**Practical: 1**

**Aim: Perform a practical to demonstrate ping of death (Denial of Service) attack in Ubuntu machine.**

**What Is a Ping of Death Attack?**

The ping of death is a form of denial-of-service (DoS) attack that occurs when an attacker crashes, destabilizes, or freezes computers or services by targeting them with oversized data packets. This form of DoS attack typically targets and exploits legacy weaknesses that organizations may have patched.

**How Does the Ping of Death Work?**

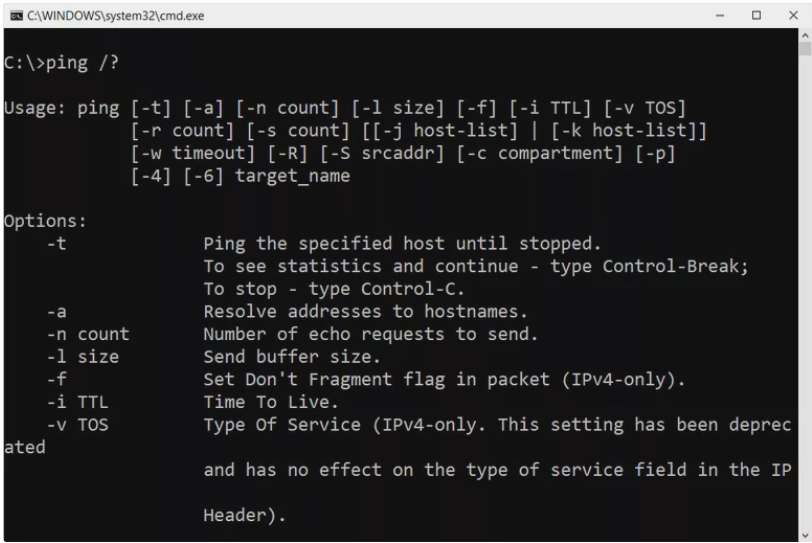
A correct Internet Protocol version 4 (IPv4) packet is formed of 65,535 bytes, and most legacy computers cannot handle larger packets. Sending a ping larger than this violates the IP, so attackers send packets in fragments which, when the targeted system attempts to reassemble, results in an oversized packet that can cause the system to crash, freeze, or reboot.

The vulnerability can be exploited by any source that sends IP datagrams, which include an ICMP echo, the Internetwork Packet Exchange (IPX), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP).

**Ping Command**

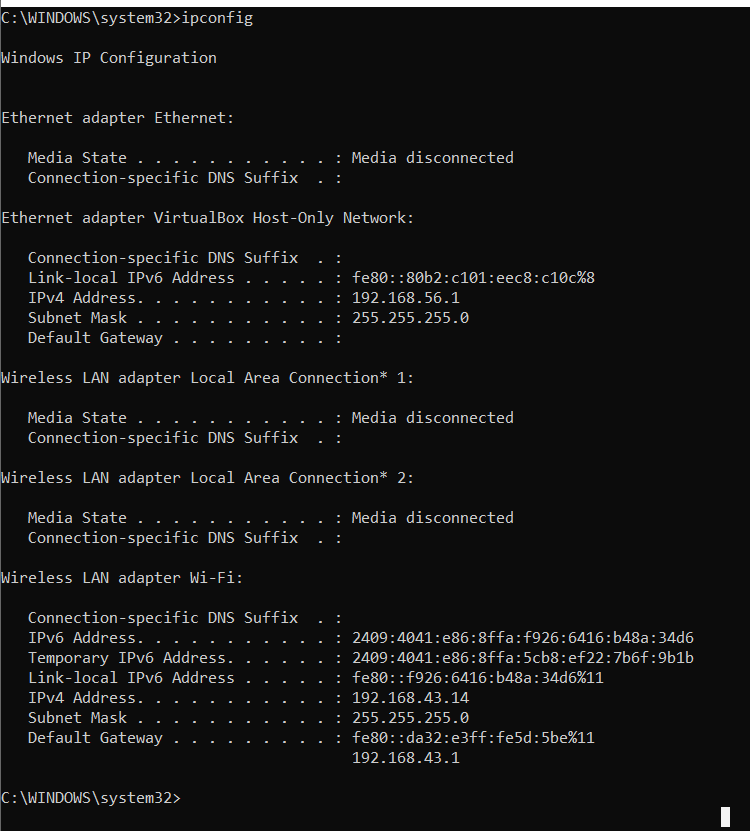
The ping command is a Command Prompt command used to test the ability of the source computer to reach a specified destination computer. It's usually used as a simple way to verify that a computer can communicate over the network with another computer or network device.

