PRACTICAL NO 1 :

Write the following programs for blockchain in python:

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

CODE :

import hashlib

import random

import string

import json

import binascii

import numpy as np

import pandas as pd

import pylab as pl

import logging

import datetime

import collections

import Crypto

from Crypto import Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

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

random = Crypto.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):

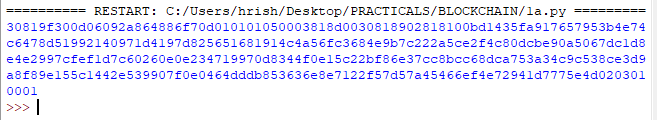
idn = binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii') # <------

return idn

Dinesh = Client()

print(Dinesh.identity)

OUTPUT :



B ] A transaction class to send and receive money and test it.

CODE :

import hashlib

import random

import string

import json

import binascii

import numpy as np

import pandas as pd

import pylab as pl

import logging

import datetime

import collections

import Crypto

from Crypto import Random

from Crypto.Hash import SHA

from Crypto.PublicKey import RSA

from Crypto.Signature import PKCS1\_v1\_5

class Client:

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

random = Crypto.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):

idn = binascii.hexlify(self.\_public\_key.exportKey(format='DER')).decode('ascii') # <------

return idn

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')

Dinesh = Client()

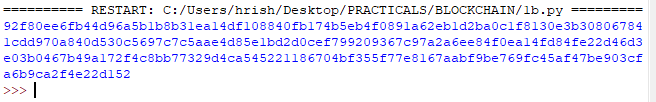
Ramesh = Client()

t = Transaction(Dinesh, Ramesh.identity, 5.0)

signature = t.sign\_transaction()

print (signature)

OUTPUT :



C ] Create multiple transactions and display them.

CODE :

OUTPUT :

D ] Create a blockchain, a genesis block and execute it.

CODE :

import hashlib

import json

from time import time

# creating the Block\_chain class

class Block\_chain(object):

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

self.chain = []

self.pendingTransactions = []

self.newBlock(previousHash = "The first Transaction", the\_proof = 100)

def newBlock(self, the\_proof, previousHash = None):

the\_block = {

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

'timestamp': time(),

'transactions': self.pendingTransactions,

'proof': the\_proof,

'previous\_hash': previousHash or self.hash(self.chain[-1]),

}

self.pendingTransactions = []

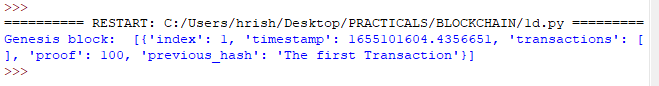
self.chain.append(the\_block)

return the\_block

block\_chain = Block\_chain()

print("Genesis block: ", block\_chain.chain)

OUTPUT :



E ] Create a mining function and test it.

CODE :

# import libraries

import Crypto

from Crypto.PublicKey import RSA

import hashlib

import random

import string

import json

import binascii

import numpy as np

import pandas as pd

import pylab as pl

import logging

import datetime

import collections

import hashlib

def sha256(message):

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

def mine(message, difficulty=1):

assert difficulty >= 1

prefix = '1' \* difficulty

for i in range(1000):

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

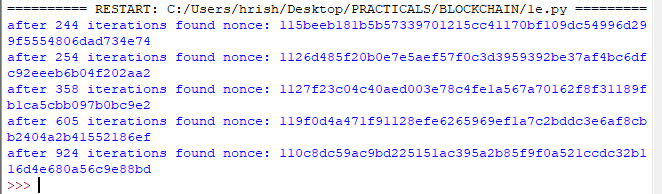
if digest.startswith(prefix):

print ("after " + str(i) + " iterations found nonce: "+ digest)

return digest

mine ("test message", 2)

OUTPUT :



F ] Add blocks to the miner and dump the blockchain.

CODE :

OUTPUT :

PRACTICAL NO 2:

Install and configure Go Ethereum and the Mist browser. Develop and test a sample application.