**Practical 1D**

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

# following imports are required by PKI

import hashlib

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 = ""

Dinesh = Client()

t0 = Transaction (

   "Genesis",

   Dinesh.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 = []

TPCoins.append (block0)

dump\_blockchain(TPCoins)

Output:

Number of blocks in the chain: 1

block # 0

sender: Genesis

-----

recipient: 30819f300d06092…..

-----

value: 500.0

-----

time: 2022-04-26 04:24:05.232662