**Cryptography and 19115045**

**Network Security Lab Kunal Sachdeva**

**LAB-5 6th Sem CSE**

**1. Write a program to implement the concept of the ElGamal Cryptosystem.**

**Python Code:**

*import random*

*from math import pow*

*a = random.randint(2, 10)*

*def gcd(a, b):*

*if a < b:*

*return gcd(b, a)*

*elif a % b == 0:*

*return b;*

*else:*

*return gcd(b, a % b)*

*# Generating large random numbers*

*def gen\_key(q):*

*key = random.randint(pow(10, 20), q)*

*while gcd(q, key) != 1:*

*key = random.randint(pow(10, 20), q)*

*return key*

*# Modular exponentiation*

*def power(a, b, c):*

*x = 1*

*y = a*

*while b > 0:*

*if b % 2 == 0:*

*x = (x \* y) % c;*

*y = (y \* y) % c*

*b = int(b / 2)*

*return x % c*

*# Asymmetric encryption*

*def encrypt(msg, q, h, g):*

*en\_msg = []*

*k = gen\_key(q)# Private key for sender*

*s = power(h, k, q)*

*p = power(g, k, q)*

*for i in range(0, len(msg)):*

*en\_msg.append(msg[i])*

*print("g^k used : ", p)*

*print("g^ak used : ", s)*

*for i in range(0, len(en\_msg)):*

*en\_msg[i] = s \* ord(en\_msg[i])*

*return en\_msg, p*

*def decrypt(en\_msg, p, key, q):*

*dr\_msg = []*

*h = power(p, key, q)*

*for i in range(0, len(en\_msg)):*

*dr\_msg.append(chr(int(en\_msg[i]/h)))*

*return dr\_msg*

*# Driver code*

*def main():*

*msg = 'encryption'*

*print("Original Message :", msg)*

*q = random.randint(pow(10, 20), pow(10, 50))*

*g = random.randint(2, q)*

*key = gen\_key(q)# Private key for receiver*

*h = power(g, key, q)*

*print("g used : ", g)*

*print("g^a used : ", h)*

*en\_msg, p = encrypt(msg, q, h, g)*

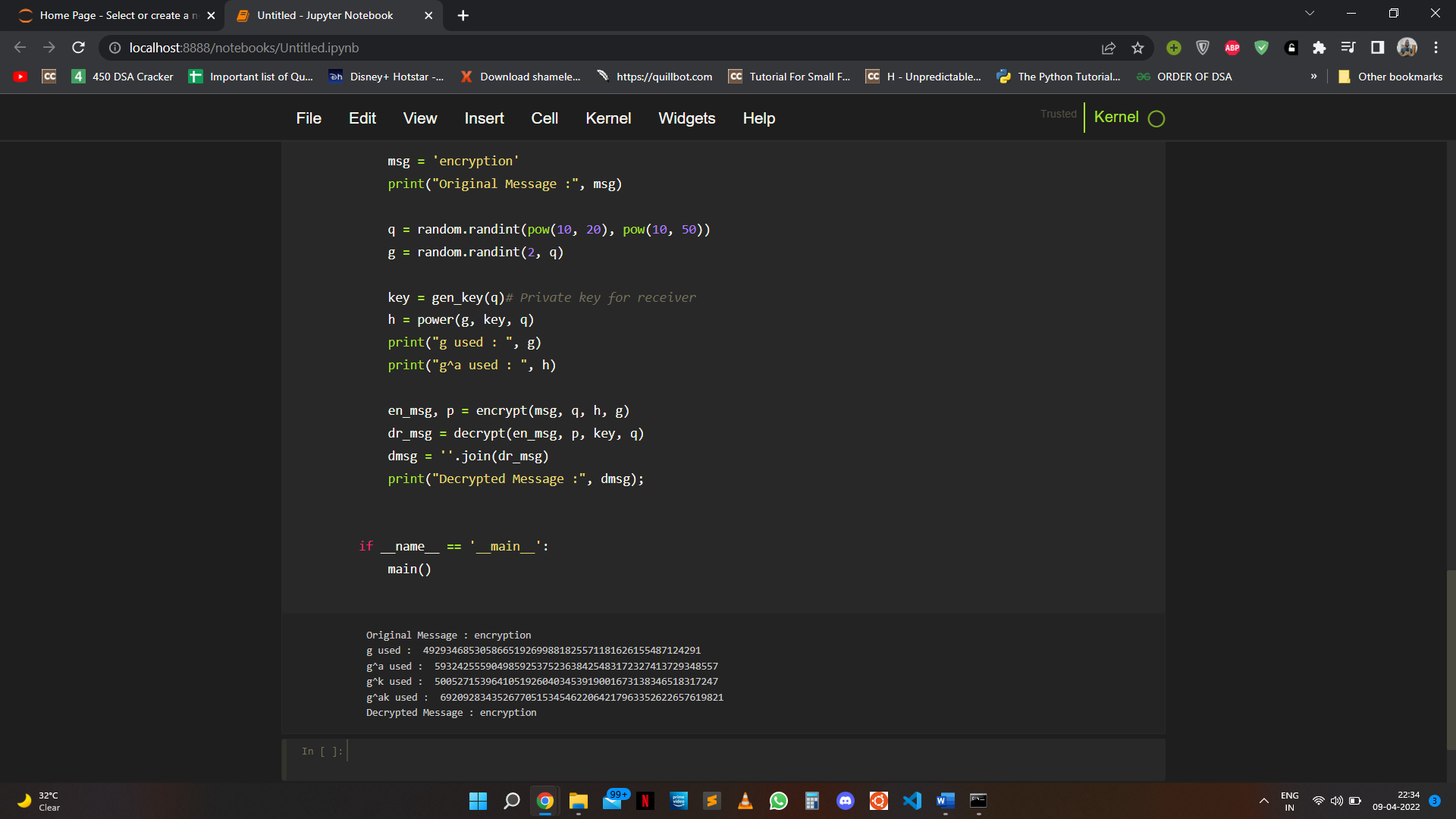
*dr\_msg = decrypt(en\_msg, p, key, q)*

*dmsg = ''.join(dr\_msg)*

*print("Decrypted Message :", dmsg);*

*if \_\_name\_\_ == '\_\_main\_\_':*

*main()*

**Output:** 

**2. Write a program to implement the concept of the Chinese Remainder Theorem.**

**Java Code:**

*import java.util.\*;*

