1.Write a simple client class that generates the private and public keys by using the built-in Python RSA algorithm and test it.

#**pip3 install pycryptodome**

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

Dinesh = Client()

print ("sender ",Dinesh.identity)

**Aim: Setting up Ethereum network by using Geth command line interface.(INSTALLATION**

Install on Ubuntu via PPAs

The easiest way to install go-ethereum on Ubuntu-based distributions is with the built-in launchpad PPAs (Personal Package Archives). We provide a single PPA repository that contains both our stable and development releases for Ubuntu versions trusty, xenial, zesty and artful.

linux:

To enable our launchpad repository run:

Step 1: open new terminal

Step 2: on terminal type this command

sudo add-apt-repository -y ppa:ethereum/ethereum

#if above command gives error then run

#sudo apt-get install --reinstall ca-certificates

Step 3: install the stable version of go-ethereum:

sudo apt-get update

sudo apt-get install ethereum

1. Transfer ethers from one **contract** to another on an Ethereum testnet.

pragma solidity ^0.8.0;

contract sendEther{

function getBalance() external view returns(uint)

{

return address(this).balance;

}

receive() external payable { }

}

1. Transfer ethers from one **account** to another on an Ethereum testnet.

//https://dev.to/sparklesix/solidity-tutorial-how-to-build-and-deploy-a-smart-contract-to-send-ether-from-one-account-to-another-n54

pragma solidity ^0.8.11;

contract Sender {

uint public amount;

address payable owner;

constructor (){

owner = payable(msg.sender); // set the deployer of contract as the owner

}

function sendEth(address payable receiver) payable public{

require(owner == msg.sender, "Only the owner can send funds");

amount = msg.value;

receiver.transfer(amount);

}

}

Ganach

https://abhibvp003.medium.com/how-to-install-and-execute-truffle-on-an-ubuntu-16-04-7d0ff6458c9b

https://ethereum.stackexchange.com/questions/93533/call-an-existing-contract-function-from-truffle-console

sudo apt-get -y install curl git vim build-essential

sudo apt-get install curl software-properties-common

sudo apt install npm

sudo npm install -g web3

sudo apt-get install nodejs

sudo apt install python3.9

curl -sL https://deb.nodesource.com/setup\_10.x | sudo bash -

sudo npm install --global node-sass@latest

sudo npm install -g truffle@latest

sudo npm install -g ganache-cli

export NODE\_OPTIONS=--openssl-legacy-provider

////to update npm//

sudo npm cache clean -f

sudo npm install -g n

sudo n latest

//////////////////////////////////////

Start from here!!!

mkdir upg1

cd upg1

truffle init

////////// create contract

nano contracts/HelloWorld.sol

pragma solidity ^0.5.0;

contract HelloWorld {

function sayHello() public pure returns(string memory){

return("hello world");

}

}

////////////////////create configuration

nano migrations/1\_initial\_migration.js

const Migrations = artifacts.require("HelloWorld");

module.exports = function (deployer) {

deployer.deploy(Migrations,"hello");

};

////////////////network configuration

nano truffle-config.js

module.exports = {

networks: {

development: {

host: "127.0.0.1",

port: 8545,

network\_id: "\*",

}

}

}

////////////////////start ganache-cli

ganache-cli

////////////////

truffle migrate

truffle console

#replace contact address

contract = await HelloWorld.at('0x37354B83aadd35516c56f24b724228f29300be77')

a = await contract.sayHello()

a

Deploy a local private blockchain over a network with Ethereum or Rust (VM )

Aim: Create your own blockchain and demonstrate its use.

