

IT6712 SECURITY LABORATORY SYLLABUS Regulation 2013

OBJECTIVES:

The student should be made to:

Step 1. Be exposed to the different cipher techniques.

Step 2. Learn to implement the algorithms DES, RSA, MD5, SHA-1.

Step 3. Learn to use network security tools like GnuPG, KF sensor, Net Strumbler.

LIST OF EXPERIMENTS:

1 Implement the following SUBSTITUTION & TRANSPOSITION TECHNIQUES concepts

- a) Caesar Cipher
- b) Playfair Cipher
- c) Hill Cipher
- d) Vigenere Cipher
- e) Rail fence – row & Column Transformation

2 Implement the following algorithms

- a) DES
- b) RSA Algorithm
- c) Diffie-Hellman
- d) MD5
- e) SHA-1

3 Implement the SIGNATURE SCHEME - Digital Signature Standard

4 Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG)

5 Setup a honey pot and monitor the honeypot on network (KF Sensor)

6 Installation of rootkits and study about the variety of options

7 Perform wireless audit on an access point or a router and decrypt WEP and WPA (Net Stumbler)

8 Demonstrate intrusion detection system (ids) using any tool (snort s/w)

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

- i) Implement the cipher techniques
- ii) Develop the various security algorithms
- iii) Use different open source tools for network security and analysis

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

C / C++ / Java or equivalent compiler

GnuPG, KF Sensor or Equivalent, Snort, Net Stumbler or Equivalent

HARDWARE:

Standalone desktops 30 Nos.(or)

Server supporting 30 terminals or more.

EX.1(A) IMPLEMENTATION OF CAESAR CIPHER

AIM:

To implement the simple substitution technique named Caesar cipher using c or java program.

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the key from the user.

STEP-3: If the key is positive then encrypt the text by adding the key with each character in the plain text.

STEP-4: Else subtract the key from the plain text.

STEP-5: Display the cipher text obtained.

PROGRAM: (Caesar Cipher)

```
#include <stdio.h> // Header files for Input and Output functions
#include<conio.h> // Header files for Console Input and Output functions
void main() //Starting point of the program
{
int i, x; // variable declaration of type integer
char str[100]; // variable declaration of type string
printf("\nPlease enter a string:\t"); //input the string to be encrypted
gets(str); //read the string
printf("\nPlease choose following options:\n"); //user's choice
printf("1 = Encrypt the string.\n");
printf("2 = Decrypt the string.\n");
scanf("%d", &x);
//using switch case statements
switch(x)
{
case 1: for(i = 0; (i < 100 && str[i] != '\0'); i++)
str[i] = str[i] + 3; //the key for encryption is 3 that is added to ASCII value
printf("\nEncrypted string: %s\n", str);
```

```

//break;

case 2: for(i = 0; (i < 100 && str[i] != '\0'); i++)

str[i] = str[i] - 3; //the key for encryption is 3 that is subtracted to ASCII value

printf("\nDecrypted string: %s\n", str);

break;

default:

printf("\nError\n"); //exceptional cases

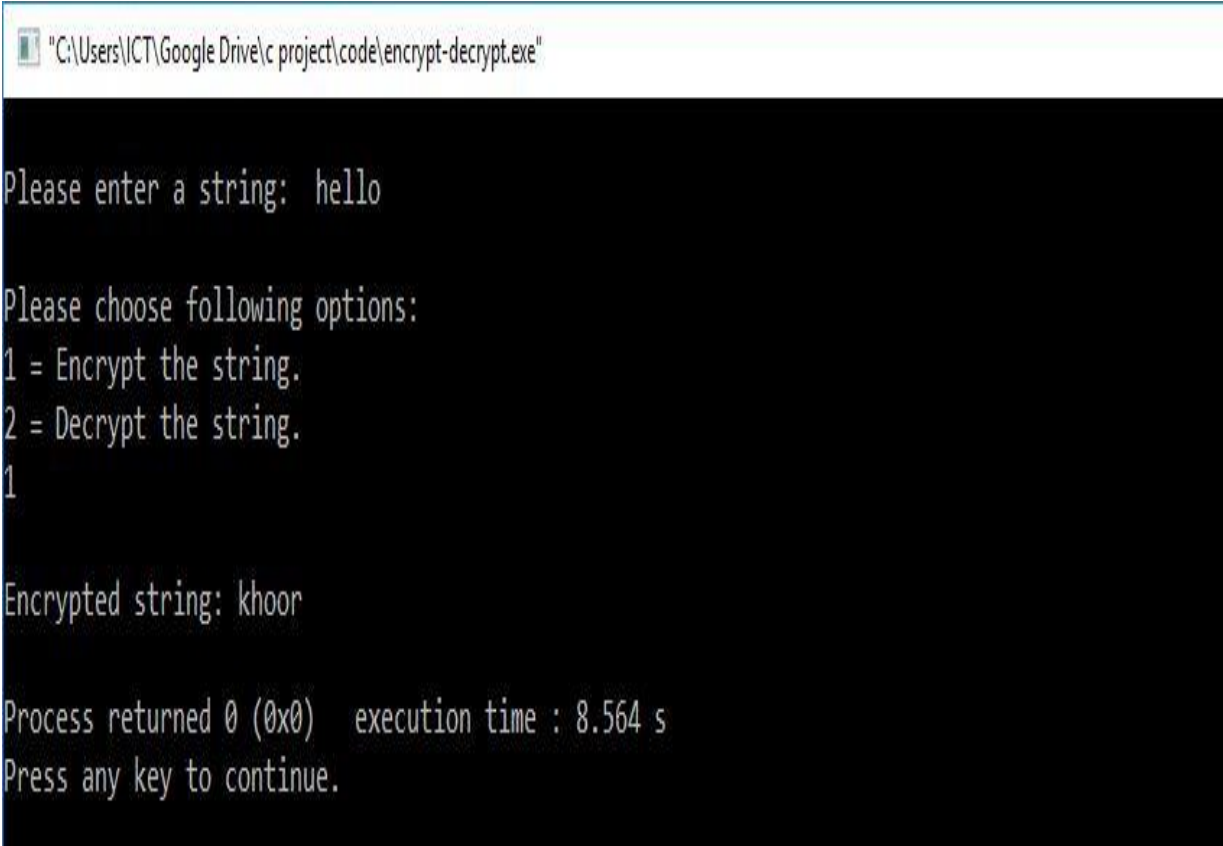
}

getch(); // Display the output screen

}

```

Sample Output



The screenshot shows a Windows command prompt window with the title bar "C:\Users\ICT\Google Drive\c project\code\encrypt-decrypt.exe". The window has a black background with white text. The text inside the window is as follows:

```

Please enter a string: hello

Please choose following options:
1 = Encrypt the string.
2 = Decrypt the string.
1

Encrypted string: khoor

Process returned 0 (0x0)   execution time : 8.564 s
Press any key to continue.

```

"C:\Users\ICT\Google Drive\c project\code\encrypt-decrypt.exe"

Please enter a string: kloor

Please choose following options:

1 = Encrypt the string.

2 = Decrypt the string.

2

Decrypted string: hello

Process returned 0 (0x0) execution time : 4.288 s

Press any key to continue.

RESULT:

Thus the implementation of Caesar cipher had been executed successfully

EXP:1(B) IMPLEMENTATION OF PLAYFAIR CIPHER

AIM:

To implement the simple substitution technique named Playfair cipher using c or java program

ALGORITHM:

STEP-1: Read the plain text from the user.

STEP-2: Read the keyword from the user.

STEP-3: Arrange the keyword without duplicates in a 5*5 matrix in the row order and fill the remaining cells with missed out letters in alphabetical order. Note that 'i' and 'j' takes the same cell.

STEP-4: Group the plain text in pairs and match the corresponding corner letters by forming a rectangular grid.

STEP-5: Display the obtained cipher text.

PROGRAM: (Playfair Cipher)

```
#include < stdio.h > // Header files for Input and Output functions
#include < conio.h > // Header files for Console Input and Output functions
#include < string.h > // Header files for String functions
#define MX 5
void playfair(char ch1, char ch2, char key[MX][MX]) //function
{
    int i, j, w, x, y, z;
    FILE * out;
    if ((out = fopen("cipher.txt", "a+")) == NULL) {
        printf("File Corrupted...");
    }
    for (i = 0; i < MX; i++) {
        for (j = 0; j < MX; j++) {
            if (ch1 == key[i][j]) {
                w = i;
                x = j;
            } else if (ch2 == key[i][j]) {
                y = i;
                z = j;
            }
        }
    }
    if (w == y) {
        x = (x + 1) % 5;
        z = (z + 1) % 5;
        printf("%c%c", key[w][x], key[y][z]);
        fprintf(out, "%c%c", key[w][x], key[y][z]);
    } else if (x == z) {
```

```

w = (w + 1) % 5;
y = (y + 1) % 5;
printf("%c%c", key[w][x], key[y][z]);
fprintf(out, "%c%c", key[w][x], key[y][z]);
} else {
printf("%c%c", key[w][z], key[y][x]);
fprintf(out, "%c%c", key[w][z], key[y][x]);
}
fclose(out);
}
void main() //Starting point of the program
{
int i, j, k = 0, l, m = 0, n;
char key[MX][MX], keyminus[25], keystr[10], str[25] = {
0
};
char
alpa[26]={ 'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V',
'W','X','Y','Z' };
printf("\nEnter key:");
gets(keystr); //read key from user
printf("\nEnter the plain text:");
gets(str); //read plain text from user
n = strlen(keystr); //get the length
//convert the characters to uppertext
for (i = 0; i < n; i++) {
if (keystr[i] == 'j') keystr[i] = 'i';
else if (keystr[i] == 'J') keystr[i] = 'T';
keystr[i] = toupper(keystr[i]);
}
//convert all the characters of plaintext to uppertext
for (i = 0; i < strlen(str); i++) {
if (str[i] == 'j') str[i] = 'i';
else if (str[i] == 'J') str[i] = 'T';
str[i] = toupper(str[i]);
}
// store all characters except key
j = 0;
for (i = 0; i < 26; i++) {
for (k = 0; k < n; k++) {
if (keystr[k] == alpa[i]) break;
else if (alpa[i] == 'J') break;
}
if (k == n) {
keyminus[j] = alpa[i];
j++;
}
}
}

```

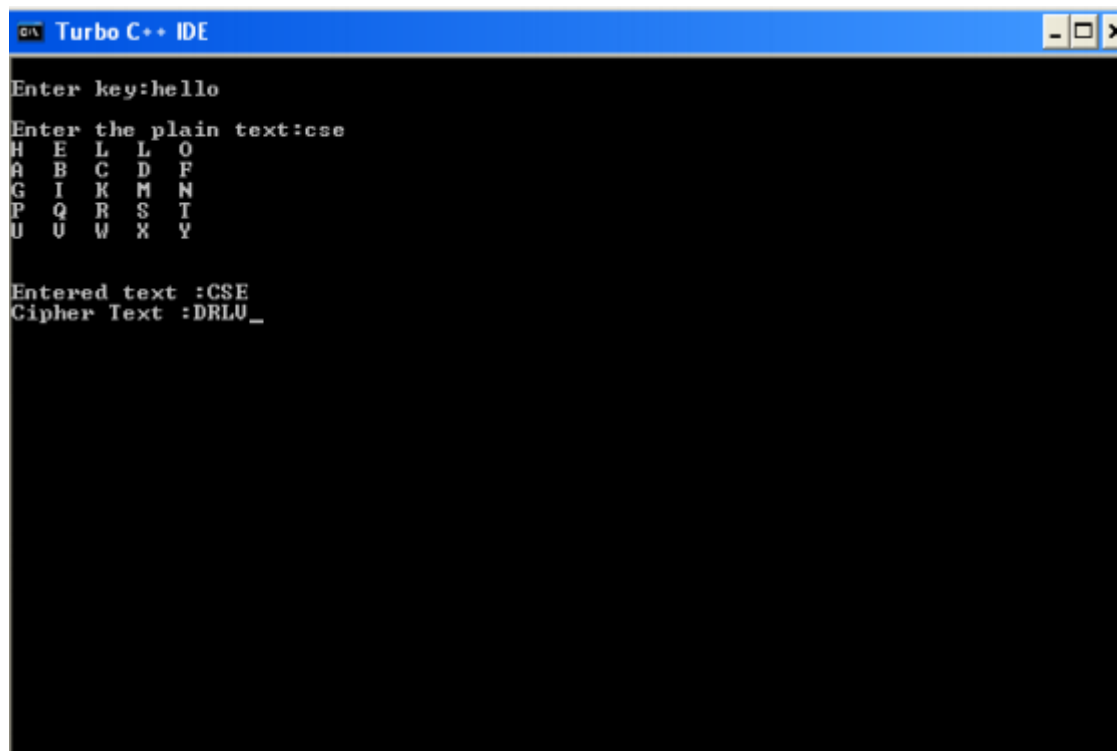
//construct key keymatrix

```
k = 0;
for (i = 0; i < MX; i++) {
for (j = 0; j < MX; j++) {
if (k < n) {
key[i][j] = keystr[k];
k++;
} else {
key[i][j] = keyminus[m];
m++;
}
printf("%c ", key[i][j]);
}
printf("\n");
}
```

// construct diagram and convert to cipher text

```
printf("\n\nEntered text :%s\nCipher Text :", str);
for (i = 0; i < strlen(str); i++) {
if (str[i] == 'J') str[i] = 'I';
if (str[i + 1] == '\0') playfair(str[i], 'X', key);
else {
if (str[i + 1] == 'J') str[i + 1] = 'I';
if (str[i] == str[i + 1]) playfair(str[i], 'X', key);
else {
playfair(str[i], str[i + 1], key);
i++;
}
} }
getch();
}
```


Sample output:



```
Enter key:hello
Enter the plain text:cse
H E L L O
A B C D F
G I K M N
P Q R S T
U V W X Y

Entered text :CSE
Cipher Text :DRLU_
```

RESULT:

Thus the Playfair cipher substitution technique had been implemented successfully.

EX. NO: 1(C)

IMPLEMENTATION OF HILL CIPHER

AIM:

To write a C program to implement the hill cipher substitution techniques.

ALGORITHM:

STEP-1: Read the plain text and key from the user.

STEP-2: Split the plain text into groups of

STEP-3: Arrange the keyword in a 3*3 matrix.

STEP-4: Multiply the two matrices to obtain the cipher text of length three

STEP-5: Combine all these groups to get the complete cipher text

PROGRAM: (Hill Cipher)

```
#include<stdio.h>//HEADER FILE FOR STANDARD INPUT AND OUTPUT

#include<conio.h>//HEADER FILE FOR CONSOLE INPUT AND OUTPUT

#include<string.h>//HEADER FILE FOR MANIPULATING CHARS

int main();//EXECUTION ENTRY POINT

{ //OPEN BRACES

unsigned int a[3][3]={ {6,24,1},{13,16,10},{20,17,15}};//REP ONLY POSITIVE NO'S AND
ZEROS

unsigned int b[3][3]={ {8,5,10},{21,8,21},{21,12,8}};// REP ONLY POSITIVE NO'S AND
ZEROS

int i,j, t=0; //DEFINES MEMORY LOCATION

unsigned int c[20],d[20];// REP ONLY POSITIVE NO'S AND ZEROS

char msg[20];//STRING LIKE A CHAR MSG
clrscr();//PREDEFINED FN IN CONSOLE

printf("Enter plain text");//PRINT FORMATTED

scanf("%s",msg);//READS WITH SPECIFIED OUTPUT

for(i=0;i<strlen(msg);i++)//FOR LOOP
{
c[i]=msg[i]-65;//DECLARATION

printf("%d ",c[i]);// PRINT FORMATTED
}
```

```

for(i=0;i<3;i++)//FOR LOOP

{

t=0;

for(j=0;j<3;j++)//FOR LOOP

{

t=t+(a[i][j]*c[j]);//CONDITION

}

d[i]=t%26;

}

printf("\nEncrypted Cipher Text :");//PREDEFINED FN IN CONSOLE

for(i=0;i<3;i++)//FOR LOOP

printf(" %c",d[i]+65);//PREDEFINED FN IN CONSOLE

for(i=0;i<3;i++)//FOR LOOP

{

t=0;//DECLARATION

for(j=0;j<3;j++)//FOR LOOP

{

t=t+(b[i][j]*d[j]);

}

c[i]=t%26;//CONDITION

}

printf("\nDecrypted Cipher Text :");//PREDEFINED FN IN CONSOLE

for(i=0;i<3;i++)//FOR LOOP

printf(" %c",c[i]+65);//PREDEFINED FN IN CONSOLE

getch();//PREDEFINED FN IN CONSOLE

return 0;//RETURNS AN INTEGER VALUE

}

```

OUTPUT:

ENTER THE PLAIN TEXT:

**A C T
0 2 19**

**ENCRYPTED CIPHER TEXT:P O H
DECRYPTED CIPHER TEXT:A C T**

Result:

Thus, the hill cipher encryption technique has been implemented successfully.