*class CodeSpeedy{*

*static int CRT(int a[], int m[], int n, int p){*

*int x = 0;*

*for(int i = 0; i<n; i++){*

*int M = p/m[i], y = 0; // M1 = p/m1, M2 = p/m2 ....., Mn = p/mn*

*for(int j=0; j<m[i]; j++){*

*if((M\*j)%m[i]==1){*

*y = j; break; // Finding the values for y1, y2,..., yn*

*}*

*}*

*x = x + a[i]\*M\*y; // x = a1\*M1\*y1 + a2\*M2\*y2 + ... + an\*Mn\*yn*

*}*

*return x%p;*

*}*

*public static void main(String args[]){*

*Scanner sc = new Scanner(System.in);*

*System.out.println("Enter the number of congruence relations: ");*

*int size = sc.nextInt();*

*System.out.println("Enter the values of a: ");*

*int a[] = new int[size];*

*for(int i=0; i<size; i++)*

*a[i] = sc.nextInt();*

*System.out.println("Enter the values of m: ");*

*int m[] = new int[size], p = 1;*

*for(int i=0; i<size; i++){*

*m[i] = sc.nextInt();*

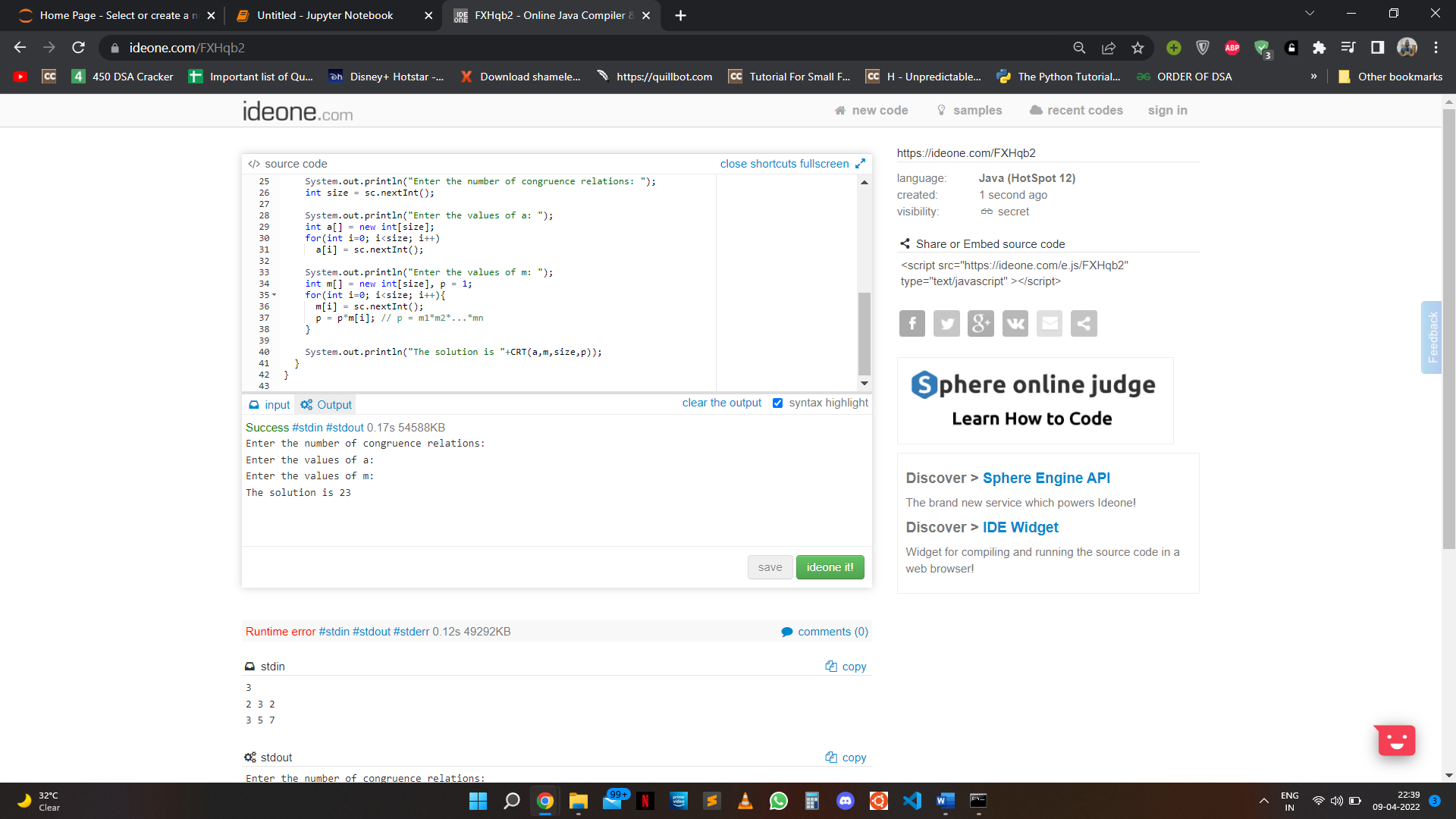
*p = p\*m[i]; // p = m1\*m2\*...\*mn*

*}*

*System.out.println("The solution is "+CRT(a,m,size,p));*

*}*

*}*

**Output:**

**3. Write a program to implement the concept of digital signature.**

**Java Code:**

*// Java implementation for Generating*

*// and verifying the digital signature*

*package java\_cryptography;*

*// Imports*

*import java.security.KeyPair;*

*import java.security.KeyPairGenerator;*

*import java.security.PrivateKey;*

*import java.security.PublicKey;*

*import java.security.SecureRandom;*

*import java.security.Signature;*

*import java.util.Scanner;*

*import javax.xml.bind.DatatypeConverter;*

*public class Digital\_Signature\_GeeksforGeeks {*

*// Signing Algorithm*

*private static final String*

*SIGNING\_ALGORITHM*

*= "SHA256withRSA";*

*private static final String RSA = "RSA";*

*private static Scanner sc;*

*// Function to implement Digital signature*

*// using SHA256 and RSA algorithm*

*// by passing private key.*

*public static byte[] Create\_Digital\_Signature(*