**Ping Command Availability**

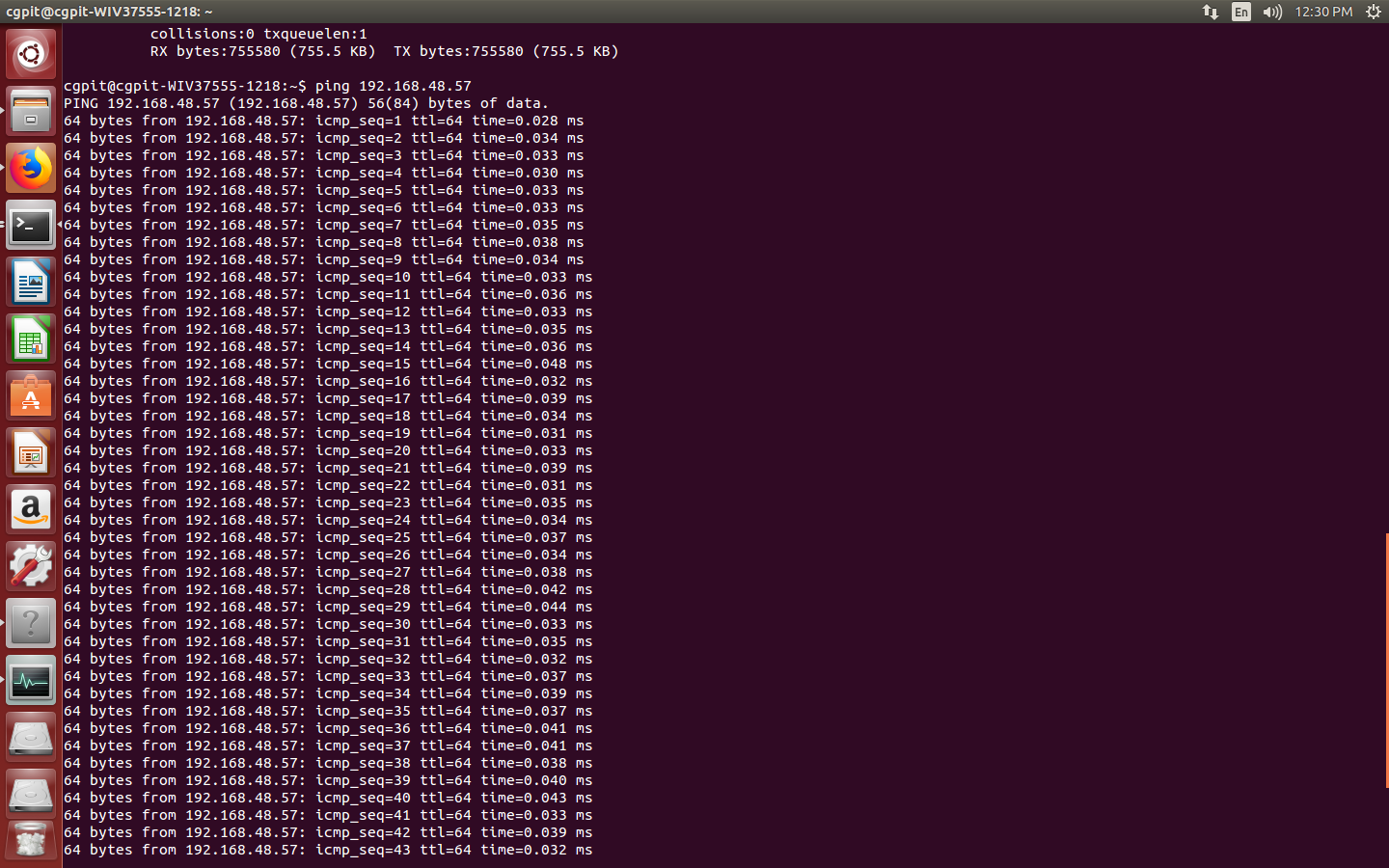


**Syntax:**

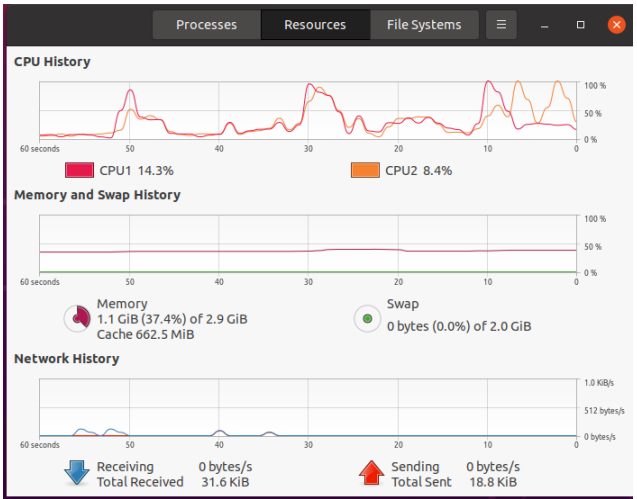
**ping** [**-t**] [**-a**] [**-n** *count*] [**-l** *size*] [**-f**] [**-i** *TTL*] [**-v** *TOS*] [**-r** *count*] [**-s** *count*] [**-w** *timeout*] [**-R**] [**-S** *srcaddr*] [**-p**] [**-4**] [**-6**] *target* [**/?**]

**How to perform Dos attack?**

**Send Ping:**

****

**System monitor in Ubuntu:**



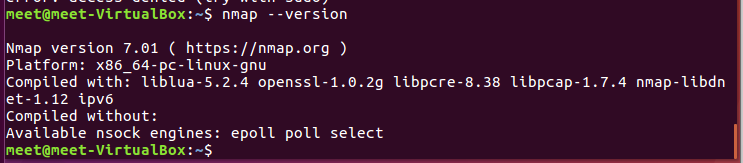
**Practical: 2**

**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

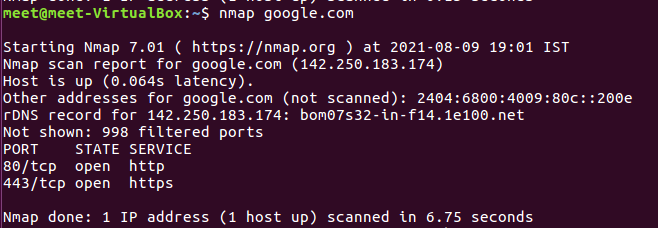
**Install n-map tool:**

Nmap is a powerful network discovery and security auditing utility that is free, open-source, and easy to install. Nmap scans for vulnerabilities on your network, performs inventory checks, and monitors host or service uptime, alongside many other useful features

