

SRI SIVASUBRAMANIYA NADAR

COLLEGE OF ENGINEERING

(AFFILIATED TO ANNA UNIVERSITY, CHENNAI)

RAJIV GANDHI SALAI (OMR), KALAVAKKAM – 603 110

LABORATORY RECORD

IT8761 – SECURITY LABORATORY

Name : **GOURAV KUMAR L**

Reg. No. : **312217205030**

Dept. : **INFORMATION TECHNOLOGY** Sem. : **VII** Sec. : **IT-A**



ANNA UNIVERSITY

BONAFIDE CERTIFICATE

Certified that this is the bonafide record of
the practical work done for the

..... lab

by

Name ...L.Gourav Kumar.....

Register Number312217205030.....

of Department ofINFORMATION TECHNOLOGY.....

Sri Sivasubramaniya Nadar College of Engineering,

Kalavakkam, Chennai

during the academic year2020.....

Staff In-Charge

Head

Department of

Submitted for the University Examination held at

Sri Sivasubramaniya Nadar College of Engineering on

Internal Examiner

External Examiner

INDEX

Name: L.Gourav Kumar Reg. No. : 312217205030

Sem. : VII Sec. : A

Ex.No.	Date of Expt.	Title of the Experiment	Page No.	Signature of the Faculty	Remarks
1	11-08-2020	Substitution Techniques - Ceaser Cipher - Playfair Cipher - Vigenere Cipher - Hill Cipher			
2	25-08-2020	Transposition Techniques - Rail fence - Row and Column - One Time Pad			
3	08-09-2020	DES Algorithm			
4	15-09-2020	AES Algorithm			
5	22-09-2020	RSA Algorithm			
6	29-09-2020	Key Exchange Algorithms - Diffie Hellman - ElGamal Key Exchange			
7	06-10-2020	SHA-1 Algorithm			
8	13-10-2020	Digital Signature Algorithms - ElGamal Digital Signature - Digital Signature Algorithm			
9	20-10-2020	Intrusion Detection System using Snort			
10	27-10-2020	Web application security using N-Stalker			
11	03-11-2020	Defeating malware Building Rootkit hunter			

SUBSTITUTION TECHNIQUE

Ex. No : 1(a)
Date:11/08/2020

Encryption and Decryption Using Ceaser Cipher

AIM:

To encrypt and decrypt the given message by using Ceaser Cipher encryption algorithm.

ALGORITHMS:

1. In Ceaser Cipher each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet.
2. For example, with a **left shift of 3**, **D** would be replaced by **A**, **E** would become **B**, and so on.
3. The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, **A = 0, B = 1, Z = 25**.
4. Encryption of a letter x by a shift n can be described mathematically as,
 $En(x) = (x + n) \bmod (\text{total no. of letters})$
5. Decryption is performed similarly,
 $Dn(x) = (x - n + \text{total no. of letters}) \bmod 26$

PROGRAM:

```
def modencrypt(text,key): result=""  
  
a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
  
if(text.isupper()):  
for i in range(0,len(text)): if(text[i]==" "):  
result=result+" " else:  
p=a.find(text[i]) result=result+a[(p+key)%26]  
else:  
text=text.upper()  
for i in range(0,len(text)): if(text[i]==" "):  
result=result+" " else:  
p=a.find(text[i]) result=result+a[(p+key)%26] result=result.lower()
```

```
return result

def moddecrypt(text,key): result=""

a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

if(text.isupper()):

for i in range(0,len(text)):

if(text[i]==" "): result=result+" "

else:

p=a.find(text[i]) result=result+a[(p-key+26)%26]

else:

text=text.upper()

for i in range(0,len(text)): if(text[i]==" "):

result=result+" " else:

p=a.find(text[i]) result=result+a[(p-key+26)%26] result=result.lower()

return result

def encrypt(text,key): result=""

a="ABCDEFGHIJKLMNOPQRSTUVWXYZ" b=""

for i in range(0,26): b=b+a[(i+key)%26]

if(text.isupper()):

for i in range(0,len(text)): if(text[i]==" "):

result=result+" " else:

p=a.find(text[i]) result=result+b[p]

else:

text=text.upper()

for i in range(0,len(text)): if(text[i]==" "):

result=result+" " else:

p=a.find(text[i]) result=result+b[p] result=result.lower()

return result
```

```
def decrypt(text,key): result=""
a="ABCDEFGHIJKLMNOPQRSTUVWXYZ" b=""
for i in range(0,26): b=b+a[(i+key)%26]
if(text.isupper()):
for i in range(0,len(text)): if(text[i]==" "):
result=result+" " else:
p=b.find(text[i]) result=result+a[p]
else:
text=text.upper()
for i in range(0,len(text)): if(text[i]==" "):
result=result+" "
else:
p=b.find(text[i]) result=result+a[p] result=result.lower()
return result

print("CAESAR CIPHER:")

while True:
text=input("Enter plain text:") s=int(input("Enter Shift/key:"))

print("\n1.Shift Encrypt\n2.Shit Decrypt\n3.Modulo Encrypt\n4.Modulo
Decrypt") option=int(input("\nEnter option:"))

if option==1:
print("Plain Text : " + text ) print("Shift : ",s )
print("Cipher: " + encrypt(text,s)) elif option==3:
print("Plain Text : " + text) print("Shift : ",s )
print("Cipher: " + modencrypt(text,s)) elif option==2:
print("Cipher Text : " + text ) print("Shift : ",s )
print("Plain Text: " + decrypt(text,s)) elif option==4:
print("Cipher Text : " + text) print("Shift : ",s )
```

```
print("Plain Text: " + moddecrypt(text,s))
```

```
else:
```

```
quit()
```

OUTPUT :

```
*Python 3.8.1 Shell*
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, 1
Type "help", "copyright", "credits"
>>>
===== RESTART: C:\U:
CAESAR CIPHER:
Enter plain text:hello
Enter Shift/key:2

1.Shift Encrypt
2.Shit Decrypt
3.Modulo Encrypt
4.Modulo Decrypt

Enter option:1
Plain Text : hello
Shift : 2
Cipher: jgnnq
Enter plain text:jgnnq
Enter Shift/key:2

1.Shift Encrypt
2.Shit Decrypt
3.Modulo Encrypt
4.Modulo Decrypt

Enter option:2
Cipher Text : jgnnq
Shift : 2
Plain Text: hello
```



```
Enter plain text:hello  
Enter Shift/key:2
```

```
1.Shift Encrypt  
2.Shit Decrypt  
3.Modulo Encrypt  
4.Modulo Decrypt
```

```
Enter option:3  
Plain Text  : hello  
Shift :    2  
Cipher: jgnnq
```

RESULT:

Thus the program for ceaser cipher encryption and decryption algorithm has been implemented and the output verified successfully

Ex. No : 1(b)
Date:11/08/2020

Encryption and Decryption Using Play fair Cipher

AIM:

To implement a program to encrypt a plain text and decrypt a cipher text using play fair Cipher substitution technique.

ALGORITHM:

1. To encrypt a message, one would break the message into digrams (groups of 2 letters)
2. For example, "HelloWorld" becomes "HE LL OW OR LD".
3. These digrams will be substituted using the key table.
4. Since encryption requires pairs of letters, messages with an odd number of characters usually append an uncommon letter, such as "X", to complete the final digram.
5. The two letters of the digram are considered opposite corners of a rectangle in the key table. To perform the substitution, apply the following 4 rules, in order, to each pair of letters in the plaintext:
 - i. If a pair is a repeated letter, insert filler like 'X'.
 - ii. If both letters fall in the same row, replace each with the letter to its right (circularly).
 - iii. If both letters fall in the same column, replace each with the the letter below it (circularly).
 - iv. Otherwise, each letter is replaced by the letter in the same row but in the column of the other letter of the pair.

PROGRAM:

```
import re

def matrixgen(key): k=0

x=0 play_fair_matrix=[]

a="ABCDEFGHJKLMNOPQRSTUVWXYZ"

for i in range(0,5): temp=[]

j=0

while j<5: if(len(key)>k):

if any(key[k] in sub for sub in play_fair_matrix)==False: temp.append(key[k])

else:

j=j-1

k=k+1 else:
```

```
if a[x] not in key: temp.append(a[x])
```

```
else:
```

```
j=j-1 x=x+1
```

```
j=j+1
```

```
play_fair_matrix.append(temp) return play_fair_matrix
```

```
def play_fair(text,key): b=""
```

```
if 'J' in text: text=text.replace('J','I')
```

```
play_fair_matrix=matrixgen(key) i=0
```

```
while i<len(text):
```

```
if text[i]==text[i+1]: text=text[:i+1] + 'X' + text[i+1:]
```

```
m1=[(index, row.index(text[i])) for index, row in enumerate(play_fair_matrix) if  
text[i] in row]
```

```
m2=[(index, row.index(text[i+1])) for index, row in enumerate(play_fair_matrix)  
if text[i+1] in row]
```

```
r1,c1=m1[0][0],m1[0][1]
```

```
r2,c2=m2[0][0],m2[0][1]
```

```
if c1==c2: b=b+play_fair_matrix[(r1+1)%5][c1]
```

```
b=b+play_fair_matrix[(r2+1)%5][c1]
```

```
elif r1==r2: b=b+play_fair_matrix[r1][(c1+1)%5]
```

```
b=b+play_fair_matrix[r2][(c2+1)%5]
```

```
else:
```

```
b=b+play_fair_matrix[r1][c2] b=b+play_fair_matrix[r2][c1]
```

```
i=i+2 return b
```

```
def play_fair1(text,key): b=""
```

```
if 'J' in text: text=text.replace('J','I')
```

```
play_fair_matrix=matrixgen(key) i=0
```

```
while i<len(text):

m1=[(index, row.index(text[i])) for index, row in enumerate(play_fair_matrix) if
text[i] in row]

m2=[(index, row.index(text[i+1])) for index, row in enumerate(play_fair_matrix)
if text[i+1] in row]

r1,c1=m1[0][0],m1[0][1]

r2,c2=m2[0][0],m2[0][1]

if c1==c2:

b=b+play_fair_matrix[(r1-1+5)%5][c1] b=b+play_fair_matrix[(r2-1+5)%5][c1]

elif r1==r2: b=b+play_fair_matrix[r1][(c1-1+5)%5]
b=b+play_fair_matrix[r2][(c2-1+5)%5]

else:

b=b+play_fair_matrix[r1][c2] b=b+play_fair_matrix[r2][c1]

i=i+2 return b

def play_fair_encrypt(text,key):

regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]') if regex.search(text)!=None:

return "The text should contain alphabets only" if(text.isupper()):

b=play_fair(text,key)

else:

text=text.upper() key=key.upper() b=play_fair(text,key) b=b.lower()

return b

def play_fair_decrypt(text,key);

regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]')

if regex.search(text)!=None:

return "The text should contain alphabets only" if(text.isupper()):

b=play_fair1(text,key)

else:
```

```
text=text.upper() b=play_fair1(text,key) b=b.lower()

return b

while True:

title="PLAY FAIR CIPHER"

print("\n\n") print(title.center(50,'*')) text=input("Enter plain text:")
text1=text.replace(" ","") s=input("Enter key:")
print("\nt1.Encrypt\n\t2.Decrypt") option=int(input("\nEnter option:")) if
option==1:

print("\tPlain Text : " + text ) print("\tShift : ",s ) a=play_fair_encrypt(text1,s)

for i in range(0,len(text)): if text[i]==" ":

a=a[:i] + ' ' + a[i:] print("\tCipher: " + a)

elif option==2:

print("\tCipher Text : " + text ) print("\tShift : ",s ) a=play_fair_decrypt(text1,s)
i=0

if 'X' in a: a=a.replace('X',"")

while i<len(text): if text[i]==" ":

a=a[:i] + ' ' + a[i:] i=i+1

print("\tPlain Text: " + a) else:

quit()
```

Output:

```
===== RESTART: C:\Users\S.Naveen\Documents\security lab\playfair.py =====

*****PLAY FAIR CIPHER*****
Enter plain text:HIDE THE GOLD IN THE TREE STUMP
Enter key:PLAYFAIREXAMPLE

    1.Encrypt
    2.Decrypt

Enter option:1
    Plain Text  : HIDE THE GOLD IN THE TREE STUMP
    Shift  :    PLAYFAIREXAMPLE
    Cipher: BMOD ZBX DNAB EK UDM UIXM MOUVIF

*****PLAY FAIR CIPHER*****
Enter plain text:BMOD ZBX DNAB EK UDM UIXM MOUVIF
Enter key:PLAYFAIREXAMPLE

    1.Encrypt
    2.Decrypt

Enter option:2
    Cipher Text  : BMOD ZBX DNAB EK UDM UIXM MOUVIF
    Shift  :    PLAYFAIREXAMPLE
    Plain Text: HIDE THE GOLD IN THE TREE STUMP
```

RESULT:

Thus the program for playfair cipher encryption and decryption algorithm has been implemented and the output verified successfully.