*byte[] input,*

*PrivateKey Key)*

*throws Exception*

*{*

*Signature signature*

*= Signature.getInstance(*

*SIGNING\_ALGORITHM);*

*signature.initSign(Key);*

*signature.update(input);*

*return signature.sign();*

*}*

*// Generating the asymmetric key pair*

*// using SecureRandom class*

*// functions and RSA algorithm.*

*public static KeyPair Generate\_RSA\_KeyPair()*

*throws Exception*

*{*

*SecureRandom secureRandom*

*= new SecureRandom();*

*KeyPairGenerator keyPairGenerator*

*= KeyPairGenerator*

*.getInstance(RSA);*

*keyPairGenerator*

*.initialize(*

*2048, secureRandom);*

*return keyPairGenerator*

*.generateKeyPair();*

*}*

*// Function for Verification of the*

*// digital signature by using the public key*

*public static boolean*

*Verify\_Digital\_Signature(*

*byte[] input,*

*byte[] signatureToVerify,*

*PublicKey key)*

*throws Exception*

*{*

*Signature signature*

*= Signature.getInstance(*

*SIGNING\_ALGORITHM);*

*signature.initVerify(key);*

*signature.update(input);*

*return signature*

*.verify(signatureToVerify);*

*}*

*// Driver Code*

*public static void main(String args[])*

*throws Exception*

*{*

*String input*

*= "GEEKSFORGEEKS IS A"*

*+ " COMPUTER SCIENCE PORTAL";*

*KeyPair keyPair*

*= Generate\_RSA\_KeyPair();*

*// Function Call*

*byte[] signature*

*= Create\_Digital\_Signature(*

*input.getBytes(),*

*keyPair.getPrivate());*

*System.out.println(*

*"Signature Value:\n "*

*+ DatatypeConverter*

*.printHexBinary(signature));*

*System.out.println(*

*"Verification: "*

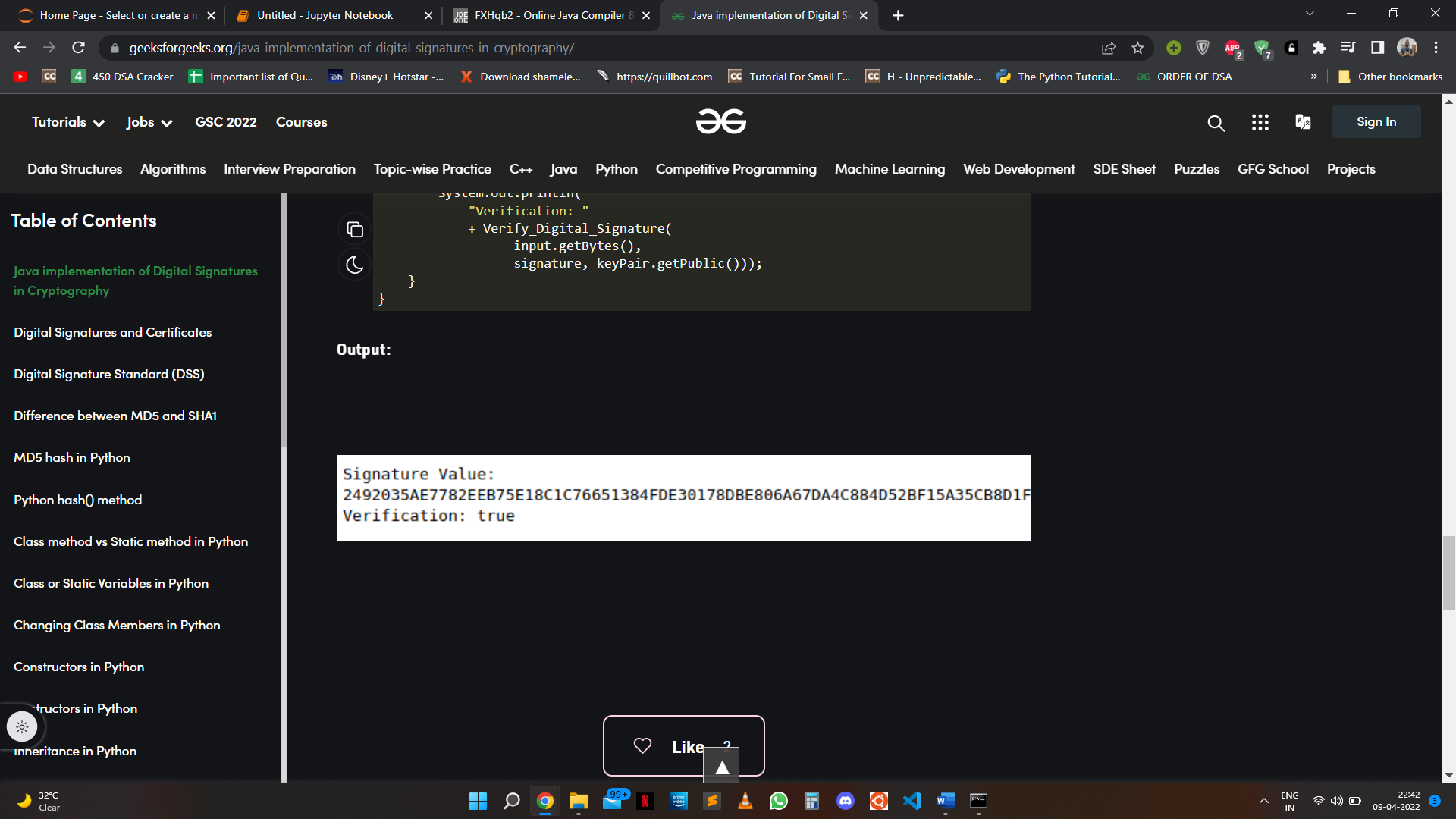
*+ Verify\_Digital\_Signature(*

*input.getBytes(),*

*signature, keyPair.getPublic()));*

*}*

*}*

**Output:**

**4. Write a program for the Kerberos Authentication Protocol.**

**Python Code:**

*import datetime, random*

*class xor\_cipher():*

*''' Very simple stream cipher '''*

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

