**BLOCKCHAIN AND CRYPTOCURRENCY**

**ASSIGNMENT # 1**

**K20-1052**

**BSE-6B**

This is a Python script implementing a simple blockchain-based cryptocurrency called "GoofyCoin". The script creates a server that listens on a specific IP address and port and accepts incoming connections. When a connection is established, it creates a new GoofyCoin and transfers it to a new user. It then transfers the coin back to the original creator and attempts to transfer a coin that the user does not own to himself. Finally, it verifies the validity of the first coin in the chain.

The script defines a class called GoofyCoin that represents the cryptocurrency. The class has three methods: CreateUser, CreateCoin, and TransferCoin. The CreateUser method generates a new public-private key pair for a new user. The CreateCoin method creates a new coin and adds it to the chain if the creator is the legitimate owner. The TransferCoin method transfers a coin from one user to another if the sender is the legitimate owner of the coin.

The script uses the ecdsa library for generating digital signatures, hashlib for hashing messages, and socket for networking. The script is not complete and is missing some parts, such as handling errors, storing the blockchain, and verifying the entire chain. Also, it is not recommended to use this implementation in a real-world application without proper security measures and auditing.

**Server(Goofy)**

import random

import string

import hashlib

import json

import socket

import threading

import os

from ecdsa import SigningKey

def handle\_client(client\_socket):

    request = client\_socket.recv(1024)

    request = request.decode()

    print(f"[\*] Received: {request}")

    def get\_random\_string(length):

        result\_str = ''.join(random.choice(request) for i in range(length))

        return result\_str

    realGoofyKey = SigningKey.generate()

    goofyVerifyingKey = realGoofyKey.verifying\_key

    class GoofyCoin(object):

        def \_\_init\_\_(self):

            self.chain = []

        def CreateUser(self):

            userSk = SigningKey.generate()

            userPk = userSk.verifying\_key

            return userSk, userPk

        def CreateCoin(self, goofyKey):

            if goofyKey == realGoofyKey:

                temp\_uniqueId = get\_random\_string(256)

                message = bytes("Goofy coin " + str(temp\_uniqueId), 'utf-8')

                signature = goofyKey.sign\_deterministic(message)

                newGoofyCoin = {

                    'Signature': signature,

                    'Data': message,

                    'Destiny': goofyVerifyingKey,

                    'HashPointer': "genesis"

                }

                # 5th we add the new coin to our chain

                self.chain.append(newGoofyCoin)

            else:

                raise Exception("Do not try to fool me")

        def TransferCoin(self, destinyPk, userSk, coinSignature):

            m = hashlib.sha256()

            m.update(coinSignature)

            for i in self.chain:

                if coinSignature == i.get('Signature'):

                    message = bytes('Pay' + str(destinyPk) +

                                    str(m.digest()), 'utf-8')

                    signature = userSk.sign\_deterministic(message)

                    self.chain.append({

                        'Signature': signature,

                        'Data': message,

                        'Destiny': destinyPk,

                        'HashPointer': m.digest()

                    })

                    return self.chain

            raise Exception("This coin do not exist")

        def VerifyCoin(self, coin):

            coinSignature = coin.get('Signature')

            coinMessage = coin.get('Data')

            coinHashPointer = coin.get('HashPointer')

            currentPos = self.chain.index(coin)-1

            while currentPos >= 0:

                m = hashlib.sha256()

                m.update(self.chain[currentPos].get('Signature'))

                if (coinHashPointer == m.digest()):

                    self.chain[currentPos].get('Destiny').verify(

                        coinSignature, coinMessage)

                    coinSignature = self.chain[currentPos].get('Signature')

                    coinMessage = self.chain[currentPos].get('Data')

                    coinHashPointer = self.chain[currentPos].get('HashPointer')

                    currentPos = currentPos - 1

                else:

                    currentPos = currentPos - 1

            return print('Good Coin')

    Goofy = GoofyCoin()

    Goofy.CreateCoin(realGoofyKey)

    HassanA, HassanS = Goofy.CreateUser()

    Goofy.TransferCoin(HassanS, realGoofyKey, Goofy.chain[0].get('Signature'))

    MaxKey = SigningKey.generate()

    Goofy.TransferCoin(goofyVerifyingKey, HassanA,

                       Goofy.chain[1].get('Signature'))

    Goofy.TransferCoin(HassanS, HassanA, Goofy.chain[2].get('Signature'))

    Goofy.VerifyCoin(Goofy.chain[0])

    client\_socket.send(b"Goofy Coins Received")

    client\_socket.close()

server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server.bind(("0.0.0.0", 12345))

server.listen(5)

print("[\*] Listening on 0.0.0.0:1234")

while True:

    client, addr = server.accept()

    print(f"[\*] Connection from {addr[0]}:{addr[1]}")

    client\_handler = threading.Thread(target=handle\_client, args=(client,))

    client\_handler.start()

**Client**

import socket

import json

client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client.connect(("127.0.0.1", 12345))

message = "Sending Goofy bit Coins "

client.send(message.encode())

response = client.recv(1024)

print(response)

client.close()

**Output:**

