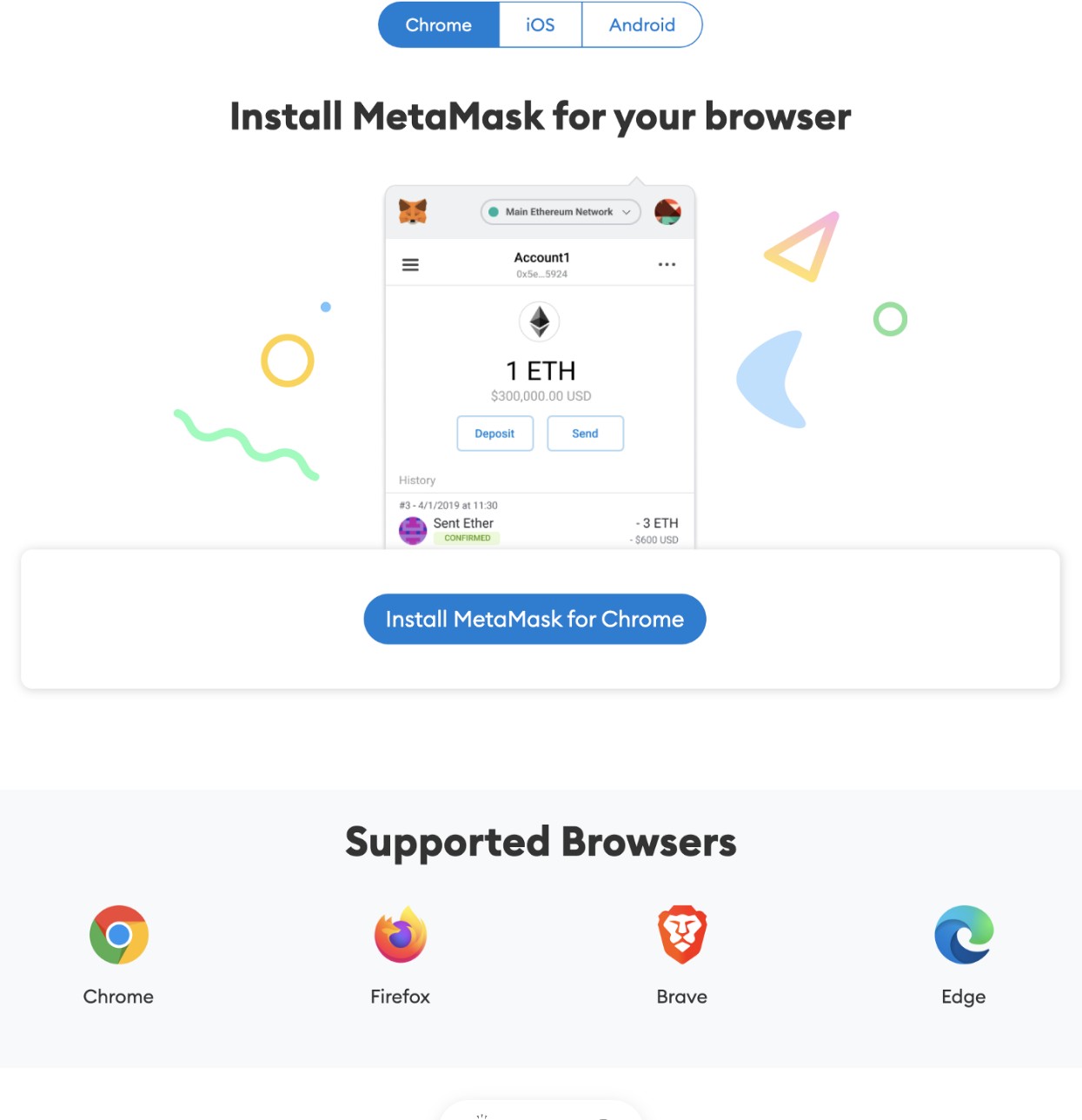


Step 4 - On the pop-up, click “Add Extension”



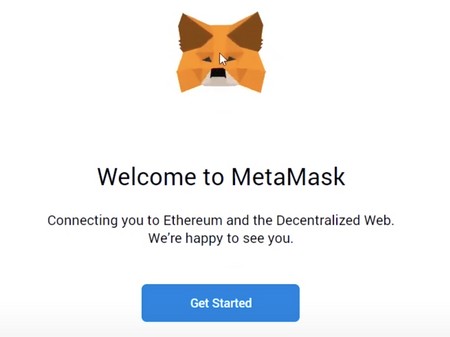
## **Steps to create a wallet using Metamask**

Step 1 - Go to **https://metamask.io/** and click on “Download”. Choose your preferred browser or mobile application and install the MetaMask extension.



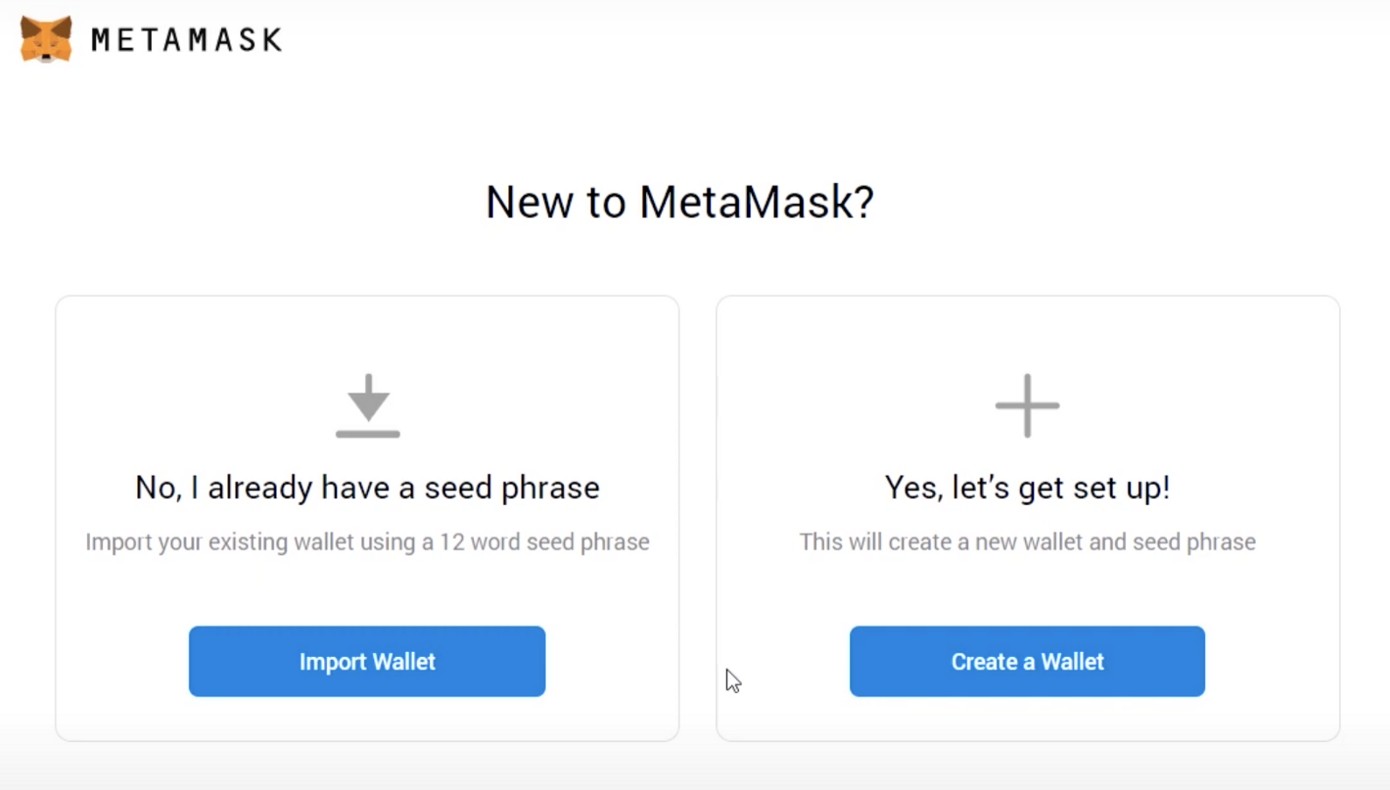
**Step 2 - MetaMask wallet installation**

Click on the MetaMask extension and click on “Get Started”.



