Practical 1

Write the following programs for Blockchain in Python

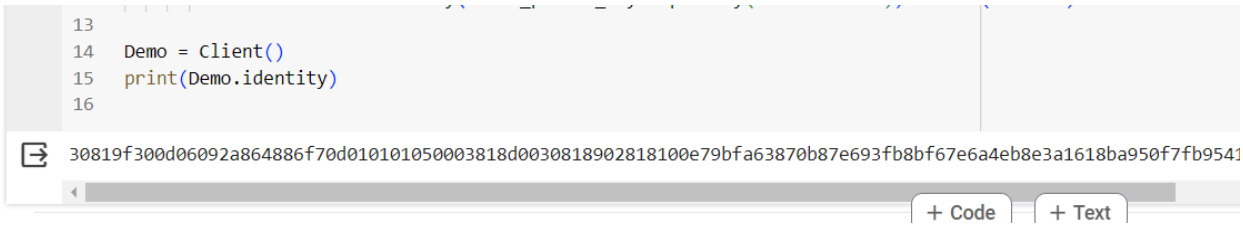
Practical 1 a)

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

Code:

```
#pip install pycryptodome

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

import Crypto
import binascii

from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5

class Client:
    def __init__(self):
        # Creating random number for key
        random = Crypto.Random.new().read
        # Creating new public key and private key
        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')
Demo = Client()
print(Demo.identity)
```

Output:



Practical 1 b)

Aim: A transaction class to send and receive money and test it.

Code:

```
#1B.- A transaction class to send and receive money and test it.
import Crypto
import binascii

import datetime
import collections

from Crypto.PublicKey import RSA
from Crypto.Signature import PKCS1_v1_5
from Crypto.Hash import SHA

class Client:
    def __init__(self):
        # Creating random number for key
        random = Crypto.Random.new().read
        # Creating new public key and private key
        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, 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_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')

Ninad = Client()
print("-"*50)
print("Ninad Key")
print(Ninad.identity)

KS = Client()
print("-"*50)
print("KS Key")
print(KS.identity)

t = Transaction(Ninad, KS.identity, 10.0)
print("-"*50)
print("Transaction Sign")
signature = t.sign_transaction()
print(signature)
print("-"*50)
```

Output:

```
--------------------------------------------------
Ninad Key
30819f300d06092a864886f70d01010105000381d003081890281810a99ad7dbf3cbfbb0d340ae1d091387896b0ef9449e032569819ac
--------------------------------------------------
KS Key
30819f300d06092a864886f70d01010105000381d003081890281810b03818bf884264881b44027dd3e654fdd258339df2d3040ad2122
--------------------------------------------------
Transaction Sign
a7a209a226c7df822f436dae08b64dab0dcc0c744ef7ff198868ee005b3e570969e4cdb817ca36c1d69b999e308c1ae6762755df2101c1e
--------------------------------------------------
```

Practical 1 c)

Aim: Create multiple transactions and display them.

Code:

```
#!pip install pycryptodome

import Crypto
import binascii

from Crypto.PublicKey import RSA
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5

import datetime
import collections

import hashlib
from hashlib import sha256

class Client:
    def __init__(self):
        # Creating random number for key
        random = Crypto.Random.new().read
        # Creating new public key and private key
        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, 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_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 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.startwith(prefix):
            print("after" + str(i) + "iteration found nonce:" + digest)
            return digest

class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

    last_block_hash = ""
```

```
def display_transaction(transaction):
    dict = transaction.to_dict()
    print("Sender: " + dict["sender"])
    print("-----")
    print("Receiver: " + dict["receiver"])
    print("-----")
    print("Value: " + str(dict["value"]))
    print("-----")
    print("Time: " + str(dict["time"]))
    print("-----")

TPCoins = []

def dump_blockchain(self):
    print("Number of blocks in chain" + str(len(self)))
    for x in range(len(Block.TPCoins)):
        block_temp = Block.TPCoins[x]
        print("block #" + str(x))
        for transaction in block_temp.verified_transactions:
            Block.display_transaction(transaction)
            print("-------")

last_transaction_index = 0

transactions = []

Ninad = Client()
ks = Client()
vighnesh = Client()
sairaj = Client()

t1 = Transaction(Ninad, ks.identity, 15.0)
t1.sign_transaction()
transactions.append(t1)

t2 = Transaction(Ninad, vighnesh.identity, 6.0)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(Ninad, sairaj.identity, 16.0)
t3.sign_transaction()
transactions.append(t3)
```

```
t4 = Transaction(vighnesh, Ninad.identity, 8.0)
t4.sign_transaction()
transactions.append(t4)

t5 = Transaction(vighnesh, ks.identity, 19.0)
t5.sign_transaction()
transactions.append(t5)

t6 = Transaction(vighnesh, sairaj.identity, 35.0)
t6.sign_transaction()
transactions.append(t6)

t7 = Transaction(sairaj, vighnesh.identity, 5.0)
t7.sign_transaction()
transactions.append(t7)

t8 = Transaction(sairaj, Ninad.identity, 12.0)
t8.sign_transaction()
transactions.append(t8)

t9 = Transaction(sairaj, ks.identity, 25.0)
t9.sign_transaction()
transactions.append(t9)

t10 = Transaction(Ninad, ks.identity, 1.0)
t10.sign_transaction()
transactions.append(t10)

for transaction in transactions:
    display_transaction(transaction)
    print("*" * 50)
```

