1.

import random

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

def multiplicative\_inverse(e, phi):

d = 0

x1 = 0

x2 = 1

y1 = 1

temp\_phi = phi

while e > 0:

temp1 = temp\_phi // e

temp2 = temp\_phi - temp1 \* e

temp\_phi = e

e = temp2

x = x2 - temp1 \* x1

y = d - temp1 \* y1

x2 = x1

x1 = x

d = y1

y1 = y

if temp\_phi == 1:

return d + phi

def generate\_keypair(p, q):

n = p \* q

phi = (p-1) \* (q-1)

e = random.randrange(1, phi)

g = gcd(e, phi)

while g != 1:

e = random.randrange(1, phi)

g = gcd(e, phi)

d = multiplicative\_inverse(e, phi)

return ((e, n), (d, n))

def encrypt(public\_key, plaintext):

key, n = public\_key

cipher = [(ord(char) \*\* key) % n for char in plaintext]

return cipher

def decrypt(private\_key, ciphertext):

key, n = private\_key

plain = [chr((char \*\* key) % n) for char in ciphertext]

return ''.join(plain)

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

p = 61

q = 53

public, private = generate\_keypair(p, q)

print("Public key:", public)

print("Private key:", private)

message = input("Enter the message to Securely Communicate")

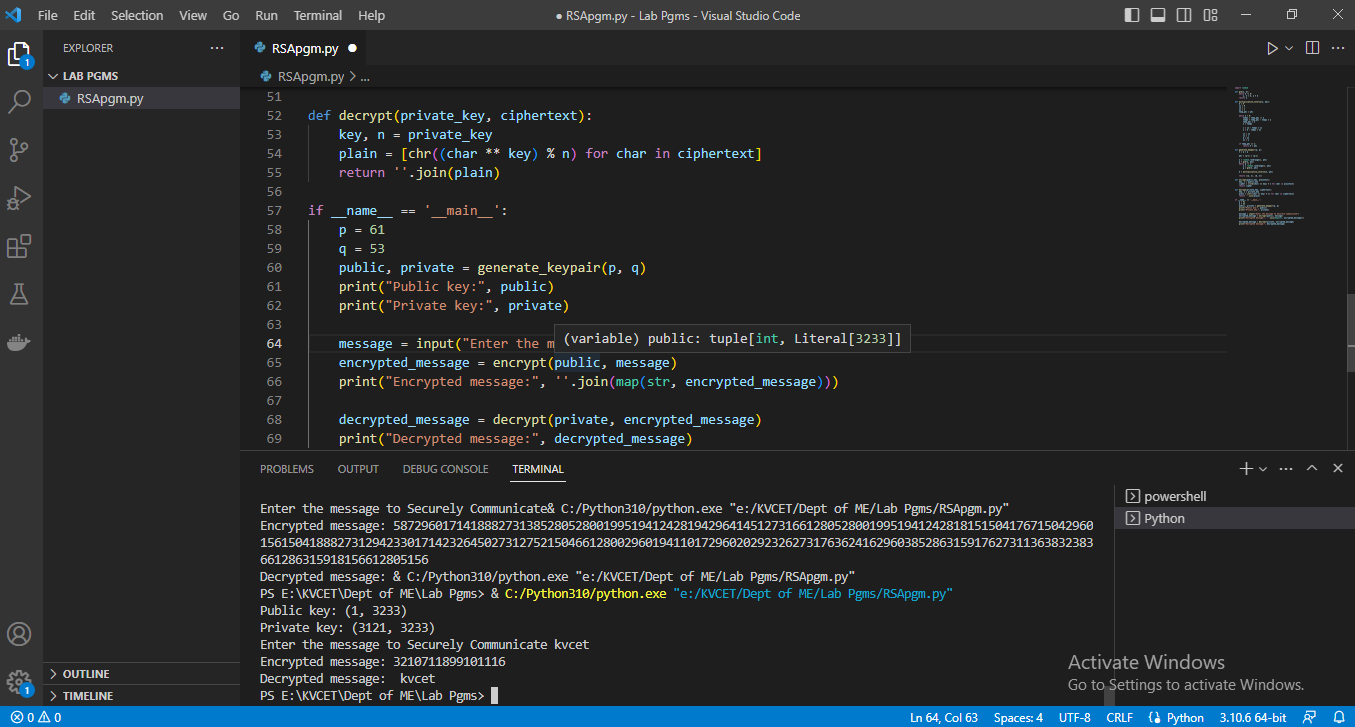
encrypted\_message = encrypt(public, message)