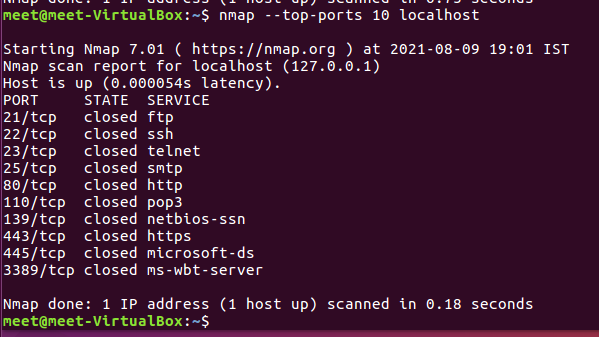


**Analysis of the open port:**

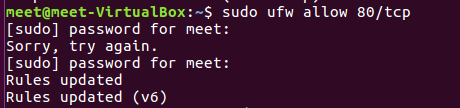
Nmap is a powerful and popular network exploration tool and port scanner. To install nmap on your system, use your default package manager as shown. To scan all open/listening ports in your Linux system, run the following command (which should take a long time to complete)

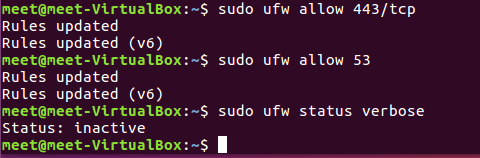


**Close all the insecure port:**

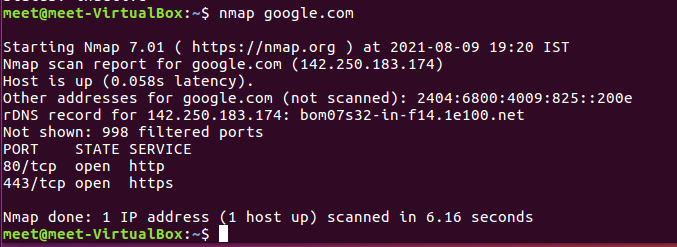


**Re-Open ports:**





**Reopen ports and demonstrate:**



**Practical: 3**

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

**Caesar cipher:**

**Code**:

pt = str(input("Enter the string:"))

key = int(input("Enter the key:"))

ct = ""

dt = ""

print("Original text:", pt)

for letter in pt:

    l = (ord(letter)+key % 26)

    ct += chr(l)

print("Encrytpted text(Cipher text):", ct)

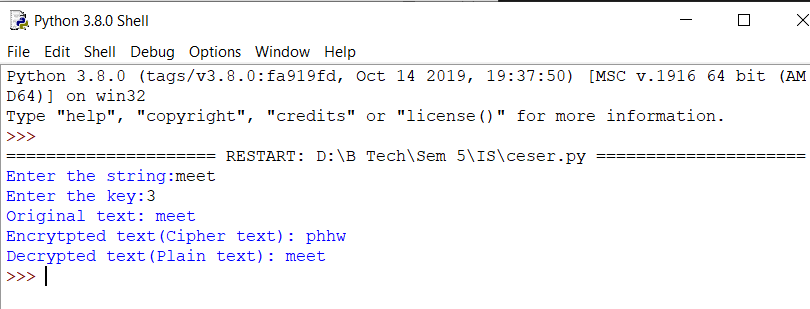
for letter in ct:

    l = (ord(letter)-key % 26)

    dt += chr(l)

print("Decrypted text(Plain text):", dt)

**Output**:



**Play fair cipher:**

**Code**:

key = input("Enter the key:")

plain\_text = input("Enter the pain text:")

if len(key) > 0 and len(plain\_text) > 0:

    plain\_text = plain\_text.replace(" ", "").lower()

    group = list()

    matrix = list()

    original\_5\_5\_matrix = list()

    list1 = list()

    # check key for value plain text

    for a in key.lower():

        if a not in matrix:

            matrix.append(a)

    # check character i and j an pattern set and check all alphabet insert but

    # check plain text value key same value

    for char in key:

        if char == 'i':

            alphabet = "abcdefghiklmnopqrstuvwxyz"

        else:

            alphabet = "abcdefghjklmnopqrstuvwxyz"

    for a in alphabet:

        if a not in matrix:

            matrix.append(a)

    print('single single charactor matrix:')

    print(matrix)

    original\_5\_5\_matrix = [matrix[i:i+5] for i in range(0, len(matrix), 5)]

    print(original\_5\_5\_matrix)

    print()

    # same alphabet insert x

    def change\_pt(i, pt):

        pt = pt[:i]+"x"+pt[i:]

        return pt

    # divide pain text 2 character an list all 2 charator inserted for 2 part

    for i in range(0, len(plain\_text), 2):

        if i == len(plain\_text):

            pass

        else:

            if plain\_text[i] == plain\_text[i+1]:

                plain\_text = change\_pt(i+1, plain\_text)

        group.append(plain\_text[i]+plain\_text[i+1])

    print("Divide plain text of 2 character:")

    print(group)

    print()

    print("Cipher text:")

    index\_of\_w1 = None

    index\_of\_w2 = None

    for word in group:

        for i in range(5):

            for j in range(5):

                if word[0] in original\_5\_5\_matrix[i][j]:

                    # first letter index for column and row

                    index\_of\_w1 = [i, j]

                if word[1] in original\_5\_5\_matrix[i][j]:

                    # second letter index for column and row

                    index\_of\_w2 = [i, j]

        if index\_of\_w1 != None and index\_of\_w2 != None:

            # for same row

            if index\_of\_w1[0] == index\_of\_w2[0]:

                if index\_of\_w1[1] == 4:  # first letter : last position of row index

                    print(original\_5\_5\_matrix[index\_of\_w1[0]][0] +

                          original\_5\_5\_matrix[index\_of\_w2[0]][index\_of\_w2[1]+1])

                elif index\_of\_w2[1] == 4:  # second letter : last position of row index

                    print(original\_5\_5\_matrix[index\_of\_w1[0]]

                          [index\_of\_w1[1]+1] + original\_5\_5\_matrix[index\_of\_w2[0]][0])

                else:

                    print(original\_5\_5\_matrix[index\_of\_w1[0]][index\_of\_w1[1]+1] +

                          original\_5\_5\_matrix[index\_of\_w2[0]][index\_of\_w2[1]+1])

            # for same column

            # first letter : last position of column index

            elif index\_of\_w1[1] == index\_of\_w2[1]:

                if index\_of\_w1[0] == 4:

                    print(original\_5\_5\_matrix[0][index\_of\_w1[1]] +

                          original\_5\_5\_matrix[index\_of\_w2[0]+1][index\_of\_w2[1]])

                # second letter : last position of column index

                elif index\_of\_w2[0] == 4:

                    print(original\_5\_5\_matrix[index\_of\_w1[0]+1]

                          [index\_of\_w1[1]] + original\_5\_5\_matrix[0][index\_of\_w2[1]])

                else:

                    print(original\_5\_5\_matrix[index\_of\_w1[0]+1][index\_of\_w1[1]] +

                          original\_5\_5\_matrix[index\_of\_w2[0]+1][index\_of\_w2[1]])

            # otherwise

            else:

                print(original\_5\_5\_matrix[index\_of\_w1[0]][index\_of\_w2[1]] +

                      original\_5\_5\_matrix[index\_of\_w2[0]][index\_of\_w1[1]])

else:

    print("Key and Pain text are required!")

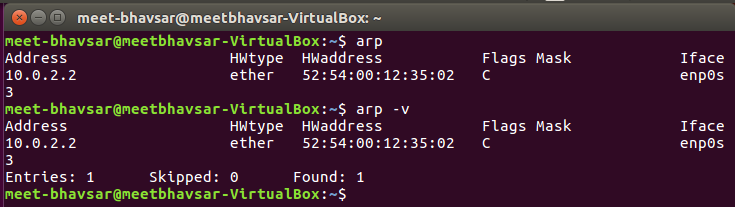
**Output**:



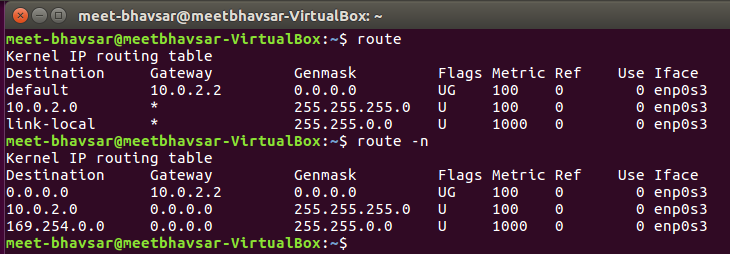
**Practical: 4**

**Aim: Perform a practical to demonstrate important Linux command of information security.**

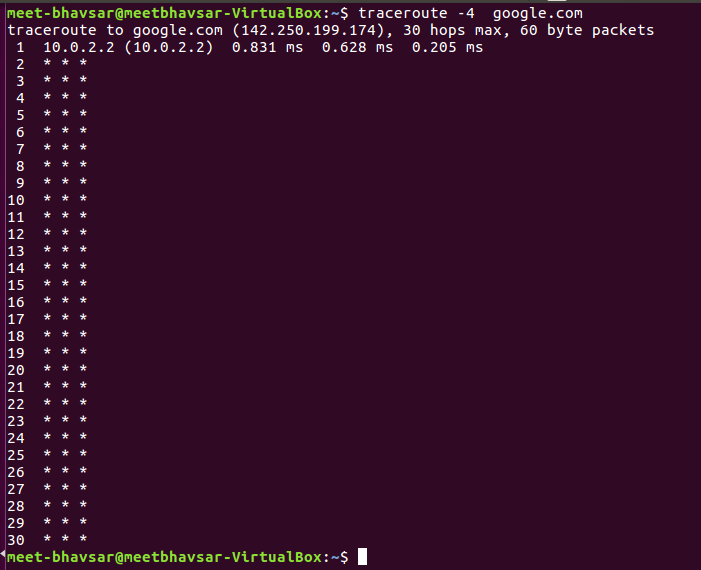
**arp:** arp command manipulates the System's ARP cache. It also allows a complete dump of the ARP cache. ARP stands for Address Resolution Protocol. The primary function of this protocol is to resolve the IP address of a system to its mac address, and hence it works between level 2(Data link layer) and level 3(Network layer).