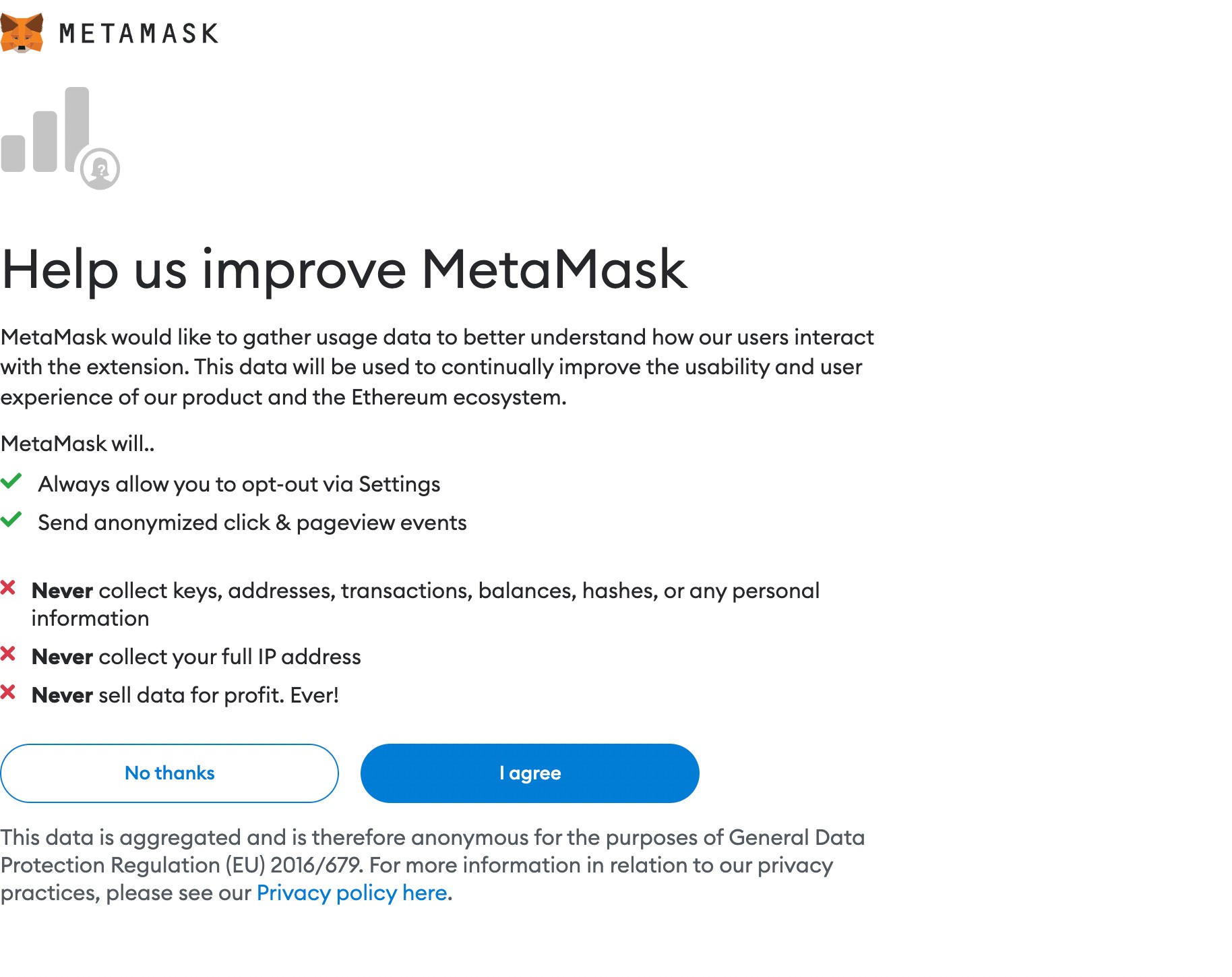
Caption

You can either import an existing wallet using the seed phrase or create a new one.



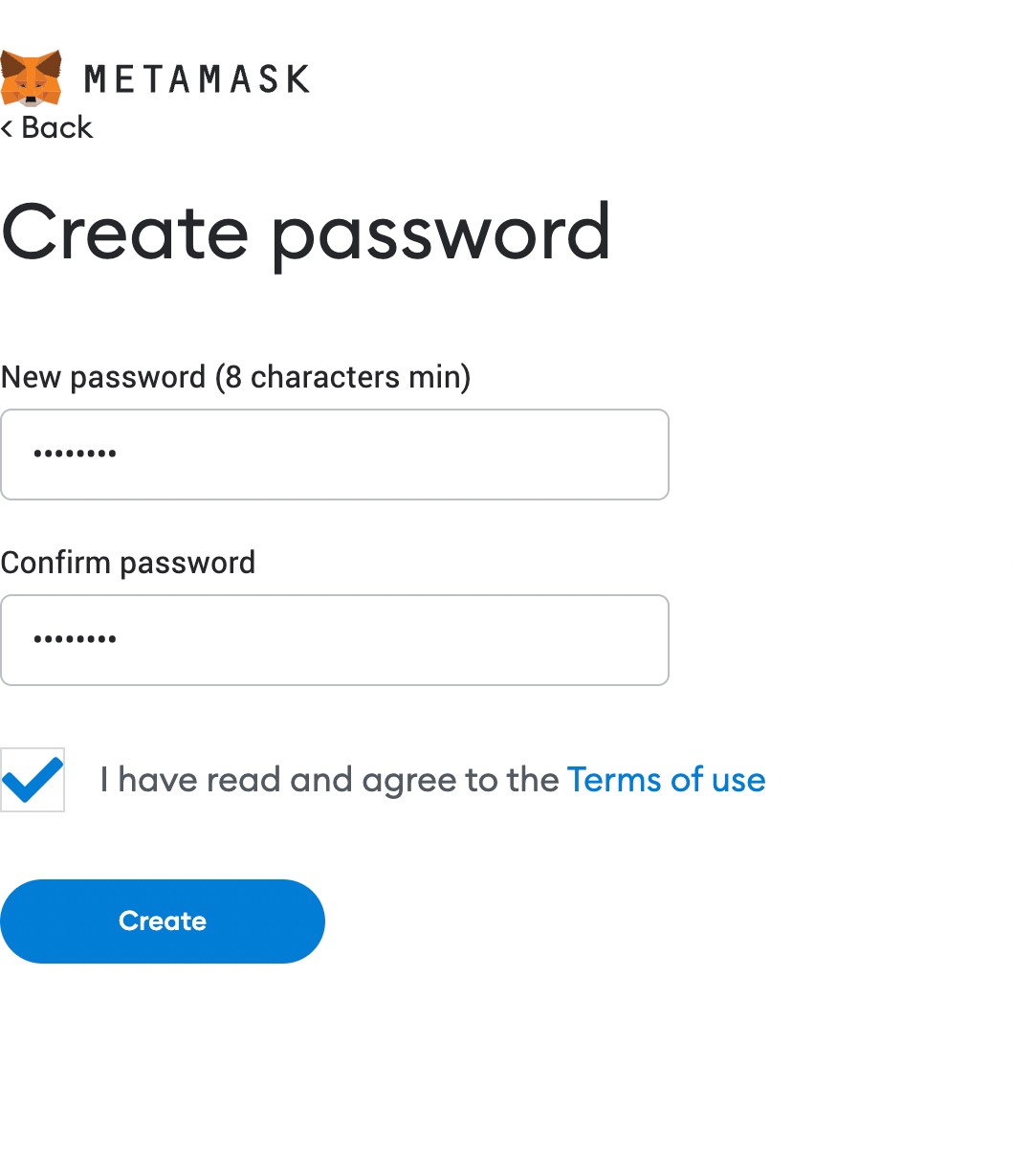
**Step 3 - How to create a new MetaMask wallet**

Click on “Create a Wallet” and on the next window click on “I agree” if you would like to help improve MetaMask or click on “No Thanks” to proceed.



