EXPERIMENT NO: 01- Create your own wallet using Coinbase app and also Metamaskfor sending cryptocurrency.

Install the Coinbase Wallet mobile app

Open the MetaMask mobile app or browser extension. Enter in your password to access your wallet if

prompted.

Click on the menu icon on the top left hand corner of the mobile app, or the top right hand corner in the

extension, and navigate to Settings. From the list, click on the “Security & Privacy” option, and tap on

“Reveal Secret Recovery Phrase”. Enter in your password to continue

Copy down your Secret Recovery Phrase. This should be 12 words. Keep this safe.

Open the Coinbase Wallet app, and select “I already have a wallet”. Then tap “Restore with Recovery

Phrase”

Enter in the Secret Recovery Phrase that you copied down on Step 5. This should be 12 words. Keep this

safe, as this is the key to access your wallet and Coinbase cannot recover this phrase for you

Select a Wallet username and set your privacy preferences

Choose your preferred security method: biometric authentication or a passcode. Keep your passcode safe,

as Coinbase cannot recover this for you

All done - your MetaMask wallet has been imported

EXPERIMENT NO: 03 - Write a smart contract on a test network, for Bank accountof a customer for following operations: Deposit money Withdraw Money

// Solidity program to

// retrieve address and

// balance of owner file name as MyContract.sol

pragma solidity ^0.6.8;

// Creating a contract

contract MyContract

{

// Private state variable

address private owner;

// Defining a constructor

constructor() public{

owner=msg.sender;

}

// Function to get

// address of owner

function getOwner(

) public view returns (address) {

return owner;

}

// Function to return current balance of owner

function getBalance(

) public view returns(uint256){

return owner.balance;

}

}

EXPERIMENT NO: 04 - Create a dApp (de-centralized app) for e-voting system

import datetime

import hashlib

import json # Corrected import statement

from flask import Flask, jsonify

class Blockchain:

def \_\_init\_\_(self):

self.chain = []

self.create\_block(proof=1, previous\_hash='0')

def create\_block(self, proof, previous\_hash):

block = {

'index': len(self.chain) + 1,

'timestamp': str(datetime.datetime.now()),

'proof': proof,

'previous\_hash': previous\_hash

}

self.chain.append(block)

return block

def print\_previous\_block(self):

return self.chain[-1]

def proof\_of\_work(self, previous\_proof):

new\_proof = 1

check\_proof = False

while not check\_proof:

hash\_operation = hashlib.sha256(

str(new\_proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:5] == '00000':

check\_proof = True

else:

new\_proof += 1

return new\_proof

def hash(self, block):

encoded\_block = json.dumps(block, sort\_keys=True).encode() # Corrected json

return hashlib.sha256(encoded\_block).hexdigest()

def chain\_valid(self, chain):

previous\_block = chain[0]

block\_index = 1

while block\_index < len(chain):

block = chain[block\_index]

# Check if the previous block's hash matches the current block's 'previous\_hash'

if block['previous\_hash'] != self.hash(previous\_block):

return False

previous\_proof = previous\_block['proof']

proof = block['proof']

hash\_operation = hashlib.sha256(

str(proof\*\*2 - previous\_proof\*\*2).encode()).hexdigest()

if hash\_operation[:5] != '00000':

return False

previous\_block = block

block\_index += 1

return True

# Initialize Flask app

app = Flask(\_\_name\_\_)

blockchain = Blockchain()

@app.route('/mine\_block', methods=['GET'])

def mine\_block():

previous\_block = blockchain.print\_previous\_block()

previous\_proof = previous\_block['proof']

proof = blockchain.proof\_of\_work(previous\_proof)

previous\_hash = blockchain.hash(previous\_block)

block = blockchain.create\_block(proof, previous\_hash)

response = {

'message': 'A block is MINED',

'index': block['index'],

'timestamp': block['timestamp'],

'proof': block['proof'],

'previous\_hash': block['previous\_hash']

}

return jsonify(response), 200

@app.route('/get\_chain', methods=['GET'])

def display\_chain():

response = {

'chain': blockchain.chain,

'length': len(blockchain.chain)

}

return jsonify(response), 200

@app.route('/valid', methods=['GET'])

def valid():

valid = blockchain.chain\_valid(blockchain.chain)

if valid:

response = {'message': 'The Blockchain is valid.'}

else:

response = {'message': 'The Blockchain is not valid.'}

return jsonify(response), 200

# Run the Flask app

if \_\_name\_\_ == "\_\_main\_\_": # Ensure Flask app only runs when script is executed directly

app.run(host='127.0.0.1', port=5000, debug=True)

#http://127.0.0.1:5000/get\_chain

#<http://127.0.0.1:5000/mine_block>

EXPERIMENT NO: 05

Write a program in solidity to create Employee data. Use the following

constructs:

• Structures

• Arrays

• Fallback

Deploy this as smart contract on Ethereum and Observe the

transaction fee and Gas values.

// Solidity program

// to store

// Employee Details

pragma solidity ^0.6.8;

// Creating a Smart Contract

contract StructDemo{

// Structure of employee

struct Employee{

// State variables

int empid;

string name;

string department;

string designation;

}

Employee []emps;

// Function to add

// employee details

function addEmployee(

int empid, string memory name,

string memory department,

string memory designation

) public{

Employee memory e

=Employee(empid,

name,

department,

designation);

emps.push(e);

}

// Function to get

// details of employee

function getEmployee(

int empid

) public view returns(

string memory,

string memory,

string memory){

uint i;

for(i=0;i<emps.length;i++)

{

Employee memory e

=emps[i];

// Looks for a matching

// employee id

if(e.empid==empid)

{

return(e.name,

e.department,

e.designation);

}

}

// If provided employee

// id is not present

// it returns Not

// Found

return("Not Found",

"Not Found",

"Not Found");

}

}