PANIMALAR ENGINEERING COLLEGE

(A CHRISITIAN MINORITY INSTITUTION)

JAISAKTHI EDUCATIONAL TRUST

ACCREDITED BY NATIONAL BOARD OF ACCREDITATION (NBA)

Bangalore Trunk Road, Varadharajapuram, Nasarathpettai,

Poonamallee, Chennai – 600 123

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

IT8761 – SECURITY LABORATORY

IV CSE - VII SEMESTER

Name:____________________________________

Reg.No:____________________________________

Roll.No:____________________________________

PANIMALAR ENGINEERING COLLEGE

(A CHRISITIAN MINORITY INSTITUTION)
JAISAKTHI EDUCATIONAL TRUST
ACCREDITED BY NATIONAL BOARD OF ACCREDITATION - NBA, NEW DELHI
(An ISO 9001:2000 Certified Institution)

Bangalore Trunk Road, Varadharajapuram, Nasarathpettai,
Poonamallee, Chennai – 600 123

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is a Certified Bonafide Record Book of

Mr. /Ms. ____

Register Number ____

Submitted for University Practical Examination held on_____________in

IT8761 – SECURITY LABORATORY during ____________________________.

Staff – In charge

External Examiner **Internal Examiner**

Ex. No	Date	Title	Page No	Marks	Sign
1.		**SUBSTITUTION TECHNIQUES**			
1a		Caesar Cipher			
1b		Playfair Cipher			
1c		Hill Cipher			
1d		Vigenere Cipher			
2.		**TRANSPOSITION TECHNIQUES**			
2a		Rail fence			
2b		Row & Column Transformation			
3.		DES Algorithm for practical applications.			
4.		AES Algorithm for practical applications.			
5.		RSA Algorithm using HTML and JavaScript			
6.		Diffie-Hellman Key Exchange Algorithm			
7.		Implement Secure Hash Algorithm (SHA - 1)			
8.		Implement Signature Scheme – Digital Signature Standard			
9.		Intrusion Detection System (IDS) - SNORT			
10.		Automated Attack and Penetration Tools Exploring N-Stalker			
11.		**DEFEATING MALWARE**			
11a		Building Trojans			
11b		Rootkit Hunter			
		ADDITIONAL EXPERIMENTS			
12.		Implement Message Digest Algorithm (MD5)			
13.		Creating digital signatures (GnuPG)			
14.		Setup a Honey pot and monitor the Honey pot Network			

1.	**ENCRYPTION/DECRYPTION FOR SUBSTITUTION TECHNIQUES**
Ex. No: 1a	**CAESAR CIPHER (SHIFT CIPHER)**

AIM

To implement a program for encrypting a plain text and decrypting a cipher text using Caesar Cipher (shift cipher) substitution technique.

ALGORITHM DESCRIPTION

It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on. The method is named after Julius Caesar, who used it in his private correspondence. The transformation can be represented by aligning two alphabets; the cipher alphabet is the plain alphabet rotated left or right by some number of positions. 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.

Encryption of a letter x by a shift n can be described mathematically as,

$$\mathbf{E_n(x)=(x+n)\ mod26}$$

Decryption is performed similarly,

$$\mathbf{D_n(x)=(x-n)\ mod26}$$

Procedure for Encryption:

1. Read 'Key' value to shift.
2. Read the **Plaintext** to encrypt.
3. Initialize **Cipher Text** with empty string.
4. Convert each character into Upper text.
5. Read each character 'current' in a PlainText.
   a. Find ASCII value of current character and add key with that.
   b. Find, **Sum**←ASCII(current)+key
   c. Subtract 65 from sum, **Sum ← Sum – 65**
   d. Do modulo division with 26, **Sum ← Sum** MOD **26**
   e. Compute encrypted char by adding 65, **Encrypted**←65+ **Sum**
   f. Add **Encrypted** character to **CipherText.**
6. Repeat step 5 until all characters encrypted.

Procedure for Decryption:

1. Read 'Key' value to shift.
2. Read the **CipherText** to Decrypt.
3. Initialize **PlainText** with empty string.
4. Convert each character into Upper text.
5. Read each character 'current' in a **CipherText**.
   a. Find ASCII value of current character and subtract key with that.
   b. Find, **Sum**←ASCII(current) MOD 65.
   c. Subtract **key** from sum, **Diff** ← **Sum** – **key**
   d. If **Diff** is less than zero, **Diff**←26-**Diff**
   e. Compute **Decrypted** character by adding 65, **Decrypted** ←65+ **Diff**
   f. Add **Decrypted** character to **PlainText.**

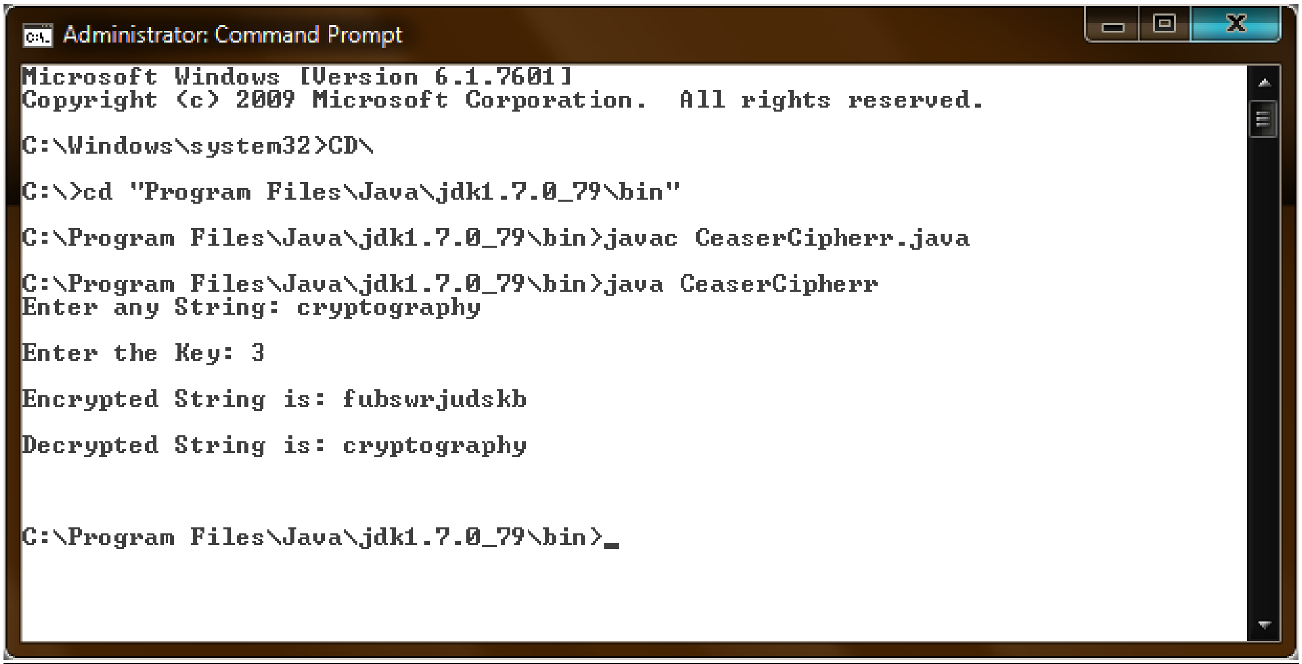
6. Repeat step 5 until all characters Decrypted.

PROGRAM

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
public class CeaserCipherr {
static Scanner sc=new Scanner(System.in);
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
public static void main(String[] args) throws IOException {
System.out.print("Enter any String: "); String str = br.readLine();
System.out.print("\nEnter the Key: ");
int key = sc.nextInt();
String encrypted = encrypt(str, key);
System.out.println("\nEncrypted String is: " +encrypted);
String decrypted = decrypt(encrypted, key);
System.out.println("\nDecrypted String is: " +decrypted);
System.out.println("\n");
}
public static String encrypt(String str, int key) { String encrypted = "";
for(int i = 0; i < str.length(); i++)
{
int c = str.charAt(i);
if (Character.isUpperCase(c))
 {
 c = c + (key % 26);
if (c > 'Z')
c = c - 26;
}
else if (Character.isLowerCase(c)) { c = c + (key % 26);
if (c > 'z')
```

```
c = c - 26;
}
encrypted += (char) c;
}
return encrypted;
}
public static String decrypt(String str, int key) { String decrypted = "";
for(int i = 0; i < str.length(); i++) { int c = str.charAt(i);
if (Character.isUpperCase(c)) { c = c - (key % 26);
if (c < 'A')
c = c + 26;
}
else if (Character.isLowerCase(c)) { c = c - (key % 26);
if (c < 'a')
c = c + 26;
}
decrypted += (char) c;
}
return decrypted;
}
}
```

OUTPUT



RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Caesar cipher (Shift Cipher) substitution technique is done using java and output is verified successfully

1.	ENCRYPTION/DECRYPTION FOR SUBSTITUTION TECHNIQUES
Ex. No: 1b	PLAYFAIR CIPHER

AIM

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

ALGORITHM DESCRIPTION

The Playfair cipher uses a 5 by 5 table containing a key word or phrase. To generate the key table, first fill the spaces in the table with the letters of the keyword, then fill the remaining spaces with the rest of the letters of the alphabet in order (usually omitting "Q" to reduce the alphabet to fit; other versions put both "I" and "J" in the same space). The key can be written in the top rows of the table, from left to right, or in some other pattern, such as a spiral beginning in the upper-left-hand corner and ending in the centre.

The keyword together with the conventions for filling in the 5 by 5 table constitutes the cipher key. To encrypt a message, one would break the message into digrams (groups of 2 letters) such that, for example, "HelloWorld" becomes "HE LL OW OR LD", and map them out on the key table. Then apply the following 4 rules, to each pair of letters in the plaintext:

1. If both letters are the same (or only one letter is left), add an "X" after the first letter. Encrypt the new pair and continue. Some variants of Playfair use "Q" instead of "X", but any letter, itself uncommon as a repeated pair, will do.
2. If the letters appear on the same row of your table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).
3. If the letters appear on the same column of your table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).
4. If the letters are not on the same row or column, replace them with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first letter of the encrypted pair is the one that lies on the same row as the first letter of the plaintext pair.

To decrypt, use the INVERSE (opposite) of the last 3 rules, and the 1st as-is (dropping any extra "X"s, or "Q"s that do not make sense in the final message when finished).

Procedure for Encryption:

1. Read the key.
2. Read the Plaintext.
3. Initialize the PlayFair table array as 5x5 two dimensional array.
4. Add each character of the key in the array without duplicates.
5. Add alphabets from A,B..Z in the remaining array positions without duplicates.
6. Read First two characters from the Plaintext and find their locations (R1, C1), (R2, C2) in array.
    a. If R1=R2, then the characters at the position (R1, (C1+1%5)) and (R2,(C2+1%5)) are the equivalent Cipher Text.
    b. Otherwise, the characters at location (R1, C2) and (R2, C1) are the equivalent Cipher Text.

Repeat step 6 until all characters converted into Cipher Text.

Procedure for Decryption:

1. Read the key.
2. Read the Cipher Text.
3. Initialize the PlayFair table array as 5x5 two dimensional array.
4. Add each character of the key in the array without duplicates.
5. Add alphabets from A, B...Z in the remaining array positions without duplicates.
6. Read First two characters from the Cipher Text and find their locations (R1, C1), (R2, C2) in array.
    a. If R1=R2, then the characters at the position (R1, (C1-1%5)) and (R2, (C2-1%5)) are the equivalent Plaintext.
    b. Otherwise, the characters at location (R1, C2) and (R2, C1) are the equivalent Cipher Text.

Repeat step 6 until all characters converted into Plaintext.

PROGRAM