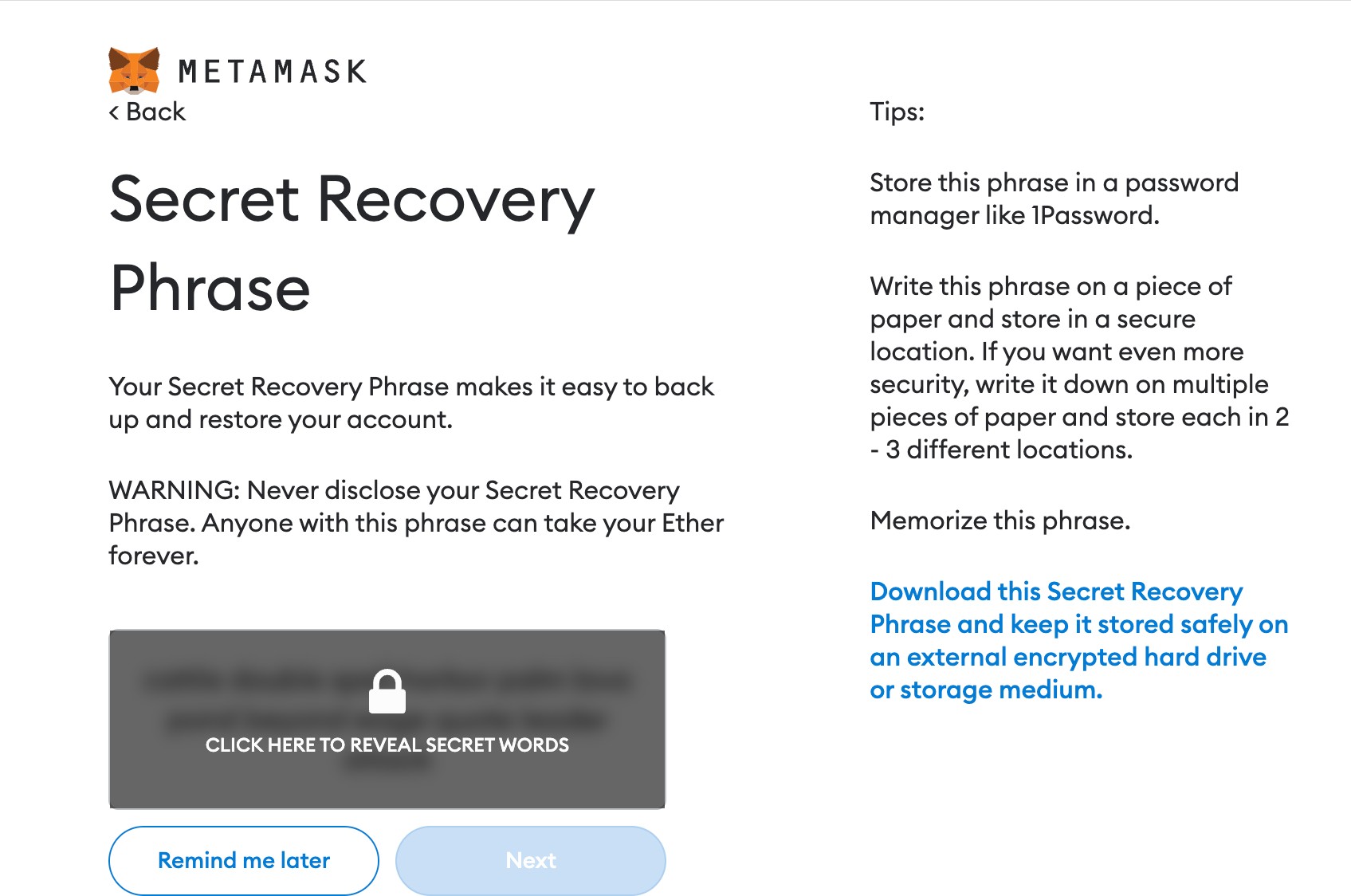
Caption

**Step 4 - Create a strong password for your wallet.**



## Step 5: Securely store the seed phrase for your wallet.

Click on “Click here to reveal secret words” to show the seed phrase.