Ex.no:1d)

VIGENERE CIPHER

AIM:

To write a C program to implement Vigenere Cipher.

Description:

Vigenere Cipher is kind of polyalphabetic substitution method. It is used for encryption of alphabetic text. For encryption and decryption Vigenere Cipher Table is used in which alphabets from A to Z are written in 26 rows.

ALGORITHM:

Step 1: Vigenere Cipher Encryption

Message Text:SECURITYLAB

Key: HELLO

Step 2: Here we have to obtain a new key by repeating the given key till its length become equal to original message length.

New Generated Key:HELLOHELLOH

Step 3: a)For encryption take first letter of message and new key i.e. S and H. Take the alphabet in Vigenere Cipher Table where S row and H column coincides i.e. Z.

b)Repeat the same process for all remaining alphabets in message text. Finally the encrypted message text is:ZINFFPXJWOI

c)Encrypted Message:

The algorithm can be expressed in algebraic form as given below. The cipher text can be generated by below equation.

$$E_i = (P_i + K_i) \bmod 26$$

Here P is plain text and K is key.

Step 4: Vigenere Cipher Decryption

Encrypted Message:ZINFFPXJWOI

Key: HELLO

New Generated Key:HELLOHELLOH

a)Take first alphabet of encrypted message and generated key i.e. Z and H. Analyze Vigenere Cipher Table, look for alphabet Z in column H, the corresponding row will be the first alphabet of original message i.e. S.Repeat the same process for all the alphabets in encrypted message.

Original Message:SECURITYLAB

b) Above process can be represented in algebraic form by following equation.

$$P_i = (E_i - K_i + 26) \bmod 26$$

We will use above algebraic equations in the program.

VIGNERE CIPHER TABLE:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

PROGRAM:

```
#include<stdio.h>    //include the necessary header files
```

```
#include<string.h>
```

```
/*This is the main() function.
```

```
*The compiled program will run the code defined here
```

```
*/
```

```
void main()
```

```
{
```

```
//declaring variables
```

```
char msg[100];
```

```
char key[100];
```

```
int msgLen = strlen(msg), keyLen = strlen(key), i, j;
```

```
char newKey[100],encryptedMsg[100],decryptedMsg[100];
```

```

clrscr();           //clears the console screen

printf("Original text:"); //prompt user for the original text

scanf("%s",&msg);    //use %s to read the string

printf("\nKey:");     //prompt user for the key

scanf("%s",&key);     //use %s to read the string

//generating new key

//Execute for loop with condition i<msgLen

for(i = 0, j = 0; i < msgLen; ++i, ++j)

{

if(j == keyLen)      //check whether j equals to KeyLen

j = 0;              //assign j=0

newKey[i] = key[j]; //new key gets generated

}

newKey[i] = '\0';    //denotes the end of new key

//encryption

//Execute the for loop with condition i<msgLen

for(i = 0; i < msgLen; ++i)

//Execute the for loop with condition i<msgLen

encryptedMsg[i] = ((msg[i] + newKey[i]) % 26) + 'A';

encryptedMsg[i] = '\0'; //denotes end of the string

//decryption

for(i = 0; i < msgLen; ++i)

//Execute the for loop with condition i<msgLen

decryptedMsg[i] = (((encryptedMsg[i] - newKey[i]) + 26) % 26) + 'A';

decryptedMsg[i] = '\0'; //denotes end of the string

//Print the new key,encrypted msg,decrypted msg

printf("\nNew Generated Key: %s",&newKey);

printf("\nEncrypted Message: %s",&encryptedMsg);

printf("\nDecrypted Message: %s",&decryptedMsg);

getch();

```

}

EXPECTED OUTPUT:

```
Original text:SECURITYLAB  
Key:HELLO  
New Generated Key: HELLOHELLOH  
Encrypted Message: ZINFFPZJWOI  
Decrypted Message: SECURITYLAB
```

RESULT:

Thus the C program of Vignere Cipher has been implemented successfully.

Ex.No:1E IMPLEMENTATION OF RAILFENCE-ROW AND COLUMN TRANSFORMATION TECHNIQUE

Aim:

To write a C program to implement the railfence transposition technique.

Algorithm :

Step1:Read the plaintext

Step2:Arrange the plaintext in row column matrix format.

Step3:Now read the keyword depending on the number of columns of the plaintext.

Step4:Arrange the characters of the keyword in sorted order and the corresponding columns of the plaintext.

Step5:Read the characters row wise or column wise in the former order to get the cipher text.

Program:

```
#include<stdio.h>

#include<conio.h>
#include<string.h>//to handle string functions//
void main()
{
//declare necessary variables//
int i,j,k,l;
char a[20],c[20],d[20];
//to clear the console screen//
clrscr();

printf("\n\t\tRAILFENCE TECHNIQUE");

printf("\n\nEnter the input string:");
//to get the input//
gets(a);
//initialize the length of the string//
l=strlen(a);
/*Ciphering*/
//to print the string in a zigzag manner//
for(i=0,j=0;i<l;i++)
{
if(i%2==0)
c[j++]=a[i];
}
for(i=0;i<l;i++)
{
if(i%2==1)
c[j++]=a[i];
}
//executing loop till the string ends with \0//
c[j]='\0';
printf("\nCipher text after applying railfence:");
```

```

//to print the ciphertext//
printf("\n%s",c);
/*Deciphering*/
if(l%2==0)

k=l/2;//initialize key value//

else
k=(l/2)+1;
//arrange the ciphertext into a rail with respect to key//
for(i=0,j=0;i<k;i++)
{
d[j]=c[i];
j=j+2;
}
//to print the string column wise until reach the key value//
for(i=k,j=1;i<l;i++)
{
d[j]=c[i];
j=j+2;
}
//executing the loop until the string ends with \0//
d[l]='\0';
printf("\nText after decryption:");
//to display the decrypted text//
printf("%s",d);
getch();

}

```

Sample output:

RAILFENCE TECHNIQUE

Enter the input string: computerscience

Ciphertext after applying rail fence: cmuececoptsrine

Text after decryption: computerscience

Result:

Thus the rail fence algorithm had been executed successfully

EX. NO: 2(A)**IMPLEMENTATION OF DES****AIM:**

To write a C program to implement Data Encryption Standard (DES) using C Language.

DESCRIPTION:

DES is a symmetric encryption system that uses 64-bit blocks, 8 bits of which are used for parity checks. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm. The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions. The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted k1 to k16. Given that "only" 56 bits are actually used for encrypting, there can be 256 different keys.

The main parts of the algorithm are as follows:

- Fractioning of the text into 64-bit blocks
- Initial permutation of blocks
- Breakdown of the blocks into two parts: left and right, named L and R
- Permutation and substitution steps repeated 16 times
- Re-joining of the left and right parts then inverse initial permutation

ALGORITHM:

STEP-1: Read the 64-bit plain text.

STEP-2: Split it into two 32-bit blocks and store it in two different arrays.

STEP-3: Perform XOR operation between these two arrays.

STEP-4: The output obtained is stored as the second 32-bit sequence and the original second 32-bit sequence forms the first part.

STEP-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 javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random ;
class DES {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String inputMessage,encryptedData,decryptedMessage;
public DES()
{
try
{
generateSymmetricKey();
inputMessage=JOptionPane.showInputDialog(null,"Enter
message to encrypt");
byte[] ibyte = inputMessage.getBytes();
byte[] ebyte=encrypt(raw, ibyte);
String encryptedData = new String(ebyte);
```

```

System.out.println("Encrypted message "+encryptedData);
JOptionPane.showMessageDialog(null,"Encrypted Data
"+"\\n"+encryptedData);
byte[] dbyte= decrypt(raw,ebyte);
String decryptedMessage = new String(dbyte);
System.out.println("Decrypted message
"+decryptedMessage);
JOptionPane.showMessageDialog(null,"Decrypted Data
"+"\\n"+decryptedMessage);
}
catch(Exception e)
{
System.out.println(e);
}
}
void generateSymmetricKey() {
try {
Random r = new Random();
int num = r.nextInt(10000);
String knum = String.valueOf(num);
byte[] knumb = knum.getBytes();
skey=getRawKey(knumb);
skeyString = new String(skey);
System.out.println("DES Symmetric key = "+skeyString);
}
catch(Exception e)
{
System.out.println(e);
}
}
private static byte[] getRawKey(byte[] seed) throws Exception
{
KeyGenerator kgen = KeyGenerator.getInstance("DES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(56, sr);
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
private static byte[] encrypt(byte[] raw, byte[] clear) throws
Exception {
SecretKeySpec keySpec = new SecretKeySpec(raw,
"DES");
Cipher cipher = Cipher.getInstance("DES");
cipher.init(Cipher.ENCRYPT_MODE, keySpec);
byte[] encrypted = cipher.doFinal(clear);
return encrypted;
}
private static byte[] decrypt(byte[] raw, byte[] encrypted)
throws Exception
{
SecretKeySpec keySpec = new SecretKeySpec(raw,
"DES");
Cipher cipher = Cipher.getInstance("DES");

```

```
cipher.init(Cipher.DECRYPT_MODE, keySpec);
byte[] decrypted = cipher.doFinal(encrypted);
return decrypted;
}
public static void main(String args[]) {
DES des = new DES();
}
}
```

OUTPUT:

Data Before Encryption :Classified Information!
Encrypted Data: [B@bc6007
Decrypted Data: Classified Information!

RESULT:

Thus the data encryption standard algorithm had been implemented successfully using C language.

EX. NO: 2(B)**IMPLEMENTATION OF RSA****AIM:**

To write a Java program to implement the RSA encryption algorithm.

DESCRIPTION :

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric means that it works on two different keys i.e. Public Key and Private Key. As the name suggests that the Public Key is given to everyone and Private Key is kept private.

ALGORITHM:

STEP-1: Select two co-prime numbers as p and q.

STEP-2: Compute N as the product of p and q.

STEP-3: Compute $(p-1)*(q-1)$ and store it in phi.

STEP-4: Select a random prime number e that is less than that of phi.

STEP-5: Compute the private key, d as $e^{-1} \pmod{\phi}$.

STEP-6: The cipher text is computed as $m^e \pmod{n}$

STEP-7: Decryption is done as $c^d \pmod{n}$.

PROGRAM :

```
import java.util.*;
import java.math.*;
class RSA
{
public static void main(String args[])
{
Scanner sc=new Scanner(System.in);
int p,q,n,z,d=0,e,i;
System.out.println("Enter the number to be encrypted and decrypted");
int msg=sc.nextInt();
double c;
BigInteger msgback;
//Choose two prime numbers p and q
System.out.println("Enter 1st prime number p");
p=sc.nextInt();
System.out.println("Enter 2nd prime number q");
q=sc.nextInt();
n=p*q;
z=(p-1)*(q-1);
System.out.println("the value of z = "+z);
for(e=2;e<z;e++)
{
//e is for public key exponent
if(gcd(e,z)==1)
{
break;
}
}
System.out.println("the value of e = "+e);
for(i=0;i<=9;i++)
{
int x=1+(i*z);
if(x%e==0)
{
//d is for private key exponent
```

```

        d=x/e;
        break;
    }
}
System.out.println("the value of d = "+d); c=(Math.pow(msg,e))%n;
System.out.println("Encrypted message is : -");
System.out.println(c);
//converting integer value of n to BigInteger
BigInteger N = BigInteger.valueOf(n);
//converting float value of c to BigInteger
BigInteger C = BigDecimal.valueOf(c).toBigInteger();
msgback = (C.pow(d)).mod(N);
System.out.println("Derypted message is :");
System.out.println(msgback);
}
static int gcd(int e, int z)
{
    if(e==0)
        return z;
    else
        return gcd(z%e,e);
}
}

```

OUTPUT :

```

Enter the number to be encrypted and decrypted :
12
Enter 1st prime number p :
3
Enter 2nd prime number q :
11
the value of z = 20
the value of e = 3
the value of d = 7
Encrypted message is :- 12.0
Derypted message is :- 12.0

```

RESULT:

Thus the Java program to implement RSA encryption technique had been implemented successfully

EX. NO: 2(C) IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

AIM:

To implement the Diffie-Hellman Key Exchange algorithm using Java.

DESCRIPTION:

Diffie–Hellman Key Exchange algorithm establishes a shared secret between two parties that can be used for communication for exchanging data over a public network. It is primarily used as a method of exchanging cryptography keys for use in symmetric encryption algorithms. The algorithm in itself is very simple. The process begins by having the two parties, Sender(User 1) and Receiver(User 2). Let's assume that User 1 wants to establish a shared secret with User 2.

ALGORITHM:

STEP-1: Both User 1 and User 2 shares the same public keys g and p .

STEP-2: User 1 selects a random public key x .

STEP-3: User 1 computes his secret key A as $gx \bmod p$.

STEP-4: Then User 1 sends A to User 2.

STEP-5: Similarly User 2 also selects a public key y and computes his secret key as B and sends the same back to User 1.

STEP-6: Now both of them compute their common secret key as the other one's secret key power of a mod p .

PROGRAM:

```
package security;
import java.util.*;

class Diffie_Hellman
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter a number(prime)");
        //get the prime number from user
        int p=sc.nextInt();
        System.out.println("Enter primitive root of "+p);
        //get the primitive root of the prime number chosen
        int g=sc.nextInt();
        System.out.println("Choose 1st secret no(User 1)");
        //selecting secret integer a which is the private key
        int a=sc.nextInt();
        System.out.println("Choose 2nd secret no(User 2)");
        //select secret integer b which is the private key
        int b=sc.nextInt();
        //from a calculating the public key X
        int X = (int)Math.pow(g,a)%p;
        //from b calculating the public key Y
        int Y = (int)Math.pow(g,b)%p;
        //User 1 and User 2 exchange values of X and Y to compute symmetric keys(secret key)
        int S_X = (int)Math.pow(Y,a)%p;
        int S_Y =(int)Math.pow(X,b)%p;
        //if both secret numbers equals,communication is established
        if(S_X==S_Y)
        {
            System.out.println("User 1 and User 2 can communicate with each other!!!");
        }
    }
}
```



```
System.out.println("They share a secret no = "+S_X);
}

else
{
System.out.println("User 1 and User 2 cannot communicate with each other!!!");
}
}
}
```

OUTPUT

```
Enter a number(prime):
11
Enter primitive root of 11:7
Choose 1st secret no(User 1):
3
Choose 2nd secret no(User 2):
6
X=2
Y=4
User 1 and User 2 can communicate with each other!!!
They share a secret key 9.
```

RESULT:

Thus the Diffie-Hellman key exchange algorithm had been successfully implemented using Java.

EX. NO: 2(E)**IMPLEMENTATION OF MD5****AIM:**

To write a C program to implement the MD5 hashing technique.

DESCRIPTION:

MD5 processes a variable-length message into fixed-length output of 128bits. The input message is broken up into chunks of 512-bit blocks. The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up 64 bits representing the length of the original message, modulo 2^{64} . The main MD5 algorithm operates on a 128-bit state, divided into four 32 denoted A,B,C,D. These are initialized to certain fixed constants. The main algorithm then uses each 512-bit message block in turn to modify the state.

ALGORITHM:

STEP-1:Read the 128-bit plain text.

STEP-2:Divide into four blocks of 32-bits named as A,B,Cand D.

STEP-3: Compute the functions f, g, h and i with operations such as, rotations, permutations, etc.,

STEP-4: The output of these functions are combined together as F and performed circular shifting and then given to key round.

STEP-5: Finally, right shift of 's' times are performed and the results are combined together to produce the final output.

PROGRAM:

```
#include <stdlib.h> //general utility function

#include <stdio.h> //input/output function

#include <string.h> //string function

#include <math.h> //mathematic function

#include <conio.h> //console input/output

typedef union uwb //which allows grouping together related data Items of different types
{

    unsigned w;    //An unsigned variable type of int can hold zero and positive numbers

    unsigned char b[4]; //store value from 0 to 255

}
```

```

MD5union;//Special datatype

typedef unsigned DigestArray[4];//define new datatype

unsigned func0( unsigned abcd[] )//It can hold zero and positive numbers
{
    return ( abcd[1] & abcd[2]) | (~abcd[1] & abcd[3]);//Terminate the execution of a function
}

unsigned func1( unsigned abcd[] )//It can hold zero and positive numbers
{
    return ( abcd[3] & abcd[1]) | (~abcd[3] & abcd[2]);//Terminate the execution of a function
}

unsigned func2( unsigned abcd[] )//it can hold zero and positive numbers
{
    return abcd[1] ^ abcd[2] ^ abcd[3];//Terminate the execution of a function
}

unsigned func3( unsigned abcd[] ){ return abcd[2] ^ (abcd[1] |~ abcd[3]);//It can hold zero and
positive number
}

typedef unsigned (*DgstFctn)(unsigned a[]);//which allows grouping together related data
items of different types

unsigned *calctable( unsigned *k)//it can hold zero and positive numbers
{
    double s, pwr;//it represent fractional as well as whole values

    int i; //define numeric variable holding whole numbers

    pwr = pow( 2, 32);//computes the base number raised to the power of exponent number

    for (i=0; i<64; i++)//it enables us to perform n number of steps together in one line
    {
        s = fabs(sin(1+i));//fabs-absolute value of the argument,sin()-sine of an angle given in radians

        k[i] = (unsigned)( s * pwr );//it can hold zero or positive numbers
    }

    return k; //terminate the execution of a function

```

```

}

unsigned rol( unsigned r, short N )//it can hold zero or positive numbers
{
unsigned mask1 = (1<<N) -1; //it can hold zero or positive numbers

return ((r>>(32-N)) & mask1) | ((r<<N) & ~mask1);//terminate the execution of a function
}

unsigned *md5( const char *msg, int mlen//it can hold zero or positive numbers
{

static DigestArray h0 = { 0x67452301, 0xEFCDAB89, 0x98BADCFE, 0x10325476
};//initialized array

static DgstFctn ff[] = { &func0, &func1, &func2, &func3};//initialized func

static short M[] = { 1, 5, 3, 7 }; //initialized short
static short O[] = { 0, 1, 5, 0 }; //initialized short
static short rot0[] = { 7,12,17,22}; //initialized short
static short rot1[] = { 5, 9,14,20}; //initialized short
static short rot2[] = { 4,11,16,23}; //initialized short
static short rot3[] = { 6,10,15,21}; //initialized short
static short *rots[] = {rot0, rot1, rot2, rot3 }; //initialized short

static unsigned kspace[64]; //initialize positive numbers
static unsigned *k; //initialize positive numbers
static DigestArray h; //initialize another array
DigestArray abcd; //initialize array
DgstFctn fctn; //initialize arrayfunction
short m, o, g; //initialize short value
unsigned f; //it can hold zero and positive numbers
short *rotn; //multiply short and rotn
union //it can hold members different sizes and type
{
unsigned w[16]; //it can hold zero and positive number
char b[64]; //to hold ascii characters

```

```

    }

mm; //length

int os = 0; //define numeric variables holding whole numbers

int grp, grps, q, p; //define variables holding whole numbers

unsigned char *msg2; //it can hold zero and positive number

if (k==NULL) k= calctable(kspace); //Boolean expression

for (q=0; q<4; q++) h[q] = h0[q]; //enable to perform n number of steps together in oneline

// initialize

{

    grps = 1 + (mlen+8)/64; //define previous line

msg2 = malloc( 64*grps); //define message

memcpy( msg2, msg, mlen); //copy a specified number Of bytes

msg2[mlen] = (unsigned char)0x80; //define message

q = mlen + 1; //length of memory

while (q < 64*grps) //allows to repetedly run the same block of code

{

    msg2[q] = 0; //define msg

    q++; //increment of q

}

{

    MD5union u; //special datatype

    u.w = 8*mlen; //multiplication of u nad v

    q -= 8; //decrement of q

    memcpy(msg2+q, &u.w, 4 ); //copy of memory

}

}

for (grp=0; grp<grps; grp++) //it enables to perform n number of steps in one line

{

    memcpy( mm.b, msg2+os, 64); //copy from memory

    for(q=0;q<4;q++) abcd[q] = h[q]; //it enables to perform n numbers of steps in one line

```

```

for (p = 0; p<4; p++) //it enables to perform n number of steps in one line
{
    fctn = ff[p]; //group of statements that together perform a task

    rotn = rots[p]; //encrypt/decrypt
m = M[p]; o= O[p];

    for (q=0; q<16; q++)//it enables to perform n number of step in one line
    {
        g = (m*q + o) % 16;
f = abcd[1] + rol( abcd[0]+ fctn(abcd)+k[q+16*p] + mm.w[g], rotn[q%4]);
abcd[0] = abcd[3];

        abcd[3] = abcd[2];

        abcd[2] = abcd[1];
abcd[1] = f;

    }}
for (p=0; p<4; p++) h[p] += abcd[p]; //it enables to perform n number of steps in one line
os += 64; //increment

    }

return h; //terminate the function
}

void main() //return type
{
    int j,k; //define numeric values holding whole numbers

    const char *msg = "The quick brown fox jumps over the lazy dog"; //function declaration

    unsigned *d = md5(msg, strlen(msg)); //it can hold zero and positive number

    MD5union u; //special datatype

    clrscr(); //clear the console screen

    printf("\t MD5 ENCRYPTION ALGORITHM IN C \n\n"); //display the output text

    printf("Input String to be Encrypted using MD5 : \n\t%s",msg); //display the output text

    printf("\n\nThe MD5 code for input string is: \n"); //display the output text

    printf("\t= 0x"); //display the output text

```

```

    for (j=0;j<4; j++){    u.w = d[j]; //it enables us to perform n number of steps together in
    oneline

    for (k=0;k<4;k++) printf("%02x",u.b[k]); //it enables us to perform n number of steps
    together in oneline

    }

    printf("\n");//display the output text

    printf("\n\t MD5 Encryption Successfully    Completed!!!\n\n"); //display the output text

    getch(); //catch the character from keyword

    system("pause"); //execute pause command and make screen clear

    getch();//catch the character from keyword

    }

```

OUTPUT:

MD5 ENCRYPTION ALGORITHM IN C

Input strings to be encrypted using MD5:

The quick brown fox jumps over the lazy day

The MD5 code for input string is:

=0*f87f8c8ff5718c8f6f6de98c8f

MD5 Encryption Successfully Completed!!!

RESULT:

The implementation of MD5 hashing algorithm had been implemented successfully using c.

EX. NO: 2(E)**IMPLEMENTATION OF SHA-1****AIM:**

To implement the SHA-1 hashing technique using Java program.

DESCRIPTION:

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function which produces a 160-bit hash value known as a message digest. The way this algorithm works is that for a message of size < 264 bits it computes a 160-bit condensed output called a message digest. The SHA-1 algorithm is designed so that it is practically infeasible to find two input messages that hash to the same output message. A hash function such as SHA-1 is used to calculate an alphanumeric string that serves as the cryptographic representation of a file or a piece of data. This is called a digest and can serve as a digital signature. It is supposed to be unique and non-reversible.

ALGORITHM:

STEP-1: Read the 256-bit key values.

STEP-2: Divide into five equal-sized blocks named A, B, C, D and E.

STEP-3: The blocks B, C and D are passed to the function F.

STEP-4: The resultant value is permuted with block E.

STEP-5: The block A is shifted right by 's' times and permuted with the result of step-4.

STEP-6: Then it is permuted with a weight value and then with some other key pair and taken as the first block.

STEP-7: Block A is taken as the second block and the block B is shifted by 's' times and taken as the third block.

STEP-8: The blocks C and D are taken as the block D and E for the final output.

PROGRAM :

```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class SHA1 {
    public static String encryptThisString(String input)
    {
        try {
            /* getInstance() method is called with algorithm SHA-1 */
            MessageDigest mm = MessageDigest.getInstance("SHA-1");
            /* digest() method is called to calculate message digest of the input string returned as array of
            byte */
            byte[] messageDigest = mm.digest(input.getBytes());
            /* Convert byte array into signum representation */
            BigInteger no = new BigInteger(1, messageDigest);
            //Convert message digest into hex value
            String hashtext = no.toString(16);
            // Add preceding 0s to make it 32 bit
            while (hashtext.length() < 32) {
                hashtext = "0" + hashtext;
            }
            // return the HashText
            return hashtext;
        }
    }
}
```



```
// For specifying wrong message digest algorithms
catch (Exception e) {
    throw new RuntimeException(e);
}
}
// Driver code
public static void main(String args[]) throws Exception
{
    System.out.println("HashCode Generated by SHA-1 for: ");
    String code1 = "Secure hash algorithm";
    System.out.println("\n" + code1 + " : " + encryptThisString(code1));
    String code2 = "Security laboratory";
    System.out.println("\n" + code2 + " : " + encryptThisString(code2));
}
}
```

OUTPUT

HashCode Generated by SHA-1 for:
Secure hash algorithm : ef7502aa5a57ebdc024f7cbabfd47304ada3d4e4
Security laboratory : 77831da398faf35d77abeaf0bc31605527c4068f

RESULT:

Thus the SHA-1 hashing technique had been implemented successfully.

EX. NO: 3 IMPLEMENTATION OF DIGITAL SIGNATURE STANDARD

AIM:

To write a C program to implement the creation and verification of Digital Signature Standards using Euclidean Algorithm.

ALGORITHM:

STEP-1: Alice and Bob are investigating a forgery case of x and y.

STEP-2: X had document signed by him but he says he did not sign that document digitally.

STEP-3: Alice reads the two prime numbers p and a.

STEP-4: He chooses a random co-primes alpha and beta and the x's original signature x.

STEP-5: With these values, he applies it to the elliptic curve cryptographic equation to obtain y.

STEP-6: Comparing this 'y' with actual y's document, Alice concludes that y is a forgery.

PROGRAM: (Digital Signature Standard)

```
import java.util.*;
import java.math.BigInteger;
//signature creating part
class dsaAlg //class initialization
{
    final static BigInteger one = new BigInteger("1");
    final static BigInteger zero = new BigInteger("0");
    public static BigInteger getNextPrime(String ans)
    {
        BigInteger test = new BigInteger(ans);//object creation part
        while (!test.isProbablePrime(99))
        e:
        {
            test = test.add(one);//calling the object
        }
        return test;
    }
}
```

```

public static BigInteger findQ(BigInteger n)//object creation
{
    BigInteger start = new BigInteger("2");
    while (!n.isProbablePrime(99))
    {
        while (!((n.mod(start)).equals(zero)))
        {
            start = start.add(one);//calling the object
        }
        n = n.divide(start);
    }
    return n;
}

public static BigInteger getGen(BigInteger p, BigInteger q,Random r)
{
    BigInteger h = new BigInteger(p.bitLength(), r); //object creation
    h = h.mod(p);//calling object
    return h.modPow((p.subtract(one)).divide(q), p);
}

public static void main (String[] args) throws
    java.lang.Exception //for exception handling
{
    Random randObj = new Random();//object creation for dsa key generation
    BigInteger p = getNextPrime("10600"); // approximate prime
    BigInteger q = findQ(p.subtract(one));
    BigInteger g = getGen(p,q,randObj);
    System.out.println(" \n simulation of Digital Signature Algorithm \n");
    System.out.println(" \n global public key components are:\n"); //key generation part
    System.out.println("\np is: " + p);
    System.out.println("\nq is: " + q);
    System.out.println("\ng is: " + g);
    BigInteger x = new BigInteger(q.bitLength(), randObj);

```

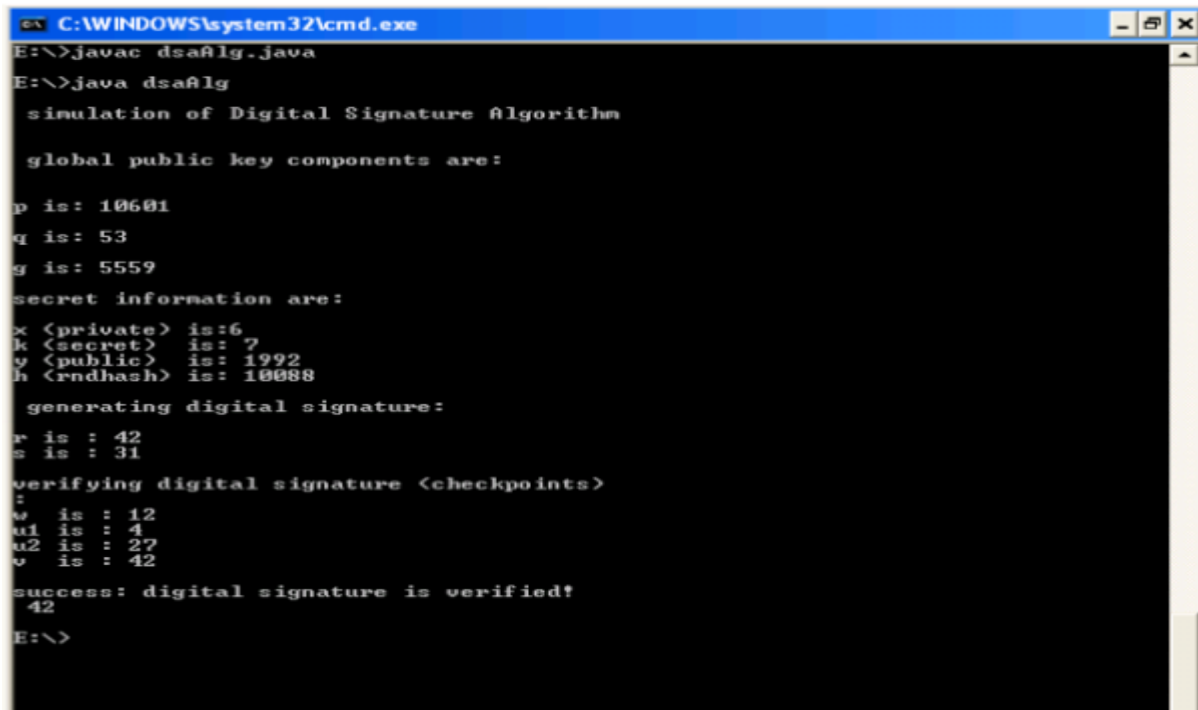
```

x = x.mod(q);
BigInteger y = g.modPow(x,p);
BigInteger k = new BigInteger(q.bitLength(), randObj);
k = k.mod(q);
BigInteger r = (g.modPow(k,p)).mod(q);
BigInteger hashVal = new BigInteger(p.bitLength(), randObj);
BigInteger kInv = k.modInverse(q);
BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));
s = s.mod(q);
System.out.println("\nsecret information are:\n");
System.out.println("x (private) is:" + x);// print the dsa key
System.out.println("k (secret) is: " + k);//print the dsa key
System.out.println("y (public) is: " + y);//print the dsa key
System.out.println("h (rndhash) is: " + hashVal);
System.out.println("\n generating digital signature:\n");
System.out.println("r is : " + r);
System.out.println("s is : " + s);
BigInteger w = s.modInverse(q);
BigInteger u1 = (hashVal.multiply(w)).mod(q);
BigInteger u2 = (r.multiply(w)).mod(q);
BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
v = (v.mod(p)).mod(q);
System.out.println("\nverifying digital signature (checkpoints)\n:");
System.out.println("w is : " + w);
System.out.println("u1 is : " + u1);
System.out.println("u2 is : " + u2);
System.out.println("v is : " + v);
if (v.equals(r))//condition for verified dsa alg
{
System.out.println("\nsuccess: digital signature is verified!\n " + r);
}
else
{

```

```
System.out.println("\n error: incorrect digital signature\n ");  
}}}
```

OUTPUT:



```
C:\WINDOWS\system32\cmd.exe  
E:\>javac dsaAlg.java  
E:\>java dsaAlg  
    simulation of Digital Signature Algorithm  
  
    global public key components are:  
  
p is: 10601  
q is: 53  
g is: 5559  
secret information are:  
x <private> is: 6  
k <secret> is: 7  
y <public> is: 1992  
h <rndhash> is: 10088  
  
    generating digital signature:  
r is : 42  
s is : 31  
  
    verifying digital signature <checkpoints>  
:  
u is : 12  
u1 is : 4  
u2 is : 27  
v is : 42  
  
success: digital signature is verified!  
42  
E:\>
```

RESULT:

Thus the implementation of Digital Signature Standard has been created and verified successfully.

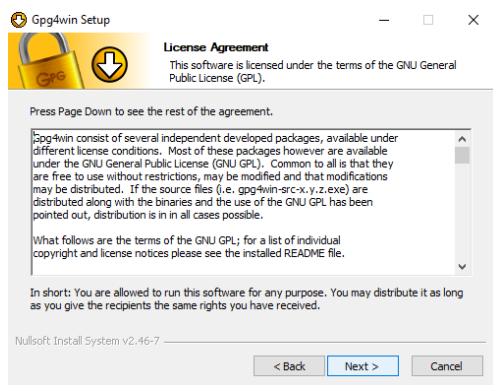
Ex.No.4. Demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG)

Aim:

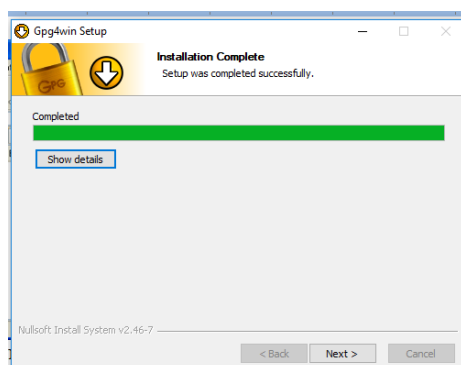
To demonstrate how to provide secure data storage, secure data transmission and for creating digital signatures (GnuPG)

Procedure:

Downloading Gpg4win from the Internet, and install.



During the **installation** process that follows, you will see a progress bar and information on which file is currently being installed.

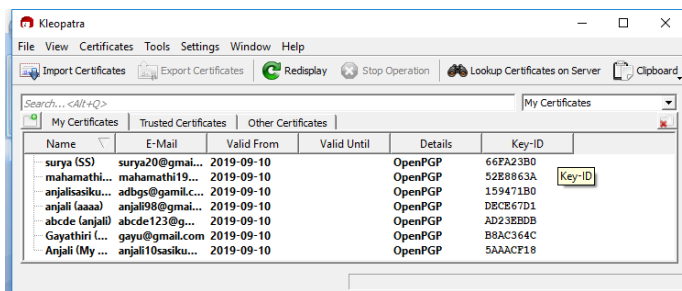


Once you have completed the installation, please click on [*Next*].

The last page of the installation process is shown once the installation has been successfully completed.

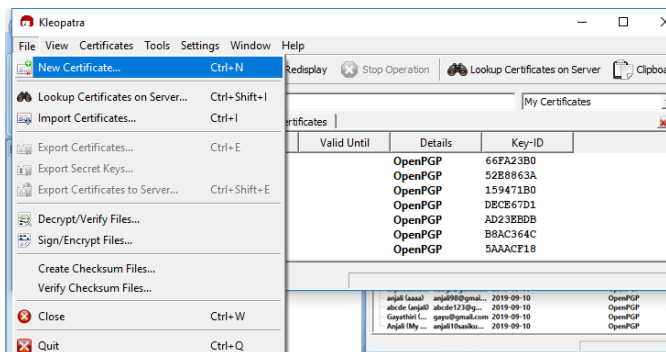
Open Kleopatra using the Windows start menu

You will see the main Kleopatra screen - the certificate administration

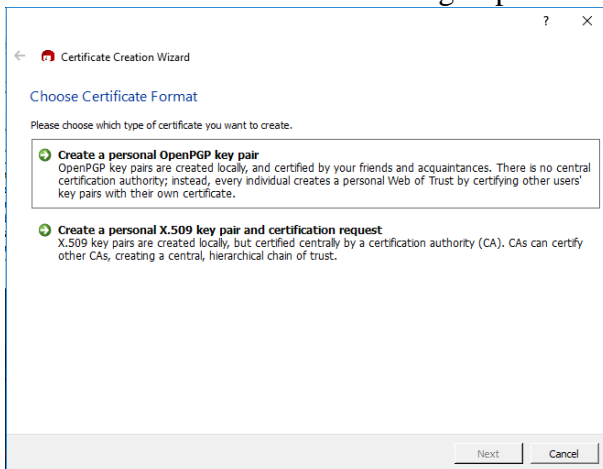


Click on *File -> New Certificate.*

In the following dialog you select the format for the certificate.



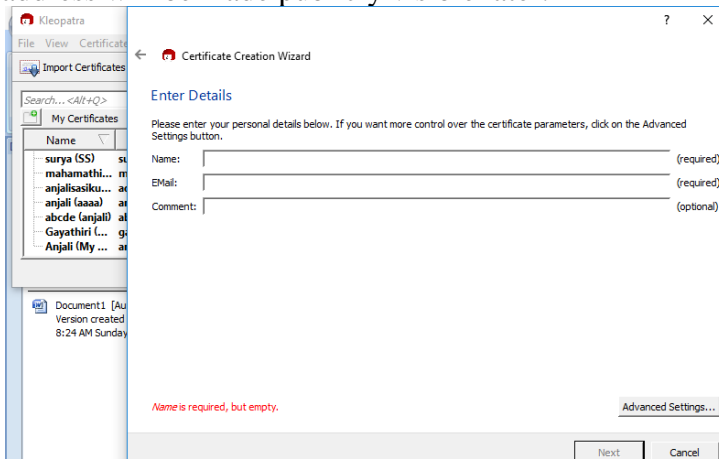
You can choose from the following: OpenPGP (PGP/MIME) or X.509 (S/MIME).

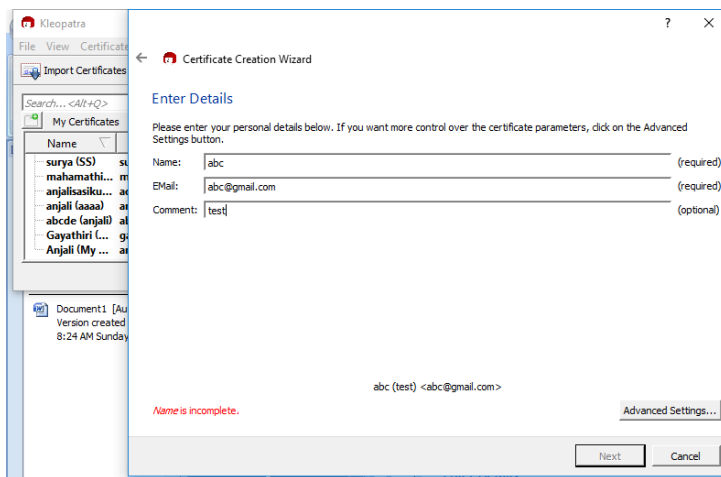


Creating an OpenPGP certificate

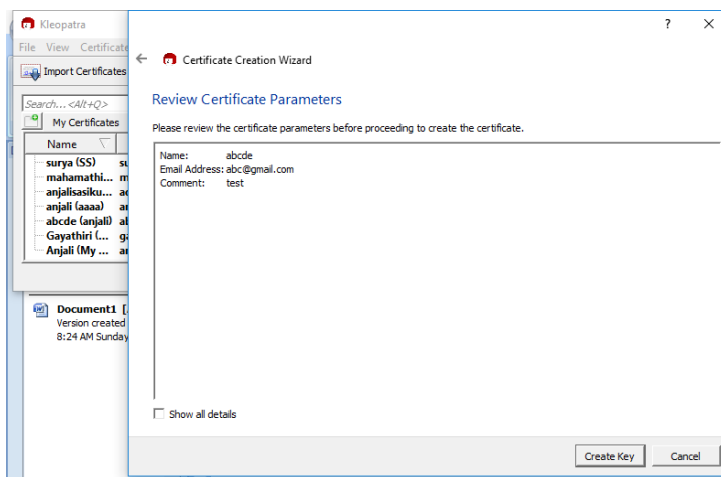
In the certificate option dialog, click on [*Create personal OpenPGP key pair*].

Now enter your e-mail address and your name in the following window. Name and e-mail address will be made publicly visible later.

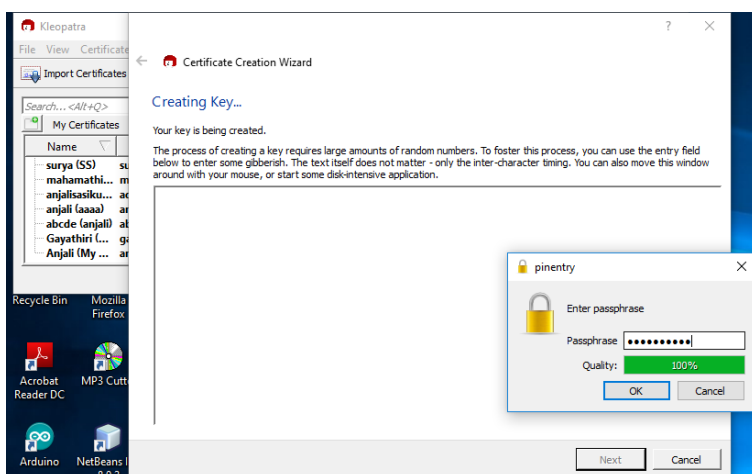




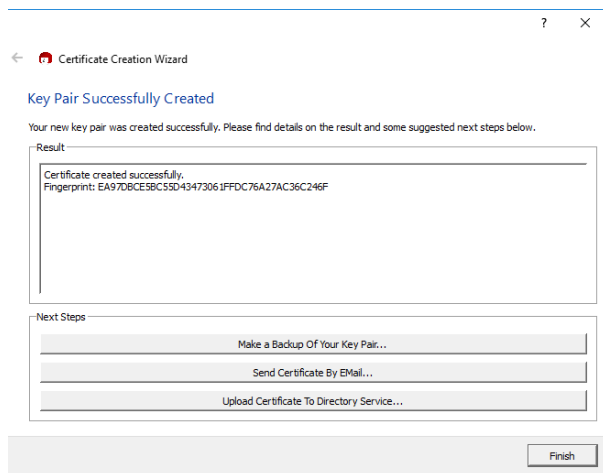
The **Advanced settings** are only be required in exceptional cases. Click on *[Next]*.



If everything is correct, click on *[Create key]*.
Now to the most important part: entering your **passphrase**!
To create a key pair, you must enter your personal passphrase.

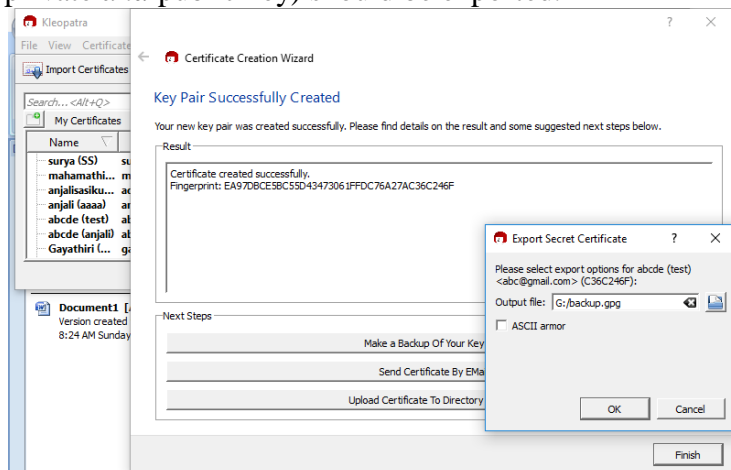


To make sure that you did not make any typing errors, the system will prompt you to enter your passphrase twice. Always confirm your entry with *[OK]*.
Now OpenPGP key pair is being created:
As soon as **the key pair creation has been successful**.

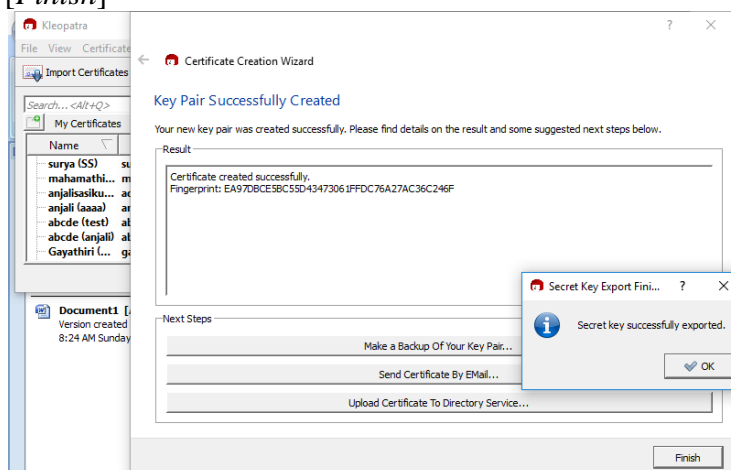


Creating a backup copy of your (private) certificate...

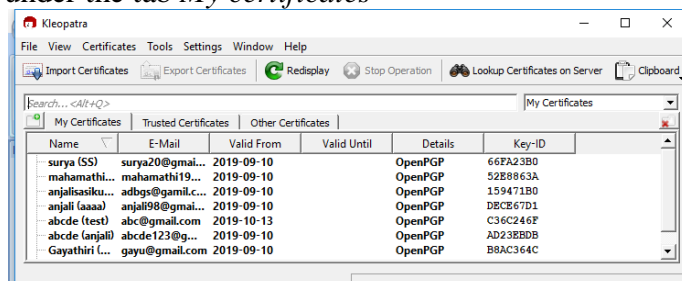
Enter the path under which your full certificate (which contains your new key pair, hence the private *and* public key) should be exported.