*''' Basic stream cipher on ascii string and ascii key'''*

*pass*

*def ascii2bin(self, string):*

*''' Converts ascii string to binary bitstring '''*

*return ''.join('{:08b}'.format(ord(asc)) for asc in string)*

*def bin2ascii(self, binary):*

*''' Converts binary bitstring to ascii string '''*

*return ''.join(chr(int(binary[i:i+8], 2)) for i in range(0, len(binary), 8))*

*def bin2hex(self, bn):*

*''' Converts binary bitstring to hex string '''*

*return ''.join('{:x}'.format(int(bn[i:i+4], 2)) for i in range(0, len(bn), 4))*

*def hex2bin(self, hx):*

*''' Converts hex string to binary bitstring '''*

*return ''.join('{:04b}'.format(int(h, 16)) for h in hx)*

*def xor(self, a, b):*

*''' Performs XOR operation of given two inputs a, b '''*

*return ''.join('0' if i == j else '1' for i, j in zip(a, b))*

*def encrypt(self, message, key):*

*''' Performs encryption on message with key, this is xor operation between message and expanded key '''*

*# expanding key to the length of message*

*key += key\*(len(message)-len(key))*

*# converting ascii strings to binary to perform XOR*

*message, key = self.ascii2bin(message), self.ascii2bin(key)*

*# performing xor operation*

*encrypted = self.xor(message, key)*

*# converting binary to hex string as cipher*

*return self.bin2hex(encrypted)*

*def decrypt(self, message, key):*

*''' Performs decryption on message with key, this is xor operation between message and expanded key '''*

*# expanding key to the length of message*

*key += key\*(len(message)-len(key))*

*# converting hex message and ascii key to binary*

*message, key = self.hex2bin(message), self.ascii2bin(key)*

*# performing xor operation*

*decrypted = self.xor(message, key)*

*# converting binary to ascii as plain text*

*return self.bin2ascii(decrypted)*

*class user():*

*'''USER class'''*

*def \_\_init\_\_(self, name, authenticate\_shared\_key, database, nonce):*