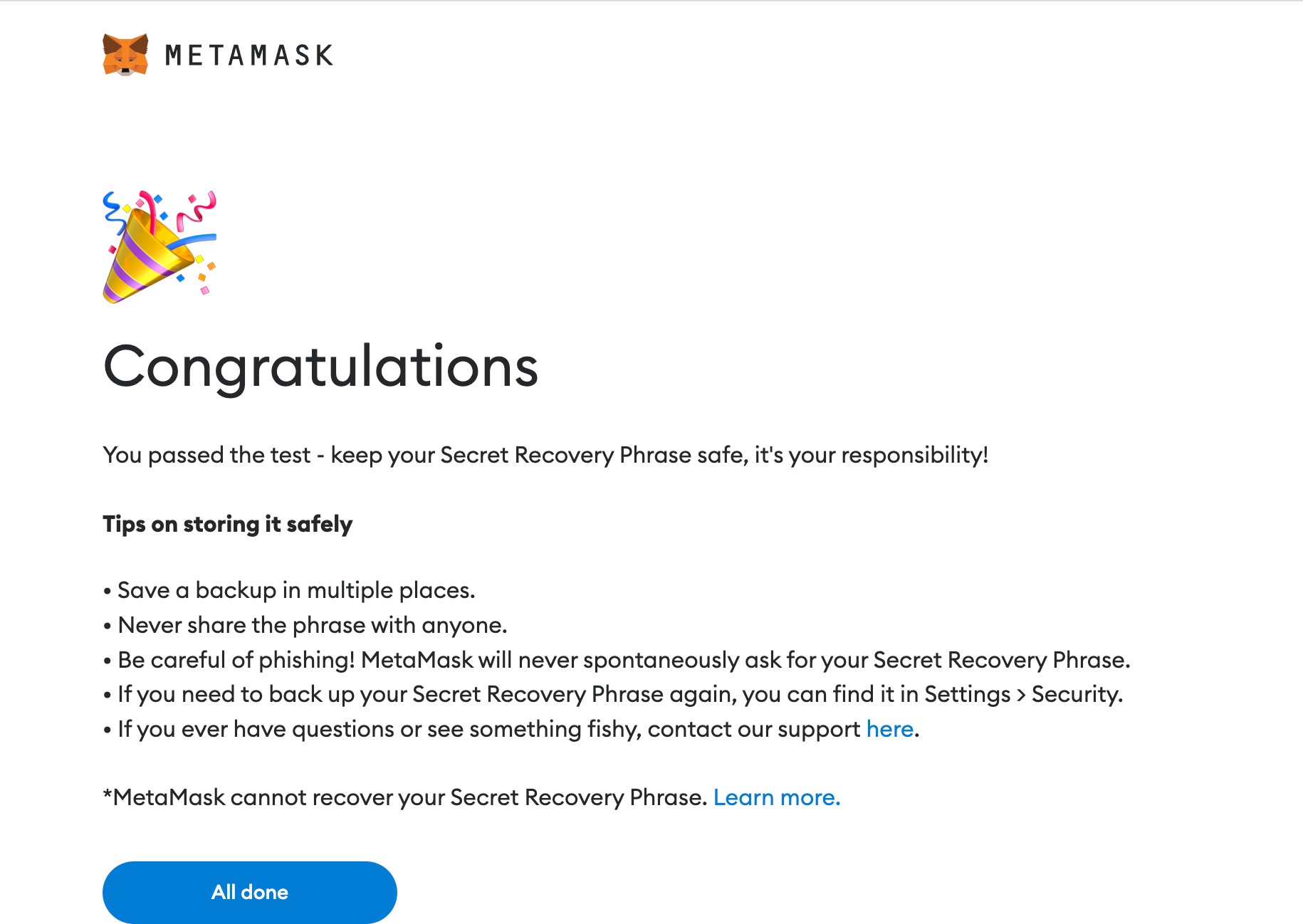
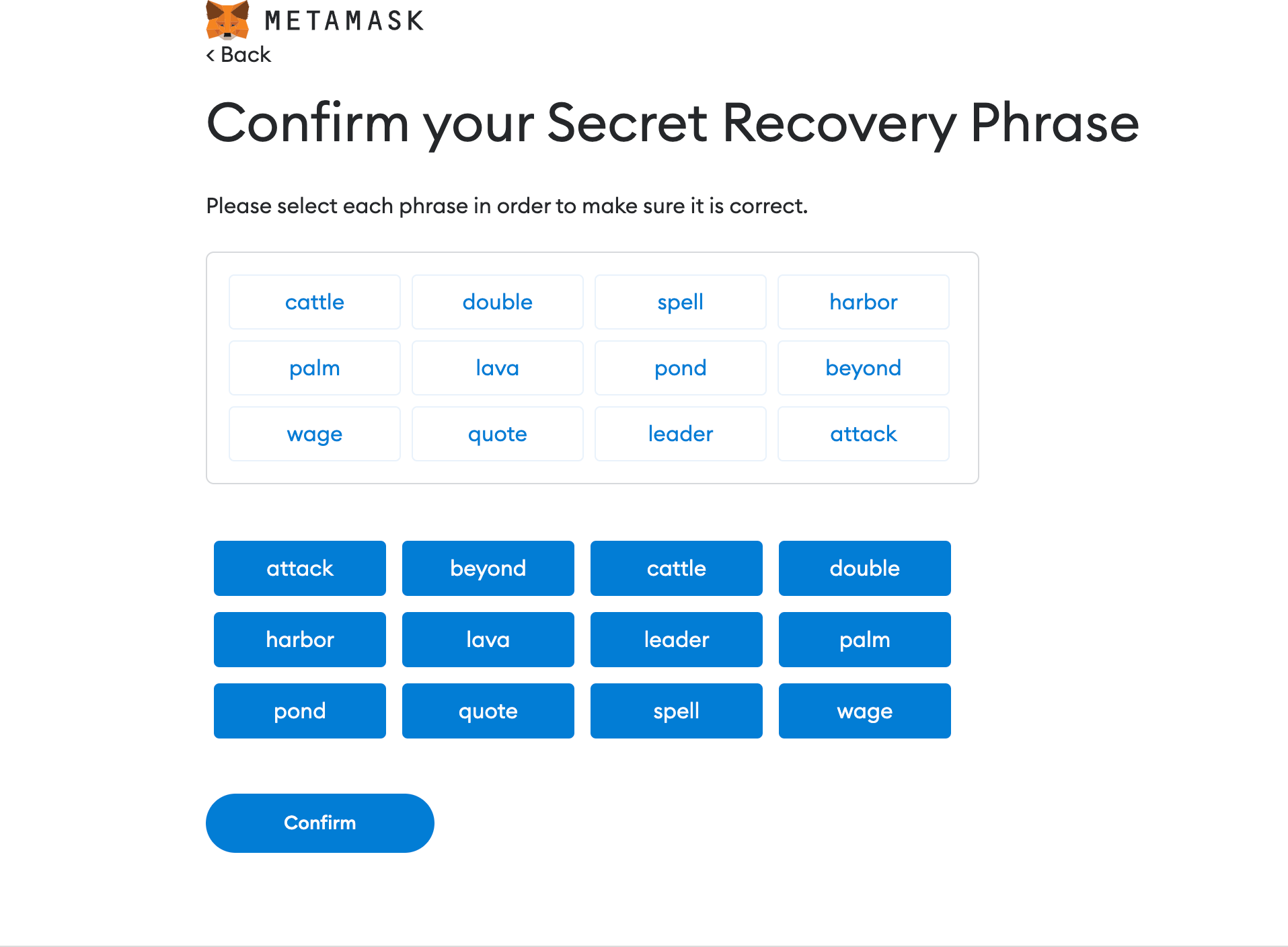


