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

**Code:**

#import random

from Crypto.PublicKey import RSA

from Crypto import Random

import binascii

from Crypto.Cipher import PKCS1\_v1\_5

class Client:

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

random=Random.new().read

self.\_private\_key=RSA.generate(1024,random) #1024->key size

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')

Rifath=Client()

print('Rifath,3--> \n',Rifath.identity)

**Output:**

**A transaction class to send and receive money and test it**

**Code:**

#import random

from Crypto.PublicKey import RSA

from Crypto import Random

import binascii

from Crypto.Cipher import PKCS1\_v1\_5

from Crypto.Hash import SHA

import datetime

import collections

from Crypto.Signature import PKCS1\_v1\_5

from collections import OrderedDict

class Client:

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

random=Random.new().read

self.\_private\_key=RSA.generate(1024,random) #1024->key size

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,receiver,value):

self.sender=sender

self.receiver=receiver

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,

"receiver":self.receiver,

"value":self.value,

"time":self.time

})

def sign\_tran(self):

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

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

h=SHA.new(str(self.to\_dict).encode('utf-8'))

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

def display\_tran(transaction):

dict=transaction.to\_dict()

print('\nsender,Rifath--> \n'+dict['sender'])

print('\nreceiver,Sara--> \n'+dict['receiver'])

print('\nvalue--> \n'+str(dict['value']))

print('\ntime--> \n'+str(dict['time']))

transactions=[]

Rifath=Client()

Sara= Client()

t1=Transaction(

Rifath,

Sara.identity,

15)

t1.sign\_tran()

display\_tran(t1)

**Create a block chain a Genesis block and execute it**.

*Noance: a randomly generated number (unique) used once in cryptography transaction*

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

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, recipent, value):

self.sender = sender

self.recipent = recipent

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,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(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("recipent:" + dict['recipent'])

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\_transaction:

display\_transaction(transaction)

print("...............")

print("====================")

class Block:

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

self.verified\_transaction=[]

self.previous\_block\_hash=""

self.Nonce=""

Rifath = Client()

t0=Transaction(

"Genesis",

Rifath.identity,

500.0

)

block0=Block()

block0.previous\_block\_hash=None

Nonce=None

block0.verified\_transaction.append(t0)

digest=hash(block0)

last\_block\_hash = digest

TPCoins=[]

TPCoins.append(block0)

dump\_blockchain(TPCoins)

1. **Create multiple transactions and display them**

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

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, recipent, value):

self.sender = sender

self.recipent = recipent

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,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(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("recipent:" + dict['recipent'])

print('-----')

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

print('-----')

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

print('-----')

transactions = []

Rifath = Client()

Armeen = Client()

Sara = Client()

t1 = Transaction(

Rifath,

Armeen.identity,

15.0

)

t1.sign\_tran()

transactions.append(t1)

t2 = Transaction(

Armeen,

Sara.identity,

17.0

)

t2.sign\_tran()

transactions.append(t2)

t3 = Transaction(

Sara,

Armeen.identity,

10.0

)

t3.sign\_tran()

transactions.append(t3)

tn = 1

for t in transactions:

print("Transaction: ", tn)

display\_transaction(t)

tn = tn + 1

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

**Output:**

**PRACTICAL 3**

1. **Create a mining function and test it.**

*Miners: verifies the transactions in block chain*

**Code:**

print("Rifath,3")

import hashlib

def sha256(message):

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

def mine(message,difficulty=1):

assert difficulty>=1 #debugging

prefix= '1'\* difficulty #verify diffficulty

print ("prefix",prefix)

for i in range(1000):

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

print("Testing --> " + digest)

if digest.startswith(prefix):

print("After" + str(i) + "iterations found nounce" + digest)

return i

mine("Rifath", 3)

1. **Add block to miner and dump the block chain.**

*Miners: verifies the transactions in block chain*

**Code:**

from Crypto.PublicKey import RSA

from Crypto import Random

from Crypto.Cipher import PKCS1\_v1\_5

import datetime

import binascii

from collections import OrderedDict

import collections

from Crypto.Hash import SHA

from Crypto.Signature import PKCS1\_v1\_5

import hashlib

print("Rifath,27")

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, recipent, value):

self.sender = sender

self.recipent = recipent

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,

'recipent': self.recipent,

'value': self.value,

'time': self.time

})

def sign\_tran(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("recipent:" + dict['recipent'])

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\_transaction:

display\_transaction(transaction)

print("...............")

print("====================")

class Block:

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

self.verified\_transaction=[]

self.previous\_block\_hash=""

self.Nonce=""

def sha256(message):

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

def mine(message,difficulty=1):

assert difficulty>=1 #debugging

prefix= '1'\* difficulty #verify diffficulty

print ("prefix",prefix)

for i in range(1000):

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

print("Testing --> " + digest)

if digest.startswith(prefix):

print("After " + str(i) + "iterations found nounce " + digest)

return i

mine("Rifath", 3)

transactions = []

Rifath = Client()

Armeen = Client()

Sara = Client()

t0=Transaction(

"Genesis",

Rifath.identity,

500.0

)

t1 = Transaction(

Rifath,

Armeen.identity,

15.0

)

t1.sign\_tran()

transactions.append(t1)

t2 = Transaction(

Armeen,

Sara.identity,

17.0

)

t2.sign\_tran()

transactions.append(t2)

t3 = Transaction(

Sara,

Armeen.identity,

10.0

)

#blockchain

TPCoins=[]

block0=Block()

block0.previous\_block\_hash=None

Nonce=None

block0.verified\_transaction.append(t0)

digest=hash(block0)

last\_block\_hash = digest

last\_block\_hash=digest

TPCoins.append(block0)

block1=Block()

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

block1.verified\_transaction.append(t1)

block1.verified\_transaction.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\_transaction.append(t3)

Nonce=mine(block2,2)

block2.Nonce=mine(block2,2)

digest=hash(block2)

last\_block\_hash=digest

TPCoins.append(block2)

dump\_blockchain(TPCoins)