Output:

```
print("*"*50)
```

```
Sender: 30819f300d06092a864886f70d010101050003818d00308189028181
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281
-----
Value: 15.0
-----
Time: 2024-04-24 11:58:22.603355
-----
**************************************************
Sender: 30819f300d06092a864886f70d010101050003818d00308189028181
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281
-----
Value: 6.0
-----
Time: 2024-04-24 11:58:22.606018
-----
**************************************************
Sender: 30819f300d06092a864886f70d010101050003818d00308189028181
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281
-----
Value: 16.0
-----
Time: 2024-04-24 11:58:22.608726
-----
**************************************************
```

```
-----
**************************************************
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a72f9c1fb19a4a4382aeaedc6be
-----
Receiver: 30819f300d06092a864886f70d010101050003818d0030818902818100bc5003b8a6f5a9a43f7739cd2
-----
Value: 8.0
-----
Time: 2024-04-24 11:58:22.611128
-----
**************************************************
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a72f9c1fb19a4a4382aeaedc6be
-----
Receiver: 30819f300d06092a864886f70d010101050003818d0030818902818100c61a7aacd1dbedddd4e7a704f
-----
Value: 19.0
-----
Time: 2024-04-24 11:58:22.614112
-----
**************************************************
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100a72f9c1fb19a4a4382aeaedc6be
-----
Receiver: 30819f300d06092a864886f70d010101050003818d0030818902818100bf3f7cc5c45ced69ddd45259a
-----
Value: 35.0
-----
Time: 2024-04-24 11:58:22.616541
-----
**************************************************
```

```
-----
************************************************
Sender: 30819f300d06092a864886f70d01010105000381 8d0030818902818100bf3f7cc5c45ced69ddd45259a1964757464
-----
Receiver: 30819f300d06092a864886f70d01010105000381 8d0030818902818100a72f9c1fb19a4a4382aeaedc6bea34e85
-----
Value: 5.0
-----
Time: 2024-04-24 11:58:22.618543
-----
************************************************
Sender: 30819f300d06092a864886f70d01010105000381 8d0030818902818100bf3f7cc5c45ced69ddd45259a1964757464
-----
Receiver: 30819f300d06092a864886f70d01010105000381 8d0030818902818100bc5003b8a6f5a9a43f7739cd2a15f9541
-----
Value: 12.0
-----
Time: 2024-04-24 11:58:22.619900
-----
************************************************
Sender: 30819f300d06092a864886f70d01010105000381 8d0030818902818100bf3f7cc5c45ced69ddd45259a1964757464
-----
Receiver: 30819f300d06092a864886f70d01010105000381 8d0030818902818100c61a7aacd1dbedddd4e7a704ffa0365d1
-----
Value: 25.0
-----
Time: 2024-04-24 11:58:22.622286
-----
************************************************
Sender: 30819f300d06092a864886f70d01010105000381 8d0030818902818100bc5003b8a6f5a9a43f7739cd2a15f9541bc
-----
Receiver: 30819f300d06092a864886f70d01010105000381 8d0030818902818100c61a7aacd1dbedddd4e7a704ffa0365d1
-----
Value: 1.0
-----
Time: 2024-04-24 11:58:22.624531
-----
************************************************
```

Practical 1 d)

Aim: Create a blockchain, a genesis block and execute it.