```
import java.awt.Point;
class playfairCipher
{
 private static char[][] charTable;
 private static Point[] positions;
 private static String prepareText(String s,boolean chgJtoI)
 {
  s=s.toUpperCase().replaceAll("[^A-Z]","");
  return chgJtoI?s.replace("J","I"):s.replace("Q","");
 }
 private static void createTbl(String key,boolean chgJtoI)
 {
  charTable=new char[5][5];
  positions=new Point[26];
  String s=prepareText(key+"ABCDEFGHIJKLMNOPQRSTUVWXYZ",chgJtoI);
  int len=s.length();
  for(int i=0,k=0;i<len;i++)
  {
   char c=s.charAt(i);
   if(positions[c-'A']==null)
   {
    charTable[k/5][k%5]=c;
    positions[c-'A']=new Point(k%5,k/5);
    k++;
   }
  }
 }
 private static String codec(StringBuilder txt,int dir)
 {
  int len=txt.length();
  for(int i=0;i<len;i+=2)
  {
   char a=txt.charAt(i);
   char b=txt.charAt(i+1);
   int row1=positions[a-'A'].y;
   int row2=positions[b-'A'].y;
   int col1=positions[a-'A'].x;
   int col2=positions[b-'A'].x;
   if(row1==row2)
   {
    col1=(col1+dir)%5;
    col2=(col2+dir)%5;
   }
   else if(col1==col2)
   {
```

```
   row1=(row1+dir)%5;
   row2=(row2+dir)%5;
  }
  else
  {
   int tmp=col1;
   col1=col2;
   col2=tmp;
  }
  txt.setCharAt(i,charTable[row1][col1]);
  txt.setCharAt(i+1,charTable[row2][col2]);
 }
 return txt.toString();
}
private static String encode(String s)
{
 StringBuilder sb=new StringBuilder(s);
 for(int i=0;i<sb.length();i+=2)
 {
  if(i==sb.length()-1)
  {
   sb.append(sb.length()%2==1?'X':"");
  }
  else if(sb.charAt(i)==sb.charAt(i+1))
  {
   sb.insert(i+1,'X');
  }
 }
return codec(sb,1);
}
private static String decode(String s)
{
return codec(new StringBuilder(s),4);
}
public static void main(String[] args)throws java.lang.Exception
{
String key="CSE";
String txt="Security Lab";/*make sure string length is even*//*change J to I*/
boolean chgJtoI=true;
createTbl(key,chgJtoI);
String enc=encode(prepareText(txt,chgJtoI));
System.out.println("Stimulating Playfair Cipher\n----------");
System.out.println("Input Message:"+txt);
System.out.println("Encrypted Message:"+enc);
System.out.println("Decrypted Message:"+decode(enc));
}
}
```

OUTPUT

```
Administrator: Command Prompt
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Windows\system32>cd\

C:\>cd "Program Files\Java\jdk1.7.0_79\bin"

C:\Program Files\Java\jdk1.7.0_79\bin>javac playfair.java

C:\Program Files\Java\jdk1.7.0_79\bin>java playfair
Enter a keyword:
monarchy
monarchybdefgiklpqstuvwxz
m o n a r

c h y b d

e f g i k

l p q s t

u v w x z

To Encrypt enter 1
To Decript enter 2
Testing both enter anything else:
3
Enter word to encrypt & decrypt:
Make Sure the length of message is even
cryptography
Encription Start
cr
yp
to
gr
ap
hy
dmhqprknosyb
Encrypted Text is->dmhqprknosyb
Decription Start
dm
hq
pr
kn
os
yb
cryptography
Decrypted Text is->cryptography

C:\Program Files\Java\jdk1.7.0_79\bin>_
```

RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Playfair Cipher substitution technique is done using java and output is verified successfully.

1.	ENCRYPTION/DECRYPTION FOR SUBSTITUTION TECHNIQUES
Ex. No: 1c	HILL CIPHER

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

ALGORITHM DESCRIPTION

The Hill cipher is a substitution cipher invented by Lester S. Hill in 1929. Each letter is represented by a number modulo 26. To encrypt a message, each block of n letters is multiplied by an invertible $n \times n$ matrix, again modulus 26.

To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible $n \times n$ matrices (modulo 26).

The cipher can, be adapted to an alphabet with any number of letters. All arithmetic just needs to be done modulo the number of letters instead of modulo 26.

Procedure for Encryption:

1. Read the key size 'N'.
2. Read the Key matrix.
3. Read the Plaintext.
4. Split Plaintext into group of N characters.
5. Convert each group into column matrix. With the ASCII (character) Mod 26.
6. Multiply each group column matrix by Key matrix. Do modulo division by 26 with the result.
7. Add 65 with each value in the resultant Matrix.
8. Add the ASCII character equivalent of the resultant matrix elements to the Cipher Text.

Repeat step 5 to 8 until all characters converted into Cipher Text.

Procedure for Decryption:

1. Read the key size 'N'.
2. Read the Key matrix.
3. Read the CipherText.
4. Split CipherText into group of N characters.
5. Convert each group into column matrix. With the ASCII (character) Mod 26.
6. Multiply each group column matrix by inverse of Key matrix. Do modulo division by 26 with the result.
7. Add 65 with each value in the resultant Matrix.
8. Add the ASCII character equivalent of the resultant matrix elements to the Plaintext.

Repeat step 5 to 8 until all characters converted into Plaintext.

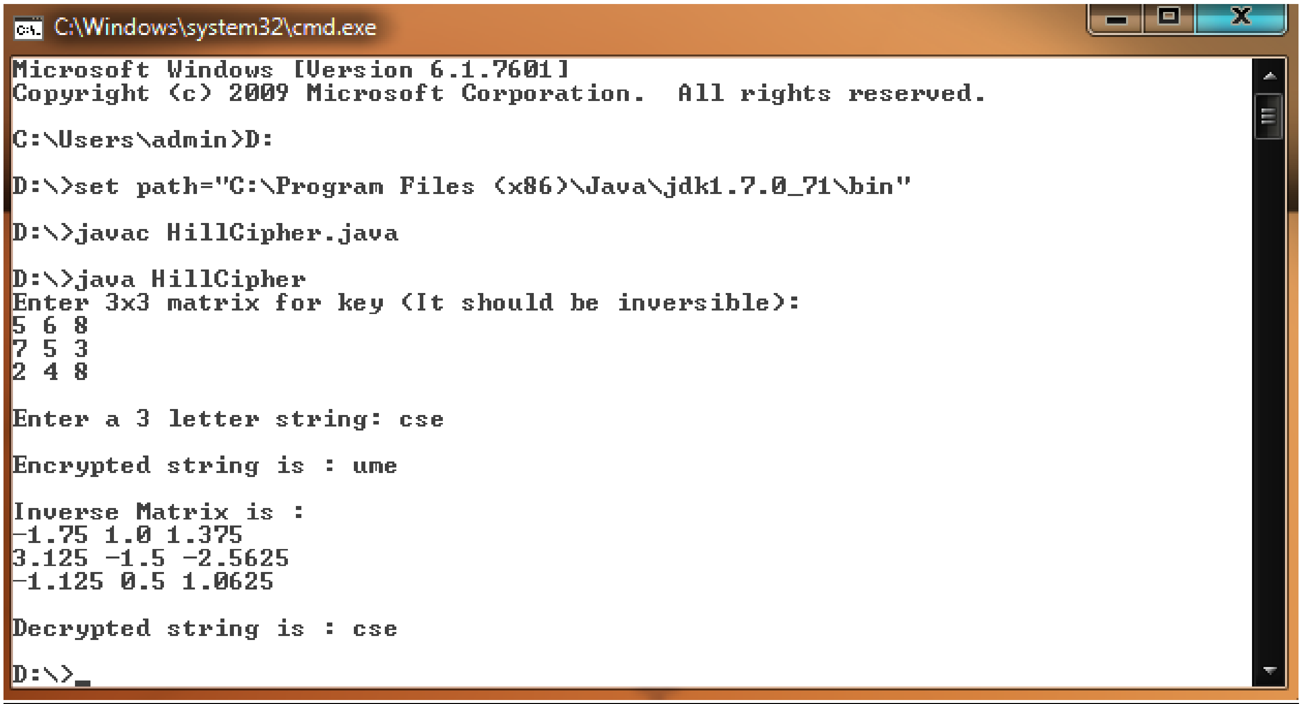
PROGRAM

```
import java.io.*;
import java.util.*;
import java.io.*;
public class HillCipher {
static float[][] decrypt = new float[3][1];
static float[][] a = new float[3][3];
static float[][] b = new float[3][3];
static float[][] mes = new float[3][1];
static float[][] res = new float[3][1];
static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
static Scanner sc = new Scanner(System.in);
public static void main(String[] args) throws IOException {
 // TODO code application logic here
getkeymes();
for(int i=0;i<3;i++)
for(int j=0;j<1;j++)
for(int k=0;k<3;k++)
{
res[i][j]=res[i][j]+a[i][k]*mes[k][j];
}
System.out.print("\nEncrypted string is : ");
for(int i=0;i<3;i++)
{
System.out.print((char)(res[i][0]%26+97));
res[i][0]=res[i][0];
 }
inverse();
for(int i=0;i<3;i++)
```

```
for(int j=0;j<1;j++)
for(int k=0;k<3;k++)
 {
decrypt[i][j] = decrypt[i][j]+b[i][k]*res[k][j];
}
System.out.print("\nDecrypted string is : ");
for(int i=0;i<3;i++)
{
System.out.print((char)(decrypt[i][0]%26+97));
 }
System.out.print("\n");
 }
public static void getkeymes() throws IOException {
System.out.println("Enter 3x3 matrix for key (It should be inversible): ");
for(int i=0;i<3;i++)
for(int j=0;j<3;j++)
a[i][j] = sc.nextFloat();
System.out.print("\nEnter a 3 letter string: ");
 String msg = br.readLine();
for(int i=0;i<3;i++)
mes[i][0] = msg.charAt(i)-97;
 }
public static void inverse() {
float p,q;
float[][] c = a;
for(int i=0;i<3;i++)
for(int j=0;j<3;j++)
{
 //a[i][j]=sc.nextFloat();
if(i==j)
```

```
b[i][j]=1;
else b[i][j]=0;
 }
for(int k=0;k<3;k++)
{
for(int i=0;i<3;i++)
{
 p = c[i][k];
 q = c[k][k];
for(int j=0;j<3;j++)
 {
if(i!=k)
{
c[i][j] = c[i][j]*q-p*c[k][j];
b[i][j] = b[i][j]*q-p*b[k][j];
 } } } }
for(int i=0;i<3;i++)
for(int j=0;j<3;j++)
{
b[i][j] = b[i][j]/c[i][i];
}
System.out.println("");
System.out.println("\nInverse Matrix is : ");
for(int i=0;i<3;i++)
{
for(int j=0;j<3;j++)
System.out.print(b[i][j] + " ");
System.out.print("\n");
} } }
```

OUTPUT



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\admin>D:

D:\>set path="C:\Program Files (x86)\Java\jdk1.7.0_71\bin"

D:\>javac HillCipher.java

D:\>java HillCipher
Enter 3x3 matrix for key (It should be inversible):
5 6 8
7 5 3
2 4 8

Enter a 3 letter string: cse

Encrypted string is : ume

Inverse Matrix is :
-1.75 1.0 1.375
3.125 -1.5 -2.5625
-1.125 0.5 1.0625

Decrypted string is : cse

D:\>_
```

RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Hill cipher substitution technique is done using java and output is verified successfully.