Caption

## Step 6: Seed phrase confirmation

Caption

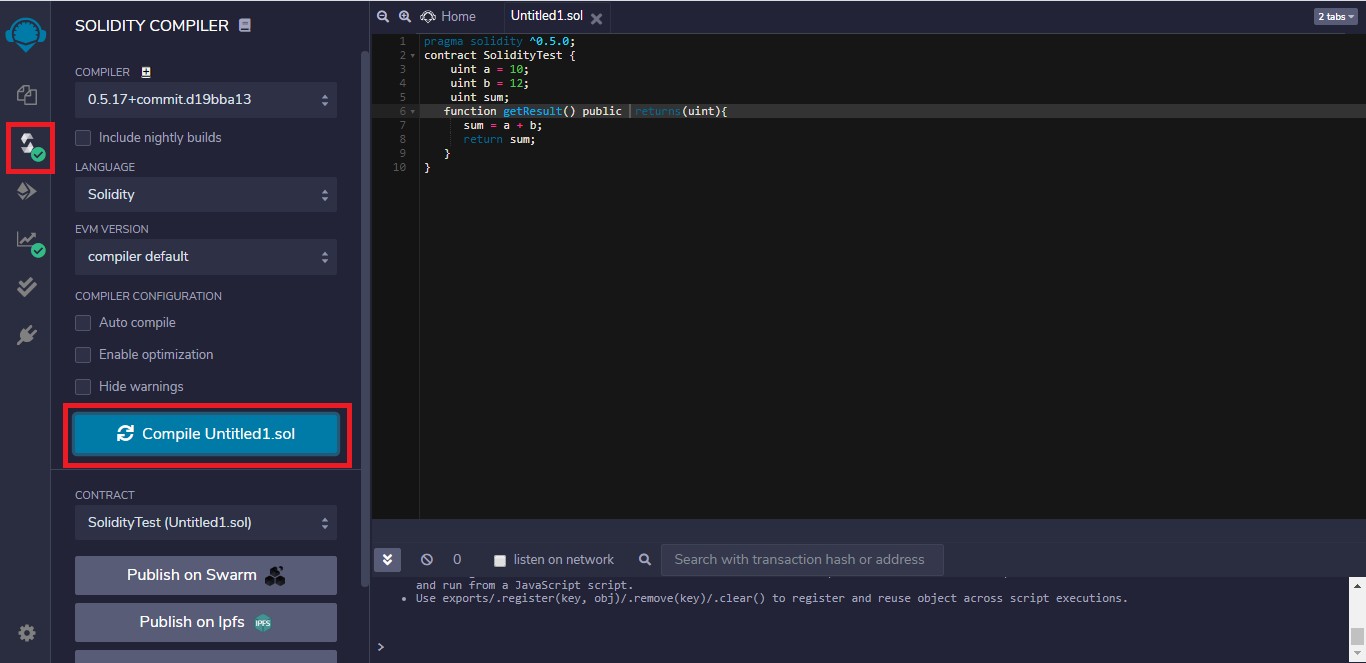
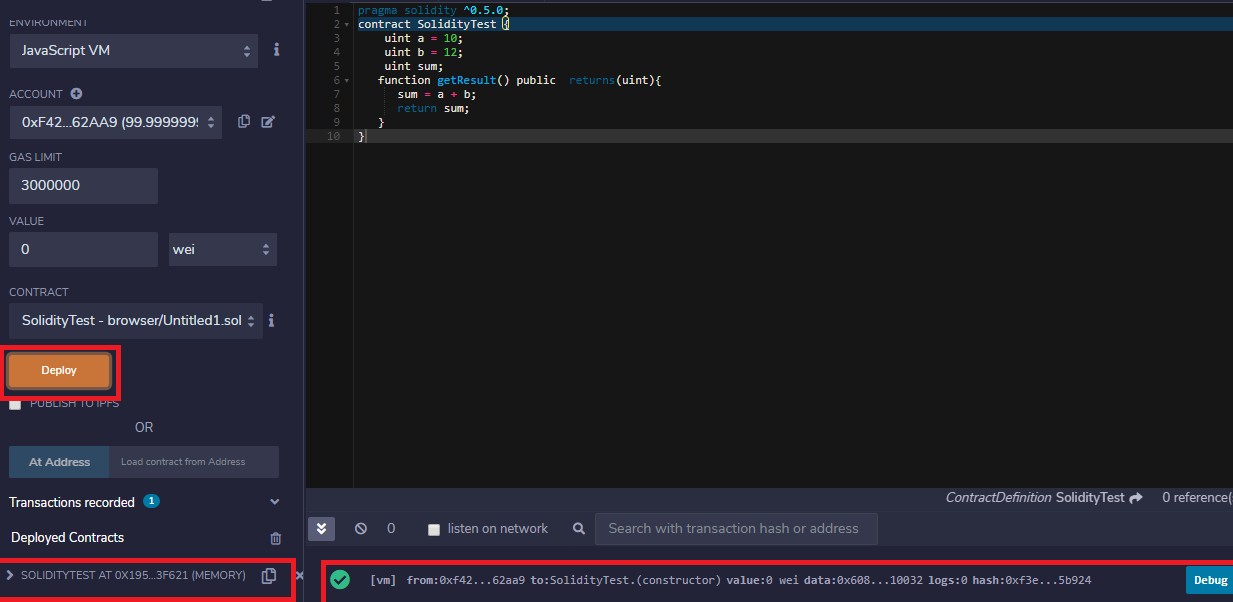
Confirm your secret backup phrase by clicking on each word in the order in which the words were presented on the previous screen. Click on “Confirm” to proceed.



## Steps for the compilation, execution, and debugging of the smart contract in Remix IDE.

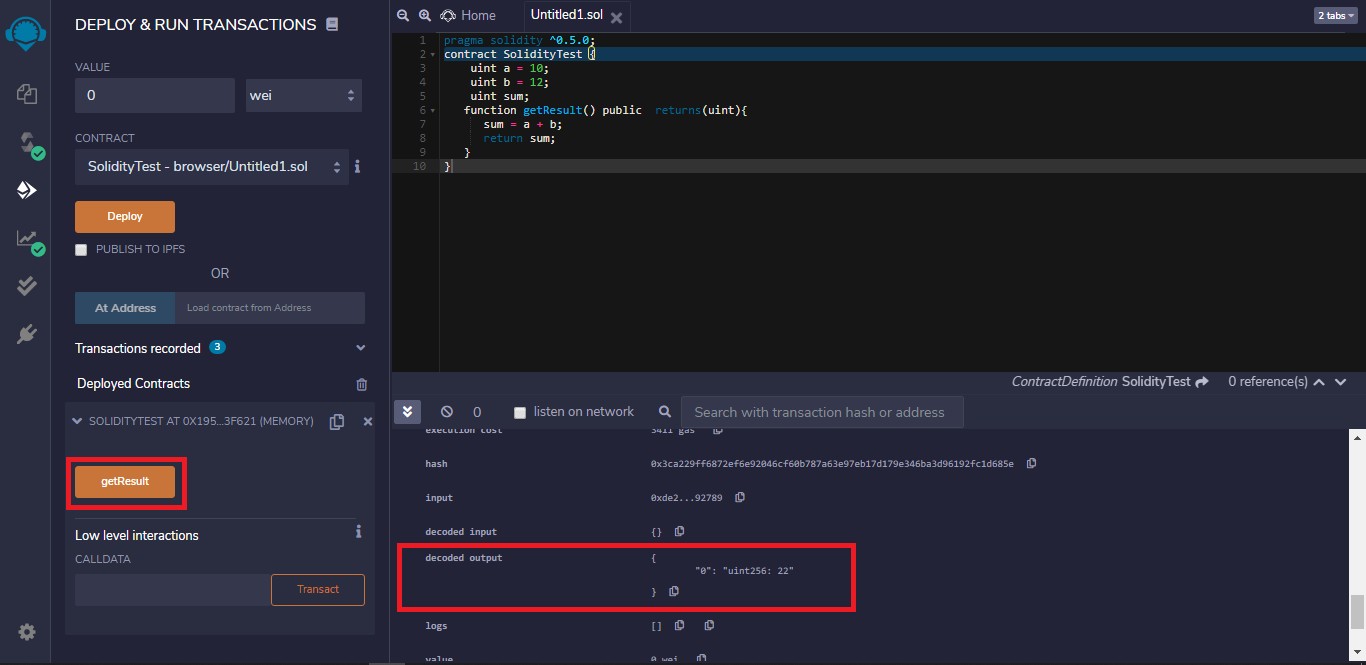
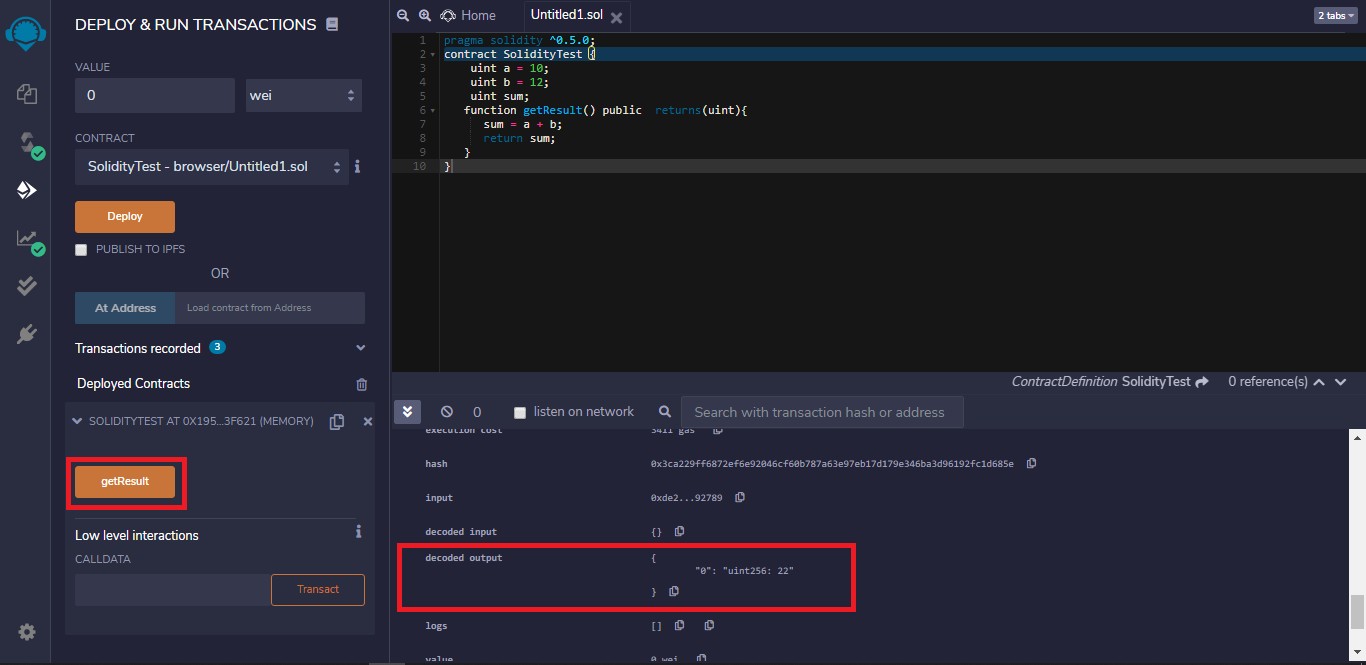
**Step 1:** Open Remix IDE on any of your browsers, select on the *New File* and click on *Solidity* to choose the environment.