Code:

```
# Aim 1D - Create a blockchain, a genesis block and execute it.

#!pip install pycryptodome

import Crypto
import binascii
import datetime
import collections

from Crypto.PublicKey import RSA
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5


class Client:
    def __init__(self):
        # Creating random number for key
        random = Crypto.Random.new().read
        # Creating new public key and private key
        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, 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_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")


class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""

    last_block_hash = ""

    def display_transaction(transaction):
        dict = transaction.to_dict()
        print("Sender: " + dict["sender"])
        print("-----")
        print("Receiver: " + dict["receiver"])  # Corrected typo
        print("-----")
        print("Value: " + str(dict["value"]))
        print("-----")
        print("Time: " + str(dict["time"]))
        print("-----")
```

```
Ninad = Client()
t0 = Transaction("Genesis", Ninad.identity, 500.0)

block0 = Block()
block0.previous_block_hash = None
Nonce = None
block0.verified_transactions.append(t0)
digest = hash(block0)
last_block_hash = digest
TPCoins = []


def dump_blockchain(self):
    print("Number of blocks in 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:
        Block.display_transaction(transaction)
        print("-" * 20)
    print("=" * 30)


TPCoins.append(block0)
dump_blockchain(TPCoins)
```

Output:

```
Number of blocks in chain: 1
block #0
Sender: Genesis
-----
Receiver: 30819f300d06092a864886f70d01010105000381 8d00308189028181 00c15f06dc4692a07cbe45da3a867658a08c996d416ab79414f4
-----
Value: 500.0
-----
Time: 2024-04-24 12:54:53.908551
-----
--------------------
==============================
```

Practical 1 e)

Aim: Create a mining function and test it.

Code:

```
# 1e. Create a mining function and test it.

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("Testmessage", 2)
```

Output:

```
(.venv) E:\GitHub\Practical_BscIT_MscIT_Ninad>e:/GitHub/Practical_BscIT_MscIT_Ninad/.venv/Scripts/python.exe "
ter 4/Blockchain/Practical01/BC_1e.py"
after 169 iterations found nonce: 112e5f10034f79dc396390060ccd5e1b3bcfc6ff550bc3deeceba860b6ec7fc6

(.venv) E:\GitHub\Practical_BscIT_MscIT_Ninad>e:/GitHub/Practical_BscIT_MscIT_Ninad/.venv/Scripts/python.exe "
ter 4/Blockchain/Practical01/BC_1e.py"
after 70 iterations found nonce: 11596db166045592adf5a9aa8eb85dfbe558f6bb0fba1d2921b6775283637c96
```

Practical 1 f)

Aim: Add blocks to the miner and dump the blockchain.