Install on Ubuntu via PPAs

The easiest way to install go-ethereum on Ubuntu-based distributions is with the built-in launchpad PPAs (Personal Package Archives). We provide a single PPA repository that contains both our stable and development releases for Ubuntu versions trusty, xenial, zesty and artful.

linux:

To enable our launchpad repository run:

Step 1: open new terminal

Step 2: on terminal type this command

sudo add-apt-repository -y ppa:ethereum/ethereum

#if above command gives error then run

#sudo apt-get install --reinstall ca-certificates

Step 3: install the stable version of go-ethereum:

sudo apt-get update

sudo apt-get install ethereum

Step 4: create new directory for storing blockchain data

mkdir myblockchain2

cd myblockchain2

geth account new --datadir data

Step 5:Create genesis.json file

sudo nano genesis.json

{

"config": {

"chainId": 12345,

"homesteadBlock": 0,

"eip150Block": 0,

"eip155Block": 0,

"eip158Block": 0,

"byzantiumBlock": 0,

"constantinopleBlock": 0,

"petersburgBlock": 0,

"istanbulBlock": 0,

"berlinBlock": 0,

"ethash": {}

},

"difficulty": "1",

"gasLimit": "8000000",

"alloc": {

"7df9a875a174b3bc565e6424a0050ebc1b2d1d82": { "balance": "300000" },

"Efaf4df069211972a7D2C3306d1F778a1603F10F": { "balance": "400000" }

}

}

save the file -> ctrl +o to write -> {enter} save -> ctrl +x exit

Step 6: initialize the block

geth init --datadir data genesis.json

Step 7: create network

geth --datadir data --networkid 12345

[do not close this terminal]

//////////////////////////////////////////////////////////////////////

Step 8: open new tab/terminal 2:

sudo geth attach data/geth.ipc

eth.getBalance(eth.accounts[0])

miner.setEtherbase(eth.accounts[0])

miner.start()

admin.addPeer(admin.nodeInfo.enode)

eth.getBalance(eth.accounts[0])

Step 10: Wait for 10-20 minutes and check balance

eth.getBalance(eth.accounts[0])

if ether balance is 0 wait for 10-20minutes for mining process to get complete and run eth.getBalance(eth.accounts[0]) again.

After balance is updated you can check current block height

eth.blockNumber

Implement the mining module of Bitcoin client . The mining module, or miner, should produce blocks that solve proof-of-work puzzle

**Refer word copy**

**Bitcoin core**

Compile and test smart contracts on a testing framework using the Ethereum Virtual Machine (EVM).

Run any program on ganache or remix

pip3 install bitcoinlib

**Aim: Demonstrate the use of Bitcoin Core API.**

Open Python IDLE and create new Script.

########################

from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet1')

key1 = w.get\_key()

print('Wallet Address:',key1.address)

w.scan()

print(w.info())

####################

Open CMD and install bitcoinlib package

**pip install bitcoinlib**

Create your own blockchain and demonstrate its use.

# following imports are required by PKI

import hashlib

import random

import binascii

import datetime

import collections

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

from collections import OrderedDict

import Crypto

import Crypto.Random

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

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

random = Random.new().read

self.\_private\_key = RSA.generate(1024, random)

self.\_public\_key = self.\_private\_key.publickey()

self.\_signer = PKCS1\_v1\_5.new(self.\_private\_key)

@property

def identity(self):

return binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii')

class Transaction:

def \_\_init\_\_(self, sender, recipient, value):

self.sender = sender

self.recipient = recipient

self.value = value

self.time = datetime.datetime.now()

def to\_dict(self):

if self.sender == "Genesis":

identity = "Genesis"

else:

identity = self.sender.identity

return collections.OrderedDict({

'sender': identity,

'recipient': self.recipient,

'value': self.value,

'time' : self.time})

def sign\_transaction(self):

private\_key = self.sender.\_private\_key

signer = PKCS1\_v1\_5.new(private\_key)

h = SHA.new(str(self.to\_dict()).encode('utf8'))

return binascii.hexlify(signer.sign(h)).decode('ascii')

def display\_transaction(transaction):