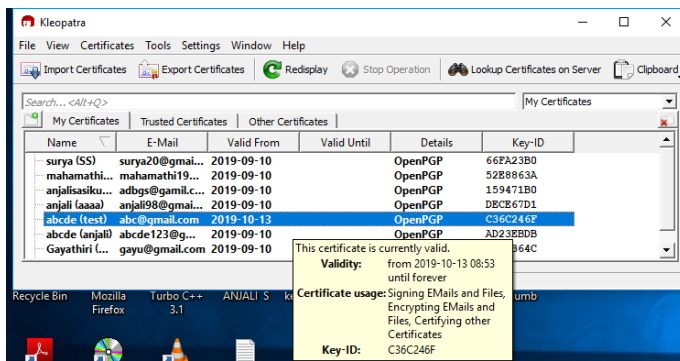


This completes the creation of your OpenPGP certificate. End the Kleopatra assistant with **[Finish]**



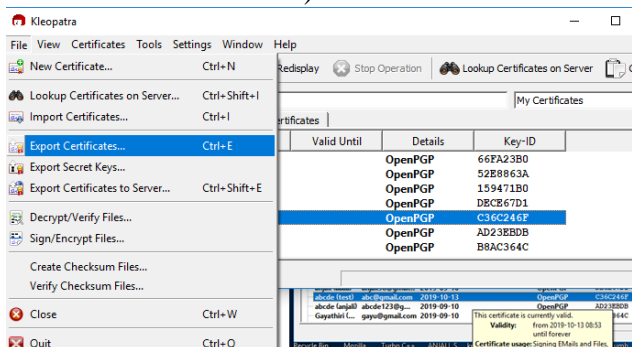
The OpenPGP certificate which was just created can be found in the certificate administration under the tab *My certificates*





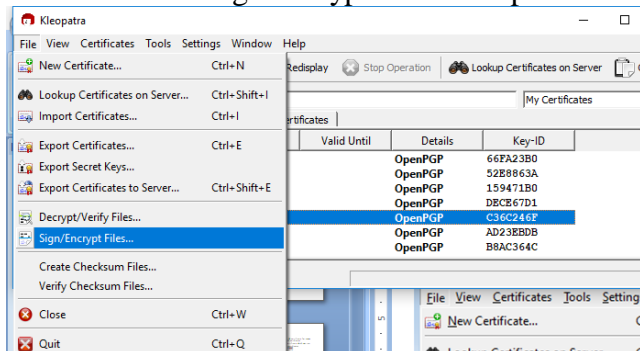
Exporting your public OpenPGP certificate

Select the public certificate to be exported in Kleopatra (by clicking on the corresponding line in the list of certificates) and then click on **File -> Export certificates...** in the menu.

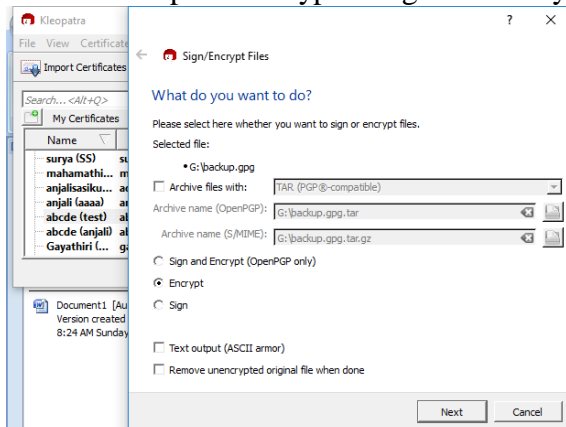


For encrypting files with the certificate created,

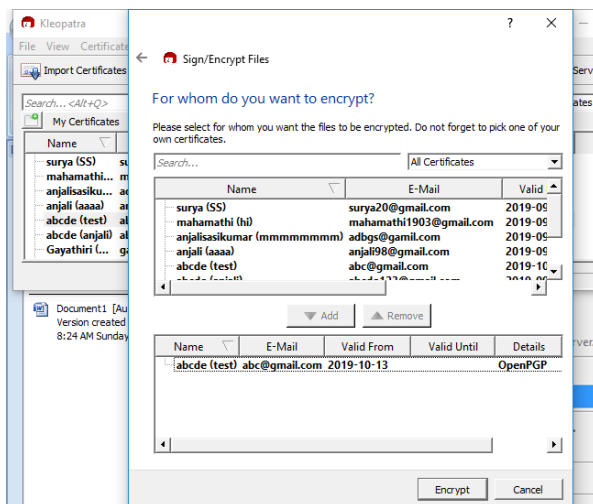
Choose **File -> Sign/Encrypt files** and upload the file to be encrypted.



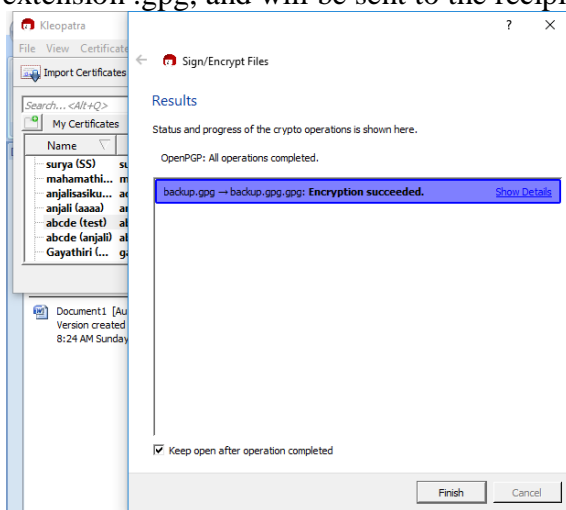
Choose the option encrypt or sign and encrypt from the options and give next..



Select the recipient for whom you want the encrypted files to be sent and click the button add, then click encrypt.

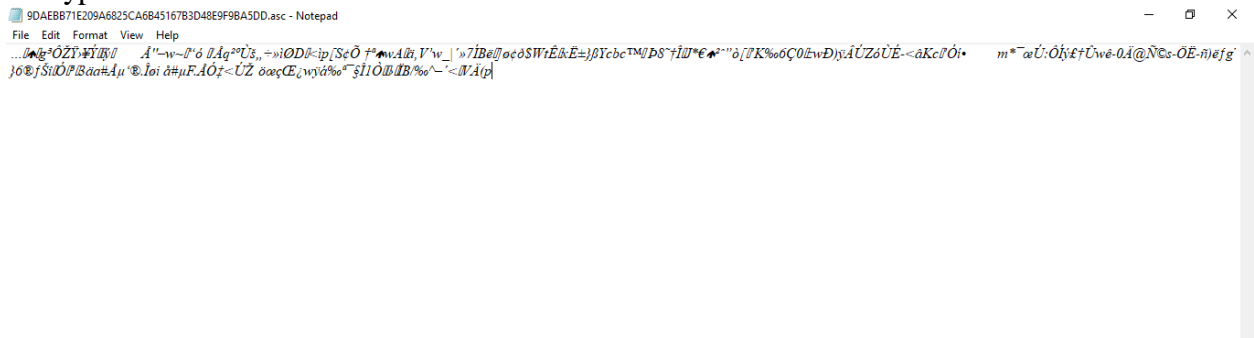


The file chosen for encryption will be encrypted and stored in the same location with the extension .gpg, and will be sent to the recipient.



Click finish

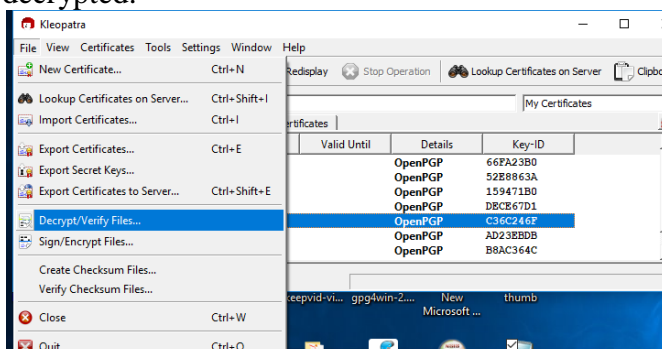
Encrypted file content will be like



For decryption,

The receiver should have the sender's public key received already through e-mail.

Choose →file→decrypt/verify files and choose your certificate, then upload the file to be decrypted.

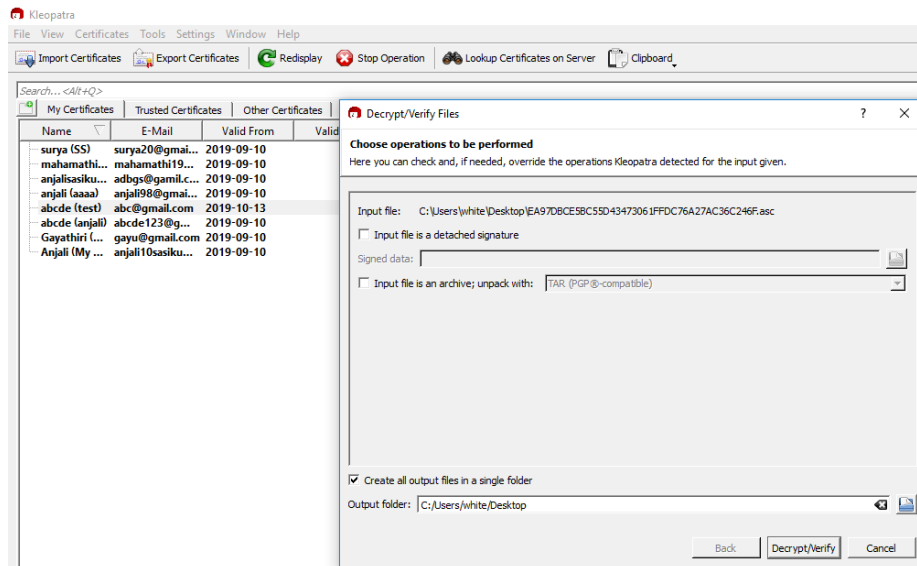


Click decrypt.

The dialog box below will be opened.

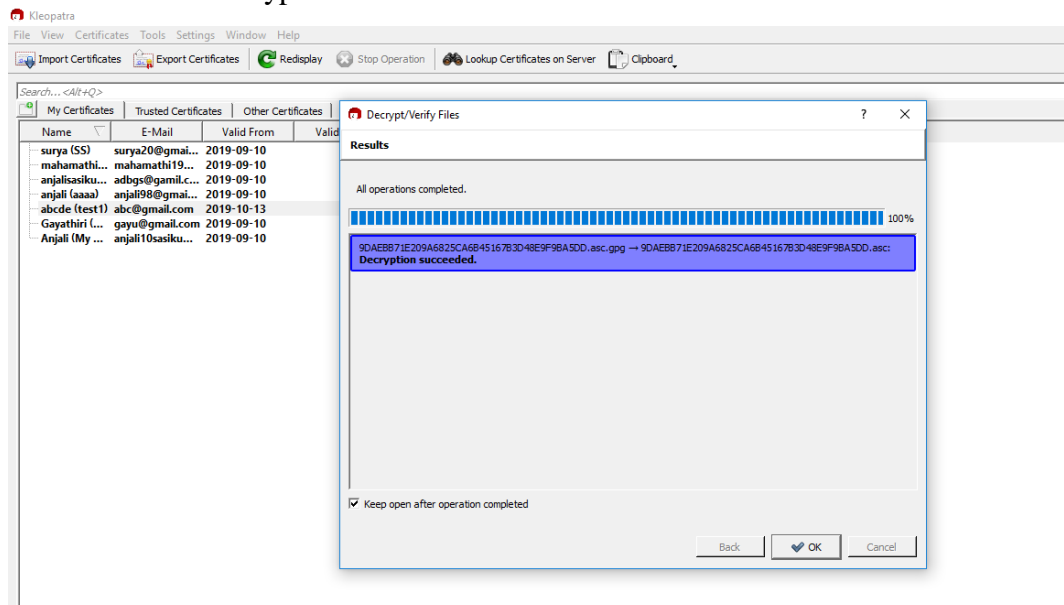
Click the check box if the encrypted message is a signed or ZIP file

If not just choose decrypt.



Enter the passphrase once again for authentication..

The file will be decrypted and will be saved in the same location



Click ok.

Result:

Thus the providing secure data storage, secure data transmission and for creating digital signatures using GnuPG is demonstrated successfully.

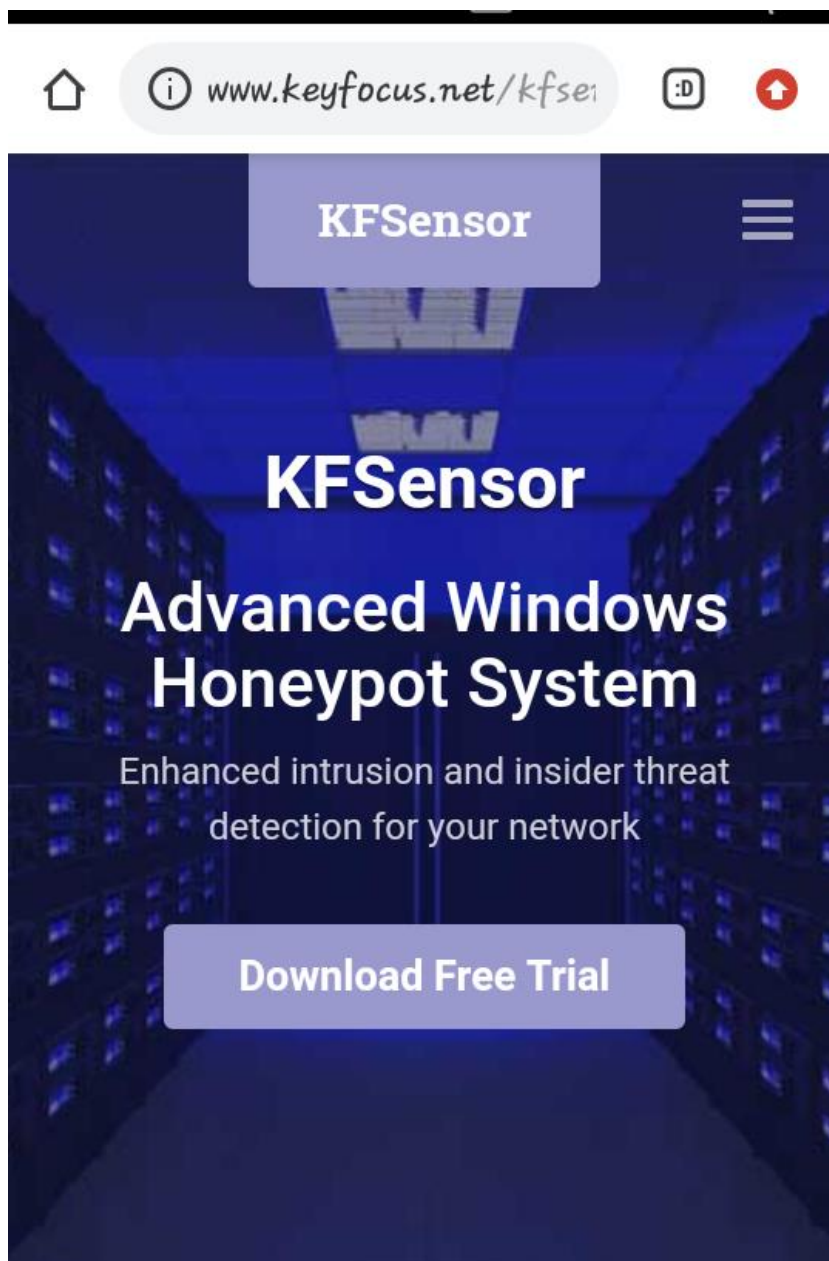
**EX NO: 5 WORKING WITH KF SENSOR TOOL FOR CREATING
AND MONITORING HONEYPOT**

AIM:

To study the setup of a honeypot and monitor the network using KF Sensor.

PROCEDURE:

STEP 1:Download KF sensor Evaluation setup file from KF sensor website.



Step2: Enter the required information to download a KFSensor.

Your Contact Information

☆ *Required fields*

First Name ☆

Last Name ☆

Company

Job Role

Country ☆

Email ☆

Submit

Step 3: Click Kfsensor professional free trial version(.MSL file) link for download.

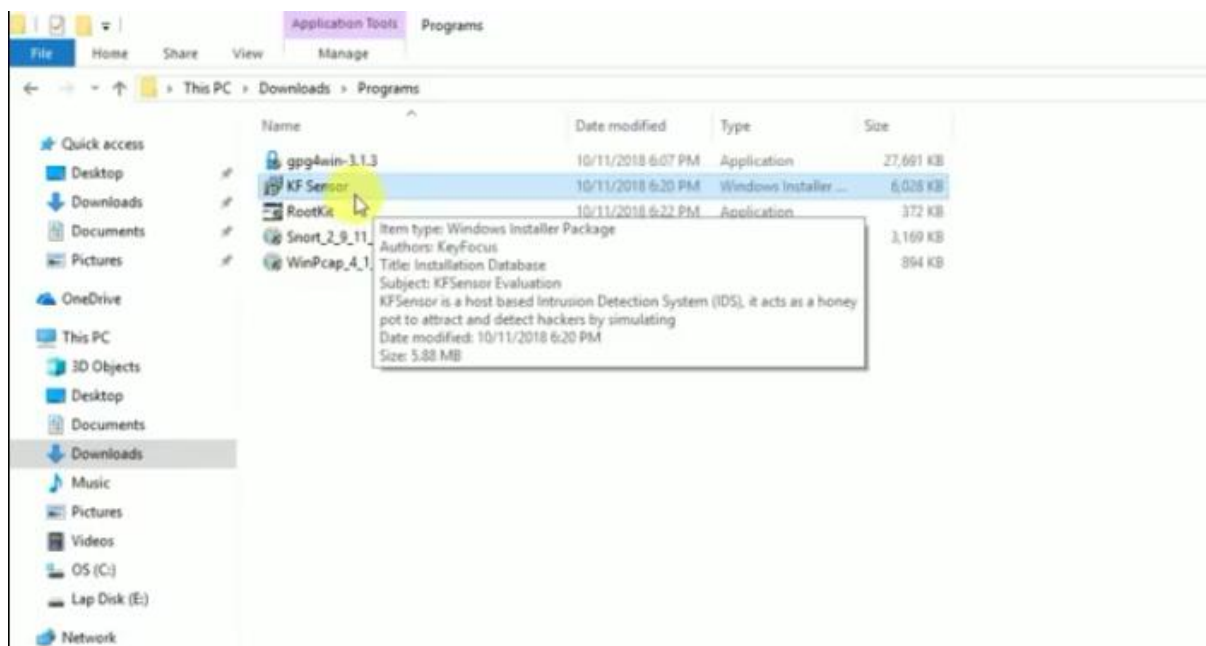
Download These Links

KFSensor Professional

 [**KFSensor Professional
Free Trial Version \(.MSI file\)**](#)

*For Windows 7, 8, 10, Windows
Server 2008R2, 2012, 2012R2,
2016*

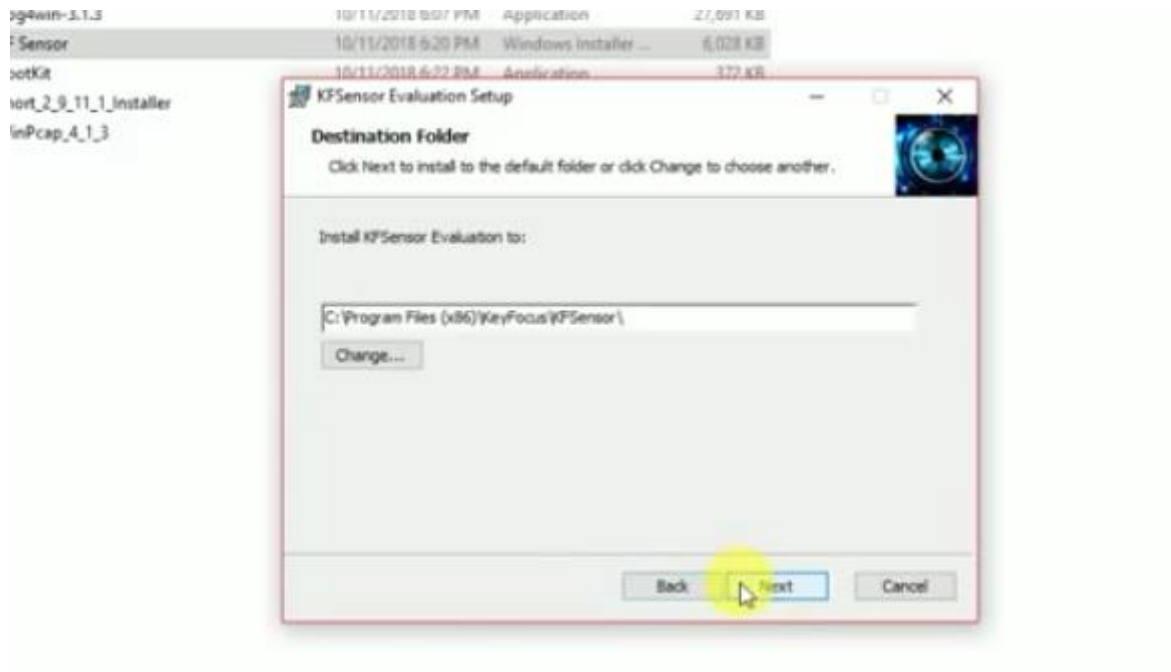
Step 4: Click kfsensor for installation process.