**Step 2:** Write the Smart contract in the code section, and click the *Compile button* under the Compiler window to compile the contract.



**Step 3:** To execute the code, click on the *Deploy button* under Deploy and Run Transactions window.

**Step 4:** After deploying the code click on the method calls under the drop-down of deployed contracts to run the program, and for output, check to click on the drop- down on the console.



**Step 5:** For debugging click on the *Debug button* corresponding to the method call in the console. Here you can check each function call and variable assignments.

# Program

Smart contract for the Bank account of the customer to do operations like Deposit, Withdraw and Show Balance

// SPDX-License-Identifier: GPL-3.0 pragma solidity >=0.7.0 <0.9.0; contract SimpleBank {

mapping (address => uint) private balances; address owner;

// Constructor is "payable" so it can receive ether, constructor() public payable {

/\* Set the owner to the creator of this contract \*/ owner = msg.sender;

}

/// @notice Deposit ether into bank, requires method is "payable"

/// @return The balance of the user after the deposit is made function deposit(uint depoamount) public payable returns (uint) {

balances[msg.sender] += depoamount; payable(msg.sender).transfer(depoamount);

//emit LogDepositMade(msg.sender, msg.value); return balances[msg.sender];

}

/// @notice Withdraw ether from bank

/// @return The balance remaining for the user

function withdraw(uint withdrawAmount) public returns (uint) {

// Check enough balance available, otherwise just return balance if (withdrawAmount <= balances[msg.sender]) {

balances[msg.sender] -= withdrawAmount; payable(msg.sender).transfer(withdrawAmount);

}

return balances[msg.sender];

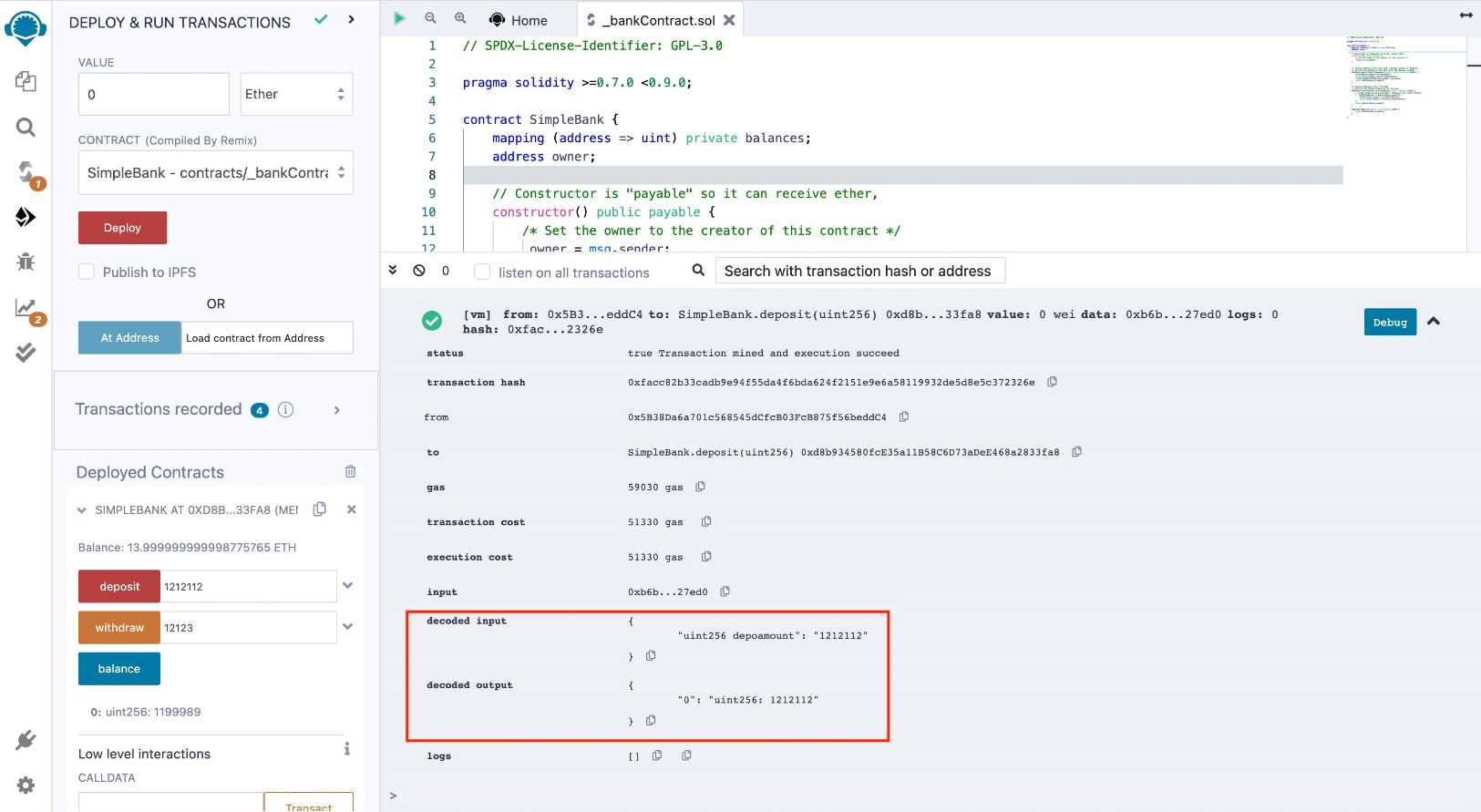
}

function balance() public view returns (uint) { return balances[msg.sender];