Ex. No : 1(c)
Date:11/08/2020

Vigenere Cipher

AIM:

To implement a program for encryption and decryption using vigenere cipher substitution technique

ALGORITHM:

1. The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword.
2. It is a simple form of *polyalphabetic* substitution.
3. To encrypt, a table of alphabets can be used, termed a Vigenere square, or Vigenere table.
4. It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers.
5. At different points in the encryption process, the cipher uses a different alphabet from one of the rows used.
6. The alphabet at each point depends on a repeating keyword.

PROGRAM:

```
import re
```

```
def encrypt(text,key):
```

```
a='ABCDEFGHIJKLMNOPQRSTUVWXYZ' b="
```

```
c=len(key) j=0
```

```
regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]') if regex.search(text)!=None:
```

```
return "The text should contain alphabets only" if(text.isupper()):
```

```
for i in range(0,len(text)): if(text[i]==" "):
```

```
b=b+" " else:
```

```
b=b+a[(a.find(text[i])+a.find(key[j%c]))%26] j=j+1
```

```
else:
```

```
text=text.upper() key=key.upper()
for i in range(0,len(text)): if(text[i]==" "):
b=b+" " else:
b=b+a[(a.find(text[i])+a.find(key[j%c]))%26] j=j+1
b=b.lower()
```

```
return b
```

```
def decrypt(text,key): a='ABCDEFGHIJKLMNOPQRSTUVWXYZ' b="
c=len(key) j=0
regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]') if regex.search(text)!=None:
return "The text should contain alphabets only" if(text.isupper()):
for i in range(0,len(text)): if(text[i]==" "):
b=b+" " else:
b=b+a[(a.find(text[i])-a.find(key[j%c]))%26] j=j+1
else:
text=text.upper()
```



```
key=key.upper()

for i in range(0,len(text)): if(text[i]==" "):

b=b+" " else:

b=b+a[(a.find(text[i])-a.find(key[j%c]))%26] j=j+1

b=b.lower()

return b

while True: title="VIGENERE CIPHER"

print(title.center(50,'*')) text=input("Enter plain text:") s=input("Enter key:")

print("\n1.Encrypt\n2.Decrypt") option=int(input("\nEnter option:")) if

option==1:

print("Plain Text : " + text ) print("Shift : ",s )

print("Cipher: " + encrypt(text,s)) elif option==2:

print("Cipher Text : " + text ) print("Shift : ",s )

print("Plain Text: " + decrypt(text,s)) else:

quit()
```

output:

```
>>>
===== RESTART: C:\Users\S.Naveen\Documents\security lab\vigenere.p
*****VIGENERE CIPHER*****
Enter plain text:SHE IS LISTENENING
Enter key:PASCAL

1.Encrypt
2.Decrypt

Enter option:1
Plain Text  : SHE IS LISTENENING
Shift :    PASCAL
Cipher: HHW KS WXSLGNPCIFI
*****VIGENERE CIPHER*****
Enter plain text:HHW KS WXSLGNPCIFI
Enter key:PASCAL

1.Encrypt
2.Decrypt

Enter option:2
Cipher Text  : HHW KS WXSLGNPCIFI
Shift :    PASCAL
Plain Text: SHE IS LISTENENING
```

RESULT:

Thus the program for vigenere cipher encryption and decryption algorithm has been implemented and the output verified successfully.

Ex. No : 1(d)
Date:11/08/2020

Hill Cipher

AIM:

To implement a program to encrypt and decrypt using the Hill cipher substitution technique

ALGORITHM:

1. In the Hill cipher, each letter is represented by a number modulo 26.
2. Divide the input string into blocks of size n and Identify $A=0, B=1, C=2, \dots, Z=25$.
3. To encrypt a message, each block of n letters is multiplied by an invertible $n \times n$ matrix, again *modulus 26*. i.e.

$$C=[P][K] \bmod 26$$

Where C is the cipher text, P is the plain text and K is the key.

4. Perform the above step for each block of the plain text.
5. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption i.e. $P=[C][K \text{ inverse}] \bmod 26$.

PROGRAM:

```
import re

import numpy as np

import math

def matrix_gen(text): l1=len(text)

t=[] a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

for i in range(0,l1): temp=[]

for j in range(0,1): temp.append(a.find(text[i]))

t.append(temp) t=np.array((t)) return t

def encrypt(text,key):

regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]') res=""

a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

if regex.search(text)!=None:

return "The text should contain alphabets only" if(text.isupper()):

t=matrix_gen(text) c=key.dot(t) m=[26]
```

```
c=np.mod(c,m)

for i in range(0,len(text)): res=res+a[c[i][0]]

else:

text=text.upper() t=matrix_gen(text) c=key.dot(t) m=[26]

c=np.mod(c,m)

for i in range(0,len(text)): res=res+a[c[i][0]]

res=res.lower() return res

def decrypt(text,key):

regex = re.compile('[@_!#$%^&*()<>?/\|}{~:]') res=""

a="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

if regex.search(text)!=None:

return "The text should contain alphabets only" if(text.isupper()):

t=matrix_gen(text)

from sympy import Matrix key_inv= Matrix(key).inv_mod(26)

k=np.array(key_inv)

c=k.dot(t) m=[26]

c=np.mod(c,m)

for i in range(0,len(text)): res=res+a[c[i][0]]

else:

text=text.upper() t=matrix_gen(text)

from sympy import Matrix key_inv= Matrix(key).inv_mod(26)

k=np.array(key_inv)

c=k.dot(t) m=[26]

c=np.mod(c,m)

for i in range(0,len(text)): res=res+a[c[i][0]]
```

```
res=res.lower() return res

while True: title="HILL CIPHER"

print("\n\n") print(title.center(50,'*')) text=input("Enter plain text:")

a=int(input("Enter no. of rows in key:")) b=int(input("Enter no. of columns in key:"))
n=4

tmp=[text[i:i+a] for i in range(0, len(text), a)] print(tmp)

key=[]

for i in range(a): temp=[]

for j in range(b):

temp.append(int(input("\nEnter the element:"))) key.append(temp)

key=np.array((key))

print("\n\t1.Encrypt\n\t2.Decrypt") option=int(input("\nEnter option:")) if option==1:

res="

print("\tPlain Text : " + text )

print("\tShift : ",key )

for i in range(0,len(tmp)): res=res+encrypt(tmp[i],key) print("\tCipher: " +
encrypt(tmp[i],key))

print("\tCipher Text: " + res) elif option==2:

print("\tCipher Text : " + text ) print("\tShift : ",key )


print("\tPlain Text: " + decrypt(text,key)) else:

break
```

Output:

```
*****HILL CIPHER*****
Enter plain text:ACT
Enter no. of rows in key:3
Enter no. of columns in key:3
['ACT']

Enter the element:6
Enter the element:24
Enter the element:1
Enter the element:13
Enter the element:16
Enter the element:10
Enter the element:20
Enter the element:17
```

 *Python 3.8.1 Shell*

File Edit Shell Debug Options Window Help

```
1.Encrypt
2.Decrypt

Enter option:1
    Plain Text   : ACT
    Shift :    [[ 6 24  1]
[13 16 10]
[20 17 15]]
    Cipher: POH
    Cipher Text: POH
```

```
1.Encrypt
2.Decrypt

Enter option:2
Cipher Text : POH
Shift : [[ 6 24 1]
[13 16 10]
[20 17 15]]
Plain Text: ACT
```

RESULT:

Thus the program for hill cipher encryption and decryption algorithm has been implemented and the output verified successfully.

TRANSPORTATION TECHNIQUE

Ex. No : 2(a) Date :	Rail Fence Cipher Transposition Technique
---------------------------------------	--

AIM:

To implement a program for encryption and decryption using rail fence transposition technique.

ALGORITHM:

1. In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
2. When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus, the alphabets of the message are written in a zig-zag manner.
3. After each alphabet has been written, the individual rows are combined to obtain the cipher-text.
4. The number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.
5. Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).
6. Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

Program:

```
import math
def rail_fence_encrypt(text,key): b=""
mat=[[" " for i in range(len(text))] for j in range(key)]
l=0
for i in range(0,len(text),2): mat[0][i]=text[l]
if (l+1)<len(text): mat[1][i+1]=text[l+1]
l=l+2
for i in range(0,2):
for j in range(0,len(text)): b=b+mat[i][j]
return b
def rail_fence_decrypt(text,key): b=""
mat=[[" " for i in range(len(text))] for j in range(key)]
l=0 m=math.ceil(len(text)/2) print(m)
```



```
for i in range(0,m): b=b+text[i]
if (m+i)<len(text): b=b+text[m+i]
return b

while True:
print("\n***** RAIL FENCE TRANSPOSITION *****")
print("\n1.Encrypt\n2.Decrypt") choice=int(input("Enter choice:"))
if choice==1: a=input("Enter text:") b=int(input("Enter key:"))
print("\nCipher Text:",rail_fence_encrypt(a,b)) elif choice==2:
a=input("Enter cipher text:") b=int(input("Enter key:"))
print("\nPlain Text:",rail_fence_decrypt(a,b)) else:
break
```

OUTPUT:

```
***** RAIL FENCE TRANSPOSITION *****

1.Encrypt
2.Decrypt
Enter choice:1
Enter text:COMPUTERGRAPHICS
Enter key:2

Cipher Text: CMUEGAHCOPTRRPIS

***** RAIL FENCE TRANSPOSITION *****

1.Encrypt
2.Decrypt
Enter choice:2
Enter cipher text:CMUEGAHCOPTRRPIS
Enter key:2
8

Plain Text: COMPUTERGRAPHICS
```

RESULT:

Thus the java program for Rail Fence Transposition Technique has been implemented and the output verified successfully.

Ex. No : 2(b)
Date:25/08/2020

Row and Column Transformation Technique

AIM:

To implement a program for encryption and decryption by using row and column transformation technique.

ALGORITHM:

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the rows are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be "3 1 2 4".
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: _).
5. Finally, the message is read off in columns, in the order specified by the keyword.
6. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
7. Then, write the message out in columns again, then re-order the columns by reforming the key word.