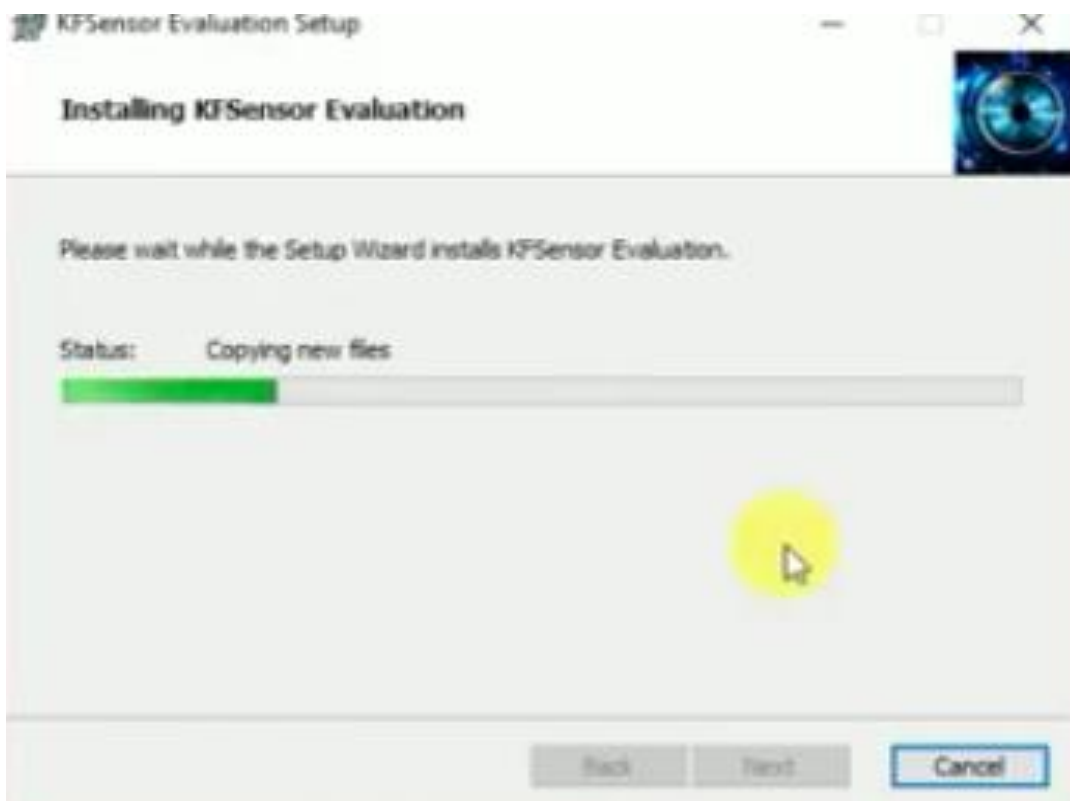
Step 5: Click next to the kfsensor setup wizard to install it.



Step 6: Select the destination folder and click next.



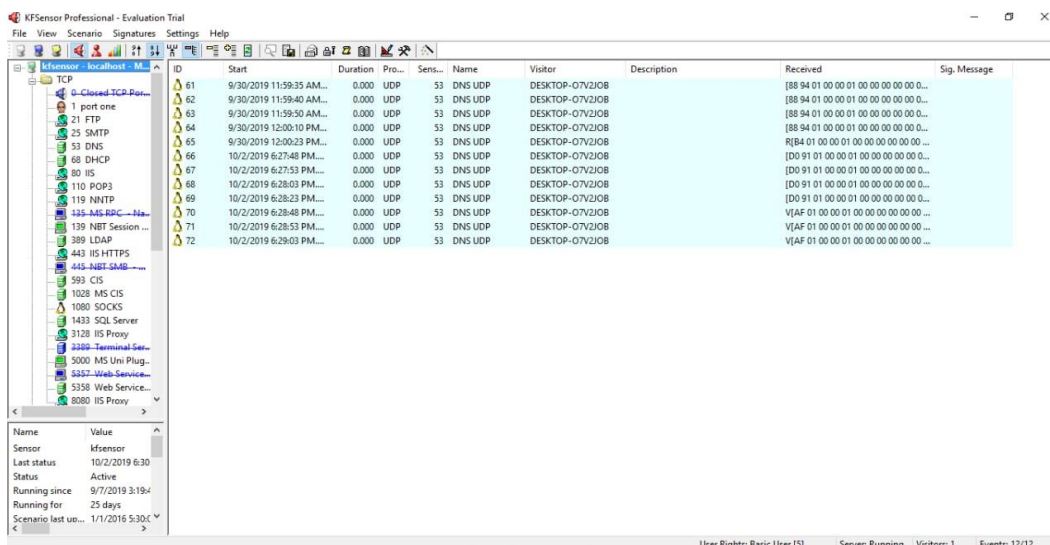
Step7: Preparing for installation and setup of a kfsensor.



Step8: click finish, after the kfsensor installation setup is completed.



Step9: kfsensor gets activated in system,it detects and displays hackers and worms also it gives a alert when it detects any intruder availability.



RESULT:

Thus the study of setup a honeypot and monitor the network has been developed successfully.

Ex.No :6**INSTALLATION OF ROOTKITS****AIM:**

Rootkit is a stealth type of malicious software designed to hide the existence of certain process from normal methods of detection and enables continued privileged access to a computer.

PROCEDURE:

STEP-1: Download Rootkit Tool from GMER website www.gmer.net.

STEP-2: This displays the Processes, Modules, Services, Files, Registry, RootKit/ Malwares, Autostart, CMD of local host.

STEP-3: Select Processes menu and kill any unwanted process if any.

STEP-4: Modules menu displays the various system files like .sys, .dll

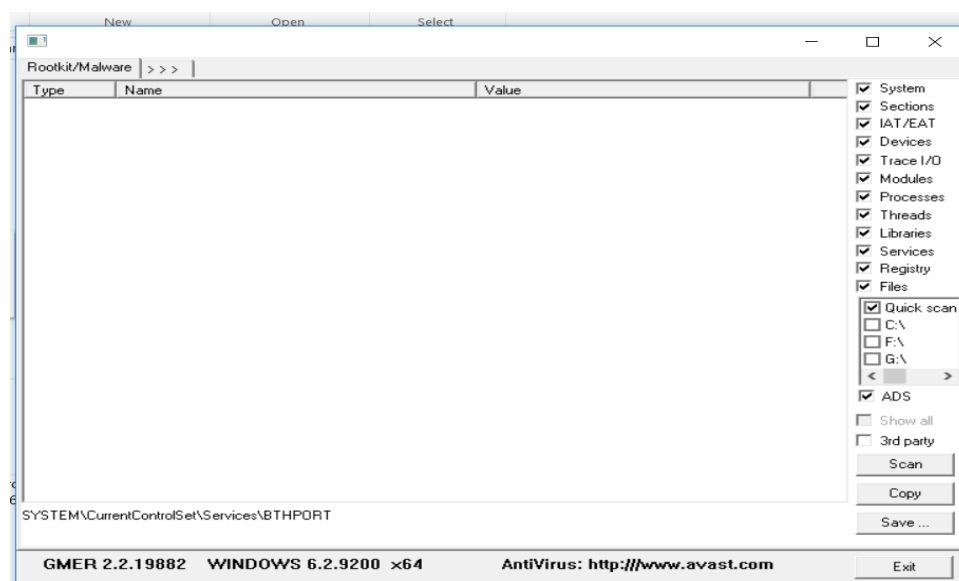
STEP-5: Services menu displays the complete services running with Autostart, Enable, Disable, System, Boot.

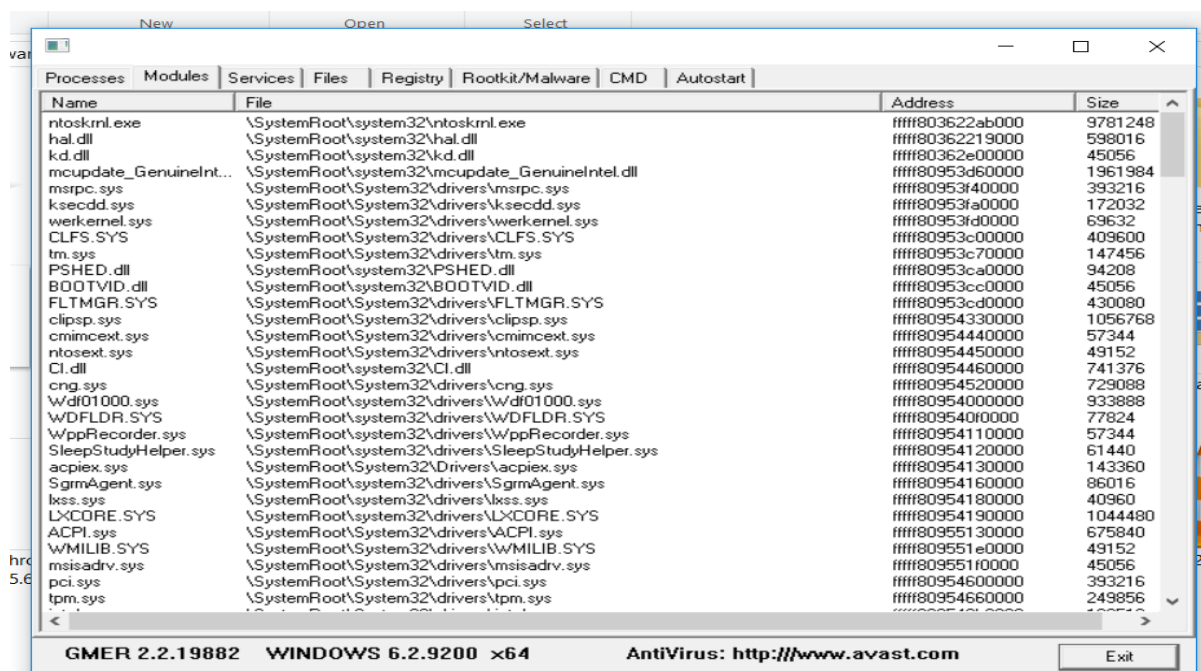
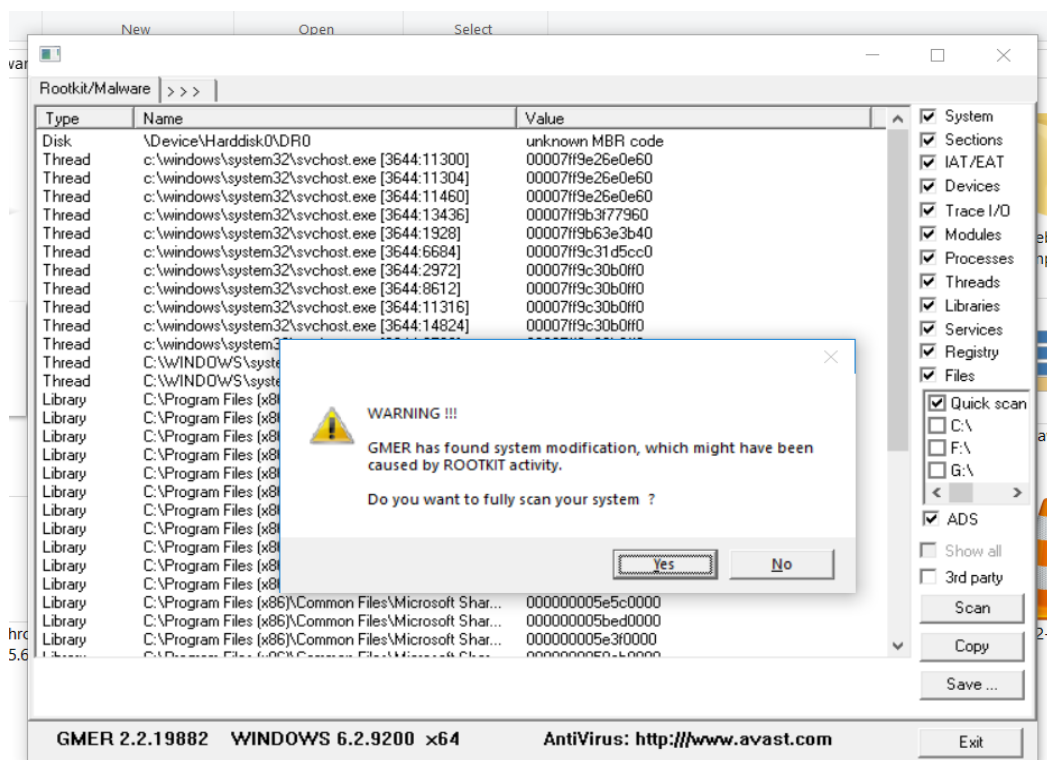
STEP-6: Files menu displays full files on Hard-Disk volumes. **STEP-7:** Registry displays Hkey_Current_user and Hkey_Local_Machine.

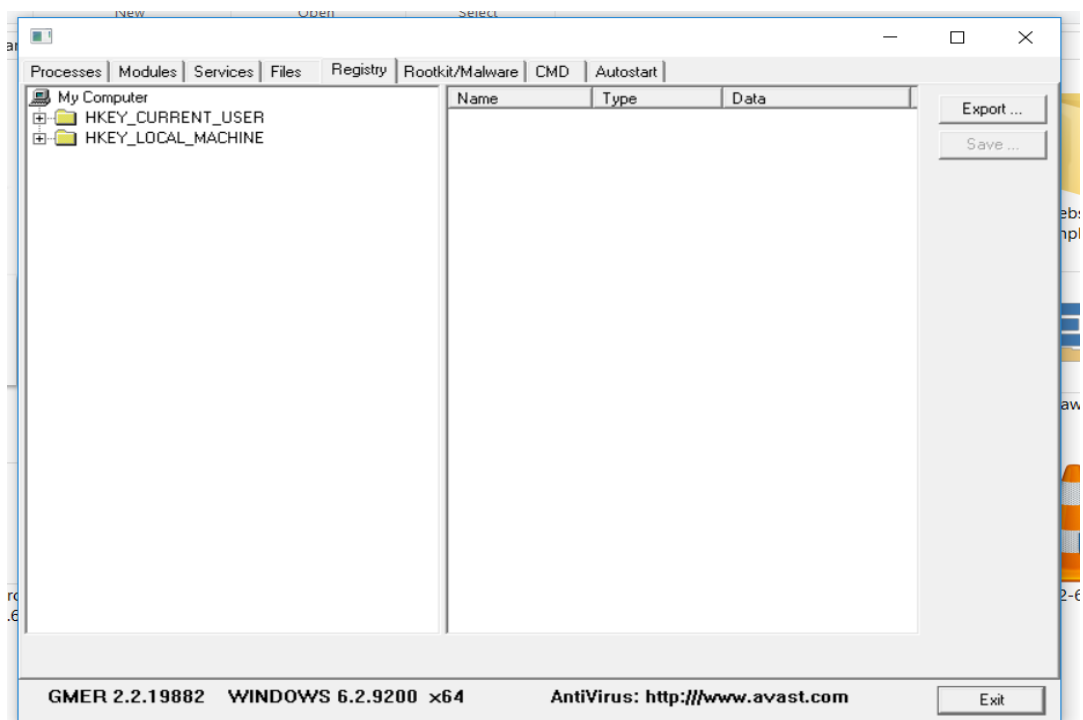
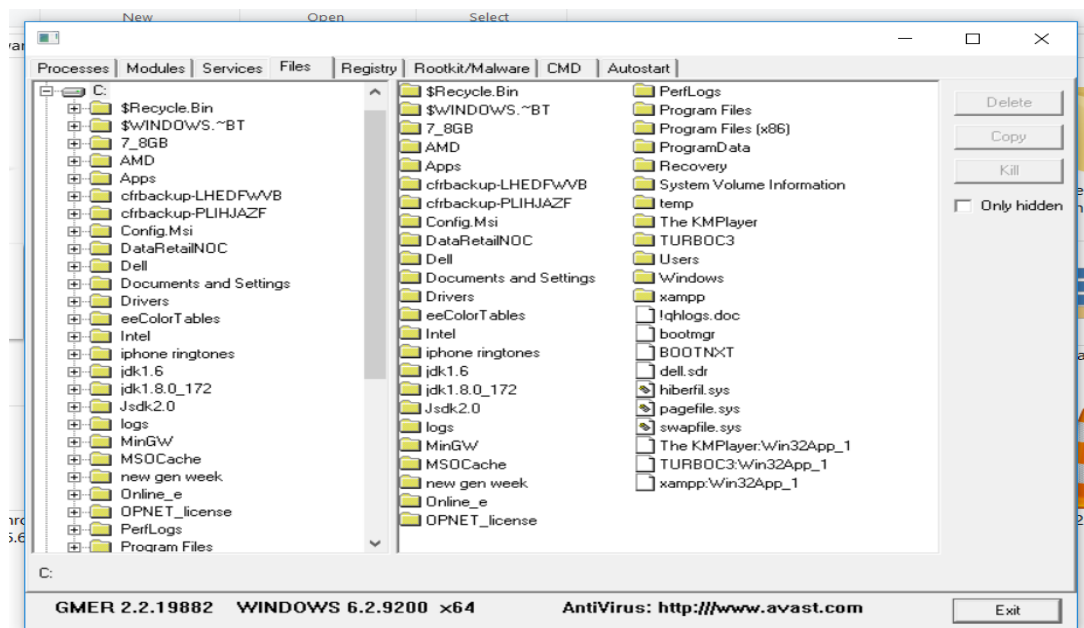
STEP-8: Rootkits / Malwares scans the local drives selected.

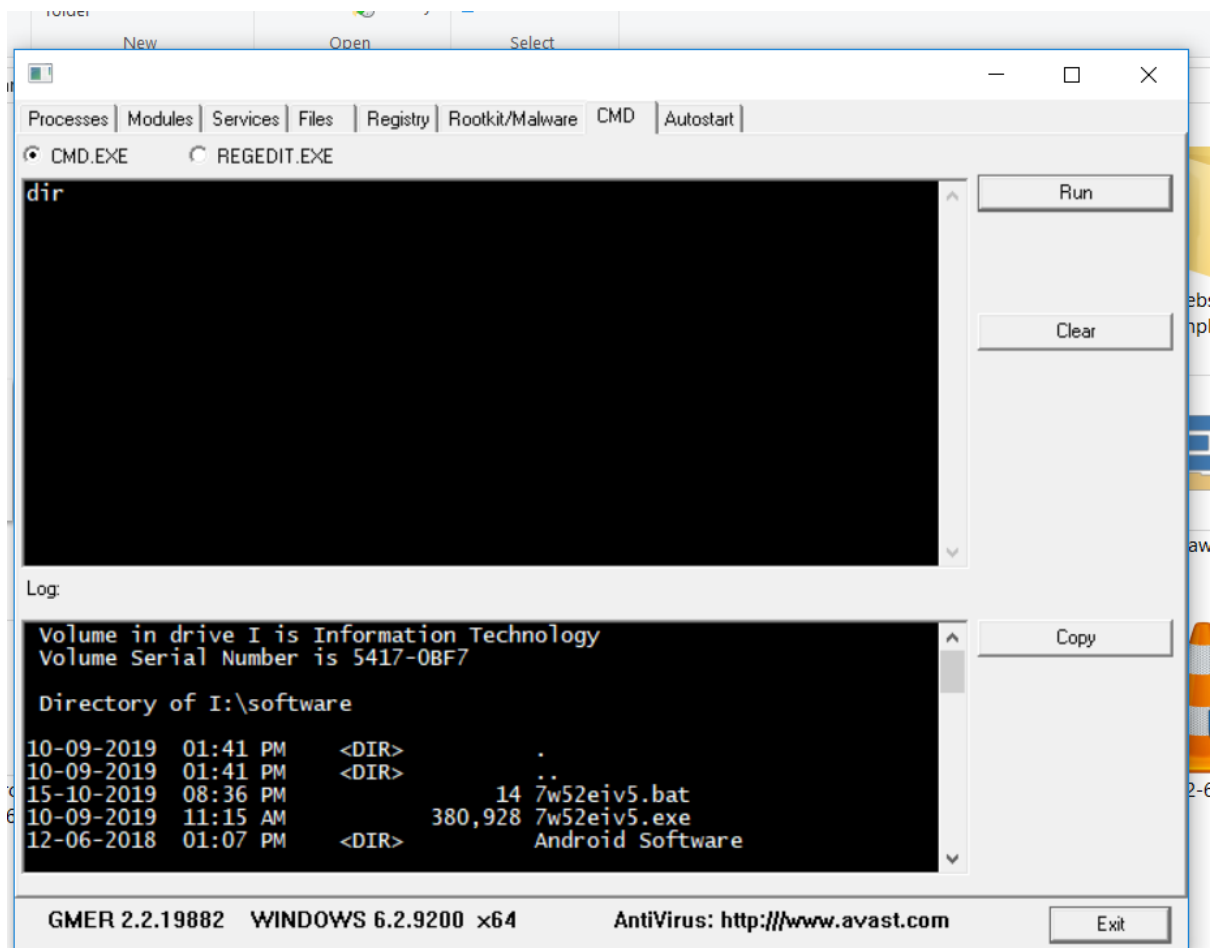
STEP-9: Autostart displays the registry base Autostart applications.

STEP-10: CMD allows the user to interact with command line utilities or Registry









RESULT:

Thus the study of installation of Rootkit software and its variety of options were developed successfully.

Exp.No: 7 **PERFORM WIRELESS AUDIT ON AN ACCESS POINT OR A ROUTER AND DECRYPT WEP AND WPA.**

AIM:

To perform wireless audit on an access point or a router and decrypt wep and wpa.

PROCEDURE:

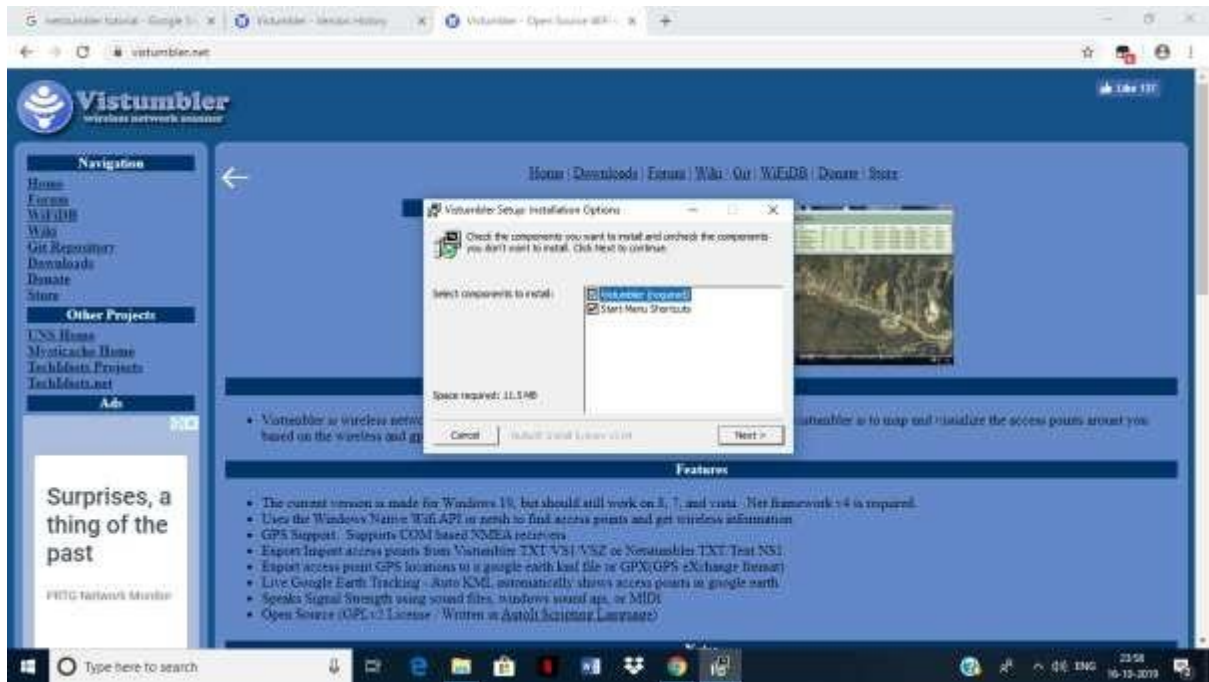
STEP 1:

Download and install the Vistumbler version-10.6.5.



STEP 2:

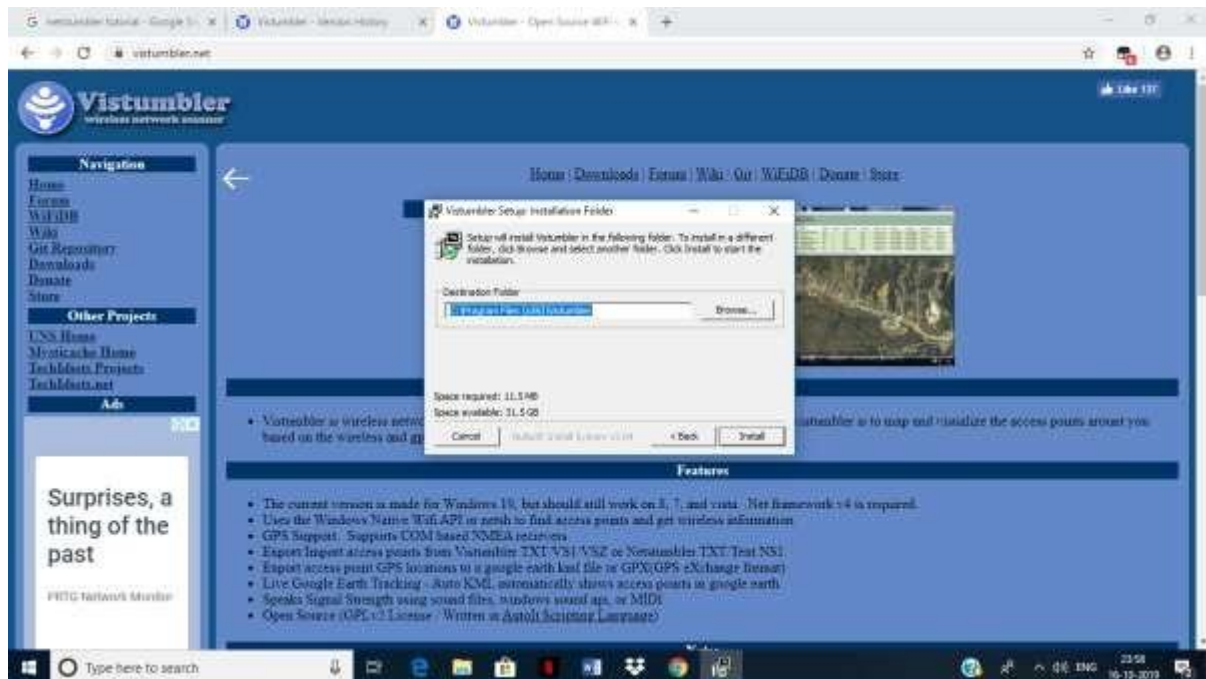
Open the Vistumbler setup □ Installation option, check the components we want to install and uncheck the components you don't want to install .Click next.



PERFORM WIRELESS AUDIT ON AN ACCESS POINT OR A ROUTER AND DECRYPT WEP AND WPA.

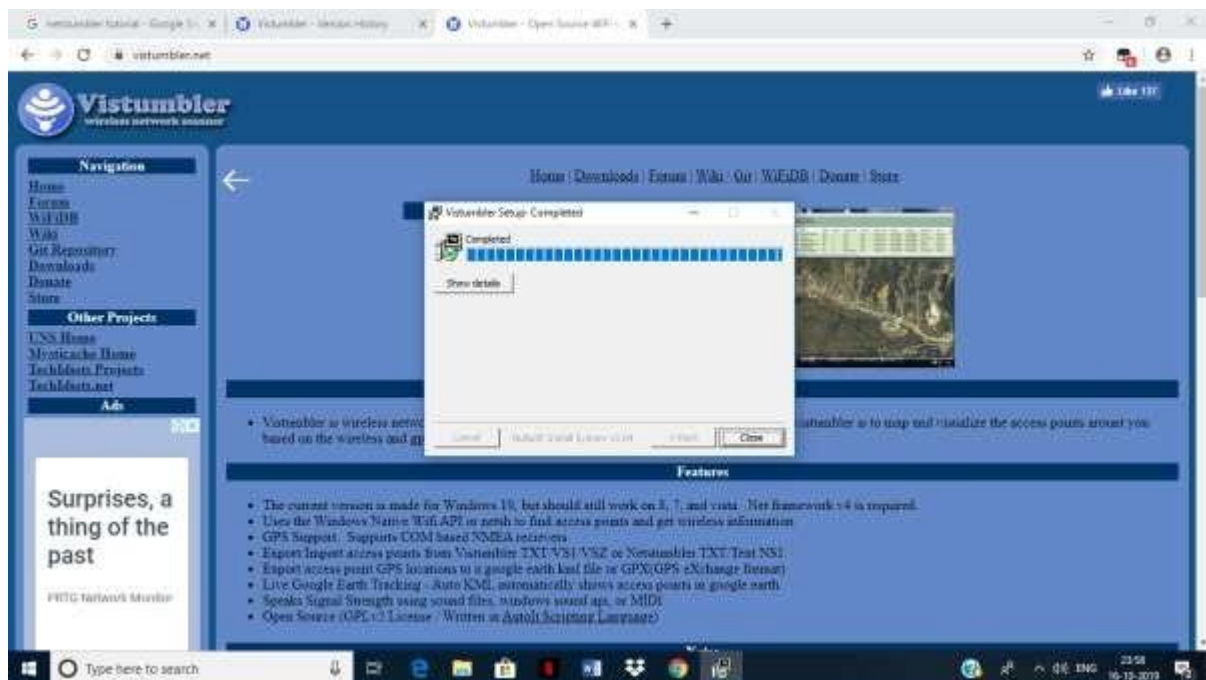
STEP 3:

Setup will install the Vistumbler in C:\Program Files(x86)\Vistumbler.To install in a different folder,click Browse and select another folder.Click Install to start the installation.



STEP 4:

Once the installation process get completed close the Vistumbler setup.Desktop shortcut would be created for the application.