*''' User is one participant in the network,*

*(name, authentication server shared key, database to communicate with, nonce) '''*

*self.name = name*

*self.auth\_key = authenticate\_shared\_key*

*self.database = database*

*self.nonce = nonce*

*def prepare\_auth\_request(self):*

*''' Prepares request for authentication server '''*

*# returns a tuple with (name, database name, nonce)*

*return (self.name, self.database, self.nonce)*

*def process\_auth\_response(self, cipher1, cipher2):*

*''' Process the response from authentication server '''*

*# eval is used to extract tuple from string*

*response\_to\_user = eval(cipher.decrypt(cipher1, self.auth\_key))*

*# unpacking tuple into individual elements, OBTAINED USER-TICKET KEY (common for user and TGS)!!*

*self.user\_ticket\_key, nonce, time, ttl, dest = response\_to\_user*

*# verify nonce*

*assert nonce == self.nonce*

*# verify destination name*

*assert dest == 'ticket\_granting\_server'*

*# 2nd element in response is ticket for TGS encrypted with TGS-auth key (common for TGS and auth)*

*self.ticket\_granting\_ticket = cipher2*

*return response\_to\_user*

*def prepare\_ticket\_request(self):*

*''' Prepare request for TGS '''*

*# request is a tuple with (name, time, database name, nonce)*

*request = str((self.name, str(datetime.datetime.now().date()), self.database, self.nonce))*

*# encrypt the request with user-ticket key (common for user and TGS)*

*encrypted\_request = cipher.encrypt(request, self.user\_ticket\_key)*

*return (encrypted\_request, self.ticket\_granting\_ticket)*

*def process\_ticket\_response(self, cipher1, cipher2):*

*''' Process the response from TGS '''*

*# extracting tuple from string*

*response\_to\_user = eval(cipher.decrypt(cipher1, self.user\_ticket\_key))*

*# unpacking elements into individual elements, "USER-DATABASE" key is obtained!! common for USER and DATABASE*