print("Encrypted message:", ''.join(map(str, encrypted\_message)))

decrypted\_message = decrypt(private, encrypted\_message)

print("Decrypted message:", decrypted\_message)

**Output**



2. Experiment on claiming ownership of digital entity

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.backends import default\_backend

# generate a new RSA key pair

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

backend=default\_backend()

)

public\_key = private\_key.public\_key()

# create a message to be signed

message = b"Hello, World!"

# sign the message using the private key

signature = private\_key.sign(

message,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH

),

hashes.SHA256()

)

# verify the signature using the public key

try:

public\_key.verify(

signature,

message,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH

),

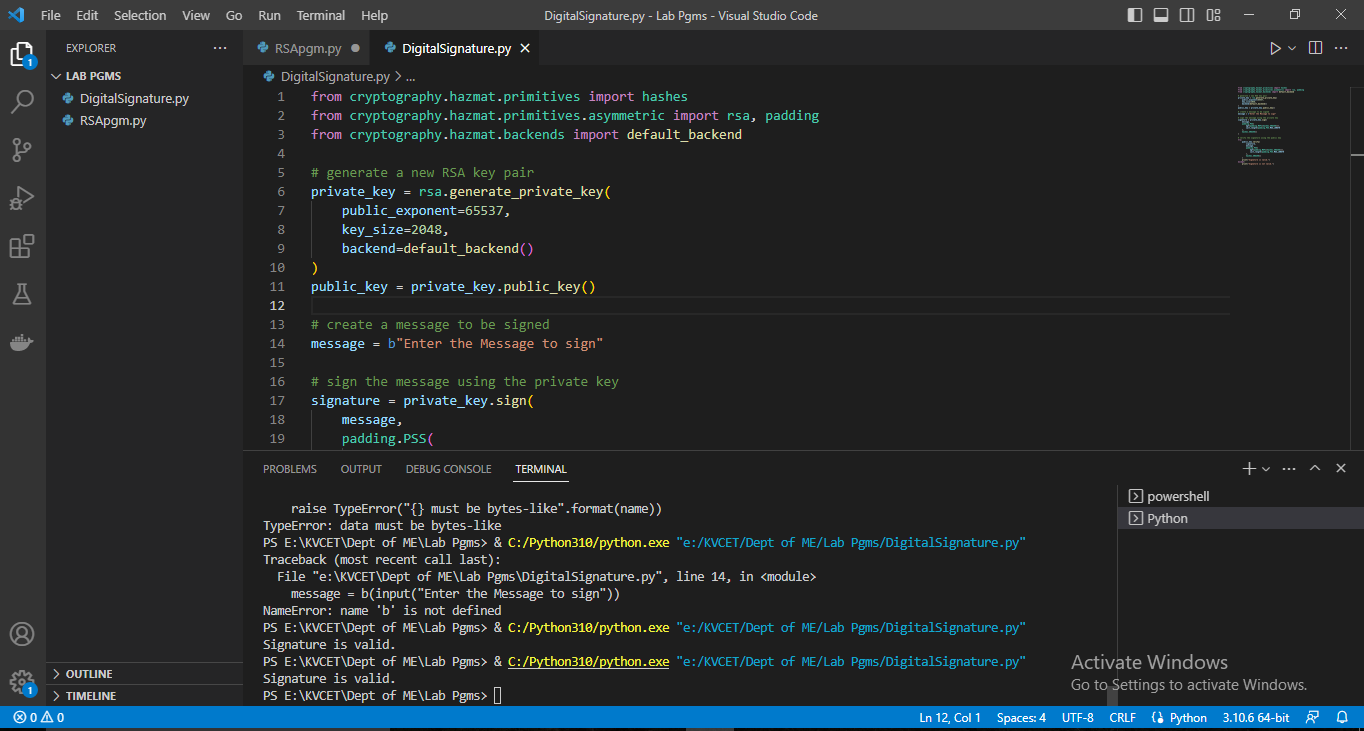
hashes.SHA256()

)

print("Signature is valid.")

except:

print("Signature is not valid.")



3. Implementation of tracing the digital theft in cyberspace

import socket

import pcapy

from struct import \*

# set the network interface and filter

dev = "eth0"

filter = "tcp"

# open the network interface for capture

cap = pcapy.open\_live(dev, 65536, 1, 0)

# set the filter

cap.setfilter(filter)

# process packets in a loop

while True:

try:

# read a packet from the network interface

(header, packet) = cap.next()

# extract the IP header from the packet

ip\_header = packet[0:20]

iph = unpack('!BBHHHBBH4s4s', ip\_header)

# extract the TCP header from the packet

tcp\_header = packet[iph[0] \* 4:iph[0] \* 4 + 20]

tcph = unpack('!HHLLBBHHH', tcp\_header)

# get the source and destination IP addresses and ports

source\_ip = socket.inet\_ntoa(iph[8])

dest\_ip = socket.inet\_ntoa(iph[9])

source\_port = tcph[0]

dest\_port = tcph[1]

# check if the packet is an HTTP request

if dest\_port == 80:

data = packet[iph[0] \* 4 + tcph[4] \* 4:]

if b"POST" in data:

print("[+] Possible digital theft detected!")

print(f" Source IP: {source\_ip}")

print(f" Source port: {source\_port}")

print(f" Destination IP: {dest\_ip}")

print(f" Destination port: {dest\_port}")

print(f" Data: {data}")

except KeyboardInterrupt:

break

3.b

import java.io.IOException;

import java.util.List;

import org.sleuthkit.autopsy.casemodule.Case;

import org.sleuthkit.autopsy.casemodule.CaseDbConnection;

import org.sleuthkit.autopsy.casemodule.services.ServiceUtils;

import org.sleuthkit.autopsy.datamodel.ContentUtils;

import org.sleuthkit.autopsy.datamodel.File;

import org.sleuthkit.autopsy.ingest.IngestModuleIngestJobSettings;

import org.sleuthkit.autopsy.ingest.IngestModuleIngestJobSettingsImpl;

import org.sleuthkit.autopsy.ingest.IngestModuleInitContext;

import org.sleuthkit.autopsy.ingest.IngestModuleOutgoingResults;

import org.sleuthkit.autopsy.ingest.IngestModuleProcessingException;

import org.sleuthkit.datamodel.TskException;

public class DigitalTheftTracing {

public static void main(String[] args) {

// connect to the case database

CaseDbConnection caseDbConnection = Case.getCurrentCase().getDbConnection();

try {

// search for files related to the digital theft

List<File> files = ContentUtils.findFilesContainingString("digital theft", caseDbConnection);

// extract relevant data from the files

for (File file : files) {

byte[] data = ServiceUtils.readBytes(file);

String extractedData = extractData(data);

System.out.println(extractedData);

}

} catch (IOException | TskException e) {

e.printStackTrace();

}

}

private static String extractData(byte[] data) {

// implement code to extract relevant data from the digital evidence

return "Extracted data: " + new String(data);

}

}

4.