#for transaction in transactions:

dict = transaction.to\_dict()

print ("sender: " + dict['sender'])

print ('-----')

print ("recipient: " + dict['recipient'])

print ('-----')

print ("value: " + str(dict['value']))

print ('-----')

print ("time: " + str(dict['time']))

print ('-----')

def dump\_blockchain (self):

print ("Number of blocks in the chain: " + str(len (self)))

for x in range (len(TPCoins)):

block\_temp = TPCoins[x]

print ("block # " + str(x))

for transaction in block\_temp.verified\_transactions:

display\_transaction (transaction)

print ('--------------')

print ('=====================================')

class Block:

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

self.verified\_transactions = []

self.previous\_block\_hash = ""

self.Nonce = ""

def sha256(message):

return hashlib.sha256(message.encode('ascii')).hexdigest()

def mine(message, difficulty=1):

assert difficulty >= 1

#if(difficulty <1):

# return

#'1'\*3=> '111'

prefix = '1' \* difficulty

for i in range(1000):

digest = sha256(str(hash(message)) + str(i))

if digest.startswith(prefix):

return i #i= nonce value

A = Client()

B =Client()

C =Client()

t0 = Transaction (

"Genesis",

A.identity,

500.0

)

t1 = Transaction (

A,

B.identity,

40.0

)

t2 = Transaction (

A,

C.identity,

70.0

)

t3 = Transaction (

B,

C.identity,

700.0

)

#blockchain

TPCoins = []

block0 = Block()

block0.previous\_block\_hash = None

Nonce = None

block0.verified\_transactions.append (t0)

digest = hash (block0)

last\_block\_hash = digest #last\_block\_hash it is hash of block0

TPCoins.append (block0)

block1 = Block()

block1.previous\_block\_hash = last\_block\_hash

block1.verified\_transactions.append (t1)

block1.verified\_transactions.append (t2)

block1.Nonce=mine (block1, 2)

digest = hash (block1)

last\_block\_hash = digest

TPCoins.append (block1)

block2 = Block()

block2.previous\_block\_hash = last\_block\_hash

block2.verified\_transactions.append (t3)

Nonce = mine (block2, 2)

block2.Nonce=mine (block2, 2)

digest = hash (block2)

last\_block\_hash = digest

TPCoins.append (block2)

dump\_blockchain(TPCoins)

#####################################################################

save the file -> ctrl +O to write -> {enter} save -> ctrl +x exit

**Run this file**

Output:

Number of blocks in the chain: 3

block # 0

sender: Genesis

-----

recipient: 30819f300d0609…..

-----

value: 500.0

-----

time: 2022-04-26 04:30:59.070952

-----

--------------

=====================================

block # 1

sender: 30819f300d06092a86…..

-----

recipient: 30819f300d06092a…..

-----

value: 40.0

-----

time: 2022-04-26 04:30:59.071076

-----

--------------

sender: 30819f300d06092a86….

-----

recipient: 30819f300d06092a….

-----

value: 70.0

-----

time: 2022-04-26 04:30:59.071174

-----

--------------

=====================================

block # 2

sender: 30819f300d06092a….

-----

recipient: 30819f300d06092a….

-----

value: 700.0

-----

time: 2022-04-26 04:30:59.071272

-----

--------------

=====================================

Solidity

6a

pragma solidity ^0.5.0;

contract SolidityTest {

uint storedData; // State variable

uint public a=10;

constructor() public {

storedData = 10;

}

function getResult(uint c) public view returns(uint){

uint a = 1; // local variable

uint b = 2;

uint result = a + b;

return result; //access the state variable

}

}

**Aim: Demonstrate the use of Bitcoin Core API.**

Open Python IDLE and create new Script.

########################

from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet1')

key1 = w.get\_key()

print('Wallet Address:',key1.address)

w.scan()

print(w.info())

####################

Open CMD and install bitcoinlib package

**pip install bitcoinlib**

After installing package run the program from Python IDLE

ore Detail:<https://pypi.org/project/bitcoinlib/>