*self.user\_database\_key, nonce, time, life, destination = response\_to\_user*

*# 2nd response from TGS is a ticket for database encrypted with database-TGS key (common for database and TGS)*

*self.database\_ticket = cipher2*

*# verify nonce*

*assert self.nonce == nonce*

*# verify database name*

*assert self.database == destination*

*# if everything is verified, which indicates user is authenticated, then create an object for database*

*# in practice, no such thing happens, because a physical database will be available*

*if self.database == destination:*

*num\_db = dataserver()*

*# return the created object, which in practice returning the permission to access database*

*return num\_db*

*def prepare\_database\_request(self):*

*''' Prepares a request for database '''*

*# create a random token, later used for acknowledgement purpose*

*self.token = random.randint(1, 100)*

*# create a request tuple with (name, token) for database*

*request = str((self.name, self.token))*

*# encrypt the request with user-database key, common for USER and DATABASE*

*encrypted\_request = cipher.encrypt(request, self.user\_database\_key)*

*return (encrypted\_request, self.database\_ticket)*

*def process\_database\_response(self, response):*

*''' Process the database response'''*

*# verify is token is incremented by 1, which is ACKNOWLEDGEMENT*

*assert self.token+1 == response*

*def prepare\_database\_data\_request(self, database, request):*

*''' Prepare data access requests for database'''*

*# data access requests and responses are encrypted with share (user-database) key*

*return cipher.encrypt(f'{database}.get\_data({request})', self.user\_database\_key)*

*class authentication\_server():*

*'''AUTHENTICATION SERVER class'''*

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

*''' Authentication Server, as a part of Key Disrtibution Centre (KDC), which authenticates users '''*

*# Authentication server should have the list of all shared passwords between users and Auth server*

*# TGS keys are inbuilt, as new users get added, their keys will be updated with set\_key method*

*self.keys = {'tgs': 'tgs123'}*

*def set\_key(self, name, key):*

*''' Sets new key for new users into network'''*

*self.keys[name] = key*

*def get\_key(self, name):*

*''' Returns the user's key '''*

*return self.keys[name]*

*def process\_request\_respond(self, request):*

*''' Process the user request and respond '''*

*# unpack user request tuple into individual elements*

*self.client\_name, self.destination, self.nonce = request*

*# return 2 encrypted responses, one for user (encrypted with user-auth key), another for TGS (encrypted with tgs\_auth key)*

*return (self.response\_for\_user(), self.response\_for\_TGS())*

*def response\_for\_user(self):*

*''' Prepare a response for user '''*

*# create a random new key as a common key for USER and TGS*

*self.user\_ticket\_key = str(random.randint(1, 100))*

*# create a response tuple with (user-ticket key, nonce, time, span, TGS name)*

*response = (self.user\_ticket\_key, self.nonce, str(datetime.datetime.now().date()), '3 days', 'ticket\_granting\_server')*

*# since, this is response for user, it is encrypted with common key of auth-server and user*

*return cipher.encrypt(str(response), self.keys[self.client\_name])*

*def response\_for\_TGS(self):*

*''' Prepare a response for TGS '''*

*# create a response tuple with (user-ticket key, client name, span)*

*response = (self.user\_ticket\_key, self.client\_name, '3 days')*

*# since, this response is for TGS, it is encrypted with common key of auth-server and TGS*

*return cipher.encrypt(str(response), self.keys['tgs'])*

*class ticket\_granting\_server():*

*'''TICKET GRANTING SERVER class'''*

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

*''' Ticket Granting Server (TGS), as a part of Key Distribution Centre (KDC), which creates session keys between users'''*

*# personal key for TGS to communicate with Authentication Server*

*self.personal\_key = 'tgs123'*

*# database keys at TGS*

*self.keys = {'number\_database': 'alpha'}*

*def process\_auth\_user\_request\_respond(self, user\_request, auth\_response):*

*''' Process user\_request and auth\_response\_for\_TGS and respond'''*

*# decrypt and exrtact the tuple with personal key, as this is encrypted by auth server*

*auth\_response = eval(cipher.decrypt(auth\_response, self.personal\_key))*

*# unpack auth response into individual elements, obtained USER-TICKET key (common for user and TGS)*

*self.user\_ticket\_key, client\_name, life = auth\_response*

*# decrypt and extract user request with USER-TICKET key obtained in above auth response*