Code:

```
#!pip install pycryptodome

import Crypto
import binascii
import datetime
import collections

from Crypto.PublicKey import RSA
from Crypto.Hash import SHA
from Crypto.Signature import PKCS1_v1_5

import hashlib

class Client:
    def __init__(self):
        # Creating random number for key
        random = Crypto.Random.new().read
        # Creating new public key and private key
        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, 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_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 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 str(digest).startswith(prefix):
            print("after " + str(i) + " iteration found nonce:" + digest)
            return digest


class Block:
    def __init__(self):
        self.verified_transactions = []
        self.previous_block_hash = ""
        self.Nonce = ""
```

```
last_block_hash = ""


def display_transaction(transaction):
    dict = transaction.to_dict()
    print("Sender: " + dict["sender"])
    print("-----")
    print("Receiver: " + dict["receiver"])
    print("-----")
    print("Value: " + str(dict["value"]))
    print("-----")
    print("Time: " + str(dict["time"]))
    print("-----")


TPCoins = []


def dump_blockchain(self):
    print("Number of blocks in 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("=" * 50)


last_transaction_index = 0

transactions = []

Ninad = Client()
ks = Client()
vighnesh = Client()
sairaj = Client()

t1 = Transaction(Ninad, ks.identity, 15.0)
t1.sign_transaction()
transactions.append(t1)
```

```
t2 = Transaction(Ninad, vighnesh.identity, 6.0)
t2.sign_transaction()
transactions.append(t2)

t3 = Transaction(Ninad, sairaj.identity, 16.0)
t3.sign_transaction()
transactions.append(t3)

t4 = Transaction(vighnesh, Ninad.identity, 8.0)
t4.sign_transaction()
transactions.append(t4)

t5 = Transaction(vighnesh, ks.identity, 19.0)
t5.sign_transaction()
transactions.append(t5)

t6 = Transaction(vighnesh, sairaj.identity, 35.0)
t6.sign_transaction()
transactions.append(t6)

t7 = Transaction(sairaj, vighnesh.identity, 5.0)
t7.sign_transaction()
transactions.append(t7)

t8 = Transaction(sairaj, Ninad.identity, 12.0)
t8.sign_transaction()
transactions.append(t8)

t9 = Transaction(sairaj, ks.identity, 25.0)
t9.sign_transaction()
transactions.append(t9)

t10 = Transaction(Ninad, ks.identity, 1.0)
t10.sign_transaction()
transactions.append(t10)

# miner 1 adds block

block = Block()

for i in range(3):  # Limit loop iterations to list length
    temp_transaction = transactions[last_transaction_index]
```

```
    # validatetransaction
    # if valid
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
digest = hash(block)
TPCoins.append(block)
last_block_hash = digest

###
# miner 2 adds block

block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validatetransaction
    # if valid
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
digest = hash(block)
TPCoins.append(block)
last_block_hash = digest

###
# miner 3 adds block

block = Block()

for i in range(3):
    temp_transaction = transactions[last_transaction_index]
    # validatetransaction
    # if valid
    block.verified_transactions.append(temp_transaction)
    last_transaction_index += 1

block.previous_block_hash = last_block_hash
block.Nonce = mine(block, 2)
```

```
digest = hash(block)
TPCoins.append(block)
last_block_hash = digest

dump_blockchain(TPCoins)
```

Output:

```
54   dump_blockchain(TPCoins)

Number of blocks in chain3
block #0
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810
-----
Value: 15.0
-----
Time: 2024-04-24 17:54:16.451041
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d
```

```
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d8
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810
-----
Value: 6.0
-----
Time: 2024-04-24 17:54:16.455375
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d0030818902818100d8
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810
-----
Value: 16.0
-----
Time: 2024-04-24 17:54:16.456722
-----
-------
==================================================
```

```
==================================================
block #1
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810c6600e615430a2b3f205c5eb34a03a20754469dc8c4
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810d8667c7877981fbb835553d64cb666c434a68c8dc
-----
Value: 8.0
-----
Time: 2024-04-24 17:54:16.458055
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810c6600e615430a2b3f205c5eb34a03a20754469dc8c4
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810bee3099e7f08dbadda4ac76098818bdaef046fc27
-----
Value: 19.0
-----
Time: 2024-04-24 17:54:16.459662
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810c6600e615430a2b3f205c5eb34a03a20754469dc8c4
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810aaef0c1d7ee2cdff143c8ee4799e1ead0ad7ba6f2
-----
Value: 35.0
-----
Time: 2024-04-24 17:54:16.461197
-----
-------
==================================================
```

```
==================================================
block #2
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810aaef0c1d7ee2cdff143c8ee4799e1ead0ad7ba6f21bbfa02891c0969fc1d8
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810c6600e615430a2b3f205c5eb34a03a20754469dc8c4cb94ff240137bd20
-----
Value: 5.0
-----
Time: 2024-04-24 17:54:16.462597
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810aaef0c1d7ee2cdff143c8ee4799e1ead0ad7ba6f21bbfa02891c0969fc1d8
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810d8667c7877981fbb835553d64cb666c434a68c8dd5733b7fc2b2b6adc72
-----
Value: 12.0
-----
Time: 2024-04-24 17:54:16.464411
-----
-------
==================================================
Sender: 30819f300d06092a864886f70d010101050003818d003081890281810aaef0c1d7ee2cdff143c8ee4799e1ead0ad7ba6f21bbfa02891c0969fc1d8
-----
Receiver: 30819f300d06092a864886f70d010101050003818d003081890281810bee3099e7f08dbadda4ac76098818bdaef046fc27b1067a91d2b69cfb52
-----
Value: 25.0
-----
Time: 2024-04-24 17:54:16.465901
-----
-------
==================================================
```

Practical 2

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 3

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 4

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 5

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 6

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 7

Write the following programs for Blockchain in Python

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

Code:

Output:

Practical 8

Write the following programs for Blockchain in Python

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

Code:

Output:
