**Practical 1**

**Aim: Perform a practical to demonstrate ping of death (Denial of**

**service) attack in ubuntu machine.**

**Ping of death:**

A ping of death attack is a type of denial-of-service (DoS) attack. It occurs when attackers overload a computer, service, or system with oversized data packets and Internet Control Message Protocol (ICMP) ping messages.

The ping of death takes advantage of this and sends data packets above

the maximum limit (65,536 bytes) that TCP/IP allows. TCP/IP

fragmentation breaks the packets into small chunks that are sent to the

server. Since the sent data packages are larger than what the server can

handle, the server can freeze, reboot, or crash.

**Command:** ping

The Linux ping command is a simple utility used to check whether a network is available and if a host is reachable. With this command, you can test if a server is up and running. It also helps with troubleshooting various connectivity issues.

The ping command allows you to:

1. Test your internet connection.
2. Check if a remote machine is online.
3. Analyze if there are network issues, such as dropped packages or high latency.

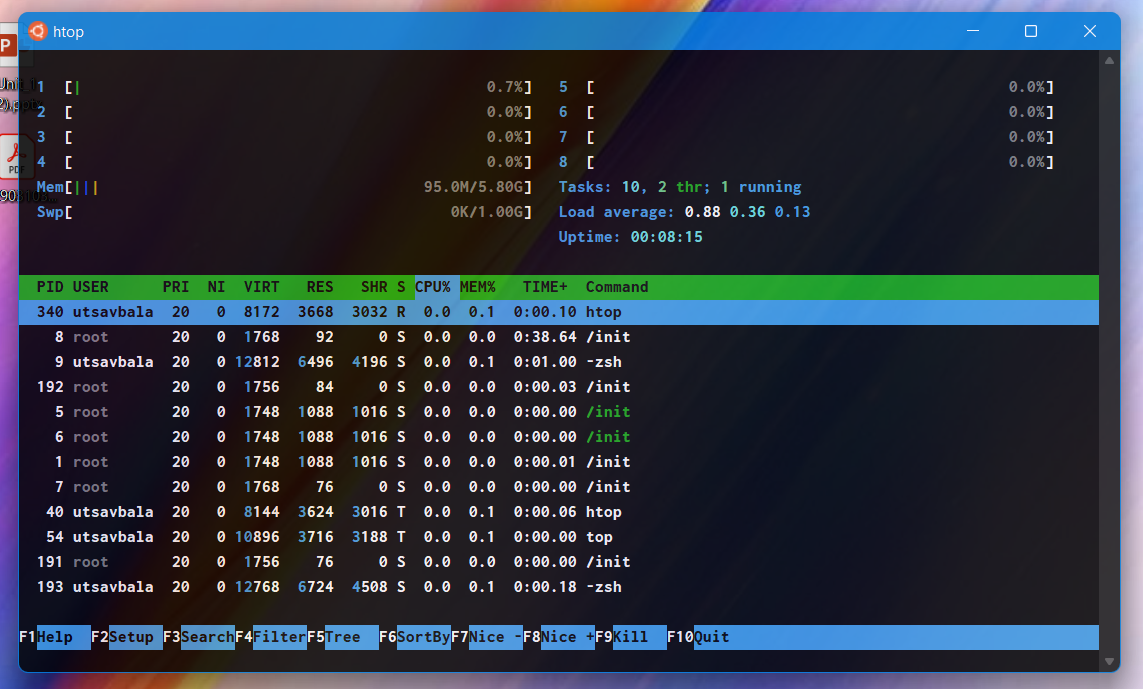
**Launching Ping of death attack.**

For this attack I am using my WSL2 Ubuntu terminal

Make sure you have **htop** installed in your system.

**htop:** It is a free interactive process viewer

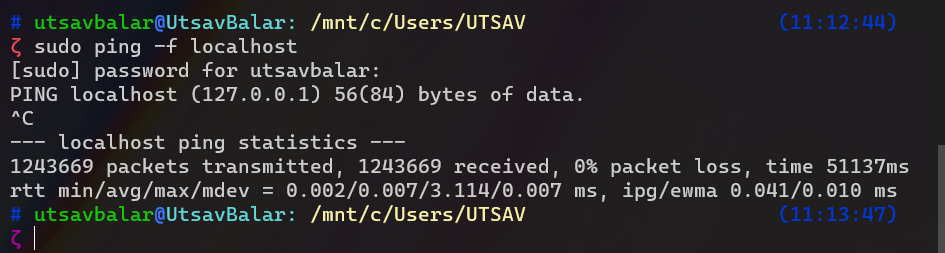
**Command:** htop



**Now lets perform ping of death attack**

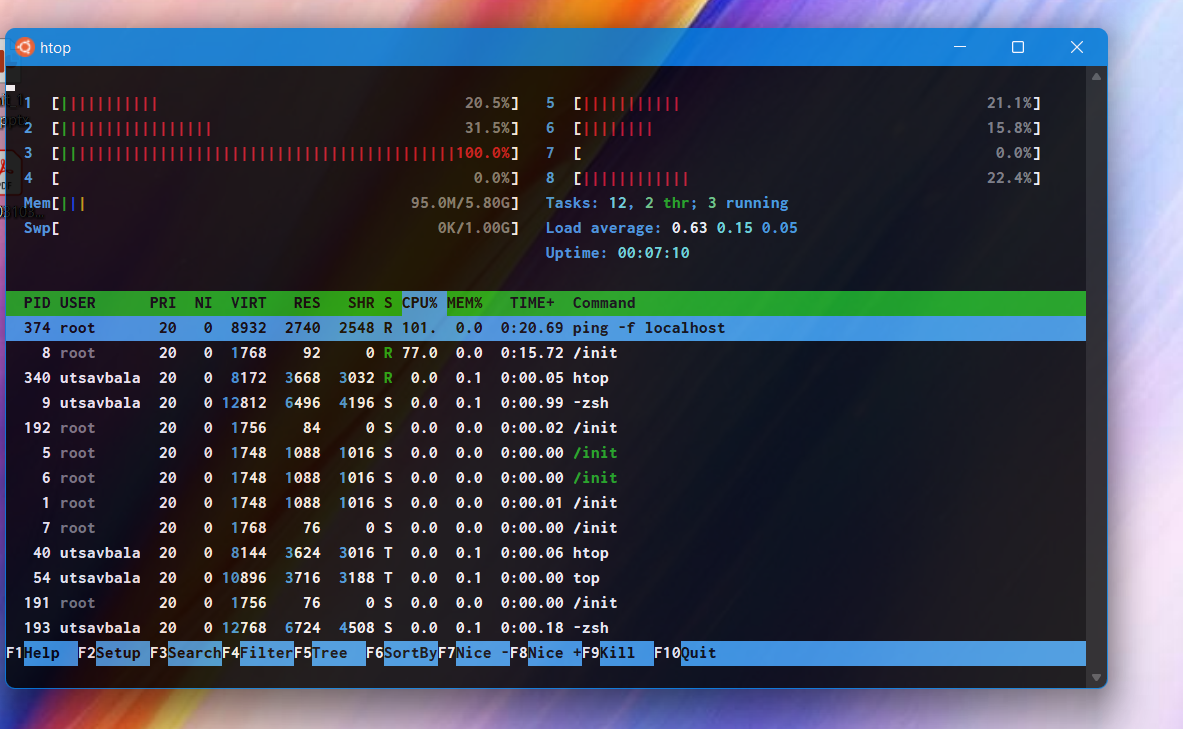
**Command:** sudo ping -f localhost

-f : Used for ping flood (to test your network performance under heavy load.)



**Output:**

Now **htop** while **sudo ping -f localhost** is running



**Practical 2**

**Aim: Perform a practical to install network mapper tool and analyze the**

**open ports in your Ubuntu machine.**

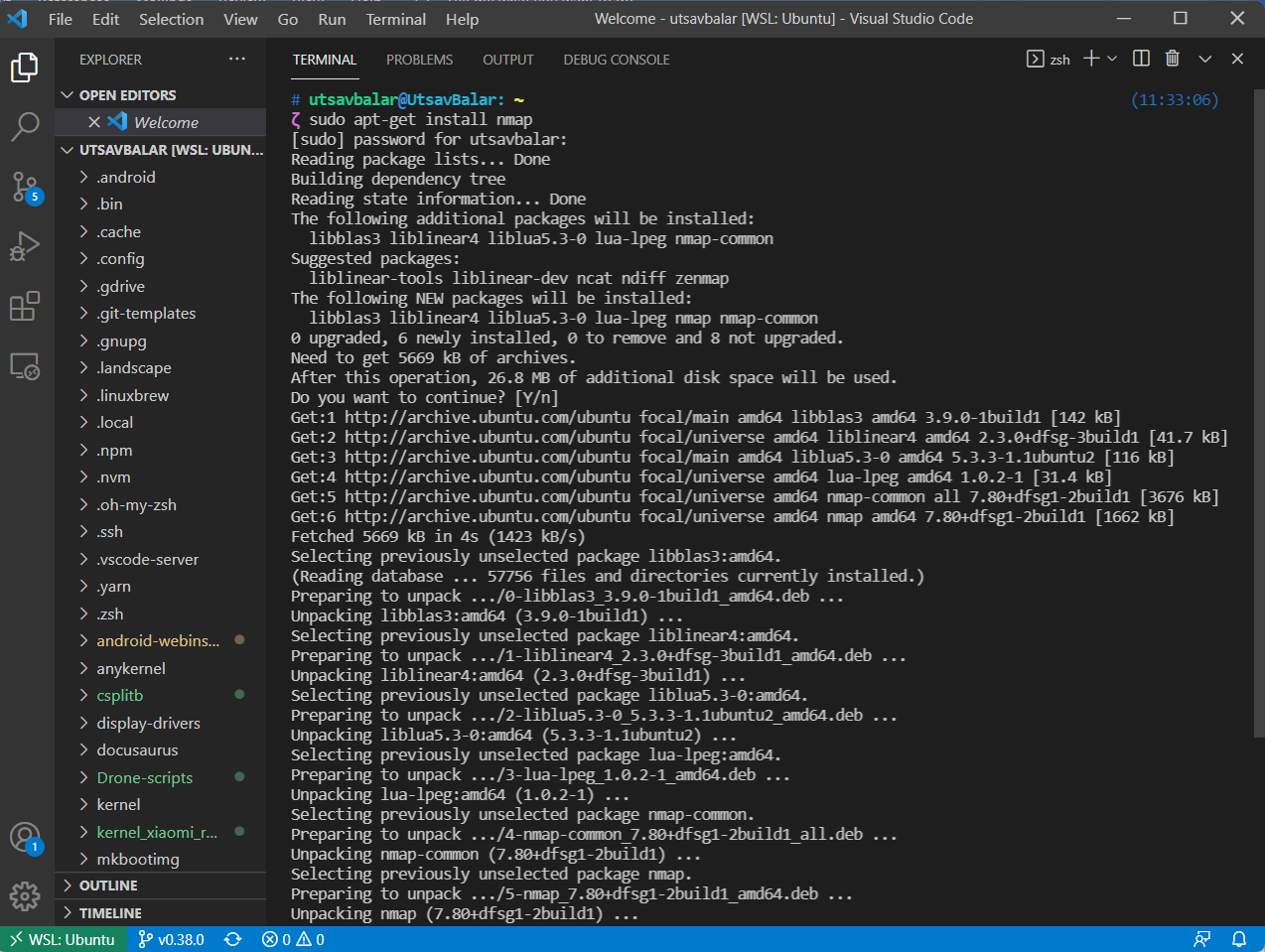
**Run a script to close all the insecure port, reopen and demonstrate.**

**Install n-map tool :**

Network Mapper (Nmap) is a free, open-source network security scanning tool. By sending IP packets and analyzing the responses, Nmap can discover information about hosts and services on remote computer networks.

Nmap can also audit the security of a device, identify the vulnerabilities of your network, or perform an inventory check with relative ease.

sudo apt-get install nmap



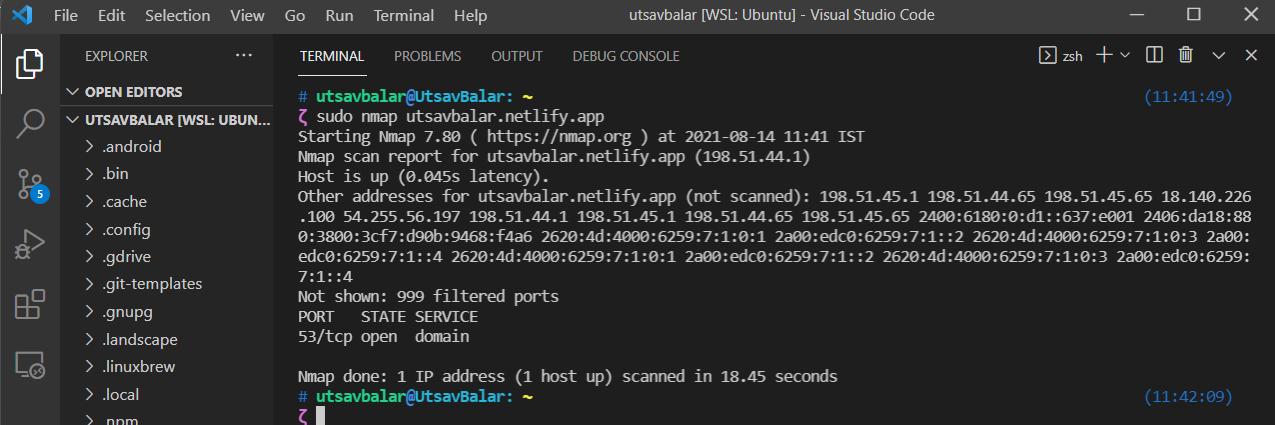
**Analysis of the open port :**

To scan all open/listening ports in your Linux system, run the following command

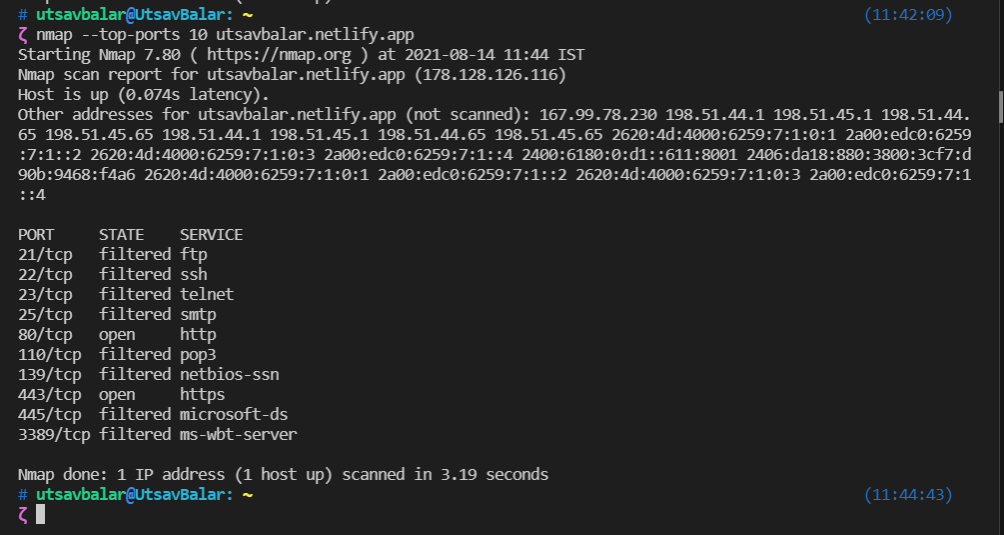
**Scan IP range or subnet**

To obtain general information of a remote system type:

sudo nmap target\_IP or domain.com

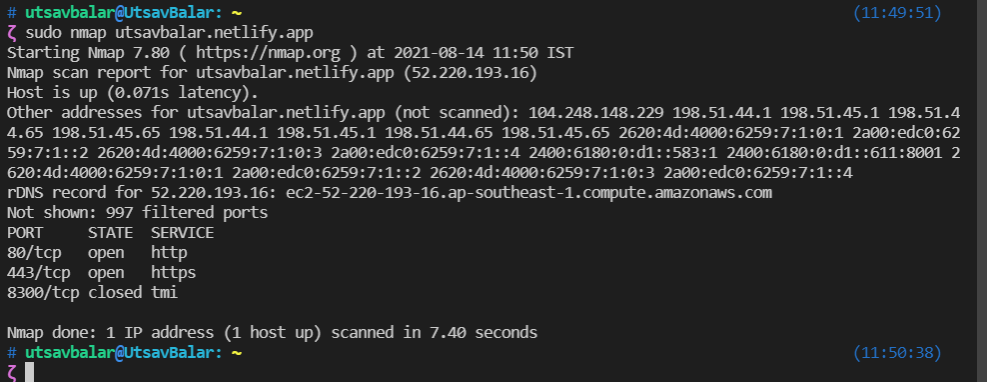


**Close all the insecure port :**



**Reopen ports :**





**Practical 3**

**Aim: Perform a practical to implement Caesar cipher and play fair**

**cipher.**

**Caesar cipher Code :**

def encrypt(plaintext, key):

encryption = ""

for text in plaintext:

if text.isupper():

text\_index = ord(text) - ord('A')

# Shift the letters (substitution)

new\_chr = chr((text\_index + key) % 26 + ord('A'))

# append new\_chr to encryption string

encryption += new\_chr

elif text.islower():

text\_index = ord(text) - ord('a')

# Shift the letters (substitution)

new\_chr = chr((text\_index + key) % 26 + ord('a'))

# append new\_chr to encryption string

encryption += new\_chr

elif text.isdigit():

# Shift the digits (substitution)

new\_chr = (int(text) + key) % 10

# append new\_chr to encryption string

encryption += str(new\_chr)

else:

encryption += text

return encryption

def decrypt(plaintext, key):

decryption = ""

for text in plaintext:

if text.isupper():

text\_index = ord(text) - ord('A')

# Shift the letters (substitution)

new\_chr = chr((text\_index - key) % 26 + ord('A'))

# append new\_chr to encryption string

decryption += new\_chr

elif text.islower():

text\_index = ord(text) - ord('a')

# Shift the letters (substitution)

new\_chr = chr((text\_index - key) % 26 + ord('a'))

# append new\_chr to encryption string

decryption += new\_chr

elif text.isdigit():

# Shift the digits (substitution)

new\_chr = (int(text) - key) % 10

# append new\_chr to encryption string

decryption += str(new\_chr)

else:

decryption += text

return decryption

key = 3

plaintext = input("Enter plain text for encryption: ")

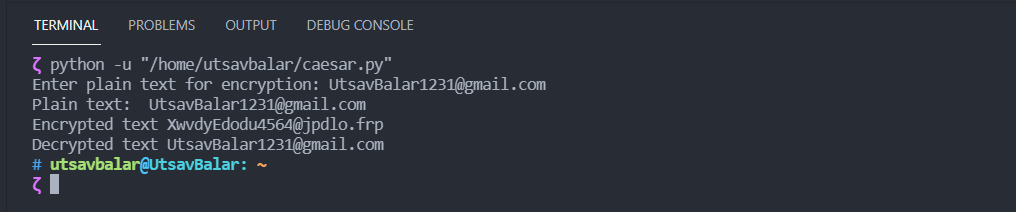
print("Plain text: ", plaintext)

ciphertext = encrypt(plaintext, key)

print("Encrypted text", ciphertext)

print("Decrypted text", decrypt(ciphertext, key))

**Output:**



**How did I implemented this?**

Firstly Caesar cipher algorithm is

Encryption:

C = (P + K) % 26

Decryption:

P = (C - K) % 26

P = plaintext

K = key or shift value

C = ciphertext

In the code

ord() is used to convert each letter into its numeric Unicode value

chr() is used to convert Unicode value back to alphabetic/character letter

So for encryption take a string from user

Check if its uppercase, lowercase or a digit

And perform the substitution letter by letter checking every time whether it’s an upper, lowercase or a digit

And return the encrypted text as output of encrypt() function

For decryption

take the same output from encrypt() and with the decryption logic of Caesar cipher return the decrypted text.

**Playfair cipher Code:**

from pprint import pprint

def matrix(key):

matrix = []

# add unique key elements

for e in key:

if e.upper() not in matrix:

matrix.append(e.upper())

letters = "ABCDEFGHIKLMNOPQRSTUVWXYZ"

# add remaining alphabets

for e in letters:

if e.upper() not in matrix:

matrix.append(e.upper())

# split 25 elements into 5x5 matrix

return [matrix[i:i + 5] for i in range(0, 25, 5)]

def messages(user\_message):

# convert string to list with all capital letters

message = list(map(str.upper, user\_message))

# remove spaces

message = list(filter(lambda x: x != ' ', message))

# add 'X' between duplicates if first one is at odd index

for i in range(len(message)):

if message[i-1] == message[i] and i % 2:

message.insert(i, 'X')

# append 'X' if list has odd number of elements to make valid pairs

if len(message) % 2 == 1:

message.append('X')

# make pairs from even number of elements

return [message[i:i+2] for i in range(0, len(message), 2)]

def position(matrixk, letter):

for i in range(5):

for j in range(5):

if matrixk[i][j] == letter:

return i, j

def encrypt(user\_message, key):

message = messages(user\_message)

matrixk = matrix(key)

cipher = []

# get indices of pairs

for a, b in message:

x1, y1 = position(matrixk, a)

x2, y2 = position(matrixk, b)

# Elements in same row

if x1 == x2:

# if y reached one end (4), %5 will make (4+1)%5 = 0

cipher.append(matrixk[x1][(y1 + 1) % 5])

cipher.append(matrixk[x1][(y2 + 1) % 5])

# Elements in same column

elif y1 == y2:

# if x reached one end (4), %5 will make (4+1)%5 = 0

cipher.append(matrixk[(x1 + 1) % 5][y1])

cipher.append(matrixk[(x2 + 1) % 5][y2])

# Elements make a box so use remaining corners

else:

cipher.append(matrixk[x1][y2])

cipher.append(matrixk[x2][y1])

# convert list to string

return ''.join(cipher)

def decrypt(user\_cipher, key):

# generate pairs from cipher

cipher = [user\_cipher[i:i+2] for i in range(0, len(user\_cipher), 2)]

matrixk = matrix(key)

plaintext = []

for a, b in cipher:

x1, y1 = position(matrixk, a)

x2, y2 = position(matrixk, b)

# Elements in same row

if x1 == x2:

# if y reached one end (0), %5 will make (0-1)%5 = 4

plaintext.append(matrixk[x1][(y1 - 1) % 5])

plaintext.append(matrixk[x1][(y2 - 1) % 5])

# Elements in same column

elif y1 == y2:

# if x reached one end (0), %5 will make (0-1)%5 = 4

plaintext.append(matrixk[(x1 - 1) % 5][y1])

plaintext.append(matrixk[(x2 - 1) % 5][y2])

# Elements make a box so use remaining corners

else:

plaintext.append(matrixk[x1][y2])

plaintext.append(matrixk[x2][y1])

# convert list to string

return ''.join(plaintext)

key = input('Please input the key: ')

plaintext = input('Please input the message: ')

cipher = encrypt(plaintext, key)

decipher = decrypt(cipher, key)

print("Plaintext Matrix: ", messages(plaintext))

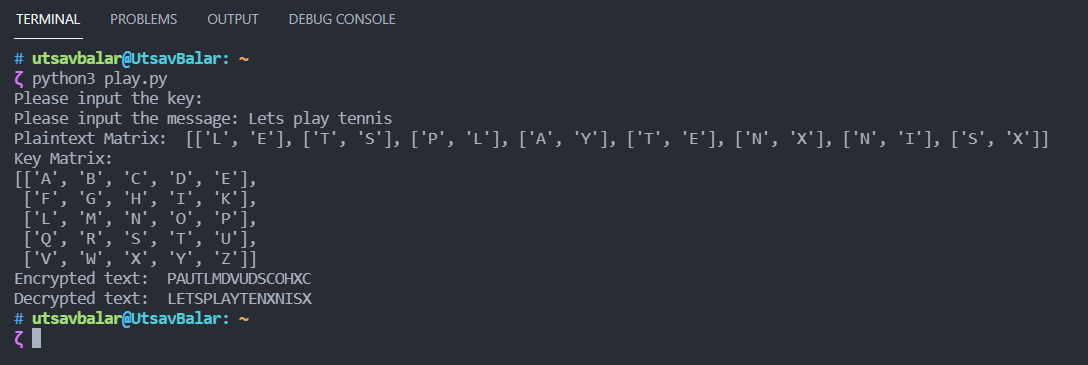
print("Key Matrix:")