}

}

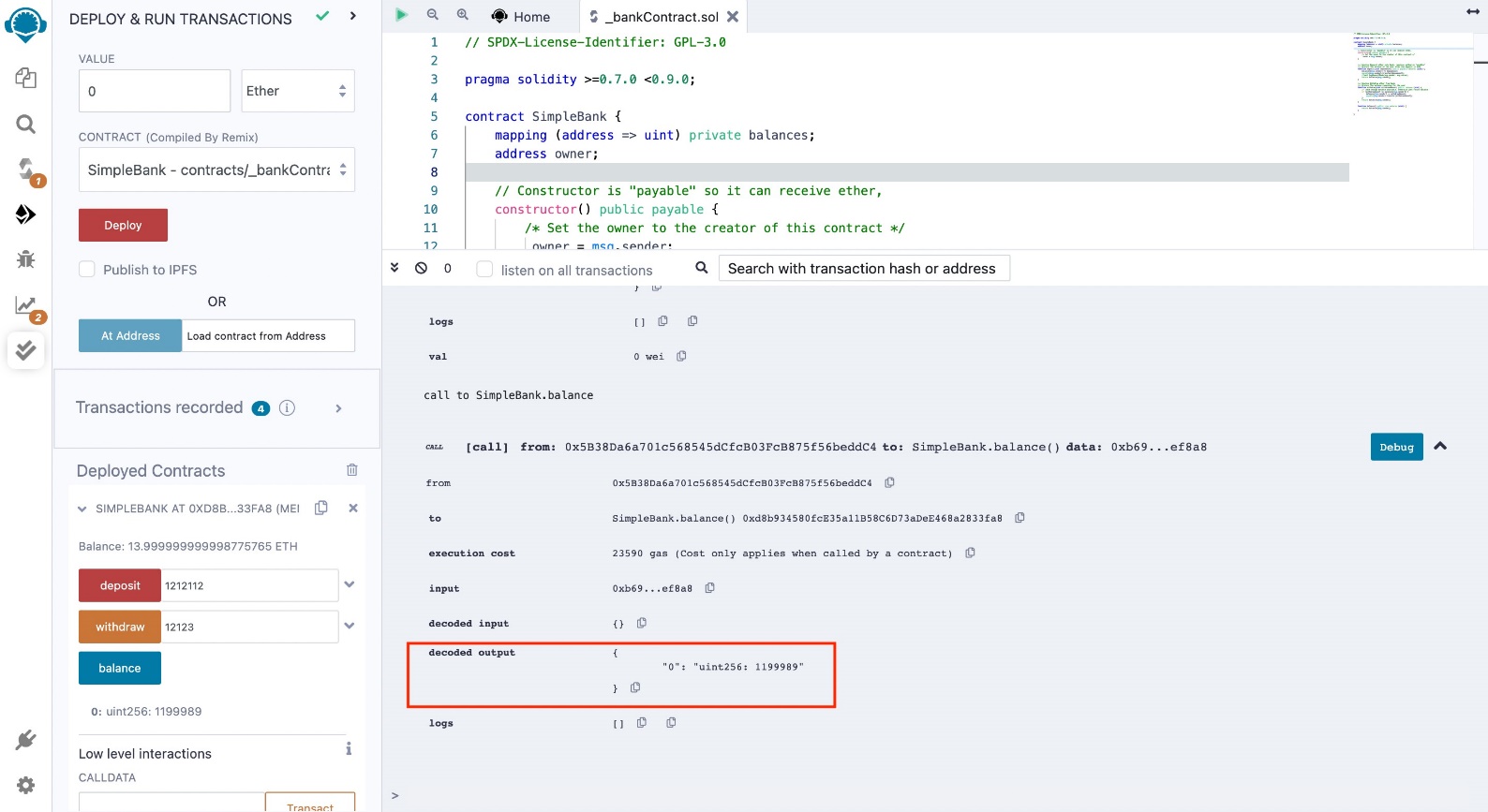
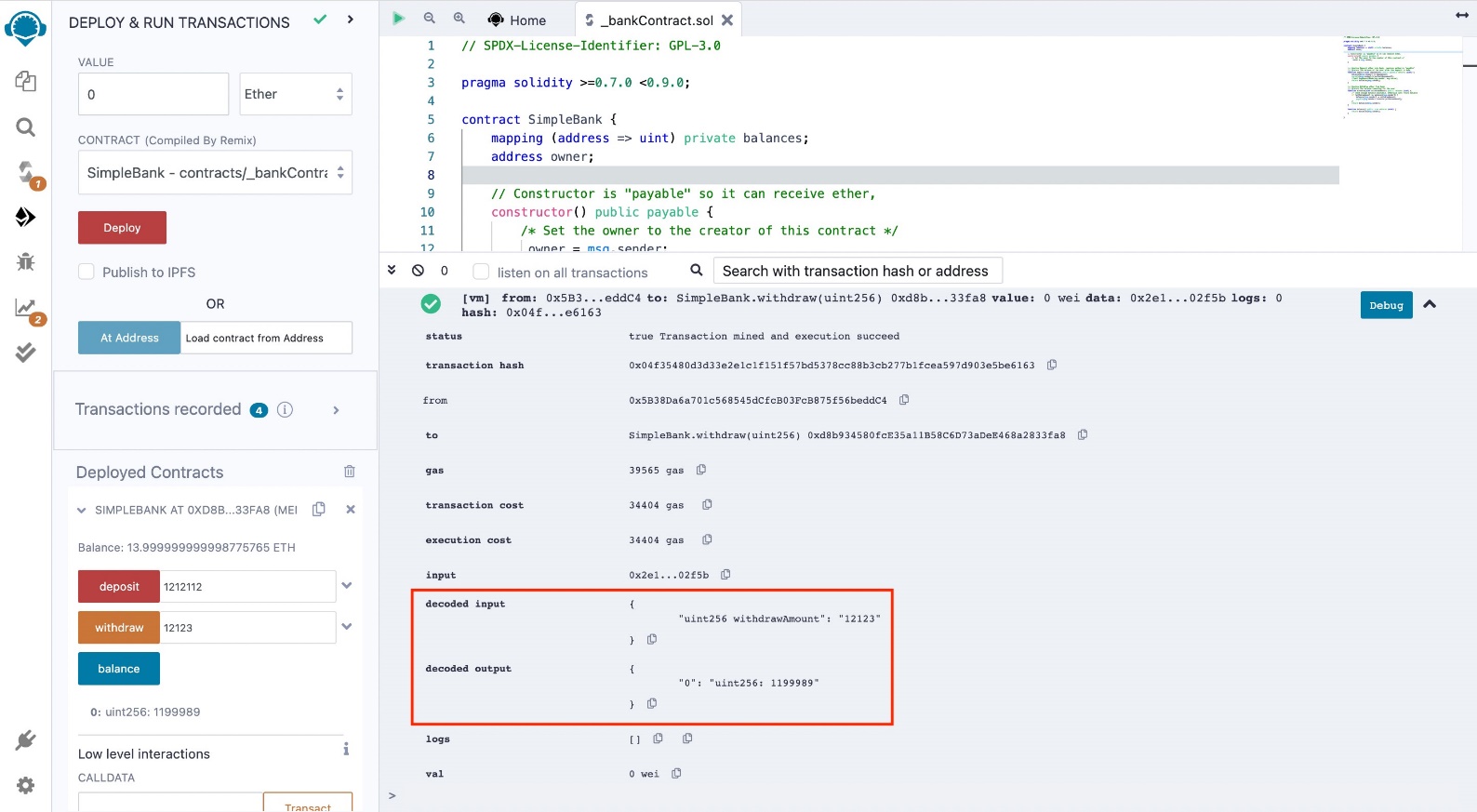
# Output



## Deploy

**Deposit Money**

**Withdraw Money**



**Balance**

**Program**

//SPDX-License-Identifier: UNLICENSED pragma solidity >= 0.7.0 <0.9.0;

// Build the Contract contract MarksManagmtSys

{

// Create a structure for

// student details

struct StudentStruct

{

uint ID;

string fName; string lName;

uint marks;

}

address owner;

uint public stdCount = 0;

//Create Array to store Student data

StudentStruct[] stdRecords;

constructor()

{

owner=msg.sender;

}

// Create a function to add

// the new records

function addNewRecords(uint \_ID,

string memory \_fName, string memory \_lName,

uint \_marks) public payable

{

// Increase the count by 1

stdCount = stdCount + 1;

//Adding data into array

stdRecords.push(StudentStruct(\_ID , \_fName , \_lName , \_marks));

}

function getAllRecords() public view returns(StudentStruct[] memory)