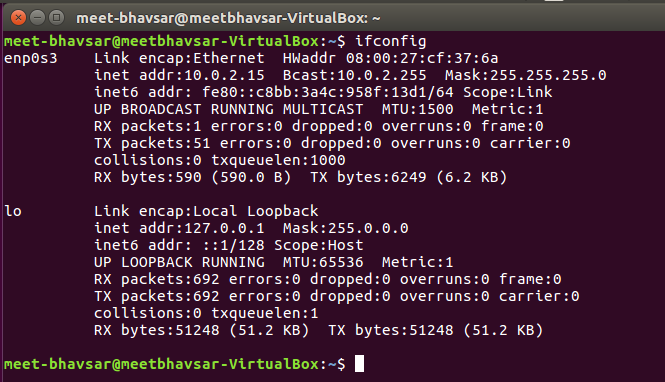
**route:** The route command allows you to make manual entries into the network routing tables. The route command distinguishes between routes to hosts and routes to networks by interpreting the network address of the Destination variable, which can be specified either by symbolic name or numeric address.



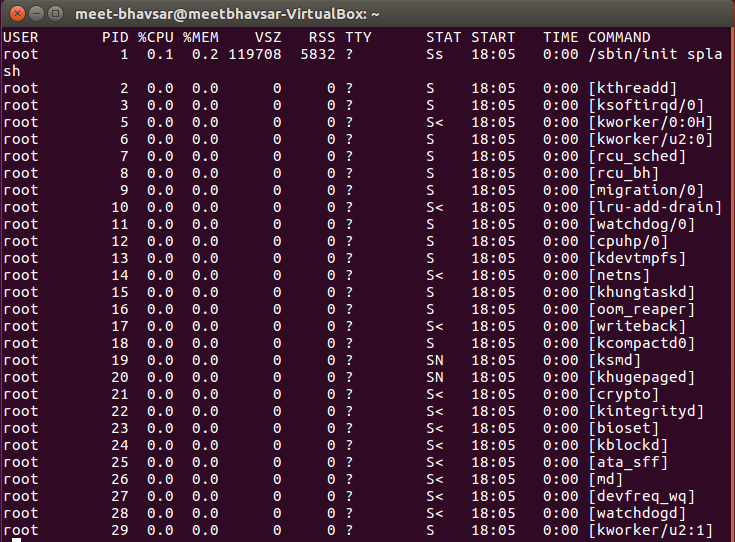
**traceroute:** traceroute command in Linux prints the route that a packet takes to reach the host. This command is useful when you want to know about the route and about all the hops that a packet takes.



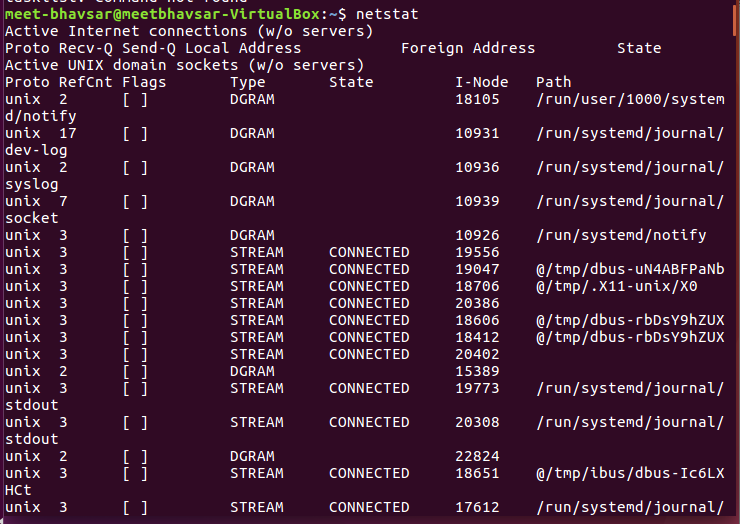
**ifconfig:** You can use the ifconfig command to assign an address to a network interface and to configure or display the current network interface configuration information.



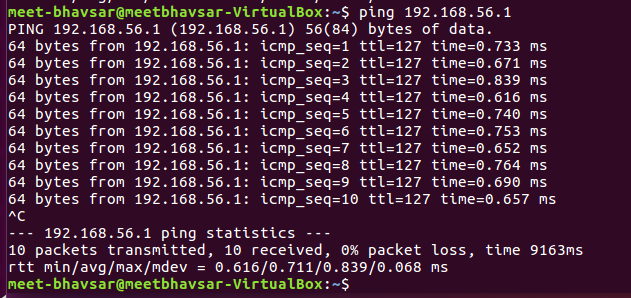
**tasklist:** You can use the tasklist command to display a list of currently-running tasks. tasklist displays the process ID number for each running task, the name of the executable program that started the task, and, when available, the window title.



**netstat:** Netstat command displays various network related information such as network connections, routing tables, interface statistics, masquerade connections, multicast memberships.



**ping:** PING (Packet Internet Groper) command is used to check the network connectivity between host and server/host. This command takes as input the IP address or the URL and sends a data packet to the specified address with the message “PING” and get a response from the server/host this time is recorded which is called latency.