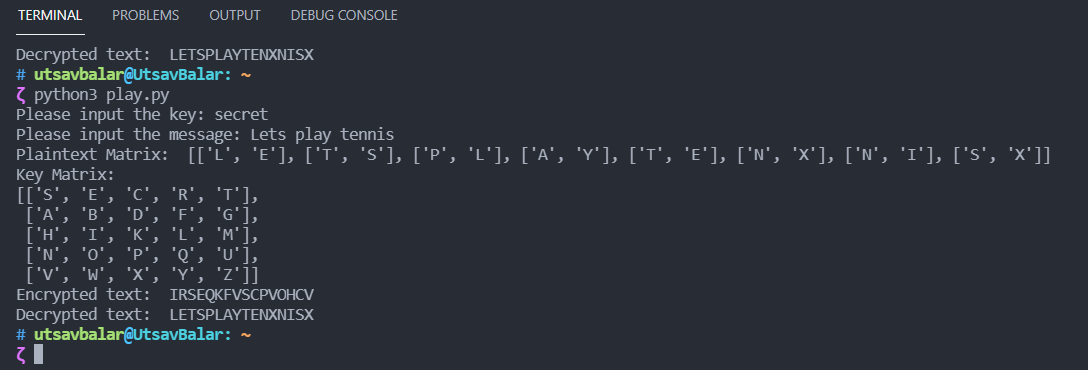
pprint(matrix(key))

print("Encrypted text: ", cipher)

print("Decrypted text: ", decipher)

**Output:**

****

****

**Practical 7: Implement S-DES encryption and decryption algorithm**

**Code:**

\_init\_permutation = (2, 6, 3, 1, 4, 8, 5, 7)

\_expP = (4, 1, 2, 3, 2, 3, 4, 1)

\_init\_permutation\_invr = (4, 1, 3, 5, 7, 2, 8, 6)

\_p10 = (3, 5, 2, 7, 4, 10, 1, 9, 8, 6)

\_p8 = (6, 3, 7, 4, 8, 5, 10, 9)

\_p4 = (2, 4, 3, 1)

s0 = ((1, 0, 3, 2),

(3, 2, 1, 0),

(0, 2, 1, 3),

(3, 1, 3, 2))

s1 = ((0, 1, 2, 3),

(2, 0, 1, 3),

(3, 0, 1, 0),

(2, 1, 0, 3))

key = '0111111101'

def permutate(original, fixed\_key):

new = ''

for i in fixed\_key:

new += original[i - 1]

return new

def left\_half(bits):

return bits[:len(bits)//2]

def right\_half(bits):

return bits[len(bits)//2:]

def shift(bits):

shift\_left\_half = left\_half(bits)[1:] + left\_half(bits)[0]

shift\_right\_half = right\_half(bits)[1:] + right\_half(bits)[0]

return shift\_left\_half + shift\_right\_half

def key1():

return permutate(shift(permutate(key, \_p10)), \_p8)

def key2():

return permutate(shift(shift(shift(permutate(key, \_p10)))), \_p8)

def xor(bits, key):

new = ''

for bit, key\_bit in zip(bits, key):

new += str(((int(bit) + int(key\_bit)) % 2))

return new

def lookup\_in\_sbox(bits, sbox):

row = int(bits[0] + bits[3], 2)

col = int(bits[1] + bits[2], 2)

return '{0:02b}'.format(sbox[row][col])

def f\_k(bits, key):

L = left\_half(bits)