*user\_request = eval(cipher.decrypt(user\_request, self.user\_ticket\_key))*

*# unpack user request into individual elements*

*self.user\_name, time, self.destination, self.nonce = user\_request*

*# return two encrypted responses, one for user (encrypted with user-tgs key), another for database (with database-tgs key)*

*return (self.response\_for\_user(), self.response\_for\_database())*

*def response\_for\_user(self):*

*''' Prepare a response for USER '''*

*# create a key common key for USER - DATABASE*

*self.user\_database\_key = str(random.randint(1, 100))*

*# create response data tuple with (user-database key, nonce, time, span, database name)*

*response = str((self.user\_database\_key, self.nonce, str(datetime.datetime.now().date()), '5 days', self.destination))*

*# encrypt the response with USER-TGS key, common for user and TGS.*

*return cipher.encrypt(response, self.user\_ticket\_key)*

*def response\_for\_database(self):*

*''' Prepare a response for DATABASE '''*

*# create a response tuple with (user-database key, user name, span)*

*response = str((self.user\_database\_key, self.user\_name, '5 days'))*

*# encrypt response with DATABASE-TGS key, common for TGS and Database*

*return cipher.encrypt(response, self.keys[self.destination])*

*class dataserver():*

*''' DATASERVER class '''*

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

*''' This can be another participant or a data server in a network '''*

*# this personal key is pre-shared with TGS*

*self.personal\_key = 'alpha'*

*# example data in database*

*self.data = {1: 'One', 2: 'Two', 3: 'Three'}*

*def \_\_str\_\_(self):*

*return 'number\_database'*

*def get\_data(self, index):*

*''' Get the data from database '''*

*# verify if request is valid*

*assert index in self.data.keys()*

*# return data*

*return self.data[index]*

*def process\_client\_request\_respond(self, client\_token, db\_ticket):*

*''' Process client request and respond '''*

*# process db ticket*

*# db\_ticket is dedicated for database, which can be decrypted with database's personal key*

*db\_ticket = eval(cipher.decrypt(db\_ticket, self.personal\_key))*

*# unpack into individual elements, obtained USER-DATABASE key !!*

*self.user\_database\_key, user\_name, life = db\_ticket*

*# process client token*

*# decrypt client token with user-database key obtained in above ticket*

*client\_token = eval(cipher.decrypt(client\_token, self.user\_database\_key))*

*# unpack token into individual elements*

*self.client\_name, self.token = client\_token*

*# verify name*

*assert self.client\_name == user\_name*

*# if name is valid, then acknowledge user by incrementing token by 1*

*return self.token + 1*

*def process\_data\_request\_respond(self, request):*

*'''Process data request and respond '''*

*# decrypt the user request with user-database key and evaluate the request*

*return eval(cipher.decrypt(request, self.user\_database\_key))*

*# setup ciphering function*

*cipher = xor\_cipher()*

*# setup database server*

*# this is created as a string, to verify across the process*

*# database object is created when requesting user is authenticated in later steps*

*db = 'number\_database'*

*# setup user*

*username = 'ajay'*

*userkey = 'secret\_key'*

*user1 = user(username, userkey, db, 12)*

*# setup authentication server*

*auth\_server = authentication\_server()*

*# add user with key in authentication server*

*auth\_server.set\_key(username, userkey)*

*# setup ticket granting server*

*ticket\_server = ticket\_granting\_server()*

*# KERBEROS PROTOCOL*

*# prepare auth request -> send to auth server -> auth authenticates and responds -> process the auth response*

*user\_request\_to\_auth = user1.prepare\_auth\_request()*

*response\_to\_user\_from\_auth, response\_to\_tgs\_from\_auth = auth\_server.process\_request\_respond(user\_request\_to\_auth)*

*auth\_response\_to\_user = user1.process\_auth\_response(response\_to\_user\_from\_auth, response\_to\_tgs\_from\_auth)*

*# prepare tgs request -> send to tgs -> tgs process tickets and responds -> process the tgs response*

*user\_request\_to\_tgs, ticket\_request\_from\_auth = user1.prepare\_ticket\_request()*

*response\_to\_user\_from\_tgs, response\_to\_db\_from\_tgs = ticket\_server.process\_auth\_user\_request\_respond(user\_request\_to\_tgs, ticket\_request\_from\_auth)*

*database1 = user1.process\_ticket\_response(response\_to\_user\_from\_tgs, response\_to\_db\_from\_tgs)*

*# prepare database request -> send to database -> database acknowledges and responds -> process the response*

*user\_request\_to\_db, db\_ticket\_from\_tgs = user1.prepare\_database\_request()*