PROGRAM:

```
def encryption():
    plain=str(input("\nEnter the plain text : "))

    row=int(input("Enter no. of rows: ")) col=int(input("Enter no. of columns: "))
    print("Enter the keys")
    key=[int(input()) for i in range(col)] encrypt=[]
    mat=[[ '0' for i in range(col)] for j in range(row)]
    k=0 val=82
    for i in range(row): for j in range(col):
        if k>=len(plain): mat[i][j]=chr(val) val=val+1
        else :
            mat[i][j]=plain[k] k=k+1
    for i in range(len(key)): for j in range(row):
        if mat[j][key[i]]!='0': encrypt.append(mat[j][key[i]])
    print("Cipher text:"+"".join(encrypt))
def decryption():
```

```
cipher=str(input("Enter the cipher text : ")) row=int(input("Enter no. of rows: "))
col=int(input("Enter no. of columns: "))
print("Enter the keys") key=[int(input()) for i in range(col)]
mat=[[ '0' for i in range(col)] for j in range(row)] k=0
for i in range(len(key)): for j in range(row):
if k<len(cipher): mat[j][key[i]]=cipher[k] k=k+1
decrypt=[]
for i in range(row): for j in range(col):
decrypt.append(mat[i][j])
print("Plain text:"+"".join(decrypt))
while True:
choice=int(input("\n1. Encryption\n2. Decryption\n3. Exit ")) if choice==1:
encryption() elif choice==2: decryption()
else :
break
```

OUTPUT:

```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 bit
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\S.Naveen\Documents\security lab\rowtransposition.py

1. Encryption
2. Decryption
3. Exit 1

Enter the plain text : ATTACKPOSTPONEDUNTILTWOAM
Enter no. of rows: 4
Enter no. of columns: 7
Enter the keys
2
3
1
0
4
5
6
Cipher text:TTNAAPTMTSUOAODWCOIRKNLSPETT
```

```
1. Encryption
2. Decryption
3. Exit 2
Enter the cipher text : TTNAAPTMTSUOAODWCOIRKNLSPETT
Enter no. of rows: 4
Enter no. of columns: 7
Enter the keys
2
3
1
0
4
5
6
Plain text:ATTACKPOSTPONEDUNTILTWOAMRST
```

RESULT:

Thus the java program for Row and Column Transposition Technique has been implemented and the output verified successfully

Ex. No : 2(c)
Date:25/08/2020

One time pad Transformation Technique

AIM:

To implement a program for encryption and decryption by using one time pad transformation technique.

ALGORITHM:

1. The plain text and the random keys are read as input
2. The plain text and key must be same in size
3. The cipher text is generated by doing bit by bit XOR of the plain text with the key.
4. For Decryption, the plain text is generated in the same way by doing bit by bit XOR of the cipher text with the shared one time key.

PROGRAM:

```
def encryption():
    plain=str(input("\nEnter the plain text :
")) res = ".join(format(ord(i), 'b') for i in
plain) print("Plain text in binary
form:",res)
    key=input("Enter the
key:") fin=""
    for i in range(len(res)):
        if res[i]==key[i]:
            fin=fin+"0"
        else:
            fin=fin+"1"

    print("Cipher text:"+fin)
def decryption():
```

```
cipher=input("Enter the cipher text :
```

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

```
fin=""
```

```
res=""
```

```
for i in
```

```
    range(len(key)): if
```

```
    cipher[i]==key[i]:
```

```
        fin=fin+"0"
```

```
    else:
```

```
        fin=fin+"1"
```

```
for i in range(0, len(fin), 7):
```

```
    temp_data = fin[i:i + 7]
```

```
    decimal_data =
```

```
    int(temp_data,2) res =res+
```

```
    chr(decimal_data)
```

```
print("Plain text in
```

```
binary:"+fin) print("Plain
```

```
text:",res)
```

```
while True:
```

```
    choice=int(input("\n1. Encryption\n2. Decryption\n3.
```

```
Exit ")) if choice==1:
```

```
        encryption()
```

```
    elif choice==2:
```

```
        decryption()
```

```
    else :
```

```
        break
```

OUTPUT:

```
*Python 3.8.1 Shell*
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.
Type "help", "copyright", "credits" or "license()" for more info
>>>
===== RESTART: C:\Users\S.Naveen\Documents\security lab\otf

1. Encryption
2. Decryption
3. Exit 1

Enter the plain text : IF
Plain text in binary form: 10010011000110
Enter the key:10101100110001
Cipher text:00111111110111

1. Encryption
2. Decryption
3. Exit 2
Enter the cipher text : 00111111110111
Enter the key:10101100110001
Plain text in binary:10010011000110
Plain text: IF
```

RESULT:

Thus the java program for Row and Column Transposition Technique has been implemented and the output verified successfully.

Ex. No : 3 Date:08/09/2020	Data Encryption Standard (DES) Algorithm (User Message Encryption)
---	--

AIM:

To use Data Encryption Standard (DES) Algorithm for a practical application like User Message Encryption.

ALGORITHM:

1. Read the 64-bit plain text.
2. Split it into two 32-bit blocks and store it in two different arrays.
3. Perform XOR operation between these two arrays.
4. The output obtained is stored as the second 32-bit sequence and the original second 32-bit sequence forms the first part.
5. Thus the encrypted 64-bit cipher text is obtained in this way. Repeat the same process for the remaining plain text characters.

PROGRAM:

DES.java

```
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.KeyGenerator;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;

public class DES
{
    public static void main(String[] argv) {

        try{
            System.out.println("Message Encryption Using DES Algorithm\n-----");
            KeyGenerator keygenerator = KeyGenerator.getInstance("DES");
            SecretKey myDesKey = keygenerator.generateKey();
            Cipher desCipher;
            desCipher = Cipher.getInstance("DES/ECB/PKCS5Padding");
            desCipher.init(Cipher.ENCRYPT_MODE, myDesKey);
            byte[] text = "Secret Information ".getBytes();
            System.out.println("Message [Byte Format] : " + text);
```



```
System.out.println("Message : " + new String(text));
byte[] textEncrypted = desCipher.doFinal(text);
System.out.println("Encrypted Message: " + textEncrypted);
desCipher.init(Cipher.DECRYPT_MODE, myDesKey);
byte[] textDecrypted = desCipher.doFinal(textEncrypted);
System.out.println("Decrypted Message: " + new
String(textDecrypted));

} catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
} catch (NoSuchPaddingException e) {
    e.printStackTrace();
} catch (InvalidKeyException e) {
    e.printStackTrace();
} catch (IllegalBlockSizeException e) {
    e.printStackTrace();
} catch (BadPaddingException e) {
    e.printStackTrace();
}
```

OUTPUT:

```
**** DES ALGORITHM ****
1.Encryption
2.Decryption
Enter choice:1
Enter the plain text:0123456789ABCDEF
Enter the key:133457799BBCDFF1
After initial permutation CC00CCFFF0AAF0AA
Round 1  F0AAF0AA  EF4A6544  1B02EFFC7072
Round 2  EF4A6544  CC017709  79AED9DBC9E5
Round 3  CC017709  A25C0BF4  55FC8A42CF99
Round 4  A25C0BF4  77220045  72ADD6DB351D
Round 5  77220045  8A4FA637  7CEC07EB53A8
Round 6  8A4FA637  E967CD69  63A53E507B2F
Round 7  E967CD69  064ABA10  EC84B7F618BC
Round 8  064ABA10  D5694B90  F78A3AC13BFB
Round 9  D5694B90  247CC67A  E0DBEBEDE781
Round 10 247CC67A  B7D5D7B2  B1F347BA464F
Round 11 B7D5D7B2  C5783C78  215FD3DED386
Round 12 C5783C78  75BD1858  7571F59467E9
Round 13 75BD1858  18C3155A  97C5D1FABA41
Round 14 18C3155A  C28C960D  5F43B7F2E73A
Round 15 C28C960D  43423234  BF918D3D3F0A
Round 16 0A4CD995  43423234  CB3D8B0E17F5
Cipher Text : 85E813540F0AB405
```

RESULT:

Thus the java program for DES Algorithm has been implemented and the output verified successfully

Ex. No : 4 Date:15/09/2020	Advanced Encryption Standard (AES) Algorithm (URL Encryption)
---	--

AIM:

To implement Advanced Encryption Standard (AES) Algorithm .

ALGORITHM:

1. Read the 128-bit plain text.
2. Add the round Key.
3. The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.
4. Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row.
5. The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.
6. Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.
7. Repeat the same process for all the rounds.
8. The final round should be done in similar way without mix columns.
9. Decryption is done in reverse way to encryption.

PROGRAM:

AES.java

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;
```

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
```

```
public class AES {
```

```
    private static SecretKeySpec secretKey;
    private static byte[] key;
```

```
public static void setKey(String myKey) {
    MessageDigest sha = null;
    try {
        key = myKey.getBytes("UTF-8");
        sha = MessageDigest.getInstance("SHA-1");
        key = sha.digest(key);
        key = Arrays.copyOf(key, 16);
        secretKey = new SecretKeySpec(key, "AES");
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

public static String encrypt(String strToEncrypt, String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        return
Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-
8"))));
    } catch (Exception e) {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}

public static String decrypt(String strToDecrypt, String secret) {
    try {
        setKey(secret);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        return new
String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
    } catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}

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

```
final String secretKey = "annaUniversity";

String originalString = "www.annauniv.edu";
String encryptedString = AES.encrypt(originalString, secretKey);
String decryptedString = AES.decrypt(encryptedString, secretKey);

System.out.println("URL Encryption Using AES Algorithm\n-----");
System.out.println("Original URL : " + originalString);
System.out.println("Encrypted URL : " + encryptedString);
System.out.println("Decrypted URL : " + decryptedString);
}
}
```

OUTPUT:

```
-----
Plain text:54776f204f6e65204e696e652054776f
Round 1 key:E232FCF191129188B159E4E6D679A293
After Round 1 : 581559CD47B6D439081CE2DF8BBAE8CE
Round 2 key:56082007C71AB18F76435569A03AF7FA
After Round 2 : 430E093DC65708F8A9C0EB7F62C8FE37
Round 3 key:D2600DE7157ABC686339E901C3031EFB
After Round 3 : 7870994B76763C39307D373454235BF1
Round 4 key:A11202C9B468BEA1D75157A01452495B
After Round 4 : B10804E7CAFCB1B25154C96CEDE1D320
Round 5 key:B1293B3305418592D210D232C6429B69
After Round 5 : 9B235D2F515F1C382022BD9168F03256
Round 6 key:BD3DC287B87C47156A6C9527AC2E0E4E
After Round 6 : 148FC05E93A4600F252B249277E84075
Round 7 key:CC96ED1674EAAA031E863F24B2A8316A
After Round 7 : 53434F8539060A528E933B575DF895BD
Round 8 key:8E51EF21FABB4522E43D7A0656954B6C
After Round 8 : 6670AFA325CED3733C5A0F1374A80A54
Round 9 key:BFE2BF904559FAB2A16480B4F7F1CBD8
After Round 9 : 09A2F07B66D1FC3B8B9AE6307865C489
Round 10 key:28FDDEF86DA4244ACCC0A4FE3B316F26
cipher text 29C3505F571420F6402299B31A02D73A
>>> |
```

RESULT:

Thus the PYTHON program for AES Algorithm has been implemented and the output verified successfully.

Ex. No : 5 Date:22/09/2020	RSA Algorithm
---	----------------------

AIM:

To implement RSA (Rivest–Shamir–Adleman) algorithm by using HTML and Javascript.

ALGORITHM:

Step 1: Start.

Step 2 : Pick two large primes numbers p and q.

Step 3: Calculate n as $n = pq$ and $\Phi(n)$ as $\Phi(n)=(p-1)(q-1)$.

Step 4: Choose a relatively small integer d such that $\text{GCD}(d, \Phi(n))=1$

Step 5: Find e, the multiplicative inverse of d mod $\Phi(n)$

Step 6: (d,n) is the public key and (e,n) is the private key.

Step 7: .To encrypt M, compute $\text{Encrypt}(M) = M^e \pmod n$.

Step 8 : To decrypt C, compute $\text{Decrypt}(C) = C^d \pmod n$.

Step 9 : Stop.

PROGRAM:

rsa.html

```
<html>
```

```
<head>
```

```
  <title>RSA Encryption</title>
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
</head>
```