import numpy as np

import pyldpc

# create a Hamming code with a code rate of 1/2

n = 7

d = 3

hamming\_code = pyldpc.make\_ldpc(n, d, systematic=True, sparse=True, parity\_check\_matrix='hamming', rng=None)

# generate a random message of length n

message = np.random.randint(2, size=n)

# encode the message using the Hamming code

encoded\_message = pyldpc.encode(hamming\_code, message)

# simulate transmission errors by flipping some of the bits

error\_pattern = np.random.randint(2, size=n)

received\_message = (encoded\_message + error\_pattern) % 2

# decode the received message using the Hamming code

decoded\_message, \_ = pyldpc.decode(hamming\_code, received\_message)

# print the original message, the encoded message, the received message, and the decoded message

print("Original message: ", message)

print("Encoded message: ", encoded\_message)

print("Received message: ", received\_message)

print("Decoded message: ", decoded\_message)

7.

Steganalysis is the process of detecting hidden messages in digital media, such as images, audio, and video files. Target steganalysis refers to the detection of hidden messages in a specific type of digital media, such as a particular image format.

Here is an example of how to perform target steganalysis on images using Python:

First, you need to install the necessary Python libraries for image processing and steganalysis. You can use the following commands to install them:

python

Copy code

pip install Pillow

pip install stegano

Next, you need to import the required libraries:

python

Copy code

from PIL import Image

from stegano import lsb

Load the image file that you want to analyze:

python

Copy code

image = Image.open("image.png")

Extract the hidden message using the Least Significant Bit (LSB) method:

python

Copy code

message = lsb.reveal(image)

Print the extracted message:

python

Copy code

print("Hidden message: " + message)

Here is the complete code:

python

Copy code

from PIL import Image

from stegano import lsb

# Load the image file

image = Image.open("image.png")

# Extract the hidden message

message = lsb.reveal(image)

# Print the extracted message

print("Hidden message: " + message)

Note that this code assumes that the image file "image.png" contains a hidden message using the LSB method. If the image does not contain a hidden message, the code will return an empty string.

You can modify this code to detect hidden messages using other steganography methods, or to analyze other types of digital media such as audio or video files.

7.

Data hiding, also known as steganography, is the process of hiding data within other data to make it more difficult to detect. One common application of steganography is hiding data within digital images.

Here is an example of how to hide data in different types of images using Python:

First, you need to install the necessary Python libraries for image processing and steganography. You can use the following commands to install them:

python

Copy code

pip install Pillow

pip install stegano

Next, you need to import the required libraries:

python

Copy code

from PIL import Image

from stegano import lsb

Load the image file that you want to hide data in:

python

Copy code

image = Image.open("image.png")

Hide the data using the Least Significant Bit (LSB) method:

python

Copy code

message = "This is a hidden message"

stego\_image = lsb.hide(image, message)

Save the stego image to a new file:

python

Copy code

stego\_image.save("stego\_image.png")

Repeat the process with different image types, such as JPEG, BMP, and GIF:

python

Copy code

jpeg\_image = Image.open("image.jpeg")

stego\_jpeg = lsb.hide(jpeg\_image, message)

stego\_jpeg.save("stego\_image.jpeg")

bmp\_image = Image.open("image.bmp")

stego\_bmp = lsb.hide(bmp\_image, message)

stego\_bmp.save("stego\_image.bmp")

gif\_image = Image.open("image.gif")

stego\_gif = lsb.hide(gif\_image, message)

stego\_gif.save("stego\_image.gif")

Note that the LSB method is just one way to hide data in images, and there are many other steganography methods that can be used. Additionally, some image formats may be better suited for steganography than others, due to factors such as compression algorithms and color depth.

Also note that data hiding can be used for both legitimate purposes, such as protecting sensitive information, and malicious purposes, such as hiding malware or illegal content. It is important to use steganography responsibly and ethically.

8.

pip install Pillow

pip install stegano