*database\_response = database1.process\_client\_request\_respond(user\_request\_to\_db, db\_ticket\_from\_tgs)*

*user1.process\_database\_response(database\_response)*

*# verify user and database successfully shared common keys*

*assert user1.user\_database\_key == database1.user\_database\_key*

*print('Key Establishment successful..!')*

*# request data from database*

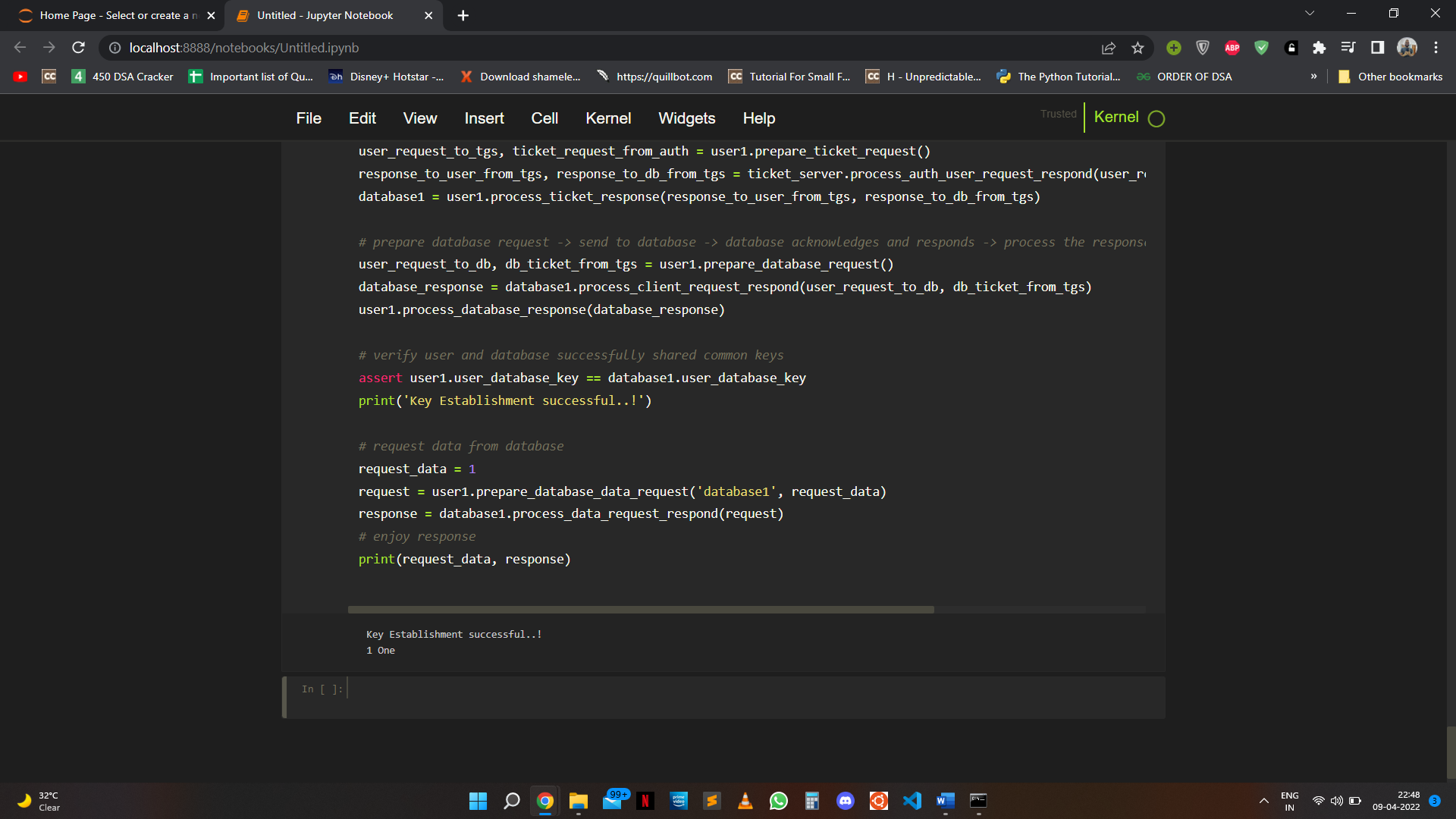
*request\_data = 1*

*request = user1.prepare\_database\_data\_request('database1', request\_data)*

*response = database1.process\_data\_request\_respond(request)*

*# enjoy response*

*print(request\_data, response)*

**Output:**