```
<body>
```

```
  <center>
```

```
    <h1>RSA Algorithm</h1>
```

```
    <h2>Implemented Using HTML & Javascript</h2>
```

```
    <hr>
```

```
    <table>
```

```
<tr>
  <td>Enter First Prime Number:</td>
  <td><input type="number" value="53" id="p"></td>
</tr>
<tr>
  <td>Enter Second Prime Number:</td>
  <td><input type="number" value="59" id="q"></p>
  </td>
</tr>
<tr>
  <td>Enter the Message(cipher text):<br>[A=1, B=2,...]</td>
  <td><input type="number" value="89" id="msg"></p>
  </td>
</tr>
<tr>
  <td>Public Key:</td>
  <td>
    <p id="publickey"></p>
  </td>
</tr>
<tr>
  <td>Exponent:</td>
  <td>
    <p id="exponent"></p>
  </td>
</tr>
<tr>
  <td>Private Key:</td>
  <td>
    <p id="privatekey"></p>
  </td>
</tr>
<tr>
  <td>Cipher Text:</td>
  <td>
    <p id="ciphertext"></p>
  </td>
</tr>
<tr>
  <td><button onclick="RSA();">Apply RSA</button></td>
</tr>
</table>
</center>
```

```
</body>
<script type="text/javascript">
  function RSA() {
    var gcd, p, q, no, n, t, e, i, x;
    gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); };
    p = document.getElementById('p').value;
    q = document.getElementById('q').value;
    no = document.getElementById('msg').value;
    n = p * q;
    t = (p - 1) * (q - 1);

    for (e = 2; e < t; e++) {
      if (gcd(e, t) == 1) {
        break;
      }
    }

    for (i = 0; i < 10; i++) {
      x = 1 + i * t
      if (x % e == 0) {
        d = x / e;
        break;
      }
    }

    ctt = Math.pow(no, e).toFixed(0);
    ct = ctt % n;

    dtt = Math.pow(ct, d).toFixed(0);
    dt = dtt % n;

    document.getElementById('publickey').innerHTML = n;
    document.getElementById('exponent').innerHTML = e;
    document.getElementById('privatekey').innerHTML = d;
    document.getElementById('ciphertext').innerHTML = ct;
  }
</script>
</html>
```


OUTPUT:

RSA Algorithm

Implemented Using HTML & Javascript

Enter First Prime Number:	<input type="text" value="53"/>
Enter Second Prime Number:	<input type="text" value="59"/>
Enter the Message(cipher text): [A=1, B=2,...]	<input type="text" value="89"/>
Public Key:	3127
Exponent:	3
Private Key:	2011
Cipher Text:	1394
<input type="button" value="Apply RSA"/>	

RESULT:

Thus the RSA algorithm has been implemented using HTML & CSS and the output has been verified successfully.

KEY EXCHANGE ALGORITHM

Ex. No : 6(a) Date:29/09/2020	Diffie-Hellman key exchange algorithm
--	--

AIM:

To implement the Diffie-Hellman Key Exchange algorithm .

ALGORITHM:

1. Both Alice and Bob share the same public keys g and p .
2. Alice selects a random public key a .
3. Alice computes his secret key A as $ga \bmod p$.
4. Then Alice sends A to Bob
5. Similarly, Bob also selects a public key b and computes his secret key as B and sends the same back to Alice.
6. Now both of them compute their common secret key as the other one's secret key power of $a \bmod p$.

PROGRAM:

DiffieHellman.java

```
q=int(input("Enter large prime integer(q):")) a=int(input("Enter primitive root(a):"))
xa=int(input("Enter Xa:")) xb=int(input("Enter Xb:"))
ya=(a**xa)%q
print("A's private key:",ya) yb=(a**xb)%q
print("B's private key:",yb) print("Shared key computation:") ka=(ya**xb)%q
print("Public key computed by A",ka) kb=(yb**xa)%q
print("Public key computed by B",kb)
```

OUTPUT:

```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64 ]
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\S.Naveen\Documents\security lab\diffiehellman.py
Enter large prime integer(q):353
Enter primitive root(a):3
Enter Xa:97
Enter Xb:233
A's private key: 40
B's private key: 248
Shared key computation:
Public key computed by A 160
Public key computed by B 160
>>> |
```

RESULT:

Thus the *Diffie-Hellman key exchange algorithm* has been implemented using PYTHON and the output has been verified successfully.

Ex. No : 6(b)
Date:29/09/2020

ELGAMMAL key exchange algorithm

AIM:

To implement the ELGAMAL Exchange algorithm for a given problem .

ALGORITHM:

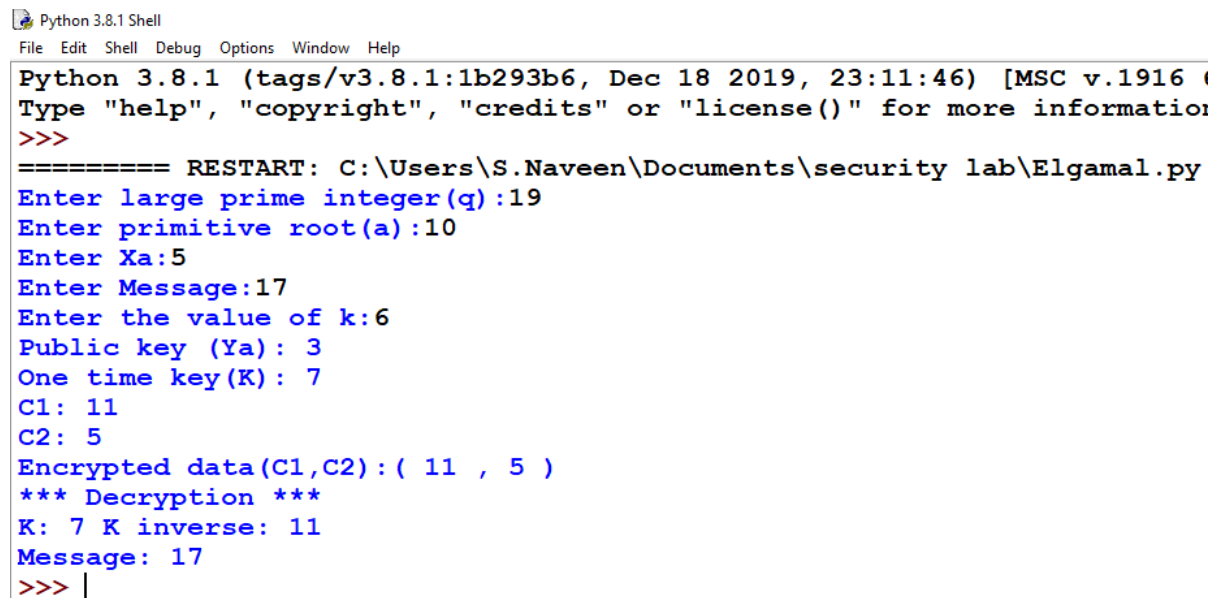
1. Bob generates public and private key :
 - Bob chooses a very large number q and a cyclic group F_q .
 - From the cyclic group F_q , he choose any element g and an element a such that $\gcd(a, q) = 1$.
 - Then he computes $h = g^a$.
 - Bob publishes F , $h = g^a$, q and g as his public key and retains a as private key.
2. Alice encrypts data using Bob's public key :
 - Alice selects an element k from cyclic group F such that $\gcd(k, q) = 1$.
 - Then she computes $p = g^k$ and $s = h^k = g^{ak}$.
 - She multiplies s with M .
 - Then she sends $(p, M*s) = (g^k, M*s)$.
3. Bob decrypts the message :
 - Bob calculates $s' = p^a = g^{ak}$.
 - He divides $M*s$ by s' to obtain M as $s = s'$.

PROGRAM:

```
q=int(input("Enter large prime
integer(q):")) a=int(input("Enter primitive
root(a):")) xa=int(input("Enter Xa:"))
ya=(a**xa)%q
m=int(input("Enter
Message:"))
k=int(input("Enter the value of
k:")) print("Public key (Ya):",ya)
K=(ya**k)%q
print("One time
key(K):",K) C1=(a**k)%q
print("C1:",C1)
C2=(K*m)%q
print("C2:",C2)
print("Encrypted data(C1,C2):(",C1,",",C2,")")
K=(C1**xa)%q
i=1
```

```
while True:
    x=(K*i)%q
    if x==1:
        break
    else:
        i=i+1
print("*** Decryption ***")
print("K:",K,"K inverse:",i)
M=(C2*i)%q
print("Message:",M)
```

OUTPUT:



```
Python 3.8.1 Shell
File Edit Shell Debug Options Window Help
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 23:11:46) [MSC v.1916 64-bit (AMD64)]
Type "help", "copyright", "credits" or "license()" for more
>>>
===== RESTART: C:\Users\S.Naveen\Documents\security lab\Elgamal.py
Enter large prime integer(q):19
Enter primitive root(a):10
Enter Xa:5
Enter Message:17
Enter the value of k:6
Public key (Ya): 3
One time key(K): 7
C1: 11
C2: 5
Encrypted data(C1,C2):( 11 , 5 )
*** Decryption ***
K: 7 K inverse: 11
Message: 17
>>> |
```

RESULT:

Thus the *EL-GAMAL key exchange algorithm* has been implemented using Java Program and the output has been verified successfully.

Ex. No : 7
Date:06/10/2020

SHA-1 Algorithm

AIM:

To Calculate the message digest of a text using the SHA-1 algorithm.

ALGORITHM:

1. Append Padding Bits
2. Append Length - 64 bits are appended to the end
3. Prepare Processing Functions
4. Prepare Processing Constants
5. Initialize Buffers
 - $h0=0x67452301,$
 - $h1=0xEFCDAB89,$
 - $h2=0x98BADCFE,$
 - $h3=0x10325476,$
 - $h4=0xC3D2E1F0$
6. Processing Message in 512-bit blocks (L blocks in total message)
7. Perform message schedule W:
 - For $t=0$ to $t=15$:
 - $W[t]=M[t]$
 - For $t=16$ to $t=79$:
 - $W[t] = (W[t-3] \text{ xor } W[t-8] \text{ xor } W[t-14] \text{ xor } W[t-16]) \text{ left rotate } 1$
8. Initialize variables $a=h0, b=h1, c=h2, d=h3, e=h4$.
9. Perform the main hash computation
 - For $t=0$ to $t=79$:
 - $T = (a \text{ left rotate } 5) + f + e + k + W[t]$
 - $e = d$
 - $d = c$
 - $c = b \text{ left rotate } 30$
 - $b = a$
 - $a = T$
10. Compute the intermediate hash value
 - $h0 = a + h0,$
 - $h1 = b + h1,$
 - $h2 = c + h2,$
 - $h3 = d + h3,$
 - $h4 = e + h4$
11. Finally print the message digest

PROGRAM:

```
import

sys class

SHA1:
    def __init__(self):
        self._H = [
            0x67452301,
            0xEFCDAB89,
            0x98BADCFE,
            0x10325476,
            0xC3D2E1F0
        ]

    def __str__(self):
        return " ".join((hex(h)[2:]).rjust(8, '0') for h in self._H)

# Private static methods used for internal
operations. @staticmethod
def _ROTL(n, x, w=32):
    return ((x << n) | (x >> w - n))

@staticmethod
def _padding(stream):
    l = len(stream) # Bytes
    hl = [int((hex(l*8)[2:]).rjust(16,
        '0')[i:i+2], 16) for i in range(0,
        16, 2)]

    l0 = (56 - l) % 64
    if not
        l0: l0
        = 64

    if isinstance(stream,
        str): stream +=
        chr(0b10000000)
        stream += chr(0)*(l0-
```

```
1)
for a in hl:
    stream += chr(a)
elif isinstance(stream,
    bytes): stream +=
    bytes([0b10000000])
    stream += bytes(10-1)
    stream +=

bytes(hl) return

stream

    @staticmethod

def
    prepare(strea
    m): M = []
    n_blocks = len(stream) // 64

    stream = bytearray(stream)

    for i in range(n_blocks): # 64 Bytes
        per Block m = []

        for j in range(16): # 16 Words
            per Block n = 0
            for k in range(4): # 4 Bytes
                per Word n <=< 8
                n += stream[i*64 +

                j*4 + k] m.append(n)