1.	**ENCRYPTION/DECRYPTION FOR SUBSTITUTION TECHNIQUES**
Ex. No: 1d	**VIGENERE CIPHER**

AIM

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

ALGORITHM DESCRIPTION

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. It is a simple form of polyalphabetic substitution.To encrypt, a table of alphabets can be used, termed a Vigenere square, or Vigenere table. 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. At different points in the encryption process, the cipher uses a different alphabet from one of the rows used. The alphabet at each point depends on a repeating keyword.

Procedure for Encryption:

1. Read the Key.
2. Read the PlainText.
3. Initialize keyindex to 0 and CipherText to empty.
4. Read each character (P) in the Plaintext, and key character(K) at position keyindex.
5. Compute PV←ASCII(P) MOD 26 and KV← ASCII(K) MOD 26
6. Add PV and KV and do modulo division, SUM←(PV+KV) MOD 26.
7. Add 65 with the Sum and Generate ASCII character corresponding to the resultant Sum. The resultant character is the Cipher equivalent of current character. Add it to the CipherText.
8. Increment keyindex. If keyindex reaches LENGTH(Key), initialize eyindex-0)

Procedure for decryption:

1. Read the Key.
2. Read the CipherText.
3. Initialize keyindex to 0 and CipherText to empty.
4. Read each character (C) in the Plaintext, and key character(K) at position keyindex.
5. Compute CV←ASCII(C) MOD 26 and KV← ASCII(K) MOD 26
6. Compute Sum ←(26-KV)+CV MOD 26.
7. Add 65 with the Sum and Generate ASCII character corresponding to the resultant Sum. The resultant character is the PlainText equivalent of current character. Add it to the PlainText.
8. Increment keyindex. If keyindex reaches LENGTH(Key), initialize keyindex-0)

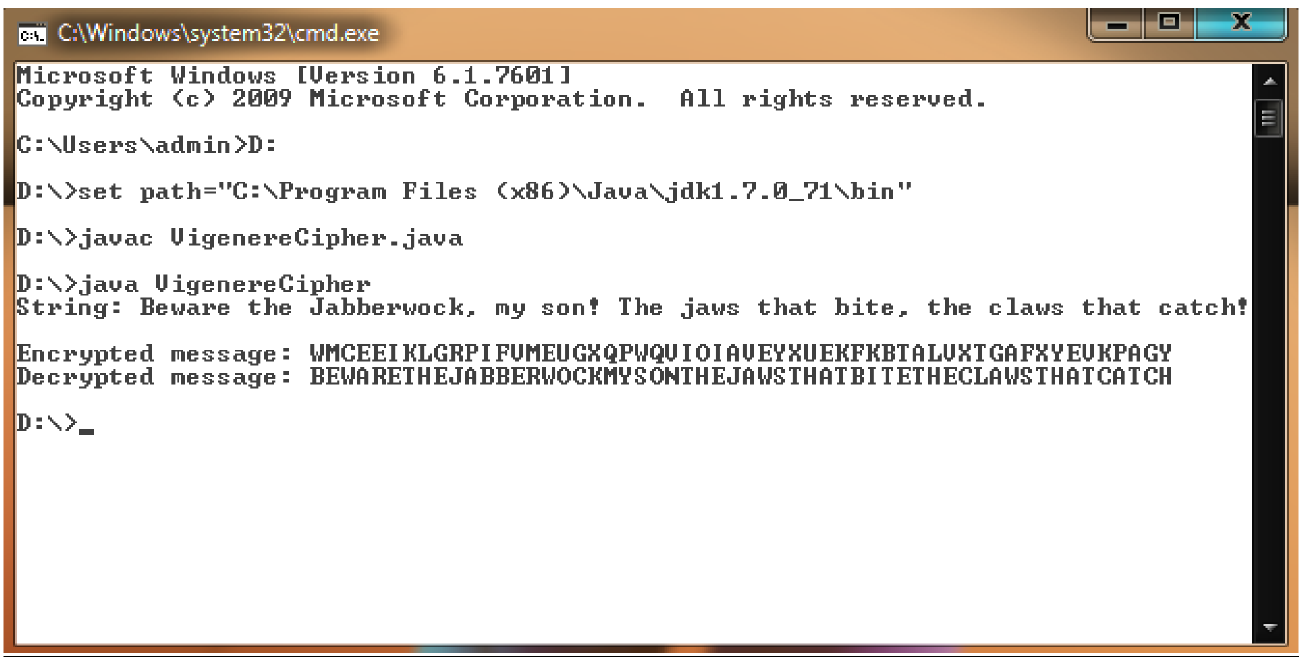
Repeat steps from 4 to 8 until all characters encrypted.

PROGRAM

```
public class VigenereCipher
{
    public static String encrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            res += (char) ((c + key.charAt(j) - 2 * 'A') % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
    }
    public static String decrypt(String text, final String key)
    {
        String res = "";
        text = text.toUpperCase();
        for (int i = 0, j = 0; i < text.length(); i++)
        {
            char c = text.charAt(i);
            if (c < 'A' || c > 'Z')
                continue;
            res += (char) ((c - key.charAt(j) + 26) % 26 + 'A');
            j = ++j % key.length();
        }
        return res;
```

```
  }
  public static void main(String[] args)
  {
     String key = "VIGENERECIPHER";
     String message = "Beware the Jabberwock, my son! The jaws that bite, the claws that catch!";
     String encryptedMsg = encrypt(message, key);
     System.out.println("String: " + message);
     System.out.println("Encrypted message: " + encryptedMsg);
     System.out.println("Decrypted message: " + decrypt(encryptedMsg, key));
  }}
```

OUTPUT



RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Vigenere cipher substitution technique is done using java and output is verified successfully.

2.	ENCRYPTION/DECRYPTION FOR TRANSPOSITION TECHNIQUES
Ex. No: 2a	RAILFENCE CIPHER

AIM

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

ALGORITHM DESCRIPTION

In the rail fence cipher, the plaintext is written downwards and diagonally on successive "rails" of an imaginary fence, then moving up when we reach the bottom rail. When we reach the top rail, the message is written downwards again until the whole plaintext is written out. The message is then read off in rows.

Procedure for Encryption:

1. Read the Key lines N.
2. Read the PlainText.
3. For each line l in N
4. Calculate the offset corresponding to the line.
5. Read the characters from the PlainText in the index l,l+offset*1,l+offset*2.
6. Add the characters to the Ciphertext.
7. Increment line by 1.Repeat steps 4 to 6 for all lines.

Procedure for Decryption:

1. Read the Key lines N.
2. Read the CipherText.
3. For each line l in N
4. Calculate the offset corresponding to the line.
5. Read the characters from the CipherText in the index l, l+offset*1, l+offset*2.
6. Add the characters to the PlainText.
7. Increment line by 1.Repeat steps 4 to 6 for all lines.

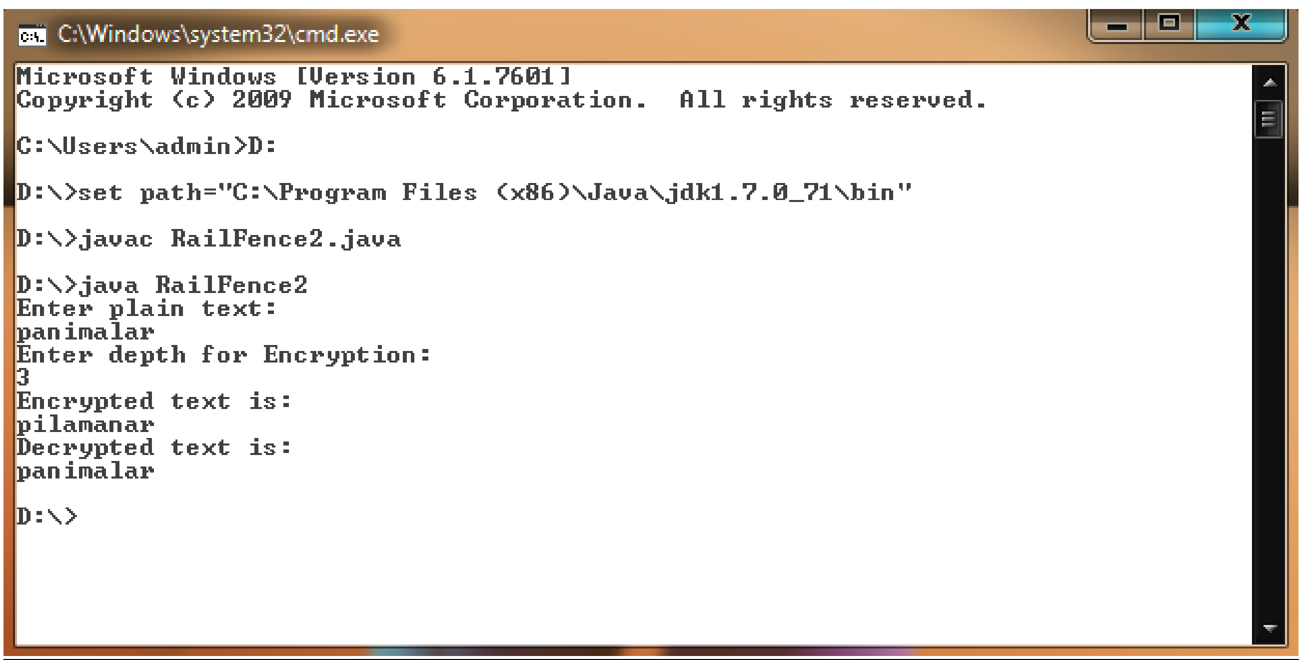
PROGRAM

```
import java.util.*;
class RailFenceBasic{
int depth;
String Encryption(String plainText,int depth)throws Exception
{
  int r=depth,len=plainText.length();
  int c=len/depth;
  char mat[][]=new char[r][c];
  int k=0;
 String cipherText="";
  for(int i=0;i< c;i++)
  {
    for(int j=0;j< r;j++)
   {
    if(k!=len)
    mat[j][i]=plainText.charAt(k++);
    else
    mat[j][i]='X';
   }
  }
  for(int i=0;i< r;i++)
  {
   for(int j=0;j< c;j++)
   {
    cipherText+=mat[i][j];
   }
  }
  return cipherText;
 }
```

```
 String Decryption(String cipherText,int depth)throws Exception
 {
  int r=depth,len=cipherText.length();
  int c=len/depth;
  char mat[][]=new char[r][c];
  int k=0;
  String plainText="";
  for(int i=0;i< r;i++)
  {
   for(int j=0;j< c;j++)
   {
    mat[i][j]=cipherText.charAt(k++);
   }
  }
  for(int i=0;i< c;i++)
  {
   for(int j=0;j< r;j++)
   {
    plainText+=mat[j][i];
   }
  }
  return plainText;
 }
}
class RailFence2{
 public static void main(String args[])throws Exception
 {
  RailFenceBasic rf=new RailFenceBasic();
  Scanner scn=new Scanner(System.in);
  int depth;
```

```
 String plainText,cipherText,decryptedText;
 System.out.println("Enter plain text:");
 plainText=scn.nextLine();
 System.out.println("Enter depth for Encryption:");
 depth=scn.nextInt();
 cipherText=rf.Encryption(plainText,depth);
 System.out.println("Encrypted text is:\n"+cipherText);
 decryptedText=rf.Decryption(cipherText, depth);
 System.out.println("Decrypted text is:\n"+decryptedText);
}}
```

OUTPUT



RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Railfence cipher transposition technique is done using java and output is verified successfully.