{

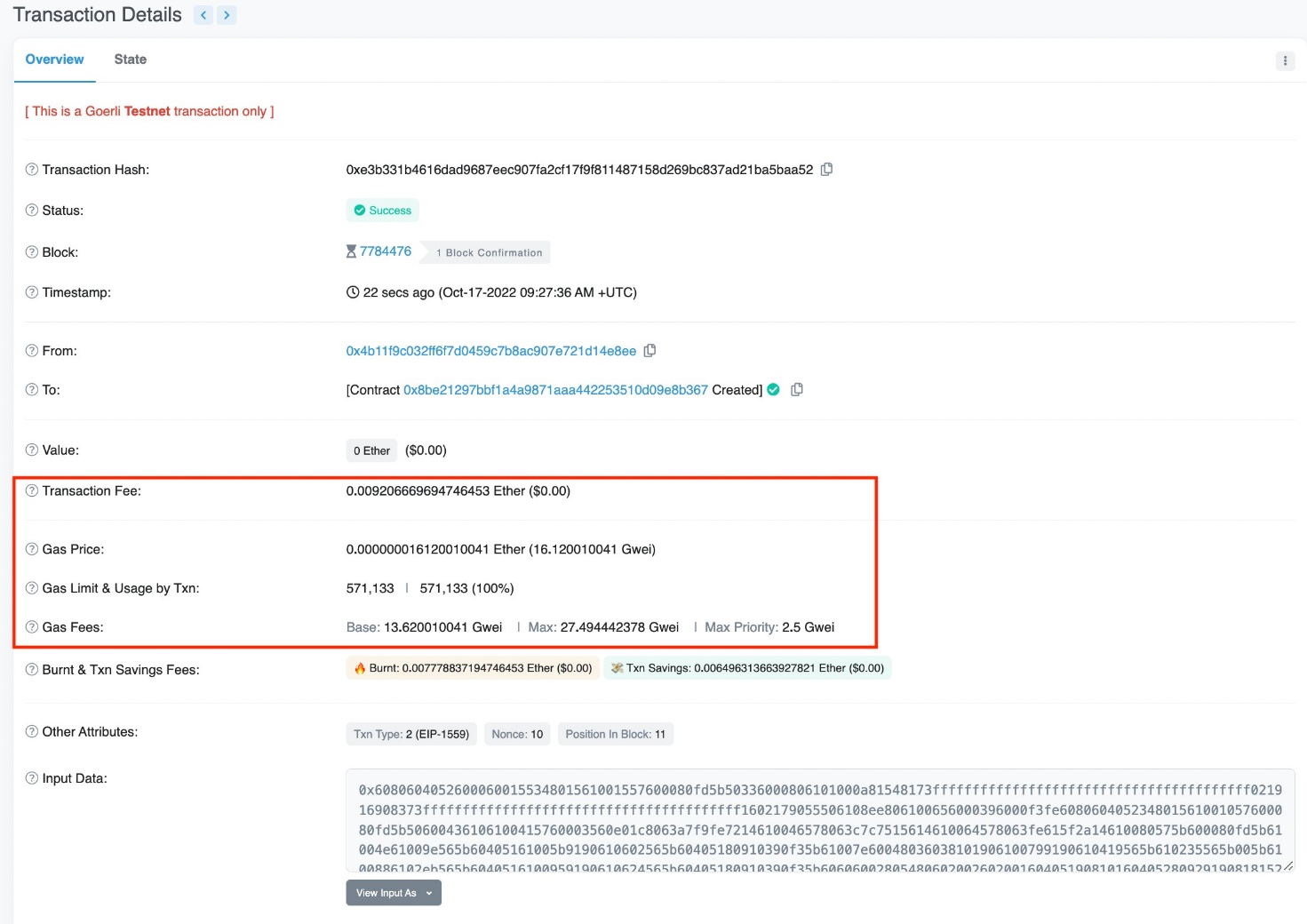
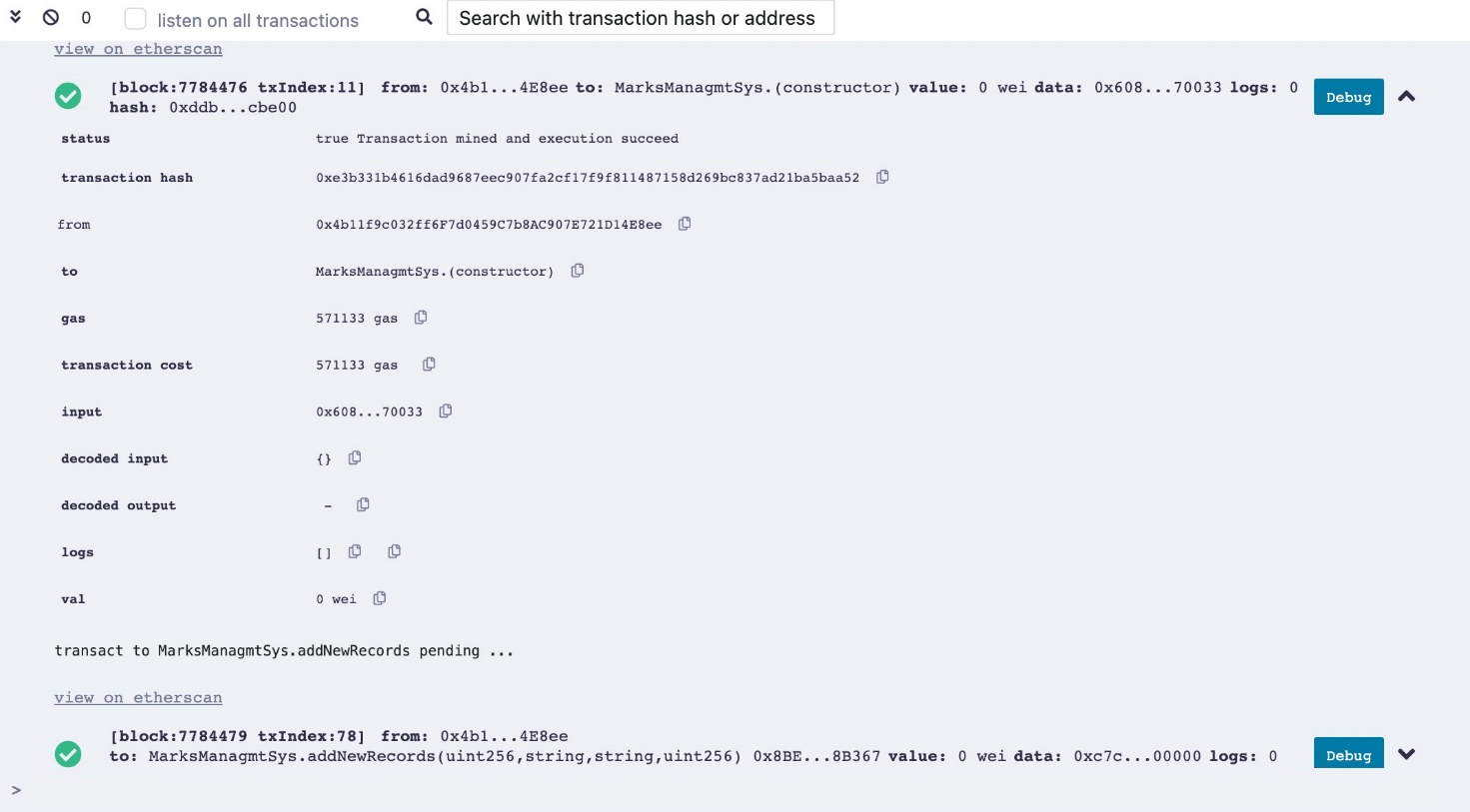
return stdRecords;

}

}

# Output

1. An analysis of the transaction fee and gas fee required for contract deployment



1. An analysis of the transaction fee and gas fee required for smart contracts transactions