        M.append(

    m[:]) return M

    @staticmethod
def_debug_print(t, a, b, c,
```



```
d, e): print('t = {0} :  
\t'.format(t),  
    (hex(a)[2:]).rjust(8, '0'),  
    (hex(b)[2:]).rjust(8, '0'),  
    (hex(c)[2:]).rjust(8, '0'),  
    (hex(d)[2:]).rjust(8, '0'),  
    (hex(e)[2:]).rjust(8, '0')  
    )
```

```
# Private instance methods used for internal  
operations. def process_block(self, block):  
    MASK = 2**32-1
```

```
    W = block[:]  
    for t in range(16, 80):  
        W.append(SHA1._ROTL(1, (W[t-3] ^ W[t-8] ^ W[t-  
            14] ^ W[t-16])) & MASK)
```

```
    a, b, c, d, e = self._H[:]
```

```
    for t in
```

```
        range(80):
```

```
        if t <= 19:
```

```
            K = 0x5a827999
```

```
            f = (b & c) ^ (~b
```

```
& d) elif t <= 39:
```

```
            K =
```

```
            0x6ed9eba1
```

```
            f = b ^ c ^ d
```

```
        elif t <= 59:
```

```
            K = 0x8f1bbcdc
```

```
            f = (b & c) ^ (b & d) ^
```

```
(c & d) else:
```

```
            K = 0xca62c1d6
```

```
            f = b ^ c ^ d
```

```
    T = ((SHA1._ROTL(5, a) + f + e + K + W[t]) & MASK)
```

```
    e =
```

```
    d d
```

```
    = c
```

```
    c = SHA1._ROTL(30, b) & MASK
```

b =
a a
= T

SHA1._debug_print(t, a,b,c,d,e)

```
self._H[0] = (a + self._H[0]) &  
MASK self._H[1] = (b + self.  
H[1]) & MASK self._H[2] = (c +  
self._H[2]) & MASK self._H[3] =  
(d + self._H[3]) & MASK self.  
H[4] = (e + self._H[4]) & MASK
```

Public methods for

class use. def

update(self, stream):

stream = SHA1.

padding(stream) stream =

SHA1._prepare(stream)

for block in stream:

self._process_block(block)

def

digest(self

): pass

def

hexdigest(sel

f): s = "

for h in self._H:

s += (hex(h)[2:]).rjust(8, '0')

```
    return s
```

```
def usage():  
    print('Usage: python SHA1.py <file> [<file>  
    ...]') sys.exit()
```

```
def main():
```

```
    if len(sys.argv)  
        < 2: usage()
```

```
    for filename in  
        sys.argv[1:]: try:  
        with open(filename,  
            'rb') as f: content =  
            f.read()
```

```
    except:  
        print ('ERROR: Input file "{0}" cannot be read.'.format(filename))
```

```
    else:  
        h = SHA1()  
        h.update(content)  
        hex_sha =  
        h.hexdigest()  
        print("Message digest:{0}".format(hex_sha))
```

```
if __name__ == '  
    main_': main()
```

Output:

```
(base) C:\Users\Navein Kumar\Downloads>python SHA1.py a.txt
t = 0 : 01f352301 7bf36ae2 98badcfe 10325476
t = 1 : 8990536d 0116fc33 59d148c0 7bf36ae2 98badcfe
t = 2 : a1390f08 8990536d c045bf0c 59d148c0 7bf36ae2
t = 3 : cdd8e11b a1390f08 626414db c045bf0c 59d148c0
t = 4 : cfd499de cdd8e11b 284e43c2 626414db c045bf0c
t = 5 : 3fc7ca40 cfd499de f3763846 284e43c2 626414db
t = 6 : 993e30c1 3fc7ca40 b3f52677 f3763846 284e43c2
t = 7 : 9e8c07d4 993e30c1 0ff1f290 b3f52677 f3763846
t = 8 : 4b6ae328 9e8c07d4 664f8c30 0ff1f290 b3f52677
t = 9 : 8351f929 4b6ae328 27a301f5 664f8c30 0ff1f290
t = 10 : fbda9e89 8351f929 12dab8ca 27a301f5 664f8c30
t = 11 : 63188fe4 fbda9e89 60d47e4a 12dab8ca 27a301f5
t = 12 : 4607b664 63188fe4 7ef6a7a2 60d47e4a 12dab8ca
t = 13 : 9128f695 4607b664 18c623f9 7ef6a7a2 60d47e4a
t = 14 : 196bee77 9128f695 1181ed99 18c623f9 7ef6a7a2
t = 15 : 20bdd62f 196bee77 644a3da5 1181ed99 18c623f9
t = 16 : 4e925823 20bdd62f c65afb9d 644a3da5 1181ed99
t = 17 : 82aa6728 4e925823 c82f758b c65afb9d 644a3da5
t = 18 : dc64901d 82aa6728 d3a49608 c82f758b c65afb9d
t = 19 : fd9e1d7d dc64901d 20aa99ca d3a49608 c82f758b
t = 20 : 1a37b0ca fd9e1d7d 77192407 20aa99ca d3a49608
t = 21 : 33a23bfc 1a37b0ca 7f67875f 77192407 20aa99ca
t = 22 : 21283486 33a23bfc 868dec32 7f67875f 77192407
t = 23 : d541f12d 21283486 0ce88eff 868dec32 7f67875f
t = 24 : c7567dc6 d541f12d 884a0d21 0ce88eff 868dec32
t = 25 : 48413ba4 c7567dc6 75507c4b 884a0d21 0ce88eff
t = 26 : be35fbd5 48413ba4 b1d59f71 75507c4b 884a0d21
t = 27 : 4aa84d97 be35fbd5 12104ee9 b1d59f71 75507c4b
t = 28 : 8370b52e 4aa84d97 6f8d7ef5 12104ee9 b1d59f71
t = 29 : c5fbaf5d 8370b52e d2aa1365 6f8d7ef5 12104ee9
t = 30 : 1267b407 c5fbaf5d a0dc2d4b 6f8d7ef5 12104ee9
t = 31 : 3b845d33 1267b407 717eebd7 a0dc2d4b d2aa1365
t = 32 : 046faa0a 3b845d33 c499ed01 717eebd7 a0dc2d4b
t = 33 : 2c0ebc11 046faa0a cee1174c c499ed01 717eebd7
t = 34 : 21796cad 2c0ebc11 811bea82 cee1174c c499ed01
t = 35 : dc1bb0cb 21796cad 4b03af04 811bea82 cee1174c
t = 36 : 0f511fd8 dc1bb0cb 085e5ab5 4b03af04 811bea82
t = 37 : dc63973f 0f511fd8 f72eec32 085e5ab5 4b03af04
t = 38 : 4c986405 dc63973f 03d447f6 f72eec32 085e5ab5
t = 39 : 32de1cba 4c986405 f718e5cf 03d447f6 f72eec32
t = 40 : fc87dedf 32de1cba 53261901 f718e5cf 03d447f6
t = 41 : 970a0d5c fc87dedf 8cb7872e 53261901 f718e5cf
t = 42 : 7f193dc5 970a0d5c ff21f7b7 8cb7872e 53261901
t = 43 : ee1b1aaf 7f193dc5 25c28357 ff21f7b7 8cb7872e
t = 44 : 40f28e09 ee1b1aaf 5fc64f71 25c28357 ff21f7b7
t = 45 : 1c51e1f2 40f28e09 fb86c6ab 5fc64f71 25c28357
t = 46 : a01b846c 1c51e1f2 503ca382 fb86c6ab 5fc64f71
t = 47 : bead02ca a01b846c 8714787c 503ca382 fb86c6ab
t = 48 : baf39337 bead02ca 2806e11b 8714787c 503ca382
t = 49 : 120731c5 baf39337 afab40b2 2806e11b 8714787c
t = 50 : 641db2ce 120731c5 eebce4cd afab40b2 2806e11b
t = 51 : 3847ad66 641db2ce 4481cc71 eebce4cd afab40b2
t = 52 : e490436d 3847ad66 99076cb3 4481cc71 eebce4cd
t = 53 : 27e9f1d8 e490436d 8e11eb59 99076cb3 4481cc71
t = 54 : 7b71f76d 27e9f1d8 792410db 8e11eb59 99076cb3
t = 55 : 5e6456af 7b71f76d 09fa7c76 792410db 8e11eb59
t = 56 : c846093f 5e6456af 5edc7ddb 09fa7c76 792410db
t = 57 : d262fff5 c846093f d79915ab 5edc7ddb 09fa7c76
t = 58 : 09d785fd d262fff5 f211824f d79915ab 5edc7ddb
t = 59 : 3f52de5a 09d785fd 3498bfd4 f211824f d79915ab
t = 60 : d756c147 3f52de5a 4275e17f 3498bfd4 f211824f
t = 61 : 548c9cb2 d756c147 8fd4b796 4275e17f 3498bfd4
Store
```

```
Anaconda Prompt
t = 44 : 40f28e09 ee1b1aaf 5fc64f71 25c28357 ff21f7b7
t = 45 : 1c51e1f2 40f28e09 fb86c6ab 5fc64f71 25c28357
t = 46 : a01b846c 1c51e1f2 503ca382 fb86c6ab 5fc64f71
t = 47 : bead02ca a01b846c 8714787c 503ca382 fb86c6ab
t = 48 : baf39337 bead02ca 2806e11b 8714787c 503ca382
t = 49 : 120731c5 baf39337 afab40b2 2806e11b 8714787c
t = 50 : 641db2ce 120731c5 eebce4cd afab40b2 2806e11b
t = 51 : 3847ad66 641db2ce 4481cc71 eebce4cd afab40b2
t = 52 : e490436d 3847ad66 99076cb3 4481cc71 eebce4cd
t = 53 : 27e9f1d8 e490436d 8e11eb59 99076cb3 4481cc71
t = 54 : 7b71f76d 27e9f1d8 792410db 8e11eb59 99076cb3
t = 55 : 5e6456af 7b71f76d 09fa7c76 792410db 8e11eb59
t = 56 : c846093f 5e6456af 5edc7ddb 09fa7c76 792410db
t = 57 : d262fff5 c846093f d79915ab 5edc7ddb 09fa7c76
t = 58 : 09d785fd d262fff5 f211824f d79915ab 5edc7ddb
t = 59 : 3f52de5a 09d785fd 3498bfd4 f211824f d79915ab
t = 60 : d756c147 3f52de5a 4275e17f 3498bfd4 f211824f
t = 61 : 548c9cb2 d756c147 8fd4b796 4275e17f 3498bfd4
t = 62 : b66c020b 548c9cb2 f5d5b051 8fd4b796 4275e17f
t = 63 : 6b61c9e1 b66c020b 9523272c f5d5b051 8fd4b796
t = 64 : 19dfa7ac 6b61c9e1 ed9b0082 9523272c f5d5b051
t = 65 : 101655f9 19dfa7ac 5ad87278 ed9b0082 9523272c
t = 66 : 0c3df2b4 101655f9 0677e9eb 5ad87278 ed9b0082
t = 67 : 78dd4d2b 0c3df2b4 4405957e 0677e9eb 5ad87278
t = 68 : 497093c9 78dd4d2b 030f7cad 4405957e 0677e9eb
t = 69 : 3f2588c2 497093c9 de37534a 030f7cad 4405957e
t = 70 : c199f8c7 3f2588c2 125c24f0 de37534a 030f7cad
t = 71 : 39859de7 c199f8c7 8fc96230 125c24f0 de37534a
t = 72 : edb42de4 39859de7 f0667e31 8fc96230 125c24f0
t = 73 : 11793f6f edb42de4 ce616779 f0667e31 8fc96230
t = 74 : 5ee76897 11793f6f 3b6d0b79 ce616779 f0667e31
t = 75 : 63f7dab7 5ee76897 c45e4fdb 3b6d0b79 ce616779
t = 76 : a079b7d9 63f7dab7 d7b9da25 c45e4fdb 3b6d0b79
t = 77 : 860d21cc a079b7d9 d8fd6ead d7b9da25 c45e4fdb
t = 78 : 5738d5e1 860d21cc 681e6dff d8fd6ead d7b9da25
t = 79 : 42541b35 5738d5e1 21834873 681e6dff d8fd6ead
Message digest:a9993e364706816aba3e25717850c26c9cd0d89d
(base) C:\Users\Navein Kumar\Downloads>
```

a.txt contents : abc

RESULT:

Thus the *Secure Hash Algorithm (SHA-1)* has been implemented and the output has been verified successfully.

Ex. No : 8(A)
Date:13/10/2020

EL-GAMAL Digital Signature Standard

AIM:

To implement the EL-GAMAL SIGNATURE SCHEME - Digital Signature Standard.

ALGORITHM:

Step 1: Start.

Step 2 : Choose a large prime p and a primitive root α .

Step 3: Input the private key X_a such $1 < X_a < q-1$.

Step 4 : Compute public key Y_a as $y_A = \alpha^{x_A} \bmod q$.

Step 5 : Input the hash message $m = H(M)$, $0 \leq m \leq (q-1)$.

Step 6 : Choose a random integer K with $1 \leq K \leq (q-1)$ and $\gcd(K, q-1) = 1$.

Step 7 : Compute temporary key: $S_1 = \alpha^k \bmod q$ and compute K^{-1} the inverse of $K \bmod (q-1)$.

Step 8 : Compute the value: $S_2 = K^{-1}(m - x_A S_1) \bmod (q-1)$ and signature is: (S_1, S_2) .

Step 9 : Compute V_1 as $V_1 = \alpha^m \bmod q$ and V_2 as $V_2 = (y_A^{S_1})^* (S_1^{S_2}) \bmod q$.

Step 10: Signature is valid if $V_1 = V_2$ else invalid.

Step 11 : Stop.

Program:

```
q=int(input("Enter large prime
```

```
integer(q:")) a=int(input("Enter
primitive root(a:"))
xa=int(input("Enter Xa:"))
ya=(a**xa)%q
m=int(input("Enter
Message:"))
k=int(input("Enter the value of k:"))
print("Public key (Ya):",ya)
K=(ya**k)%q
print("One time
key(K):",K)
C1=(a**k)%q
print("C1:",C1)
C2=(K*m)%q
print("C2:",C2)
print("Encrypted data(C1,C2):(",C1,",",C2,")")
K=(C1**xa)
%q i=1
while True:
    x=(K*i)
    %q if
    x==1:
        break
    else:
        i=i+1
print("*** Decryption
***") print("K:",K,"K
inverse:",i)
M=(C2*i)%q
print("Message:",M)
```

Output:

```
(base) C:\Users\Navein Kumar\Desktop>python eds.py
Enter large prime integer(q):19
Enter primitive root(a):10
Enter Xa:16
Enter the value of k:5
Enter Message:14
Public key (Ya): 4
S1: 3
S2: 4
Signature(S1,S2):( 3 , 4 )
V1: 16
V2: 16
    *** Verifivation ***
Signature is valid
(base) C:\Users\Navein Kumar\Desktop>
```

RESULT:

Thus the EL-GAMAL Digital Signature Standard Signature Scheme has been implemented and the output has been verified successfully.

Ex. No : 8(b) Date:13/10/2020	Digital Signature Standard
--	-----------------------------------

AIM:

To implement the SIGNATURE SCHEME - Digital Signature Standard.

ALGORITHM:

Step 1: Start.

Step 2 : Input a prime number q .

Step 3 : Input a large prime p with $2^{L-1} < p < 2^L$, where $L = 512$ to 1024 bits & is a multiple of 64.

Step 4 : Input the primitive root choose g such that $g = h^{((p-1)/q)}$.

Step 5 : Input the private key X and compute public key Y as $Y = (g^x) \bmod p$.

Step 6 : Input a random integer k , such $k < q$.

Step 7 : Input the hash value m of the message $(H(M))$.

Step 8 : Compute r as $r = (g^k \bmod p) \bmod q$ and s as $s = [k^{-1} * (H(M) + xr)] \bmod q$.

Step 9 : Compute :

w as $w = s^{-1} \bmod q$,

u_1 as $u_1 = [H(M)w] \bmod q$

u_2 as $u_2 = (r^*w) \bmod q$

v as $v = [(g^{u_1} * y^{u_2}) \bmod p] \bmod q$.

Step 10 : The signature is valid if $v == r$ else invalid.

Step 11 : Stop.

PROGRAM:

```
q=int(input("Enter large prime
integer(q):")) p=int(input("Enter
large prime integer(p):"))
g=int(input("Enter primitive
root(g):")) xa=int(input("Enter
Xa:"))
ya=(g**xa)%p
k=int(input("Enter the value of
k:")) m=int(input("Enter hash
value:")) print("Public key
(Ya):",ya) r=((g**k)%p)%q
print("r:",r)
i=1
while True:
    x=(k*i)
    %q if
    x==1:
        break
    else:
        i=i+1
        s=(i*(m+xa*r))
print("s:", %q
s)

print("Signature(r,s):(",r,"
",s,")") i=1
while True:
    x=(s*i)
    %q if
    x==1:
        break
    else:
        i=i+1
W=(i)%q
U1=(m*W)
%q
U2=(r*W)%
q
```

```
V=(((g**U1)*(ya**U2))%p)%q
print("W:",W)
print("U1:",U1)
print("U2:",U2)
print("V:",V)
print("\t*** Verifivation
***") if V==r:
    print("Signature is valid")
else:
    print("Signature is not valid!!!")
```

Output:

```
(base) C:\Users\Navein Kumar\Desktop>python DSA.py
Enter large prime integer(q):11
Enter large prime integer(p):23
Enter primitive root(g):4
Enter Xa:7
Enter the value of k:5
Enter hash value:3
Public key (Ya): 8
r: 1
s: 2
Signature(r,s):( 1 , 2 )
W: 6
U1: 7
U2: 6
V: 1
*** Verifivation ***
Signature is valid
(base) C:\Users\Navein Kumar\Desktop>
```

RESULT:

Thus the Digital Signature Standard Signature Scheme has been implemented and the output has been verified successfully.

Ex. No : 9
Date:20/10/2020

Demonstration of Intrusion Detection System(IDS)

AIM:

To demonstrate Intrusion Detection System (IDS) using Snort software tool.

STEPS ON CONFIGURING AND INTRUSION DETECTION:

1. Download Snort from the Snort.org website. (<http://www.snort.org/snort/downloads>)
2. Download Rules(<https://www.snort.org/snort-rules>). You must register to get the rules.
3. Double click on the .exe to install snort. This will install snort in the “C:\Snort” folder.It is important to have WinPcap (<https://www.winpcap.org/install/>) installed
4. Extract the Rules file. You will need WinRAR for the .gz file. 5. Copy all files from the “rules” folder of the extracted folder. Now paste the rules into “C:\Snort\rules” folder.
6. Copy “snort.conf” file from the “etc” folder of the extracted folder. You must paste it into “C:\Snort\etc” folder. Overwrite any existing file. Remember if you modify your snort.conf file and download a new file, you must modify it for Snort to work.
7. Open a command prompt (cmd.exe) and navigate to folder “C:\Snort\bin” folder. (at the Prompt, type cd\snort\bin)
8. To check the interface list, use following command:

`snort -W`

Finding an interface

```
C:\Snort\bin>snort -W

-*) Snort( <+
o*)
****
Version 2.9.16.1-WIN32 GRE (Build 149)
By Martin Roesch & The Snort Team: http://www.snort.org/contact@team
Copyright (C) 2014-2020 Cisco and/or its affiliates. All rights reserved.
Copyright (C) 1998-2013 Sourcefire, Inc., et al.
Using PCRE version: 8.10 2010-06-25
Using ZLIB version: 1.2.3

Index:  Physical Address      IP Address      Device Name      Description
-----
1  5E:01:9F:34:F5:07      disabled      \Device\NPF_{446CC80A-8907-4F11-9349-D68C00E980BC}  VPN Client Adapter - VPN
2  10:02:ES-DE:3F:74      disabled      \Device\NPF_{9F8E5003-D8E3-4A84-A8EE-7B73032F9C8E}  Realtek PCIe GbE Family Controller
3  00:00:00:00:00:00      192.168.1.7    \Device\NPF_{058C1331-5EEA-4403-9F38-07C7BF8FFA55}  Microsoft
4  00:00:00:00:00:00      disabled      \Device\NPF_{05E83290-2215-4E7F-A839-3AED1A713000}  Microsoft
5  00:00:00:00:00:00      disabled      \Device\NPF_{C35052C7-825A-4390-9C3C-A67433CE880A}  Microsoft
```

You can tell which interface to use by looking at the Index number and finding Microsoft. As you can see in the above example, the other interfaces are disabled. My interface is 3.

9. To run snort in IDS mode, you will need to configure the file “snort.conf” according to your network environment.

10. To specify the network address that you want to protect in snort.conf file, look for the following line.

var HOME_NET 192.168.1.0/24 (You will normally see any here)

11. Change the RULE_PATH variable to the path of rules folder.

var RULE_PATH c:\snort\rules

path to rules

12. Change the path of all library files with the name and path on your system. And you must change the path of **snort_dynamicpreprocessorvariable**.

C:\Snort\lib\snort_dynamicccpreprocessor

13. You need to do this to all library files in the “C:\Snort\lib” folder. The old path might be: “/usr/local/lib/...”. you will need to replace that path with your system path. Using C:\Snort\lib

14. Change the path of the “**dynamicengine**” variable value in the “snort.conf” file..

Example:

dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll

15 Add the paths for “include classification.config” and “include reference.config” files.

include c:\snort\etc\classification.config

include c:\snort\etc\reference.config

16. Remove the comment (#) on the line to allow ICMP rules, if it is commented with a #.

include \$RULE_PATH/icmp.rules

17. You can also remove the comment of ICMP-info rules comment, if it is

commented.

include \$RULE_PATH/icmp-info.rules

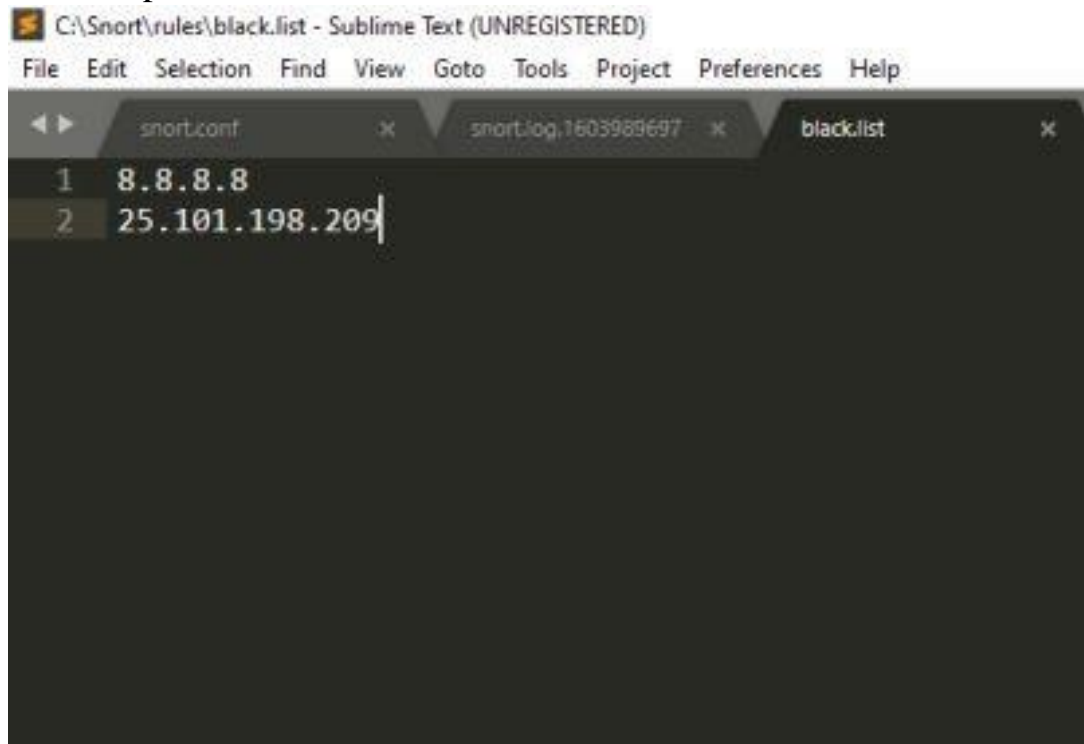
18. To add log files to store alerts generated by snort, search for the “output log” test in snort.conf and add the following line:

`output alert_fast: snort-alerts.ids`

19. Find \$WHITE_LIST_PATH/white_list.rules and change it to white.list and similarly black_list.rules to black.list

Create white.list and black.list file in C:\Snort\rules.

Add the ip address that needs to be blacklisted in black.list file.



20. Comment out (#) following lines:

`#preprocessor normalize_ip4`

`#preprocessor normalize_tcp: ips ecn stream`

`#preprocessor normalize_icmp4`

`#preprocessor normalize_ip6`

`#preprocessor normalize_icmp6`

21. Save the “snort.conf” file.

22. To start snort in IDS mode, run the following command:

`snort -i 3 -c C:\Snort\etc\snort.conf -A console`

(Note: 3 is used for my interface card)

If a log is created, select the appropriate program to open it. You can use WordPard or NotePad++ to read the file.

To generate Log files in ASCII mode, you can use following command while running snort in IDS mode:

```
snort -A console -i3 -c c:\Snort\etc\snort.conf -l c:\Snort\log -K ascii
```

[illegible]

Snort monitoring traffic –

```
SSL Preprocessor:
  SSL packets decoded: 16
    Client Hello: 2
    Server Hello: 2
    Certificate: 2
    Server Done: 4
  Client Key Exchange: 2
  Server Key Exchange: 0
  Change Cipher: 4
  Finished: 0
  Client Application: 3
  Server Application: 2
  Alert: 0
  Unrecognized records: 5
  Completed handshakes: 0
  Bad handshakes: 0
  Sessions ignored: 2
  Detection disabled: 0
=====
SIP Preprocessor Statistics
  Total sessions: 0
=====
IMAP Preprocessor Statistics
  Total sessions                : 0
  Max concurrent sessions      : 0
=====
POP Preprocessor Statistics
  Total sessions                : 0
  Max concurrent sessions      : 0
=====
Reputation Preprocessor Statistics
  Total Memory Allocated: 329964
  Number of packets blacklisted: 1
=====
Snort exiting
```

Log files--


```

1 11/03-12:40:17.998576 192.168.1.7:58272 -> 52.114.159.32:443
2 TCP TTL:128 TOS:0x0 ID:16874 IpLen:20 DgmLen:1009 DF
3 ***AP*** Seq: 0x462E46D6 Ack: 0x9ACEC700 Win: 0xFB85 TcpLen: 20
4
5
6
7 11/03-12:40:17.999820 192.168.1.7:58272 -> 52.114.159.32:443
8 TCP TTL:128 TOS:0x0 ID:16875 IpLen:20 DgmLen:1480 DF
9 ***A**** Seq: 0x462E4A9F Ack: 0x9ACEC700 Win: 0xFB85 TcpLen: 20
10
11
12
13
14 11/03-12:40:17.999832 192.168.1.7:58272 -> 52.114.159.32:443
15 TCP TTL:128 TOS:0x0 ID:16876 IpLen:20 DgmLen:1480 DF
16 ***A**** Seq: 0x462E503F Ack: 0x9ACEC700 Win: 0xFB85 TcpLen: 20
17
18
19
20
21 11/03-12:40:17.999845 192.168.1.7:58272 -> 52.114.159.32:443
22 TCP TTL:128 TOS:0x0 ID:16877 IpLen:20 DgmLen:1205 DF
23 ***AP*** Seq: 0x462E55DF Ack: 0x9ACEC700 Win: 0xFB85 TcpLen: 20
24
25
26
27
28
29

```

RESULT:

Thus the Intrusion Detection System(IDS) has been demonstrated by using the Open Source Snort Intrusion Detection Tool.

Ex. No : 10 Date:27/10/2020	Exploring N-Stalker, a Vulnerability Assessment Tool
--	---

AIM:

To download the N-Stalker Vulnerability Assessment Tool and exploring the features.

EXPLORING N-STALKER:

- N-Stalker Web Application Security Scanner is a Web security assessment tool.
 - It incorporates with a well-known N-Stealth HTTP Security Scanner and 35,000 Web attack signature database.
 - This tool also comes in both free and paid version.
 - Before scanning the target, go to “License Manager” tab, perform the update.
 - Once update, you will note the status as up to date.
 - You need to download and install N-Stalker from www.nstalker.com.
-
1. Start N-Stalker from a Windows computer. The program is installed under Start ⇨ Programs ⇨ N-Stalker ⇨ N-Stalker Free Edition.
 2. Enter a host address or a range of addresses to scan.
 3. Click Start Scan.
 4. After the scan completes, the N-Stalker Report Manager will prompt
 5. you to select a format for the resulting report as choose Generate HTML.
 6. Review the HTML report for vulnerabilities.



Now goto “Scan Session”, enter the target URL.

In scan policy, you can select from the four options,

- Manual test which will crawl the website and will be waiting for manual attacks.
- full xss assessment
- owasp policy
- Web server infrastructure analysis.

Once, the option has been selected, next step is “Optimize settings” which will crawl the whole website for further analysis.


In review option, you can get all the information like host information, technologies used, policy name, etc.

N-Stalker Scan Wizard

Start Web Application Security Scan Session



You must enter an URL and choose policy. Scan Settings may be configured.

Enter Web Application URL





(E.g: <http://www.example.tl/>, <https://www.test.tl/VirtualDirectory/>, etc)

Choose Scan Policy



 (choose one) 

Load Scan Session

 (choose one) 

(You may load scan settings from previously saved scan sessions)

Load Spider Data

 Not available in N-Stalker Free Edition 

(You may load spider data from previously saved scan sessions)