2.	ENCRYPTION/DECRYPTION FOR TRANSPOSITION TECHNIQUES
Ex. No: 2b	ROW AND COLUMN TRANSPOSITION

AIM

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

ALGORITHM DESCRIPTION

A Columnar transposition also known as row-column transpose is a very simple cipher to perform by hand. The name of the cipher comes after the operations on a matrix that are performed during both, encryption and decryption. The number of columns of the matrix is determined by the secret key. The secret key is usually a word (or just a sequence of letters). It has to be converted into a sequence of numbers. The numbers are defined by an alphabetical order of the letters in the keyword. The letter which is first in the alphabet will be the number 1; the second letter in the alphabetical order will be 2, and so on. If there are multiple identical letters in the keyword, each next occurrence of the same letter should be converted into a number that is equal to the number for the previous occurrence increased by one.

Procedure for Encryption:

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

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, theorder 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.

Procedure for Decryption:

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

PROGRAM

```
import java.util.*;
import java.io.*;
import java.lang.*;
public class columnarTranspose {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      String line = System.getProperty("line.separator");
      scan.useDelimiter(line);
      System.out.print("1. Encryt 2.Decrypt : ");
      int option = scan.nextInt();
      switch (option) {
         case 1:
            System.out.print("Enter String:");
            String text = scan.next();
            System.out.print("Enter Key:");
            String key = scan.next();
            System.out.println(encryptCT(key, text).toUpperCase());
            break;
         case 2:
            System.out.print("Enter Encrypted String:");
            text = scan.next();
            System.out.print("Enter Key:");
            key = scan.next();
            System.out.println(decryptCT(key, text));
            break;
         default:
            break;
      }
   }
```

```
public static String encryptCT(String key, String text) {
    int[] arrange = arrangeKey(key);
    int lenkey = arrange.length;
    int lentext = text.length();
    int row = (int) Math.ceil((double) lentext / lenkey);
    char[][] grid = new char[row][lenkey];
    int z = 0;
    for (int x = 0; x < row; x++) {
        for (int y = 0; y < lenkey; y++) {
            if (lentext == z) {
                // at random alpha for trailing null grid
                grid[x][y] = RandomAlpha();
                z--;
            } else {
                grid[x][y] = text.charAt(z);
            }
            z++;
        }
    }
    String enc = "";
    for (int x+1 = 0; x < lenkey; x++)
        { for (int y = 0; y < lenkey; y++)
        {
            if (x == arrange[y]) {
                for (int a = 0; a < row; a++) {
                    enc = enc + grid[a][y];
                }
            }
        }
    }
    return enc;
```

```
}
public static String decryptCT(String key, String text) {
    int[] arrange = arrangeKey(key);
    int lenkey = arrange.length;
    int lentext = text.length();
    int row = (int) Math.ceil((double) lentext / lenkey);
    String regex = "(?<=\\G.{" + row + "})";
    String[] get = text.split(regex);
    char[][] grid = new char[row][lenkey];
    for (int x = 0; x < lenkey; x++) {
        for (int y = 0; y < lenkey; y++) {
            if (arrange[x] == y) {
                for (int z = 0; z < row; z++) {
                    grid[z][y] = get[arrange[y]].charAt(z);
                }
            }
        }
    }
    String dec = "";
    for (int x = 0; x < row; x++) {
        for (int y = 0; y < lenkey; y++) {
            dec = dec + grid[x][y];
        }
    }
    return dec;
}
public static char RandomAlpha() {
    //generate random alpha for null space
    Random r = new Random();
    return (char)(r.nextInt(26) + 'a');
```

```
}
public static int[] arrangeKey(String key) {
    //arrange position of grid
    String[] keys = key.split("");
    Arrays.sort(keys);
    int[] num = new int[key.length()];
    for (int x = 0; x < keys.length; x++) {
        for (int y = 0; y < key.length(); y++) {
            if (keys[x].equals(key.charAt(y) + "")) {
                num[y] = x;
                break;
            }
        }
    }
    return num;
}}
```

OUTPUT

```
E:\>javac columnarTranspose.java

E:\>java columnarTranspose
1. Encryt 2.Decrypt : 1
Enter String:NETWORKSECURITY
Enter Key:31452
EKROCYNRUTSIWET

E:\>java columnarTranspose
1. Encryt 2.Decrypt : 2
Enter Encrypted String:EKROCYNRUTSIWET
Enter Key:31452
NETWORKSECURITY

E:\>
```

RESULT

Thus the program for encrypting a plain text and decrypting a cipher test using Row and Column transposition technique is done using java and output is verified successfully.

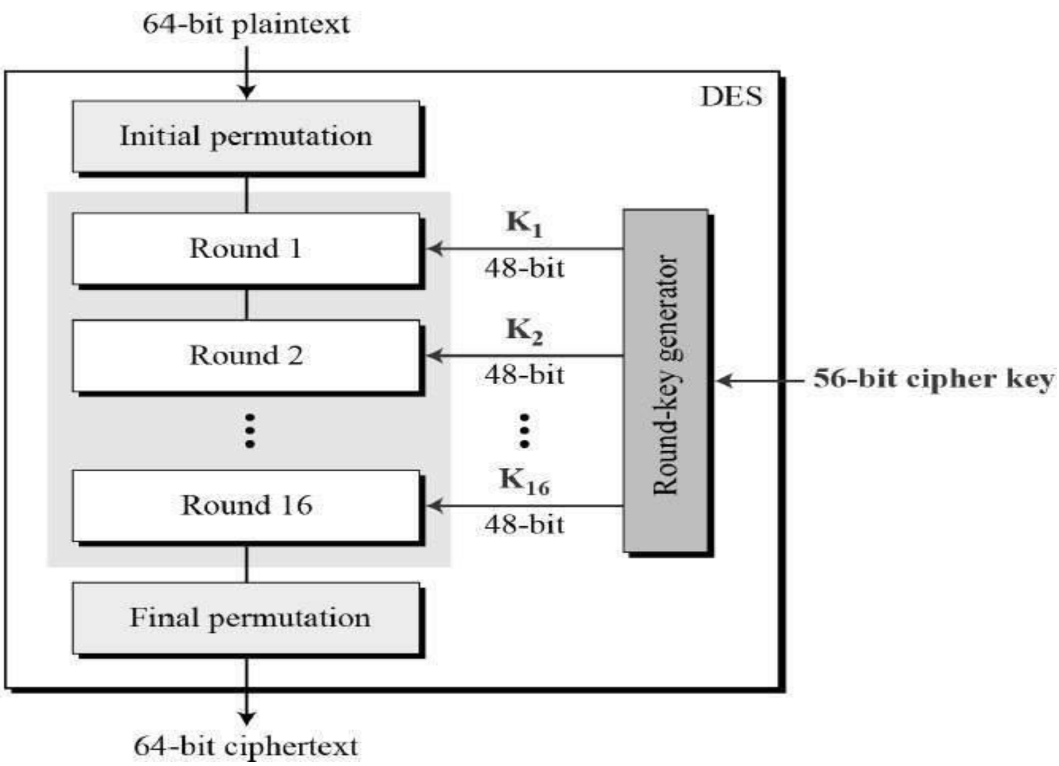
Ex. No: 3	NETWORK SECURITY ALGORITHMS
	DATA ENCRYPTION STANDARD (DES) ALGORITHM

AIM

To develop a program to implement Data Encryption Standard for Encryption and Decryption for practical applications.

ALGORITHM DESCRIPTION

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST). DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DEShas an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration



Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation

ALGORITHM

1. The 64-bit plain text block is handed over to an **Initial Permutation (IP)** function.
2. The Initial Permutation (IP) produces two halves of the permuted block; say **Left Plain Text (LPT)** and **Right Plain Text (RPT).**
3. Now each of LPT and RPT go through 16 rounds of encryption process.
4. In the end. LPT and RPT are joined and a Final Permutation (FP) is performed on the combined block.
5. The result of this process produces 64-bit cipher text.

PROGRAM

```
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);
```

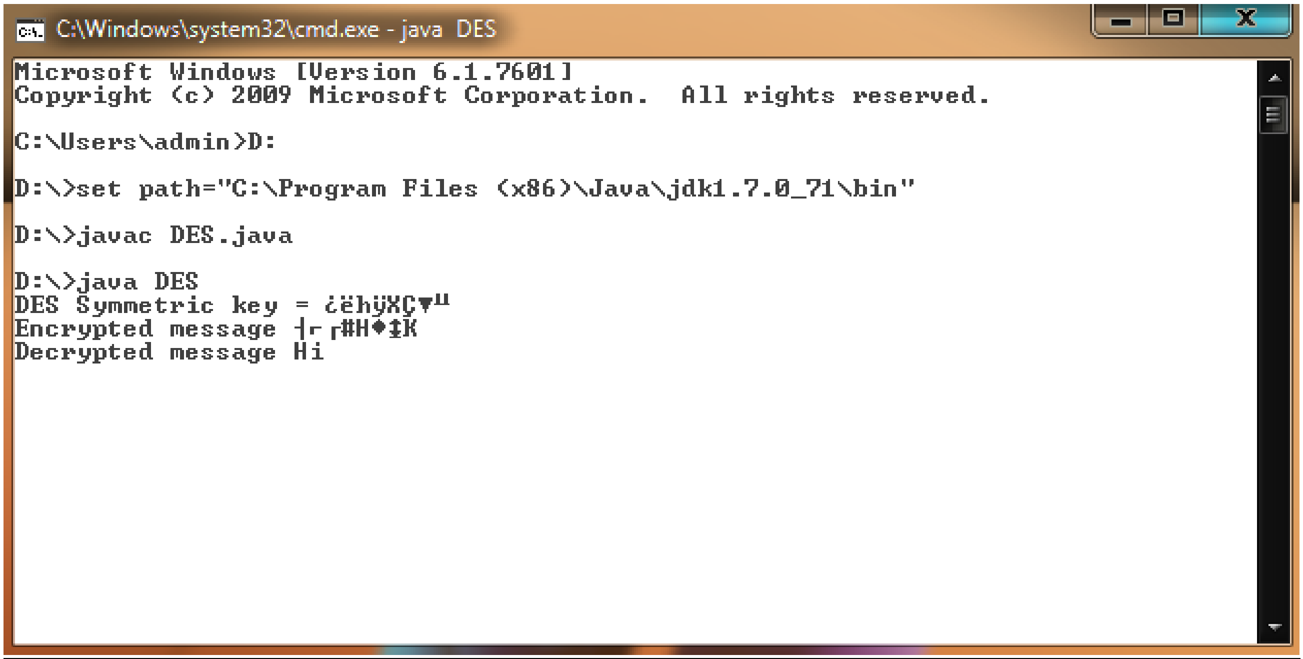
```
            System.out.println("Encrypted message "+encryptedData);
            JOptionPane.showMessageDialog(null,"Encrypted Data "+"\n"+encryptedData);

            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 skeySpec = new SecretKeySpec(raw, "DES");
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
        byte[] encrypted = cipher.doFinal(clear);
        return encrypted;
    }
    private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
        SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");
        Cipher cipher = Cipher.getInstance("DES");
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
        byte[] decrypted = cipher.doFinal(encrypted);
        return decrypted;
    }

            public static void main(String args[]) {
                    DES des = new DES();
            }
}
```

OUTPUT


C:\Windows\system32\cmd.exe - java DES
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.
C:\Users\admin>D:
D:\>set path="C:\Program Files (x86)\Java\jdk1.7.0_71\bin"
D:\>javac DES.java
D:\>java DES
DES Symmetric key = ¿ëhÿXÇ
Encrypted message
Decrypted message Hi


Input
Enter message to encrypt
Hi
OK
Cancel


Message
Encrypted Data
OK


Message
Decrypted Data
Hi
OK

RESULT

Thus the program for encrypting a plain text and decrypting a cipher text using DES algorithm is done using java and output is verified successfully

Ex. No: 4	NETWORK SECURITY ALGORITHMS
	ADVANCED ENCRYPTION STANDARD (AES) ALGORITHM

AIM

To develop a program to implement Advanced Encryption Standard for encryption and decryption for practical applications.