PERFORM WIRELESS AUDIT ON AN ACCESS POINT OR A ROUTER AND DECRYPT WEP AND WPA.

STEP 5:

Open ☐ Vistumbler. It consists of four main components ☐ Start, Use GPS, Graph1, Graph2.



STEP 6:

Click the start button. Automatically nearby wireless connection would be plotted in the table along with its description.

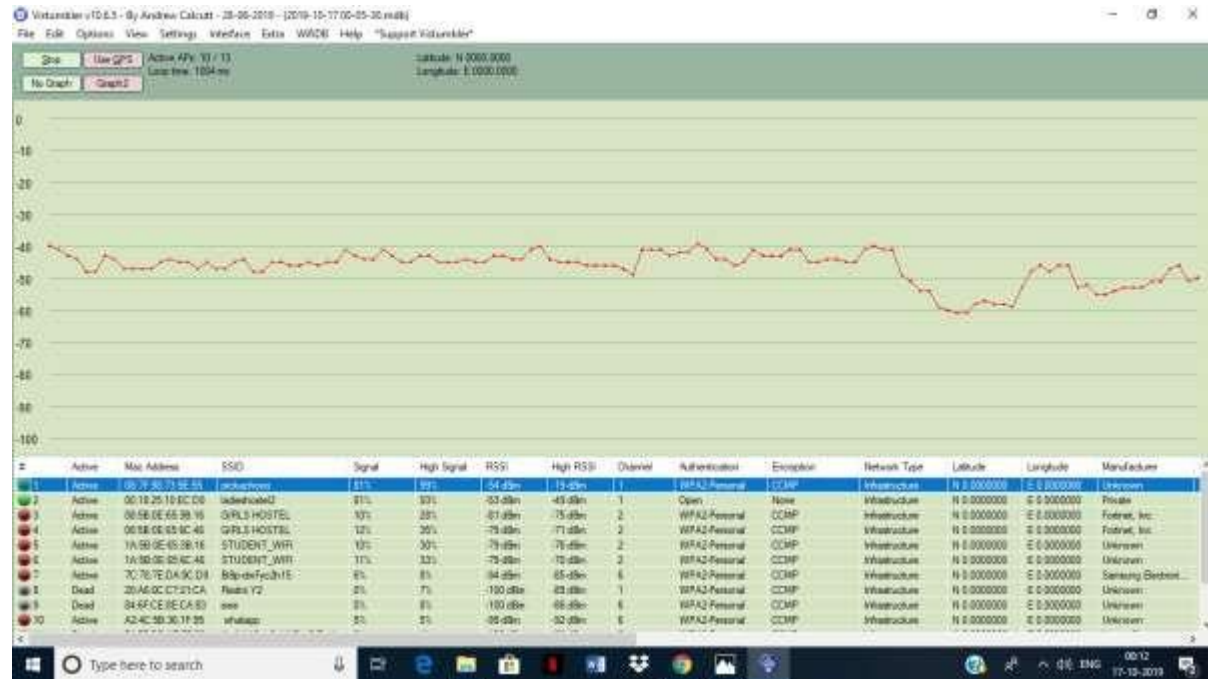


STEP 7:

Major description about the Wireless networks are SSID , Mac Address, channel, network type,Encryption,Radio Type,Authentication,Basic Transfer rate,Manufacturer,Label.

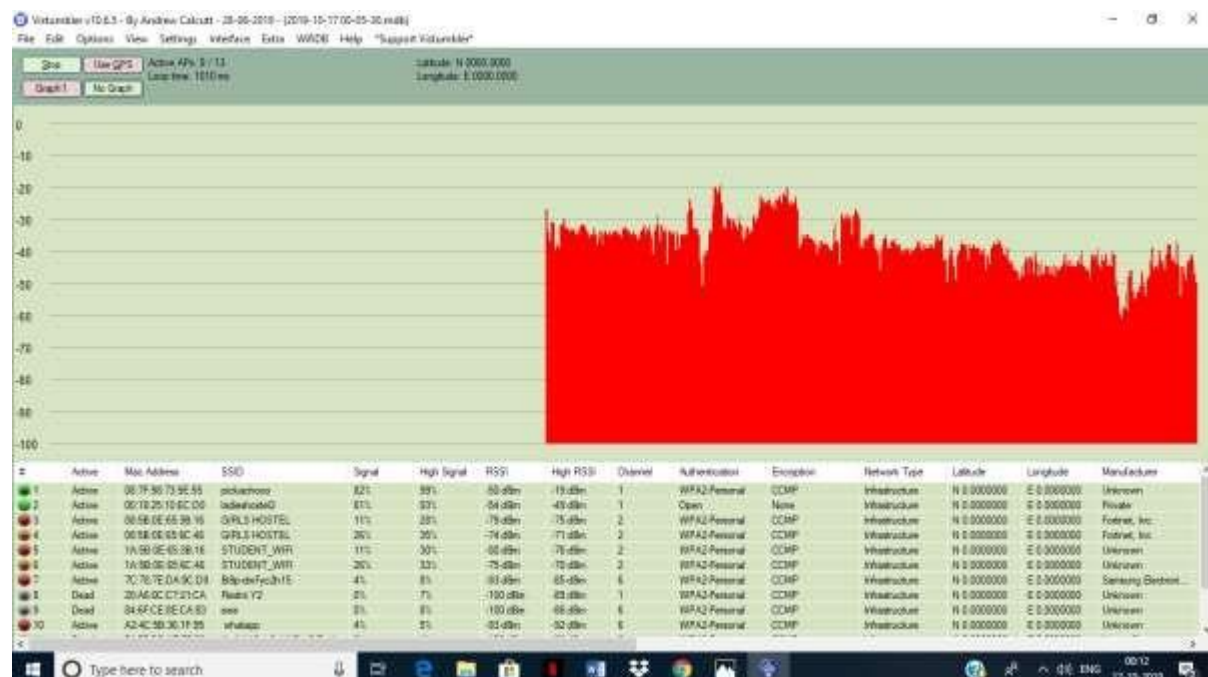
STEP 8:

Visualize the particular signal connection strength using graph 1.



STEP 9:

Visualize the particular signal connection strength using graph 2.



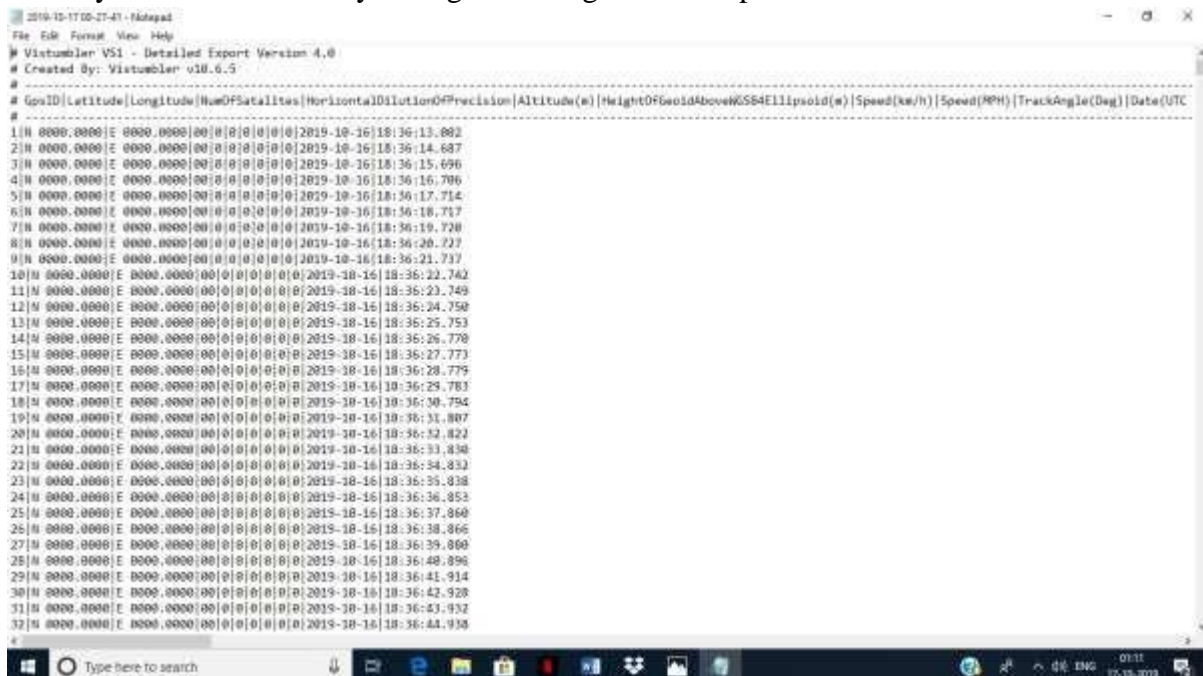
STEP 10:

We can modify the GUI settings accordingly in the vistumbler settings.



STEP 11:

Finally close the session by saving the auto generated report of the session.

**RESULT:**

Thus the wireless audit on an access point or a router and decrypt WEP and has been done successfully.

Ex.No.8 Demonstrate intrusion detection system (ids) using snort tool

Aim:

To demonstrate intrusion detection system using snort tool in packet sniffer mode.

INTRUSION DETECTION SYSTEM :

Intrusion Detection System is a software or device that monitors a network or systems for malicious activity or policy violations

SNORT TOOL:

Snort is an open source Network Intrusion Detection System (NIDS) and it is a packet sniffer that monitors network traffic in real time

Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IP traffic sniffers and analyzers.

Through protocol analysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealthport scans, and SMB probes.

When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window.

So Snort is currently the most popular free network intrusion detection software.

Sniffer mode:

Command: Snort -v

Print out the TCP/IP packets header on the screen

Procedure:

1. Download the following files from the webpages.
 - Snort version : Snort_2_9_14_1_installer.exe
 - Snort rules: snortrules-snapshot-29141.tar.gz
2. Install the Snort.
3. Extract rules from the “snortrules-snapshot-2982.tar.gz” file and move the rules, etc, preproc rules into Snort Home Folder.
(i.e. C:\Snort)
4. Go to the C:\Snort\etc.
5. Edit the “snort.conf” file using notepad editor or equivalent editor(notepad++ is recommended).
6. Set the network variables i.e. first step of Snort
 - ➔ Change HOME_NET to your home network IP address range **your IP Address/24** after checking with command **ipconfig** in command prompt.
 - ➔ Change EXTERNAL_NET to !\$HOME_NET
 - ➔ Change the RULE_PATH - actual path of rule files. i.e “c:\Snort\rules”
 - ➔ Change the PREPROC_RULE_PATH - actual path of preprocessor rule files
i.e c:\Snort\preproc_rules
 - ➔ Comment (put #)SO_RULE_PATH - as windows Snort doesn't use shared object rules
 - ➔ Configure trusted “white.list” file and untrusted “black.list” - reputation preprocessor