☐ Use local cache from previously saved session (Avoid new web crawling)

N-Stalker Scan Wizard

Start Web Application Security Scan Session

You must enter an URL and choose policy. Scan Settings may be configured.

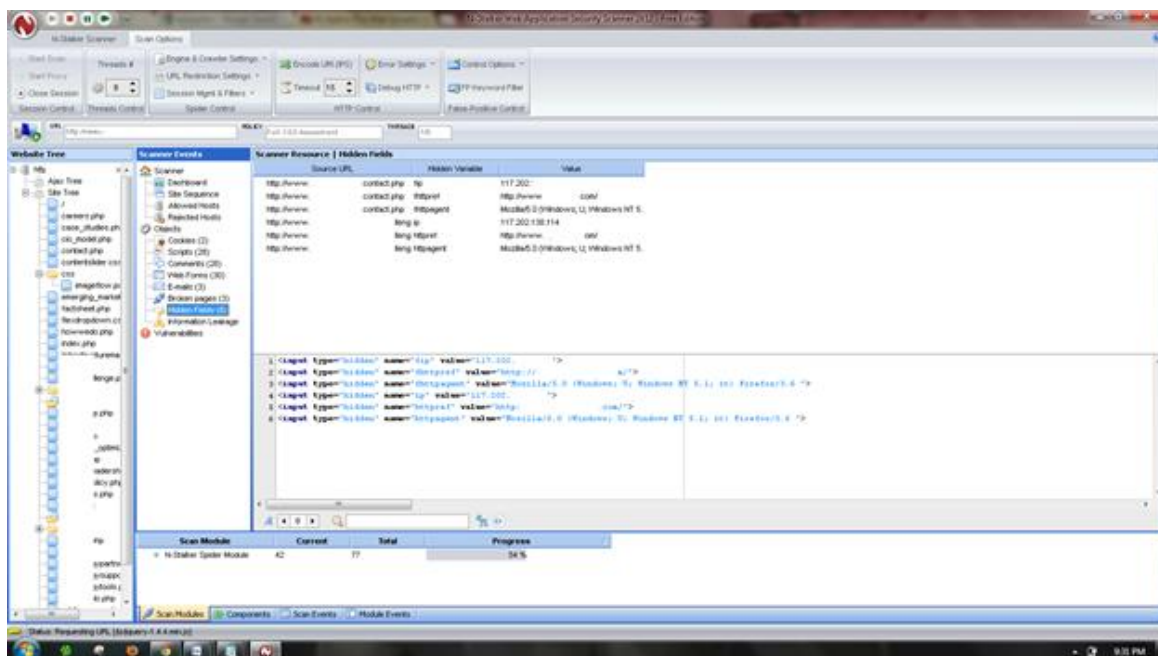
Review Summary



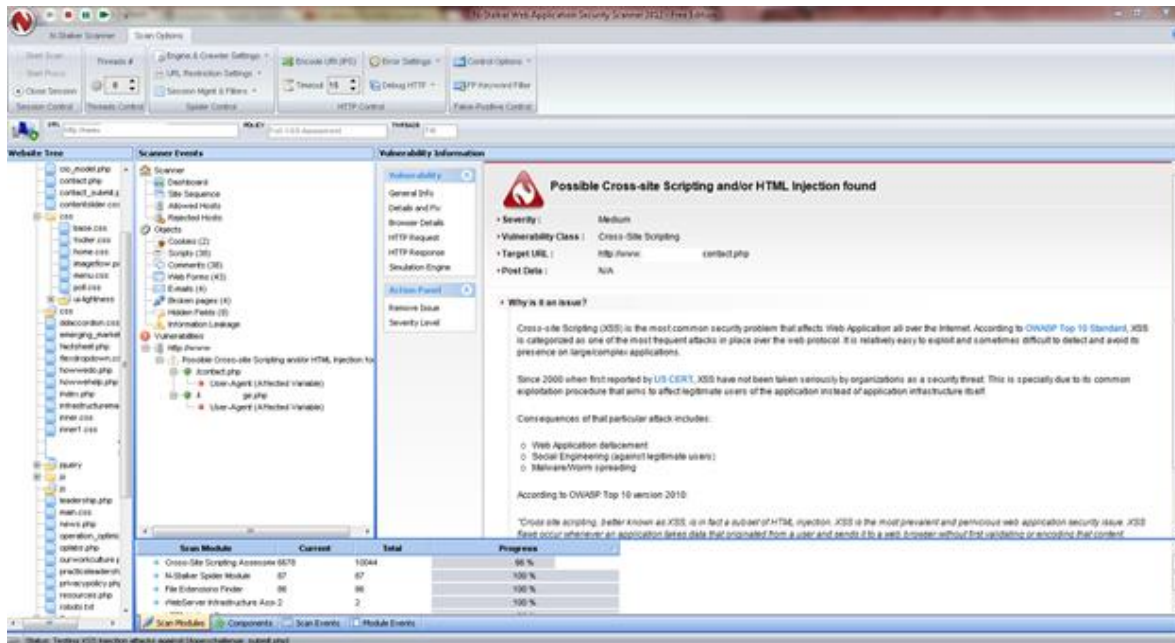
Scanning Settings

Scan Setting	Value
• Host Information	IP: [125.56.222.19] Port: [80] SSL: [no]
• Restricted Directory	Not configured.
• Policy Name	Spider Only
• False-Positive Settings	Enabled for Multiple Extensions. Enabled for 404 pages. Nk
• New Server Discovery	Enabled (recommended in most cases)
• Spider Engine	Max URLs: [500] Max Per Node [30] Max Depth [0]
• HTML Parser	JS: [Execute/Parse] External JS [Deny] JS Events [Execute]
• Server Technologies	N/A
• Allowed Hosts	No additional hosts configured.

The scanner will crawl the whole website and will show the scripts, broken pages, hidden fields, information leakage, web forms related information which helps to analyze further.



Once the scan is completed, the NStalker scanner will show details like severity level, vulnerability class, why is it an issue, the fix for the issue and the URL which is vulnerable to the particular vulnerability?



RESULT:

Thus the N-Stalker Vulnerability Assessment tool has been downloaded, installed and the features has been explored by using a vulnerable website.

Ex. No : 11
Date: 3/11/2020

Defeating Malware - Rootkit hunter

AIM:

To install a rootkit hunter and find the malwares in a computer.

ROOTKIT HUNTER:

- rkhunter (Rootkit Hunter) is a Unix-based tool that scans for rootkits, backdoors and possible local exploits.
- It does this by comparing SHA-1 hashes of important files with known good ones in online databases, searching for default directories (of rootkits), wrong permissions, hidden files, suspicious strings in kernel modules, and special tests for Linux and FreeBSD.
- rkhunter is notable due to its inclusion in popular operating systems (Fedora, Debian, etc.)
- The tool has been written in Bourne shell, to allow for portability. It can run on almost all UNIX-derived systems.

GMER ROOTKIT TOOL:

- GMER is a software tool written by a Polish researcher Przemysław Gmerek, for detecting and removing rootkits.
- It runs on Microsoft Windows and has support for Windows NT, 2000, XP, Vista, 7, 8 and 10. With version 2.0.18327 full support for Windows x64 is added.

Step 1

GMER <http://www.gmer.net>
 all your rootkits are belong to us [*]

Start

Files

News

Rootkits

FAQ

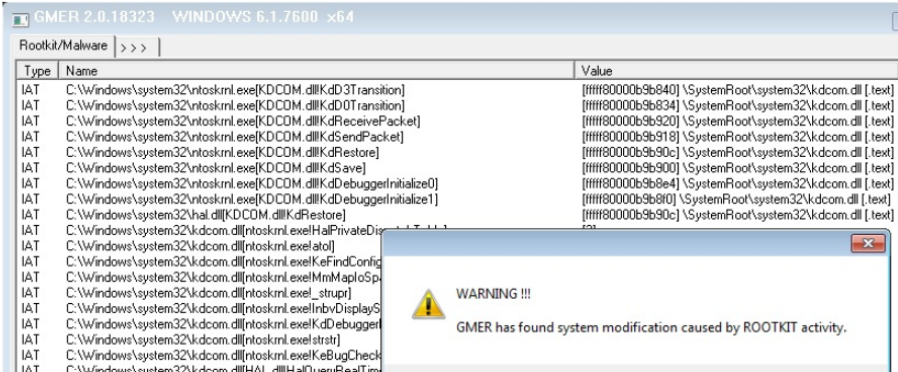
Contact

Start

GMER is an application that detects and removes rootkits .

It scans for:

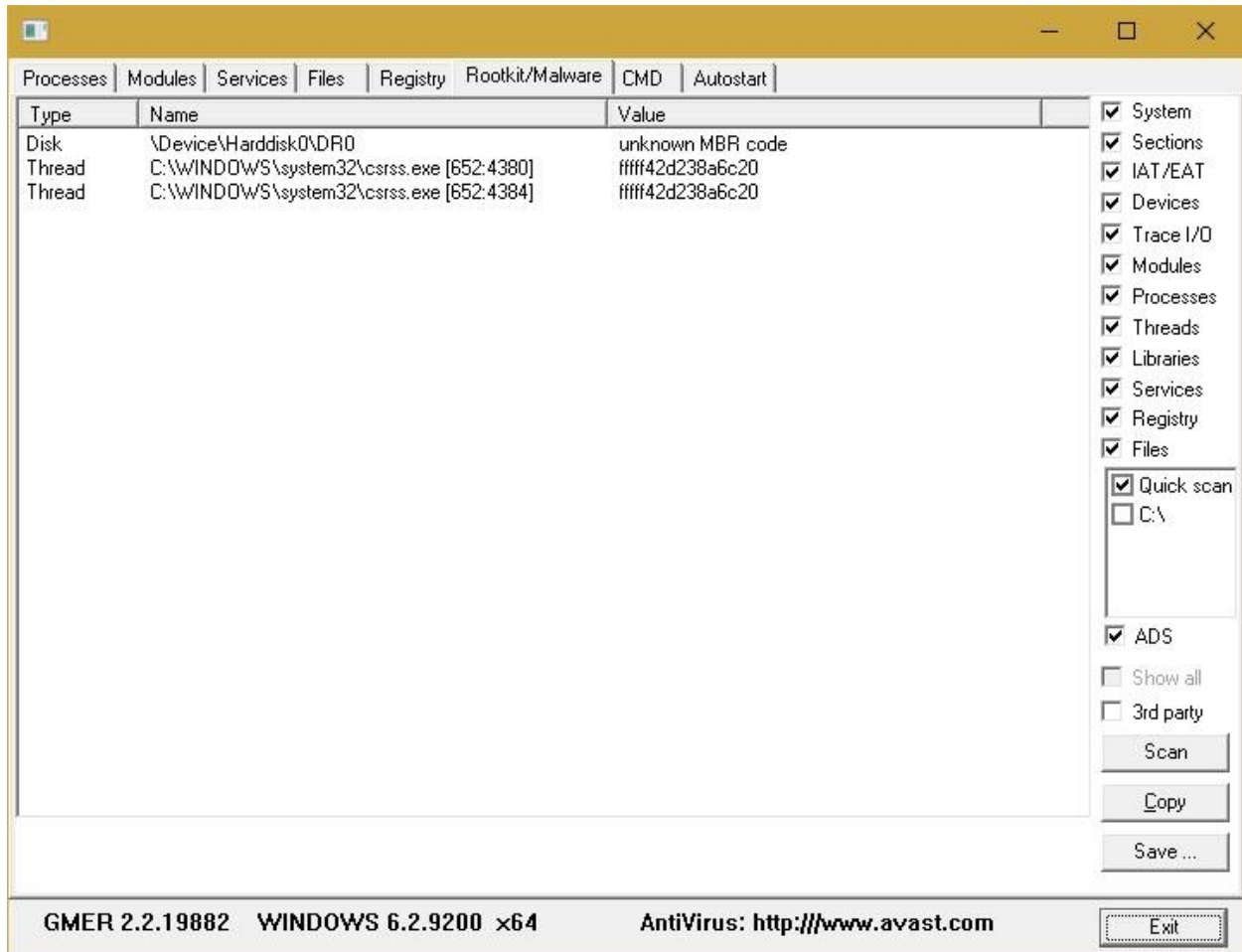
- hidden processes
- hidden threads
- hidden modules
- hidden services
- hidden files
- hidden disk sectors (MBR)
- hidden Alternate Data Streams
- hidden registry keys
- drivers hooking SSDT
- drivers hooking IDT
- drivers hooking IRP calls
- inline hooks



Visit GMER's website (see Resources) and download the GMER executable.

Click the "Download EXE" button to download the program with a random file name, as some rootkits will close "gmer.exe" before you can open it.

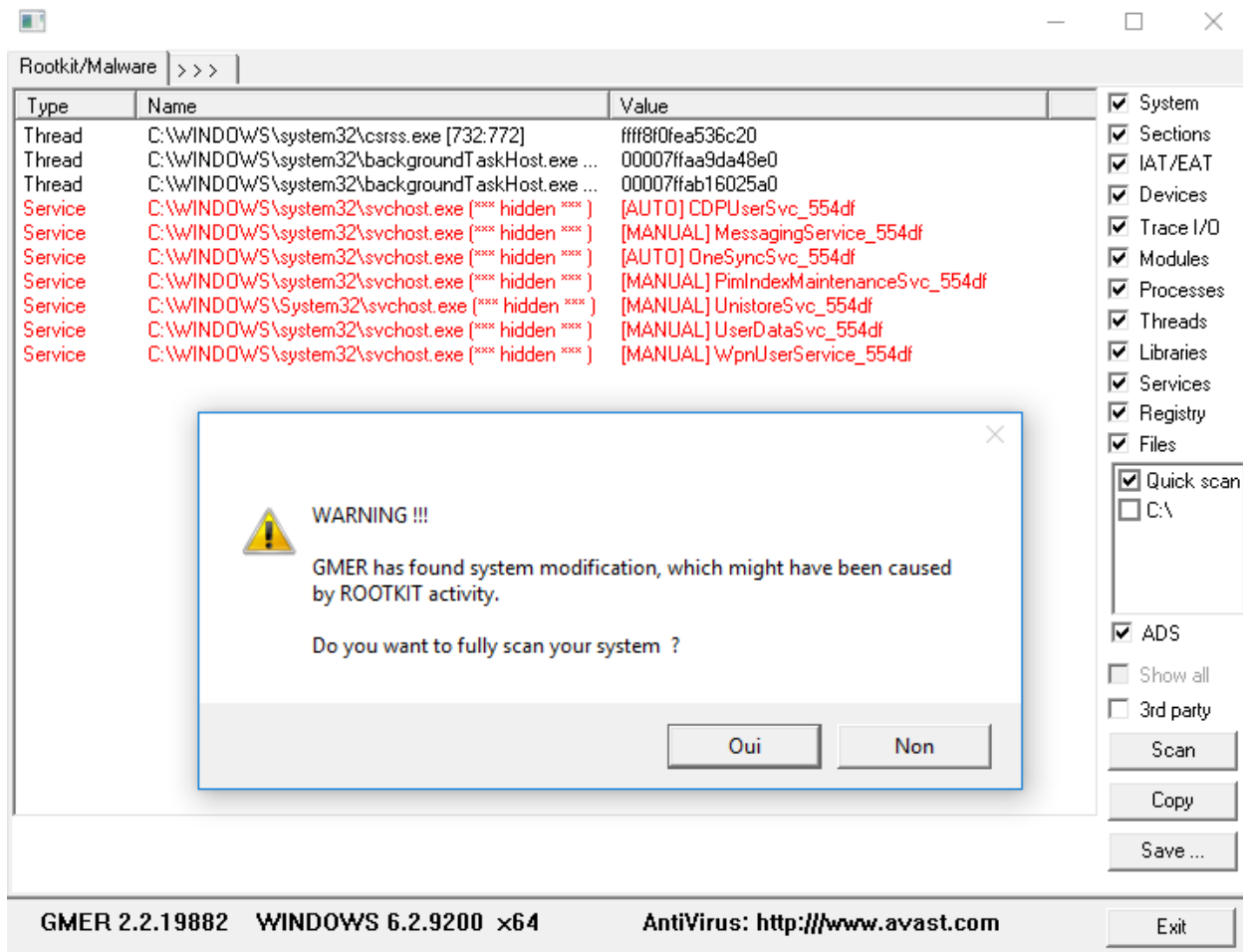
Step 2



Double-click the icon for the program.

Click the "Scan" button in the lower-right corner of the dialog box. Allow the program to scan your entire hard drive.

Step 3



When the program completes its scan, select any program or file listed in red. Right-click it and select "Delete."

If the red item is a service, it may be protected. Right-click the service and select "Disable." Reboot your computer and run the scan again, this time selecting "Delete" when that service is detected.

When your computer is free of Rootkits, close the program and restart your PC.

RESULT:

In this experiment a rootkit hunter software tool has been installed and the rootkits have been detected.