R = right\_half(bits)

bits = permutate(R, \_expP)

bits = xor(bits, key)

bits = lookup\_in\_sbox(left\_half(bits), s0) + \

lookup\_in\_sbox(right\_half(bits), s1)

bits = permutate(bits, \_p4)

return xor(bits, L)

def encrypt(plain\_text):

bits = permutate(plain\_text, \_init\_permutation)

temp = f\_k(bits, key1())

bits = right\_half(bits) + temp

bits = f\_k(bits, key2())

return permutate(bits + temp, \_init\_permutation\_invr)

def decrypt(cipher\_text):

bits = permutate(cipher\_text, \_init\_permutation)

temp = f\_k(bits, key2())

bits = right\_half(bits) + temp

bits = f\_k(bits, key1())

return permutate(bits + temp, \_init\_permutation\_invr)

choice = int(input("Enter\n1 = Encryption\n0 = Decryption\n"))

if (choice == 1):

plaintext = input("Enter 8bit plaintext: ")

while (len(plaintext) != 8):

plaintext = input("Invalid bit lenght! Enter again: ")

ciphertext = encrypt(plaintext)

print("Plaintext: {}\nCiphertext: {}". format(plaintext, ciphertext))

elif (choice == 0):

ciphertext = input("Enter 8bit plaintext: ")

while (len(ciphertext) != 8):

ciphertext = input("Invalid bit lenght! Enter again: ")

plaintext = decrypt(ciphertext)

print("Ciphertext: {}\nPlaintext: {}".format(ciphertext, plaintext))

else:

print("Invalid input")

choice = int(input("Enter\n1 = encryption\n0 = decryption: "))

**Output (Encryption):**

**Text

Description automatically generated**

**Output (Decryption):**

**Text

Description automatically generated**

**Practical8: Implement RSA as encryption algorithm**

**Code:**

def gcd(a, b):

if b == 0:

return a

else:

return gcd(b, a % b)

def phi(p, q):

# Totient Function

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

return phi\_n

def generate\_keys(p, q):

phi\_n = phi(p, q)

n = p \* q

e = 2

"""Selection of e: such that 1 < e < phi(n) && gcd(e, phi(n)) == 1"""

while (e > 1 and e < phi\_n):

if gcd(e, phi\_n) == 1:

break

else:

e = e + 1

"""Calculation of d: such that d \* e mod phi(n) == 1"""

d = 1

while(((d \* e) % phi\_n) != 1):

d = d + 1

return [(e, n), (d, n)]

def encrypt(p, q, M):

"""Encryption: C = M^e mod n"""

pub\_key, priv\_key = generate\_keys(p, q)

e, n = pub\_key

d, n = priv\_key

print('Public Key: ', pub\_key)

print('Private Key: ', priv\_key)

return ((M \*\* e) % n)

def decrypt(p, q, C):

"""Decryption: M = C^d mod n"""

pub\_key, priv\_key = generate\_keys(p, q)

e, n = pub\_key

d, n = priv\_key

print('Public Key: ', pub\_key)

print('Private Key: ', priv\_key)

return ((C \*\* d) % n)

p = int(input("Enter value of p: "))

q = int(input("Enter value of q: "))

choice = int(input("Enter\n1 = encryption\n0 = decryption\n"))

if (choice == 1):

msg = int(input("Enter plaintext: "))

print("Ciphertext: ", encrypt(p, q, msg))

elif (choice == 0):

msg = int(input("Enter cipthertext: "))

print("Plaintext: ", decrypt(p, q, msg))

else:

print("Invalid input")

choice = int(input("Enter\n1 = encryption\n0 = decryption: "))

**Output: Encryption**

Text

Description automatically generated

**Output: Decryption**

**Text

Description automatically generated**

**Practical 9: Implement Diffie-Hellman Key exchange algorithm.**

**Code:**

q = int(input("Enter Value of q: "))

Xa = int(input("Enter Private key of Alice: "))

Xb = int(input("Enter Private key of Bob: "))

a = int(input("Enter Alpha (Primitive root): "))

while (a > q):

print(f"Alpha cannot be greater than {q}")

a = int(input("Enter alpha again: "))

print('The Value of Xa is:', Xa)

print('The Value of Xb is:', Xb)

Ya = int(a \*\* Xa % q)

Yb = int(a \*\* Xb % q)

ka = int(Yb \*\* Xa % q)

kb = int(Ya \*\* Xb % q)

print('Secret key for the Alice is: ', ka)

print('Secret Key for the Bob is:', kb)

**Output:**

**Text

Description automatically generated**