**Practical: 7**

**Aim: Implement S-DES Key generation and Encryption.**

**Code**:

# plain\_text = "01110010"

# key = "1010000010"

plain\_text = str(input("Enter plain text: "))

key = str(input("Enter the key: "))

p10 = [3, 5, 2, 7, 4, 10, 1, 9, 8, 6]

p8 = [6, 3, 7, 4, 8, 5, 10, 9]

p4 = [2, 3, 4, 1]

ip = [2, 6, 3, 1, 4, 8, 5, 7]

ip\_inv = [4, 1, 3, 5, 7, 2, 8, 6]

ep = [4, 1, 2, 3, 2, 3, 4, 1]

s0\_box = [['01', '00', '11', '10'], ['11', '10', '01', '00'],

          ['00', '10', '01', '11'], ['11', '01', '11', '10']]

s1\_box = [['00', '01', '10', '11'], ['10', '00', '01', '11'],

          ['11', '00', '01', '00'], ['10', '01', '00', '11']]

p10\_per, key1, key2, per\_pt, exp\_right\_pt = "", "", "", "", ""

left\_s0, right\_s1, per\_sbox\_output, ip\_inverse = "", "", "", ""

def permutation(per\_table, key):

    a = ""

    for i in per\_table:

        a = a+(key[i-1])

    return a

p10\_per = permutation(p10, key)

print("P10 = "+p10\_per+"\n")

left\_half = p10\_per[:5]

right\_half = p10\_per[5:]

print("Left half before shift:"+left\_half)

print("Right half before shift:"+right\_half+"\n")

def shift(left\_half, d):

    Lfirst = left\_half[0: d]

    Lsecond = left\_half[d:]

    return Lsecond + Lfirst

left\_half = shift(left\_half, 1)

right\_half = shift(right\_half, 1)

print("Left half after shift:"+left\_half)

print("Right half after shift:"+right\_half+"\n")

p8\_key = left\_half+right\_half

print("P8 = "+p8\_key+"\n")

key1 = permutation(p8, p8\_key)

print("Key 1 = "+key1+"\n")

left\_half = shift(left\_half, 1)

right\_half = shift(right\_half, 1)

print("Left half after 2nd shift:"+left\_half)

print("Right half after 2nd shift:"+right\_half+"\n")

p8\_key2 = left\_half+right\_half

key2 = permutation(p8, p8\_key2)

print("Key 2 = "+key2+"\n")

# Encryption

print("Plain text:"+str(plain\_text))

per\_pt = permutation(ip, plain\_text)

print("Permuted plain text:"+per\_pt)

left\_pt = per\_pt[:4]

right\_pt = per\_pt[4:]

exp\_right\_pt = permutation(ep, right\_pt)

print("Expanded right plain text:"+exp\_right\_pt)

def ex\_or\_operation(length, first\_word, second\_word):

    ex\_or = ""

    for i in range(0, len(length)):

        if str(first\_word[i]) == "1" and str(second\_word[i]) == "1" or str(first\_word[i]) == "0" and str(second\_word[i]) == "0":

            ex\_or += "0"

        else:

            ex\_or += "1"

    return ex\_or

ex\_or = ex\_or\_operation(ip, key1, exp\_right\_pt)

print("Ex-or with key 1:"+ex\_or)

left\_exor = ex\_or[:4]

right\_exor = ex\_or[4:]

a = int((left\_exor[0]+left\_exor[3]), 2)

b = int((left\_exor[1]+left\_exor[2]), 2)

left\_s0 = s0\_box[a][b]

c = int((right\_exor[0]+right\_exor[3]), 2)

d = int((right\_exor[1]+right\_exor[2]), 2)

right\_s1 = s1\_box[c][d]

s\_box\_output = str(left\_s0)+str(right\_s1)

print("S box output:"+str(s\_box\_output))

per\_sbox\_output = permutation(p4, s\_box\_output)

print("Permuted S box value:"+str(per\_sbox\_output))

ex\_or2 = ex\_or\_operation(left\_pt, left\_pt, per\_sbox\_output)

print("Ex or with left half:"+str(ex\_or2)+"\n")

# swap

print("Swap\n")

left\_pt2 = right\_pt

right\_pt2 = ex\_or2

exp\_right\_pt, per\_sbox\_output = "", ""

exp\_right\_pt = permutation(ep, right\_pt2)

print("Expanded right plain text:"+exp\_right\_pt)

ex\_or = ex\_or\_operation(key2, key2, exp\_right\_pt)

print("Ex-or with key 2:"+ex\_or)

left\_exor = ex\_or[:4]

right\_exor = ex\_or[4:]

a = int((left\_exor[0]+left\_exor[3]), 2)

b = int((left\_exor[1]+left\_exor[2]), 2)

left\_s0 = s0\_box[a][b]

c = int((right\_exor[0]+right\_exor[3]), 2)

d = int((right\_exor[1]+right\_exor[2]), 2)

right\_s1 = s1\_box[c][d]

s\_box\_output = str(left\_s0)+str(right\_s1)

print("S box output:"+str(s\_box\_output))

per\_sbox\_output = permutation(p4, s\_box\_output)

print("Permuted S box value:"+str(per\_sbox\_output))

ex\_or2 = ex\_or\_operation(per\_sbox\_output, per\_sbox\_output, left\_pt2)

print("Ex or with left half:"+str(ex\_or2))