```

108
109 # If you are using reputation preprocessor set these
110 # Currently there is a bug with relative paths, they are relative to where snort is
111 # not relative to snort.conf like the above variables
112 # This is completely inconsistent with how other vars work, BUG 89986
113 # Set the absolute path appropriately
114 var WHITE_LIST_PATH c:\Snort\rules
115 var BLACK_LIST_PATH c:\Snort\rules
116

```

7. Configure the decoder

Set the default directory for Snort logs i.e. “c:\Snort\logs”

```

179
180 # Configure default bpf_file to use for filtering what traffic reaches snort.
    line options (-F)
181 #
182 # config bpf_file:
183 #
184
185 # Configure default log directory for snort to log to. For more information s
186 #
187 config logdir: |c:\Snort\log
188
189

```

8. Configure dynamic loaded libraries

Remove the #(comment) and Change the dynamic loaded library path references
dynamicpreprocess directory “c:\Snort\lib\snort_dynamicpreprocessor”
dynamicengine “c:\Snort\lib\snort_dynamicengine\sف_engine.dll”

```

46
47 # path to dynamic preprocessor libraries
48 dynamicpreprocessor directory c:\Snort\lib\snort_dynamicpreprocessor
49
50 # path to base preprocessor engine
51 dynamicengine c:\Snort\lib\snort_dynamicengine\sف_engine.dll
52
53 # path to dynamic rules libraries
54 # dynamicdetection directory /usr/local/lib/snort_dynamicrules
55
56 #####
57 # Step #5: Configure preprocessors
58 # For more information, see the Snort Manual, Configuring Snort - Preprocessors
59 #####
60

```

9. Configure preprocessors

Many Preprocessors are used by Snort - Check Snort manual before setting them.

Comment on “inline packet normalization preprocessor”

This preprocessor is used when Snort is in-line IPS mode

For general purpose Snort usage - check these preprocessors are active
frag3, stream5, http_inject, ftp_telnet, smtp, dns, ssl, sensitive_data

```

263
264 # Inline packet normalization. For more information, see README.normalize
265 # Does nothing in IDS mode
266 # preprocessor normalize_ip4
267 # preprocessor normalize_tcp: ips ecn stream
268 # preprocessor normalize_icmp4
269 # preprocessor normalize_ip6
270 # preprocessor normalize_icmp6
271

```

11. Remove the comment from “preprocessor sfportscan”

```

417
418 # Portscan detection. For more information, see README.sfportscan
419 preprocessor sfportscan: proto { all } memcap { 10000000 } sense_level { low }
420

```

12. change the lines

➔ whitelist \$WHITE_LIST_PATH\white.list, \
blacklist \$BLACK_LIST_PATH\black.list

```

506
507 # Reputation preprocessor. For more information see README.reputation
508 preprocessor reputation: \
509     memcap 500, \
510     priority whitelist, \
511     nested_ip inner, \
512     whitelist $WHITE_LIST_PATH\white.list, \
513     blacklist $BLACK_LIST_PATH\black.list
514

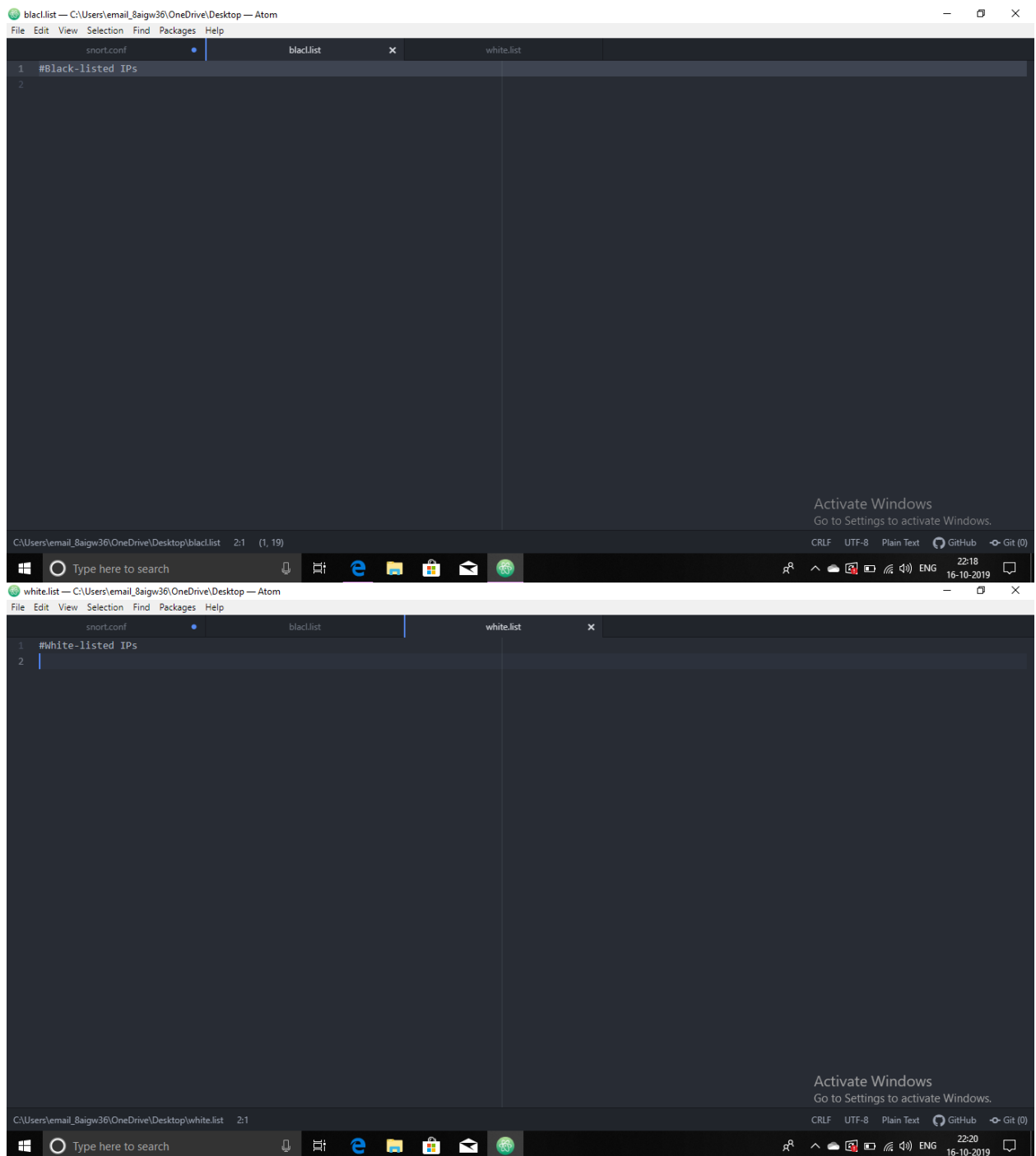
```

13. to create black.list file and white.list files

Open any Editor like Notepad++ and Type the following lines into the file.

#Black-listed IPs and #White-listed IPs

Then save this file in the location of “C:\Snort\rules” and the filename is
black.list” and “white.list”.



14. Customise your rule set

Uncomment “local.rules” but leave include as commented

Change the ‘/’ symbol to ‘\’ symbol cause it denotes the Windows OS

```

546 # site specific rules
547 # include $RULE_PATH\community.rules
548 include $RULE_PATH\local.rules
549
550 #include $RULE_PATH\app-detect.rules
551 #include $RULE_PATH\attack-responses.rules
552 #include $RULE_PATH\backdoor.rules
553 #include $RULE_PATH\bad-traffic.rules
554 #include $RULE_PATH\blacklist.rules
555 #include $RULE_PATH\botnet-cnc.rules
556 #include $RULE_PATH\browser-chrome.rules
557 #include $RULE_PATH\browser-firefox.rules
558 #include $RULE_PATH\browser-ie.rules
559 #include $RULE_PATH\browser-other.rules
560 #include $RULE_PATH\browser-plugins.rules
561 #include $RULE_PATH\browser-webkit.rules
562 #include $RULE_PATH\chat.rules
563 #include $RULE_PATH\content-replace.rules
564 #include $RULE_PATH\ddos.rules
565 #include $RULE_PATH\dns.rules
566 #include $RULE_PATH\dos.rules
567 #include $RULE_PATH\experimental.rules
568 #include $RULE_PATH\exploit-kit.rules
569 #include $RULE_PATH\exploit.rules
570 #include $RULE_PATH\file-executable.rules
571 #include $RULE_PATH\file-flash.rules
572 #include $RULE_PATH\file-identify.rules

```

15. Customise preprocessor and decoder rule set

Uncomment these lines in Step 8

```

"include $PREPROC_RULE_PATH\preprocessor.rules"
"include $PREPROC_RULE_PATH\decoder.rules"
"include $PREPROC_RULE_PATH\sensitive-data.rules"

```

```

655 #####
656 # Step #8: Customize your preprocessor and decoder alerts
657 # For more information, see README.decoder_preproc_rules
658 #####
659
660 # decoder and preprocessor event rules
661 include $PREPROC_RULE_PATH\preprocessor.rules
662 include $PREPROC_RULE_PATH\decoder.rules
663 include $PREPROC_RULE_PATH\sensitive-data.rules
664

```

16. Now Save the snort.conf file.

17. Go to "C:\Snort\rules" and edit the "local.rules" file using notepad++ if necessary

18. Type the following and save the file.

Open CMD and change the Snort folder as a Working Folder.

Command: cd C:\Snort\bin

19. To check the installed version for Snort:

Command: snort -V \

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

,,_
o" )~
'''

-*> Snort! <*-
Version 2.9.14.1-WIN32 GRE (Build 15003)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2019 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

C:\Snort\bin>
```

20. To check to see what network adapters are on your system

Command: snort -W

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

,,_
o" )~
'''

-*> Snort! <*-
Version 2.9.14.1-WIN32 GRE (Build 15003)
By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
Copyright (C) 2014-2019 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       00:00:00:00:00:00      0000:0000:fe80:0000:0000:0000:98da:624f \Device\NPF_{
2       54:EE:75:66:E5:E2      0000:0000:fe80:0000:0000:0000:248e:7e7b \Device\NPF_{

C:\Snort\bin>
```

Important Snort Commands:

Flag	Function
-v	View packet headers at the console.
-d	View application data with IP headers.
-D	Run Snort as a daemon.
-e	Show data-link layer headers.
-l	Run in packet logger mode.
-h	Log information relative to the home network.
-b	Log information to a single binary file in the logging directory.
-r	Read packets contained in a log file.
N	Disable packet logging.
-c	Specifies which file will be used to provide a ruleset for intrusion detection.

Result:

Thus the Intrusion Detection System(IDS) using Snort has been demonstrated successfully.