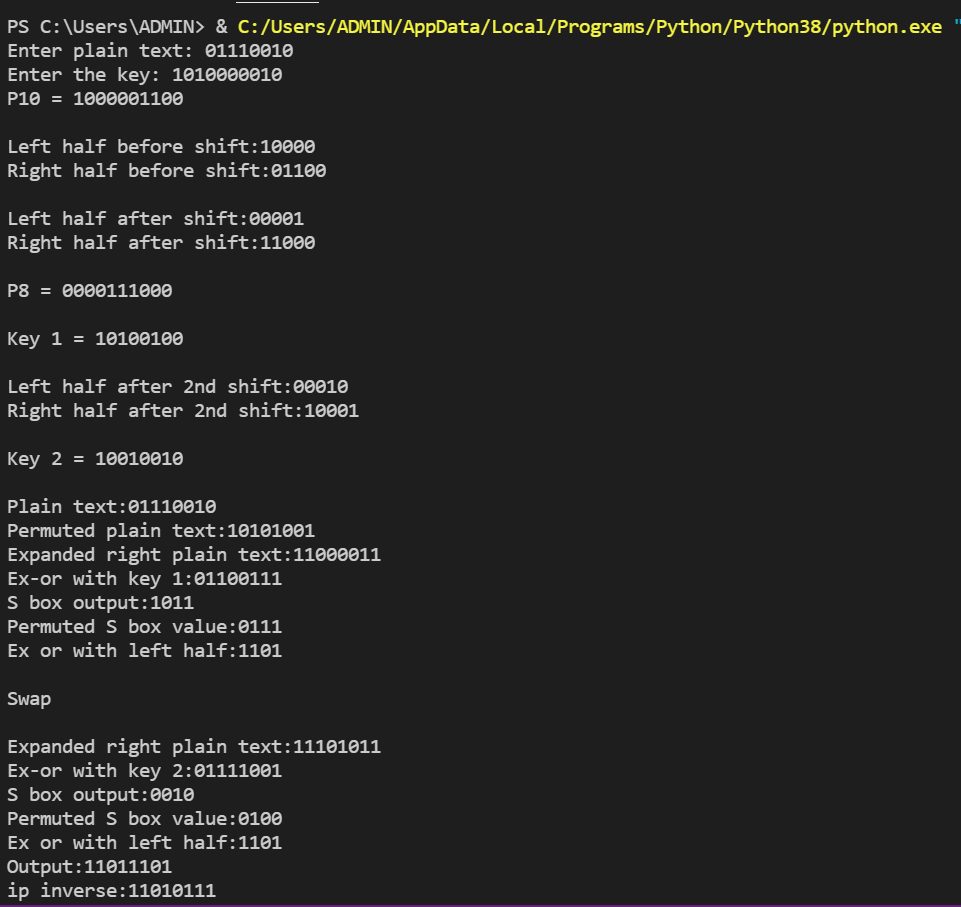
str1 = ex\_or2+right\_pt2

print("Output:"+str(str1))

ip\_inverse = permutation(ip\_inv, str1)

print("ip inverse:"+str(ip\_inverse))

**Output**:



**Practical: 8**

**Aim: Perform a practical to implement RSA algorithm.**

**Code**:

def check\_prime(num):

    for i in range(2, num):

        if (num % i) == 0:

            return True

def gcd(a, b):

    if a < b:

        a, b = b, a

    if(b == 0):

        return a

    else:

        return gcd(b, a % b)

p = int(input("Enter value of p:"))

q = int(input("Enter value of q:"))

M = int(input("Enter message:"))

e\_list = list()

if p < 1 or q < 1 or M < 1:

    print("sorry,Invalid number")

elif check\_prime(p) == True:

    print(p, "is not a prime number")

elif check\_prime(q) == True:

    print(q, "is not a prime number")

else:

    n = p\*q

    print("P =", p)

    print("Q =", q)

    print("N = (p \* q) = (", p, "\*", q, ") =", n)

    print("Message =", M)

    fin\_n = (p-1)\*(q-1)

    print("fi(N) = (p-1)\*(q-1) = (", p, "- 1 ) \* (", q, "- 1 ) = ", fin\_n)

    for i in range(2, fin\_n):

        final\_gcd = gcd(i, fin\_n)

        if final\_gcd == 1:

            e\_list.append(i)

    print("E list = ", str(e\_list))

    e = e\_list[0]

    print("Taking e =", e)

    for i in range(1, 100):

        if (i\*e) % fin\_n == 1:

            d = i

            break

    print("d =", d)

    print("Public key = [", e, ",", n, "]")

    print("Private key = [", d, ",", n, "]")

    # Encryption

    c\_t = (M\*\*e) % n

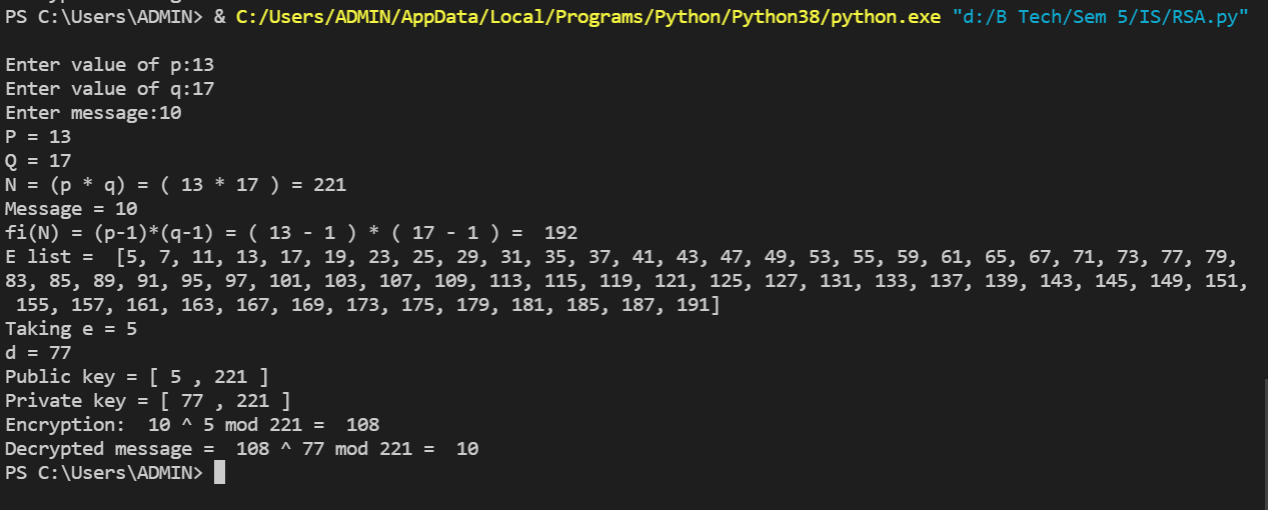
    print("Encryption: ", M, "^", e, "mod", n, "= ", c\_t)

    # decryption

    decrypt = (c\_t\*\*d) % n

    print("Decrypted message = ", c\_t, "^", d, "mod", n, "= ", decrypt)

**Output**:



**Practical: 9**

**Aim: Implement Diffie-Hellman Key exchange algorithm.**

**Code**:

def primitive\_root(q):

    c = [i for i in range(1, q)]

    for i in range(2, q):

        actual\_set = [((i\*\*power) % q) for power in range(1, q)]

        if c == sorted(actual\_set):

            return i

def check\_prime(num):

    for i in range(2, num):

        if (num % i) == 0:

            return True

q = int(input("Enter a Prime number: "))

if q < 1:

    print("sorry,Invalid number")

elif check\_prime(q) == True:

    print(q, "is not a prime number")

else:

    xA = 4

    yB = 3

    g = primitive\_root(q)

    print("g = ", g)

    A = (g\*\*xA) % q

    print("Public key A =", A)

    B = (g\*\*yB) % q

    print("Public key B =", B)

    k1 = (B\*\*xA) % q

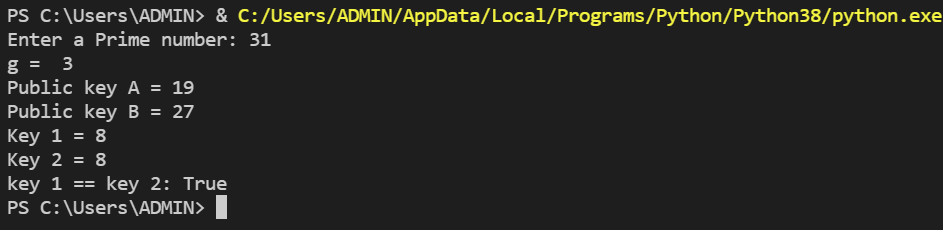
    print("Key 1 =", k1)

    k2 = (A\*\*yB) % q

    print("Key 2 =", k2)

    print("key 1 == key 2:", k1 == k2)

**Output**:



**Practical: 10**

**Aim: Demonstration of python cryptography package to perform symmetric encryption algorithm.**

**Code:**

from cryptography.fernet import Fernet

key = Fernet.generate\_key()

f = Fernet(key)

token = f.encrypt(b"welcome to C G Patel Institute of Technology")

print("Encrypted Text: ")

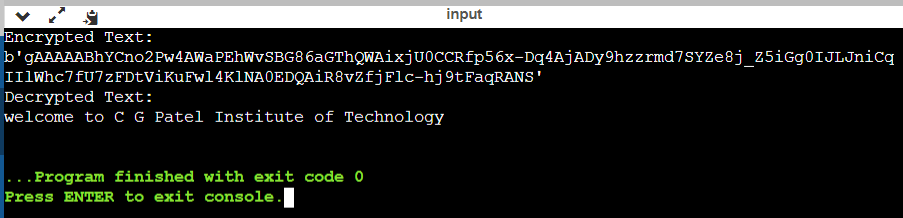
print(token)

print("Decrypted Text: ")

d = f.decrypt(token)

print(d.decode())

**Output:**



**Practical: 11**

**Aim: Demonstration of python cryptography package to perform asymmetric encryption algorithm.**

**Code:**

from cryptography.hazmat.backends import default\_backend

from cryptography.hazmat.primitives.asymmetric import rsa

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import padding

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

public\_exponent=65537, key\_size=2048, backend=default\_backend())

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

print("Private Key : ", private\_key)

print("Public Key : ", public\_key)

pem\_pr = private\_key.private\_bytes(encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.PKCS8,encryption\_algorithm=serialization.NoEncryption()

)

with open('private\_key.pem', 'wb') as f:

f.write(pem\_pr)

pem\_pu = public\_key.public\_bytes(encoding=serialization.Encoding.PEM,

format=serialization.PublicFormat.SubjectPublicKeyInfo)

with open('public\_key.pem', 'wb') as f:

f.write(pem\_pu)

with open("private\_key.pem", "rb") as key\_file:

private\_key = serialization.load\_pem\_private\_key(

key\_file.read(),password=None,backend=default\_backend())

with open("public\_key.pem", "rb") as key\_file:

public\_key = serialization.load\_pem\_public\_key(

key\_file.read(),backend=default\_backend())

message = b'encrypt me!'

encrypted = public\_key.encrypt(

message,

padding.OAEP(

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

algorithm=hashes.SHA256(),

label=None

)

)

print(encrypted)

original\_message = private\_key.decrypt(

encrypted,

padding.OAEP(

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

algorithm=hashes.SHA256(),

label=None

)

)

print(original\_message)

**Output:**