ALGORITHM DESCRIPTION

Advanced Encryption Standard (AES) is a symmetric block cipher chosen by the U.S government to protect classified information. AES is implemented in software and hardware throughout the world to encrypt sensitive data. The National Institute of Standards and Technology (NIST) started development of AES in 1997 when it announced the need for an alternative to the Data Encryption Standard (DES), which was starting to become vulnerable to brute-force attacks.

The three variants of AES are based on different key sizes (128, 192, and 256 bits). In this article, we will focus on the 128-bit version of the AES key schedule, which provides sufficient background to understand the 192 and 256 bit variants as well. At the end, we'll include a note the other variants, and how they differ from the 128-bit version.

Encryption with AES

The encryption phase of AES can be broken into three phases: the initial round, the main rounds, and the final round. All of the phases use the same sub-operations in different combinations as follows:

- Initial Round

  AddRoundKey

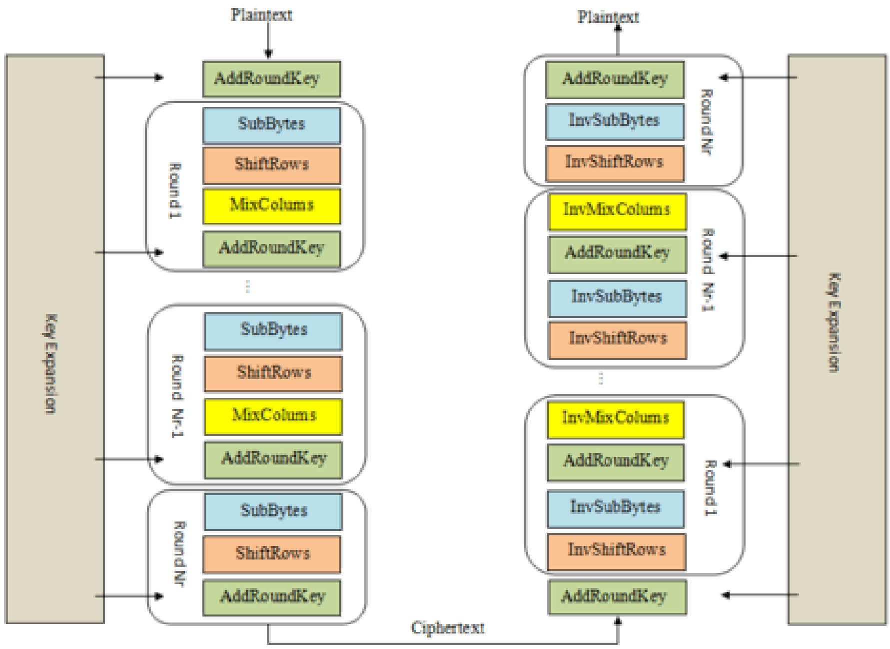
- Main Rounds

  SubBytes
  ShiftRows
  MixColumns
  AddRoundKey

- Final Round

  SubBytes
  ShiftRows
  AddRoundKey

The main rounds of AES are repeated a set number of times for each variant of AES. AES-128 uses 9 iterations of the main round, AES-192 uses 11, and AES-256 uses 13.



Overall structure of encryption and decryption in Advanced Encryption Standard

ALGORITHM

(Encryption of a 128 bit block)

1. Derive the set of round keys from the cipher key
2. Initialize the state array with the block data (plaintext)
3. Add the initial round key to the starting state array
4. Perform nine rounds of state manipulation.
5. Perform the tenth and final round of state manipulation
6. Copy the final state array out as the encrypted data (ciphertext)

The reason that the rounds have been listed as “nine followed by a final tenth round” is because the tenth round involves a slightly different manipulation from the others.

PROGRAM

```
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 AES {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String inputMessage,encryptedData,decryptedMessage;

public AES() {
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("AES Symmetric key = "+skeyString);
}
catch(Exception e) {
System.out.println(e);
}
}
private static byte[] getRawKey(byte[] seed) throws Exception {
KeyGenerator kgen = KeyGenerator.getInstance("AES");
SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
sr.setSeed(seed);
kgen.init(128, sr); // 192 and 256 bits may not be available
SecretKey skey = kgen.generateKey();
raw = skey.getEncoded();
return raw;
}
```

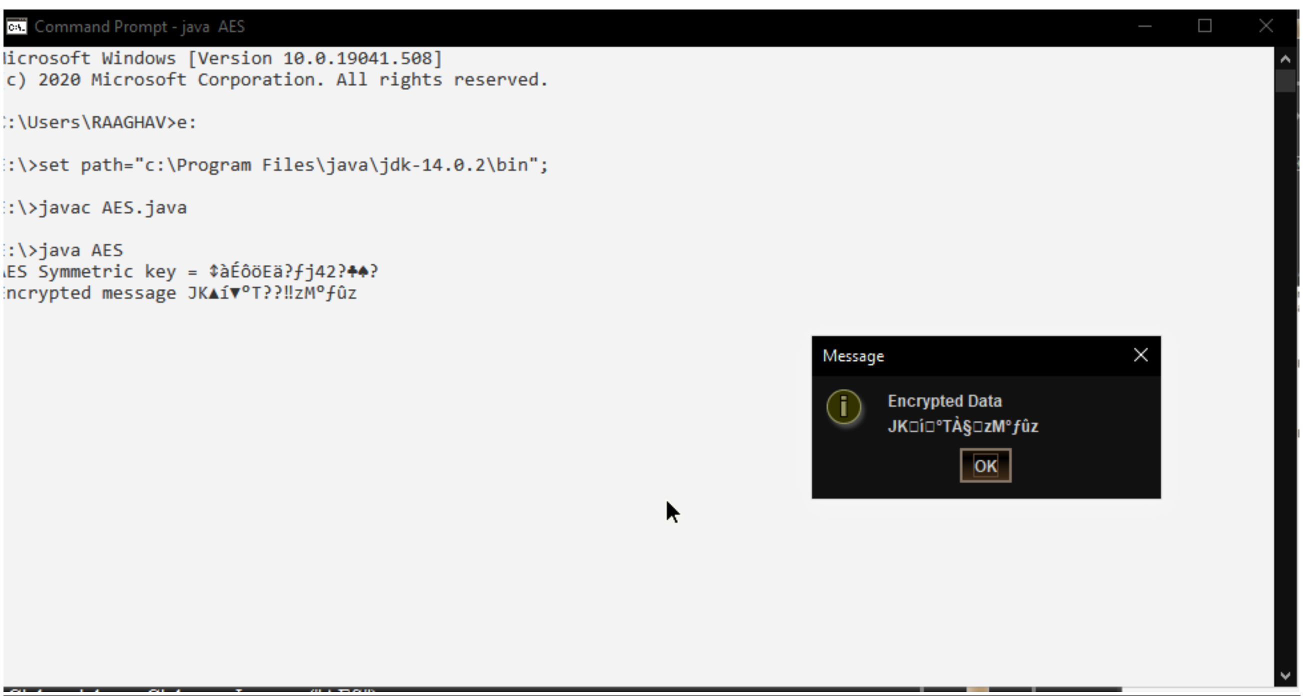
```
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
byte[] encrypted = cipher.doFinal(clear);
return encrypted;
}

private static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception {
SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, skeySpec);
byte[] decrypted = cipher.doFinal(encrypted);
return decrypted;
}

public static void main(String args[]) {
AES aes = new AES();
}
}
```

OUTPUT

Select Command Prompt - java AES
Microsoft Windows [Version 10.0.19041.508]
(c) 2020 Microsoft Corporation. All rights reserved.
C:\Users\RAAGHAV>e:
E:\>set path="c:\Program Files\java\jdk-14.0.2\bin";
E:\>javac AES.java
E:\>java AES
AES Symmetric key = ‡àÉôöEä?ƒj42?♣♠?
Input
Enter message to encrypt
Hello
OK
Cancel



Command Prompt - java AES
icrosoft Windows [Version 10.0.19041.508]
c) 2020 Microsoft Corporation. All rights reserved.
:\Users\RAAGHAV>e:
:\>set path="c:\Program Files\java\jdk-14.0.2\bin";
:\>javac AES.java
:\>java AES
ES Symmetric key = ‡àÉôöEä?ƒj42?♣♠?
ncrypted message JK▲í▼°T??‼zM°ƒûz
Message
Encrypted Data
OK

RESULT

Thus the program for encrypting a plain text and decrypting a cipher text using AES algorithm is done using java and output is verified successfully

Ex. No: 5	NETWORK SECURITY ALGORITHMS
	RIVEST SHAMIR ADLEMAN (RSA) ALGORITHM

AIM

Develop a program to implement RSA algorithm for encryption and decryption using HTML and JavaScript

This cryptosystem is one the initial system. It remains most employed cryptosystem even today. The system was invented by three scholars **Ron Rivest, Adi Shamir,** and **Len Adleman** and hence, it is termed as RSA cryptosystem. The two aspects of the RSA cryptosystem, first the generation of key pair and second the encryption-decryption algorithms.

ALGORITHM DESCRIPTION

Generation of RSA Key Pair

Each person or a party who desires to participate in communication using encryption needs to generate a pair of keys, namely public key and private key. The process followed in the generation of keys is described below −

- **Generate the RSA modulus (n)**
    - Select two large primes, p and q.
    - Calculate n=p*q. For strong unbreakable encryption, let n be a large number, typically a minimum of 512 bits.
- **Find Derived Number (e)**
    - Number **e** must be greater than 1 and less than (p − 1)(q − 1).
    - There must be no common factor for e and (p − 1)(q − 1) except for 1. In other words two numbers e and (p – 1)(q – 1) are coprime.
- **Form the public key**
    - The pair of numbers (n, e) form the RSA public key and is made public.
    - Interestingly, though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.
- **Generate the private key**
    - Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.

- Number d is the inverse of e modulo (p - 1)(q – 1). This means that d is the number less than (p - 1)(q - 1) such that when multiplied by e, it is equal to 1 modulo (p - 1)(q - 1).
- This relationship is written mathematically as follows ed = 1 mod (p − 1)(q − 1)

The Extended Euclidean Algorithm takes p, q, and e as input and gives d as output

ENCRYPTION AND DECRYPTION

Once the key pair has been generated, the process of encryption and decryption are relatively straightforward and computationally easy.
Interestingly, RSA does not directly operate on strings of bits as in case of symmetric key encryption. It operates on numbers modulo n. Hence, it is necessary to represent the plaintext as a series of numbers less than n.

RSA Encryption

- Suppose the sender wish to send some text message to someone whose public key is (n, e).
- The sender then represents the plaintext as a series of numbers less than n.
- To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as $C = P^e \bmod n$
- In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.
- Returning to our Key Generation example with plaintext P = 10, we get ciphertext C

$C = 10^5 \bmod 91$

RSA Decryption

- The decryption process for RSA is also very straightforward. Suppose that the receiver of public-key pair (n, e) has received a ciphertext C.
- Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

  Plaintext = $C^d \bmod n$
- Returning again to our numerical example, the ciphertext C = 82 would get decrypted to number 10 using private key 29 − Plaintext = $82^{29} \bmod 91 = 10$

PROGRAM

```
<html>
<head>
<title>Input</title>
<script language="JavaScript">
<!-- hide from old browsers
function gcd (a, b)
{
  var r;
  while (b>0)
  {
    r=a%b;
    a=b;
    b=r;
  }
  return a;
}
function rel_prime(phi)
{
  var rel=5;
  while (gcd(phi,rel)!=1)
    rel++;
  return rel;
}
function power(a, b)
{
  var temp=1, i;
  for(i=1;i<=b;i++)
    temp*=a;
   return temp;
```

```
}
function encrypt(N, e, M)
{
  var r,i=0,prod=1,rem_mod=0;
  while (e>0)
  {
    r=e % 2;
    if (i++==0)
      rem_mod=M % N;
    else
      rem_mod=power(rem_mod,2) % N;
    if (r==1)
    {
      prod*=rem_mod;
      prod=prod % N;
    }
    e=parseInt(e/2);
  }
  return prod;
}
function calculate_d(phi,e)
{
  var x,y,x1,x2,y1,y2,temp,r,orig_phi;
  orig_phi=phi;
  x2=1;x1=0;y2=0;y1=1;
  while (e>0)
  {
    temp=parseInt(phi/e);
    r=phi-temp*e;
    x=x2-temp*x1;
```

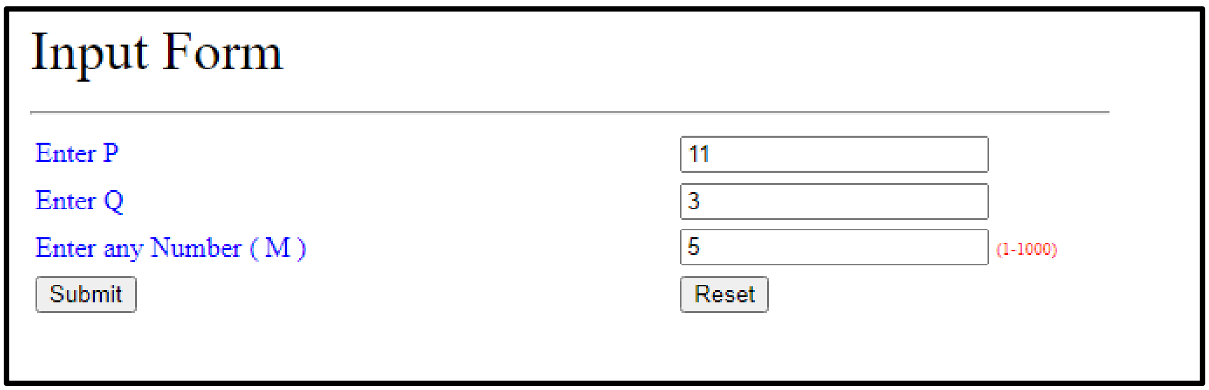
```
    y=y2-temp*y1;
    phi=e;e=r;
    x2=x1;x1=x;
    y2=y1;y1=y;
    if (phi==1)
    {
      y2+=orig_phi;
      break;
    }
  }
  return y2;
}
function decrypt(c, d, N)
{
  var r,i=0,prod=1,rem_mod=0;
  while (d>0)
  {
    r=d % 2;
    if (i++==0)
      rem_mod=c % N;
    else
      rem_mod=power(rem_mod,2) % N;
    if (r==1)
    {
      prod*=rem_mod;
      prod=prod % N;
    }
    d=parseInt(d/2);
  }
  return prod;
```

```
}
function openNew()
{
Var subWindow=window.open("Output.htm","Obj","HEIGHT=400,WIDTH=600,
SCROLLBARS=YES");
  var p=parseInt(document.Input.p.value); var
  q=parseInt(document.Input.q.value); var
  M=parseInt(document.Input.M.value);var
  N=p * q;
  var phi=(p-1)*(q-1); var
  e=rel_prime(phi); var
  c=encrypt(N,e,M);
  var d=calculate_d(phi,e); var
 dis1="Encrypted Text:";
 document.write(dis1);
 document.write(c);
var dis2="\n";
  var pt=decrypt(c,d,N);
  var dis="Decrypted Text:";
  document.write(dis);
  document.write(pt);
/*
  subWindow.document.Output.N.value=N;
  subWindow.document.Output.phi.value=phi;
  subWindow.document.Output.e.value=e;
  subWindow.document.Output.c.value=c;
  subWindow.document.Output.d.value=d;
  subWindow.document.Output.M.value=decrypt(c,d,N);
*/
}
```

```
// end scripting here -->
</script>
</head>
<body>
<p><font size="6">Input Form</font></p>
<hr>
<form name="Input">
<table border="0" width="100%" height="109">
  <tr>
    <td width="24%" height="23">
        <font color="#0000FF">Enter P</font></td>
    <td width="76%" height="23">
         <input type="text" name="p" size="20"></td>
  </tr>
  <tr>
    <td width="24%" height="23"><font color="#0000FF">
             Enter Q</font></td>
    <td width="76%" height="23">
          <input type="text" name="q" size="20"></td>
  </tr>
  <tr>
    <td width="24%" height="20">
          <font color="#0000FF">Enter any Number ( M )</font></td>
    <td width="76%" height="20"><input type="text" name="M" size="20">
        <font size="1" color="#FF0000">(1-1000)</font></td>
  </tr>
  <tr>
    <td width="24%" height="19"><input type="button"
         value="Submit" name="Submit" onClick="openNew()"></td>
    <td width="76%" height="19"><input type="reset"
```

```
        value="Reset" name="Reset"></td>
  </tr>
</table>
</form>
<p> </p>
</body>
</html>
```

OUTPUT



Encrypted Text:14
Decrypted Text:5

RESULT:

Thus the program for encrypting a plain text and decrypting a cipher text using RSA algorithm is done using HTML/Javascript and output is verified successfully

Ex. No: 6	NETWORK SECURITY ALGORITHMS DIFFIE-HELLMAN KEY EXCHANGE ALGORITHM

AIM

Develop a program to implement Diffie Hellman Key Exchange Algorithm for encryption and Decryption.

ALGORITHM DESCRIPTION

Diffie–Hellman key exchange (**D–H**) is a specific method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric key cipher.

Algorithm

1. **Global Public Elements:**

   Let q be a prime number and α where $\alpha < q$ and α is a primitive root of q.

2. **User A Key Generation:**

   Select private X_A where $X_A < q$

   Calculate public Y_A where $Y_A = \alpha^{XA} \bmod q$

3. **User B Key Generation:**

   Select private X_B where $X_B < q$

   Calculate public Y_B where $Y_B = \alpha^{XB} \bmod q$

4. **Calculation of Secret Key by User A**

   $K = (Y_B)^{XA} \bmod q$

5. **Calculation of Secret Key by User B:**

   $K = (Y_A)^{XB} \bmod q$

PROGRAM

```
import java.io.*;
import java.math.BigInteger;
class Diffie
{
    public static void main(String[]args)throws IOException
    {
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter prime number:");
        BigInteger p=new BigInteger(br.readLine());
        System.out.print("Enter primitive root of "+p+":");
        BigInteger g=new BigInteger(br.readLine());
        System.out.println("Enter value for x less than "+p+":");
        BigInteger x=new BigInteger(br.readLine());
        BigInteger R1=g.modPow(x,p);
        System.out.println("R1="+R1);
        System.out.print("Enter value for y less than "+p+":");
        BigInteger y=new BigInteger(br.readLine());
        BigInteger R2=g.modPow(y,p);
        System.out.println("R2="+R2);
        BigInteger k1=R2.modPow(x,p);
        System.out.println("Key calculated at Alice's side:"+k1);
        BigInteger k2=R1.modPow(y,p);
        System.out.println("Key calculated at Bob's side:"+k2);
        System.out.println("Diffie Hellman secret key Encryption has Taken");
    }
}
```

OUTPUT

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\admin>D:

D:\>set path="C:\Program Files (x86)\Java\jdk1.7.0_71\bin"

D:\>javac Diffie.java

D:\>java Diffie
Enter prime number:
11
Enter primitive root of 11:7
Enter value for x less than 11:
3
R1=2
Enter value for y less than 11:6
R2=4
Key calculated at Alice's side:9
Key calculated at Bob's side:9
Diffie Hellman secret key Encryption has Taken

D:\>_
```

RESULT

Thus the program for encrypting a plain text using Diffie Hellman key exchange algorithm is done using java and output is verified successfully.

Ex. No: 7	MESSAGE DIGEST
	SECURE HASH FUNCTION (SHA-1) ALGORITHM

AIM

To calculate the message digests of a text using Secure Hash Algorithm (SHA-1)

The **Secure Hash Algorithm** is a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS), including:

- **SHA-0**: A retronym applied to the original version of the 160-bit hash function published in 1993 under the name "SHA". It was withdrawn shortly after publication due to an undisclosed "significant flaw" and replaced by the slightly revised version SHA-1.
- **SHA-1**: A 160-bit hash function which resembles the earlier MD5 algorithm. This was designed by the National Security Agency (NSA) to be part of the Digital Signature Algorithm. Cryptographic weaknesses were discovered in SHA-1, and the standard was no longer approved for most cryptographic uses after 2010.
- **SHA-2**: A family of two similar hash functions, with different block sizes, known as *SHA- 256* and *SHA-512*. They differ in the word size; SHA-256 uses 32-bit words where SHA-512 uses 64-bit words. There are also truncated versions of each standard, known as *SHA-224*, *SHA-384*, *SHA-512/224* and *SHA-512/256*. These were also designed by the NSA.
- **SHA-3**: A hash function formerly called *Keccak*, chosen in 2012 after a public competition among non-NSA designers. It supports the same hash lengths as SHA-2, and its internal structure differs significantly from the rest of the SHA family.

ALGORITHM DESCRIPTION

Secured Hash Algorithm-1 (SHA-1):

Step 1: Append Padding Bits….

Message is “padded” with a 1 and as many 0’s as necessary to bring the message length to 64 bits fewer than an even multiple of 512.

Step 2: Append Length....

64 bits are appended to the end of the padded message. These bits hold the binary format of 64 bits indicating the length of the original message.

Step 3: Prepare Processing Functions….

SHA1 requires 80 processing functions defined as:

f(t;B,C,D) = (B AND C) OR ((NOT B) AND D) (0 <= t <= 19)

f(t;B,C,D) = B XOR C XOR D (20 <= t <= 39)

f(t;B,C,D) = (B AND C) OR (B AND D) OR (C AND D) (40 <= t<=59)

f(t;B,C,D) = B XOR C XOR D (60 <= t <= 79)

Step 4: Prepare Processing Constants....

SHA1 requires 80 processing constant words defined as:

K(t) = 0x5A827999 (0 <= t <= 19)

K(t) = 0x6ED9EBA1 (20 <= t <= 39)

K(t) = 0x8F1BBCDC (40 <= t <= 59)

K(t) = 0xCA62C1D6 (60 <= t <= 79)

Step 5: Initialize Buffers….

SHA1 requires 160 bits or 5 buffers of words (32 bits):

H0 = 0x67452301

H1 = 0xEFCDAB89

H2 = 0x98BADCFE

H3 = 0x10325476

H4 = 0xC3D2E1F0

Step 6: Processing Message in 512-bit blocks (L blocks in total message)….

This is the main task of SHA1 algorithm which loops through the padded and appended message in 512-bit blocks.

Input and predefined functions: M[1, 2, ..., L]: Blocks of the padded and appended message f(0;B,C,D), f(1,B,C,D), ..., f(79,B,C,D): 80 Processing Functions K(0), K(1),...... , K(79): 80 Processing Constant Words

H0, H1, H2, H3, H4, H5: 5 Word buffers with initial values

Step 6: Pseudo Code….

For loop on k = 1 to L

(W(0),W(1), ..,W(15)) = M[k] */* Divide M[k] into 16 words */*

For t = 16 to 79 do:

W(t) = (W(t-3) XOR W(t-8) XOR W(t-14) XOR W(t-16)) <<< 1

A = H0, B = H1, C = H2, D = H3, E = H4

For t = 0 to 79 do:

TEMP = A<<<5 + f(t;B,C,D) + E + W(t) + K(t) E = D, D = C,

C = B<<<30, B = A, A = TEMP

End of for loop

H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E

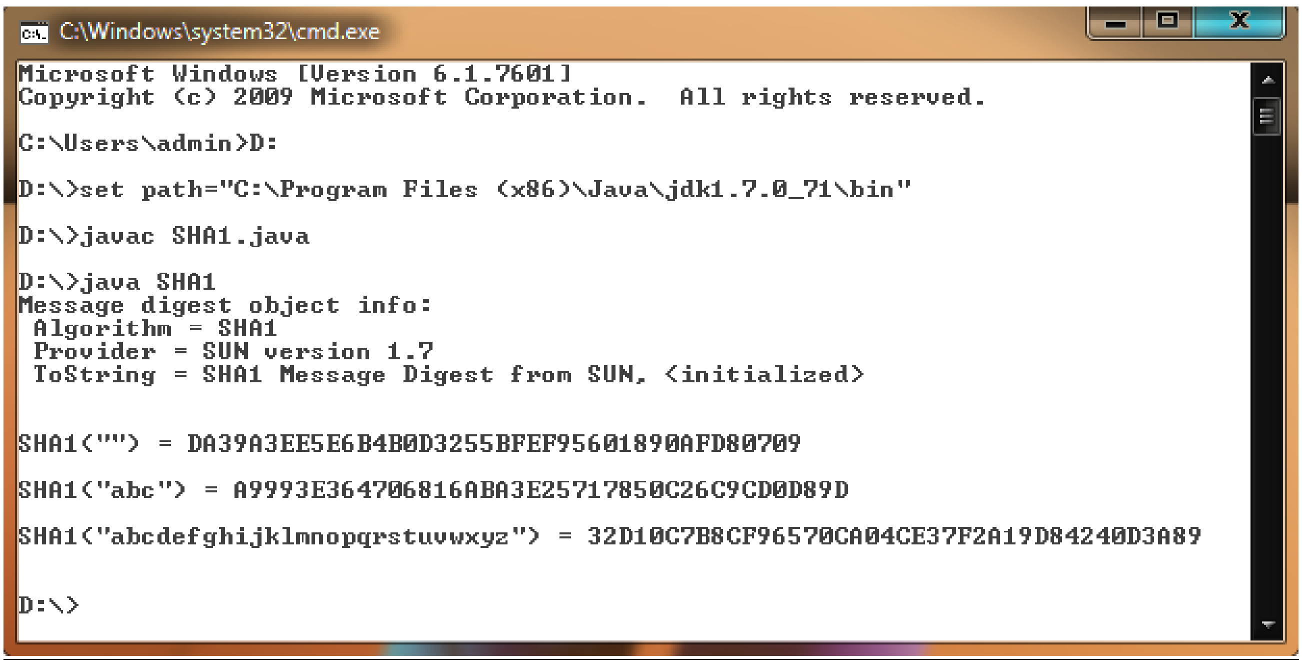
End of for loop

PROGRAM

```
import java.security.*;
public class SHA1 {
public static void main(String[] a) {
try {
MessageDigest md = MessageDigest.getInstance("SHA1");
System.out.println("Message digest object info: ");
System.out.println(" Algorithm = " +md.getAlgorithm());
System.out.println(" Provider = " +md.getProvider());
System.out.println(" ToString = " +md.toString());
String input = "";
md.update(input.getBytes());
byte[] output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
input = "abc";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"\") = " +bytesToHex(output));
input = "abcdefghijklmnopqrstuvwxyz";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("SHA1(\"" +input+"\") = " +bytesToHex(output));
System.out.println("");
}
catch (Exception e) {
System.out.println("Exception: " +e);
 }  }
```

```
public static String bytesToHex(byte[] b) {
 char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
StringBuffer buf = new StringBuffer();
for (int j=0; j<b.length; j++) {
buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
buf.append(hexDigit[b[j] & 0x0f]); }
return buf.toString();
} }
```

OUTPUT



RESULT

Thus the program for implementing Secure Hash Function (SHA-1) algorithm is done using java and output is verified successfully.

Ex. No: 8	**SIGNATURE SCHEME**
	DIGITAL SIGNATURE STANDARD

AIM

To write a program to implement the digital signature scheme in java

ALGORITHM DESCRIPTION

The use and verification of digital signatures is a standard engine that is included in the security provider architecture. Like the other engines, the classes that implement this engine have both a public interface and an SPI for implementers of the engine.

In the JDK, the most common use of digital signatures is to create signed classes. Users have the option of granting additional privileges to these signed classes using the mechanics of the access controller. In addition, a security manager and a class loader can use this information to change the policy of the security manager.

DESIGN

The following global public key components are chosen in key generation

P is a random l-bit prime, $512 \le l \le 1024$, $l = 64t$, where $t = 8,\ldots, 16$

q is a random 160-bit prime dividing p-1

$r = h^{(p-1)/q} \bmod p$, where h is a random primitive element of Z_p, such that $r>1$

User Key Components

x is a private key which is random integer $0 < x < q$

$y = r^x \bmod p$

Therefore the key = (p, q, r, x, y)

After computing the key, User A publish the key (p, q, r, y) in public directory.

SIGNATURE

Signature of a 160-bit plaintext w to be sent

Choose a random number k, where $0 < k < q$ such that gcd (k, q) = 1

Compute $a = (r^k \bmod p) \bmod q$

Compute $b = k^{-1} (w + xa) \bmod q$, (or) $bk = (w + xa) \bmod q$ where $kk^{-1} \equiv 1 \pmod q$

Signature: sig (w, k) = (a, b)

VERIFICATION

Verification of Signature (a, b)

Compute $z = b^{-1} \bmod q$

Compute $u_1 = wz \bmod q$, $u_2 = az \bmod q$

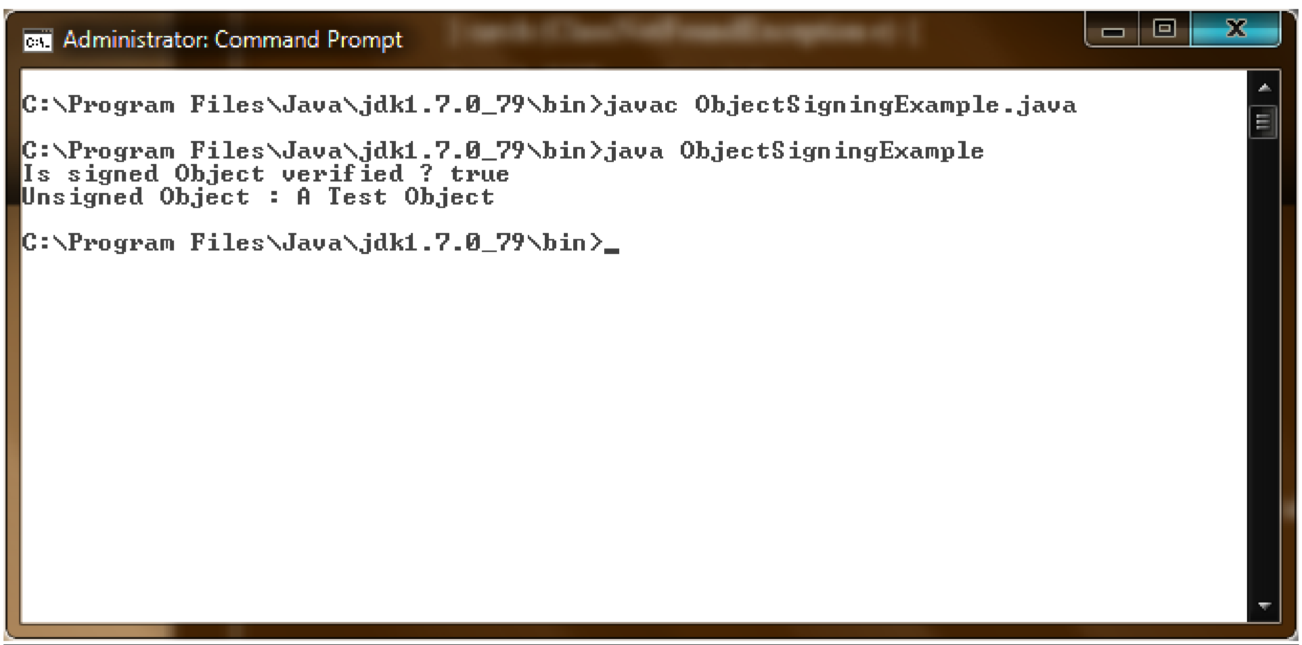
Verification: $Ver_k (w, a, b) = true$

PROGRAM →SIGNATURE VERIFICATION

```
import java.io.IOException;
import java.security.InvalidKeyException;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.SignedObject;
public class ObjectSigningExample {
public static void main(String[] args) {
try {
	// Generate a 1024-bit Digital Signature Algorithm (DSA) key pair
	KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("DSA");
	keyPairGenerator.initialize(1024);
	KeyPair keyPair = keyPairGenerator.genKeyPair();
	PrivateKey privateKey = keyPair.getPrivate();
	PublicKey publicKey = keyPair.getPublic();
		// We can sign Serializable objects only
	String unsignedObject = new String("A Test Object");
	Signature signature = Signature.getInstance(privateKey.getAlgorithm());
	SignedObject signedObject = new SignedObject(unsignedObject, privateKey,  signature);
		// Verify the signed object
```

```
Signature sig = Signature.getInstance(publicKey.getAlgorithm());
boolean verified = signedObject.verify(publicKey, sig);
System.out.println("Is signed Object verified ? " + verified);
        // Retrieve the object
unsignedObject = (String) signedObject.getObject();
System.out.println("Unsigned Object : " + unsignedObject);
} catch (SignatureException e) {
} catch (InvalidKeyException e) {
} catch (NoSuchAlgorithmException e) {
} catch (ClassNotFoundException e) {
} catch (IOException e) {
}}}
```

OUTPUT



PROGRAM →SIGNATURE GENERATION

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import  java.security.PrivateKey;
import java.security.PublicKey;
class GenerateKey {
public static void main(String[] args) {
try {
        // Generate a 1024-bit Digital Signature Algorithm (DSA) key pair
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");
        keyGen.initialize(1024);
        KeyPair keypair = keyGen.genKeyPair();
        PrivateKey privateKey = keypair.getPrivate();
        PublicKey publicKey = keypair.getPublic();
        System.out.println(privateKey + "n" + publicKey);
        // Generate a 576-bit DH key pair
        keyGen = KeyPairGenerator.getInstance("DH");
        keyGen.initialize(576);
        keypair = keyGen.genKeyPair();
        privateKey = keypair.getPrivate();
        publicKey = keypair.getPublic();
        System.out.println(privateKey + "n" + publicKey);
        // Generate a 1024-bit RSA key pair
        keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(1024);
        keypair = keyGen.genKeyPair();
        privateKey = keypair.getPrivate();
        publicKey = keypair.getPublic();
        System.out.println(privateKey + "n" + publicKey);
```

```
   }
catch (java.security.NoSuchAlgorithmException e) {
   } }}
```

OUTPUT

```
Administrator: Command Prompt

C:\Program Files\Java\jdk1.7.0_79\bin>javac GenerateKey.java

C:\Program Files\Java\jdk1.7.0_79\bin>java GenerateKey
sun.security.provider.DSAPrivateKey@fffe5aeanSun DSA Public Key
    Parameters:
    p:
    fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669
    455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7
    6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb
    83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7
    q:
    9760508f 15230bcc b292b982 a2eb840b f0581cf5
    g:
    f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267
    5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1
    3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b
    cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a

  y:
    992182a9 94b293a6 d01ca1db 58afc9a3 80e2c143 b6d71c1b c2613c14 8ed59c9f
    8b9dc5bf 944b552a 04be8d13 ca5bdf2f b244d426 88c6a8cf f7aa2c8c b0c6e7a2
    26716ee1 c1ea0a79 3c079ed0 5bed4bb8 aedc1cd5 ac4de2fe d00a51fb d019fd2a
    cfc19967 26998f3f fbad3372 eefcedd7 8279f322 8cc91e26 3719a1dc ca9dc582

com.sun.crypto.provider.DHPrivateKey@ffff9a77nSunJCE Diffie-Hellman Public Key:
y:
    3ddb0052 64c1135b 791cad71 cad17650 2897e48c 257babdb 0d51f6ca 963e7ff0
    183c31ed 61c2f83c 588d3e90 072ee978 7ca245ac bcec9f37 b61c97c3 0b51aa09
    106e2df7 d2a93472
p:
    88efe82a 78921486 c77f944f 98fb5007 f22b3dfd 47976624 d6b48a51 172163da
    96f09f2a 0c0fb040 49278a40 dacc5c3b 47b1410b a1deb8fc fa34b7ac 17d35b39
    51dcb589 3589e261
g:
    38ba577a 7383f234 add60407 fbc93e25 8e9a233d 5b82e79a abe48dcb 4c64d7ca
    6c33e347 1dc0a620 8e221367 7ee354f2 c1ff7f25 646d93c5 e38b8e25 77127218
    d6473f39 ea8e248c
l:
    384
sun.security.rsa.RSAPrivateCrtKeyImpl@fffbed3bnSun RSA public key, 1024 bits
  modulus: 937027338145420051011369636076155215398325522642171611984231070977272
77829882735851350554021045972877614371328091977126201574278367432180310235610238
62325163544581310830567964144877675809723318145984959514127532943158006052072529
9651755574833221535248595243160831642805819718976190596779868391477835706448787
  public exponent: 65537

C:\Program Files\Java\jdk1.7.0_79\bin>_
```

RESULT:

Thus the program for implementing Digital Signature Scheme (DSS) using Digital Signature Standard (DSA) algorithm is done using java and output is verified successfully.

Ex. No: 9	**INSTRUSION DETECTION SYSTEM (IDS)**
	SNORT

AIM

To demonstrate Intrusion Detection System (IDs) using Snort Tool.

Snort Description:

Snort is an open source network intrusion detection system (NIDS) has the ability to perform real-time traffic analysis and packet logging on internet protocol (IP) networks. Snort performs protocol analysis, content searching and matching.

Snort can be configured in three main modes: sniffer, packet logger, and network intrusion detection.

SNIFFER MODE

In sniffer mode, the program will read network packets and display them on the console.

snort –v	Show the TCP/IP packets header on the screen
snort –vd	Show the TCP/IP ICMP header with application data in transmit

PACKET LOGGER MODE

In packet logger mode, the program will log packets to the disk.

snort –dev –l c:\snort\log	[create this directory in the C drive] and snort will automatically know to go into packet logger mode, it collects every packet it sees and places it in log directory
snort –dev –l c:\snort\log –h ipaddress/24	This rule informs snort that you want to print out the data link and TCP/IP headers as well as application data into the log directory
snort –l c:\snort\log –b	This is binary mode logs everything into a single file

NETWORK INTRUSION DETECTION SYSTEM MODE

In intrusion detection mode, the program will monitor network traffic and analyze it against a rule set defined by the user. The program will then perform a specific action based on what has been identified.

snort –d c:\snort\log –h ipaddress/24 –c snort.conf	This is a configuration file applies rule to each packet to decide it an action based upon the rule type in the file
snort –d –h ipaddress/24 –l c:\snort\log –c	This will configure snort to run in its most

snort.conf	basic NIDS (Network Intrusion Detection System) form, logging packets that trigger rules specified in the snort.conf

PROCEDURE

1. Download SNORT from snort.org

2. Get Started [Download the latest version] from the windows tab at this point the lastest version is **snort_2_9_8_2_installer.exe**

3. Rules [Download the latest version] based on version downloaded **snortrules-snapshot-2982.tat.gz**

4. Download WinPcap from WinPcap.org [version 4.1.3]

5. Place the installer, rules, winPcap in snort folder

6. Download zenmap from nmap.org **nmap-7.12-setup.exe**

a. How to Configure snort?

Move to c drive →snort→etc folder→snort.conf [open with wordpad]

Step 1: Change the path

setup the network addresses you are protecting

ipvar HOME_NET any

change this to your current system ip address using

ipvar HOME_NET 192.168.1.2/24 [you could get the ip of the system using ipconfig command]

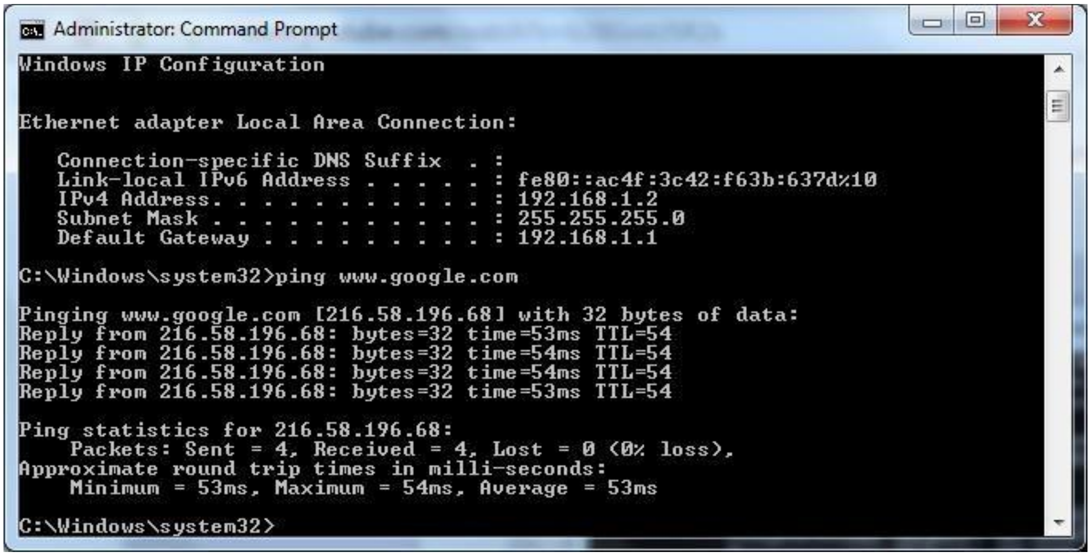


Fig. IPConfig Command

#setup the external network address

ipvar EXTERNAL_NET $HOME_NET

Rule Path

var RULE_PATH …/rules

var PREPROC_RULE_PATH …/preproc_rules

Change the path as

var RULE_PATH C:\Snort\rules

var SO_RULE_PATH ../so_rules

var PREPROC_RULE_PATH C:\Snort\preproc_rules

If you are using reputation preprocessor set these

var WHITE_LIST_PATH C:\Snort\rules

var BLACK_LIST_PATH C:\Snort\rules

Step 2: Configure the decoder

In the last line config logdir: add config logdir: C:\Snort\log [remove # symbol]

Step 3: Set path for Dynamic Preprocessor and Dynamic Engine

path to dynamic preprocessor libraries

dynamicpreprocessor directory C:\Snort\lib\snort_dynamicpreprocessor

path to base preprocessor engine

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

path to dynamic rules libraries [Assign # symbol to dynamic detection directory]

dynamicdetection directory /usr/local/lib/snort_dynamicrules

Step 4: Configure Preprocessors

Assign # symbol in following lines

```
# Inline packet normalization. For more information, see README.normalize
# Does nothing in IDS mode
# preprocessor normalize_ip4
# preprocessor normalize_tcp: block, rsv, pad, urp, req_urg,
# req_pay, req_urp, ips, ecn stream
# preprocessor normalize_icmp4
# preprocessor normalize_ip6
# preprocessor normalize_icmp6
```

In the last line add

```
# Reputation preprocessor. For more information see README.reputation
preprocessor reputation: \
  memcap 500, \
  priority whitelist, \
  nested_ip inner, \
  whitelist $WHITE_LIST_PATH/white.list, \
  blacklist $BLACK_LIST_PATH/black.list
```

Step 5: Customizing Rule Sheet

Change backward slash to forward slash in the following codes

```
# site specific rules
include $RULE_PATH\local.rules
include $RULE_PATH\app-detect.rules
include $RULE_PATH\attack-responses.rules
.
.
.
```

include $RULE_PATH\x11.rules

Step 6: Customizing Preprocessor

Change backward slash to forward slash in the following codes [also remove # symbol in the below three lines]

decoder and preprocessor event rules

include $PREPROC_RULE_PATH\preprocessor.rules

include $PREPROC_RULE_PATH\decoder.rules

include $PREPROC_RULE_PATH\sensitive-data.rules

Check whether you have the line threshold.conf in the last line of snort.conf

b. How to use Snort to detect Ping?

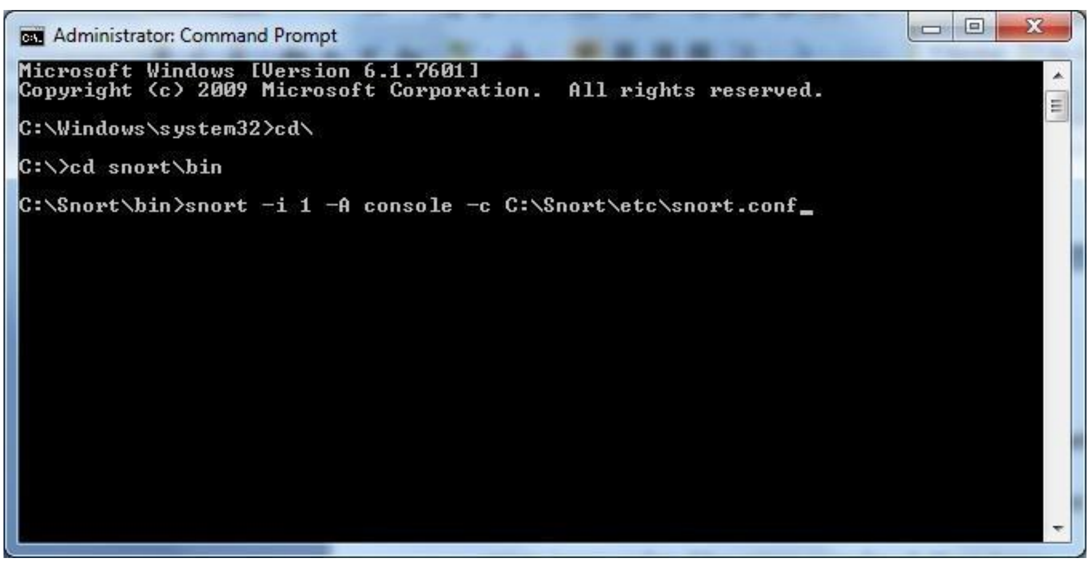
Step 1: In snort folder in c drive go to rules folder

open icmp.rules file using wordpad then type

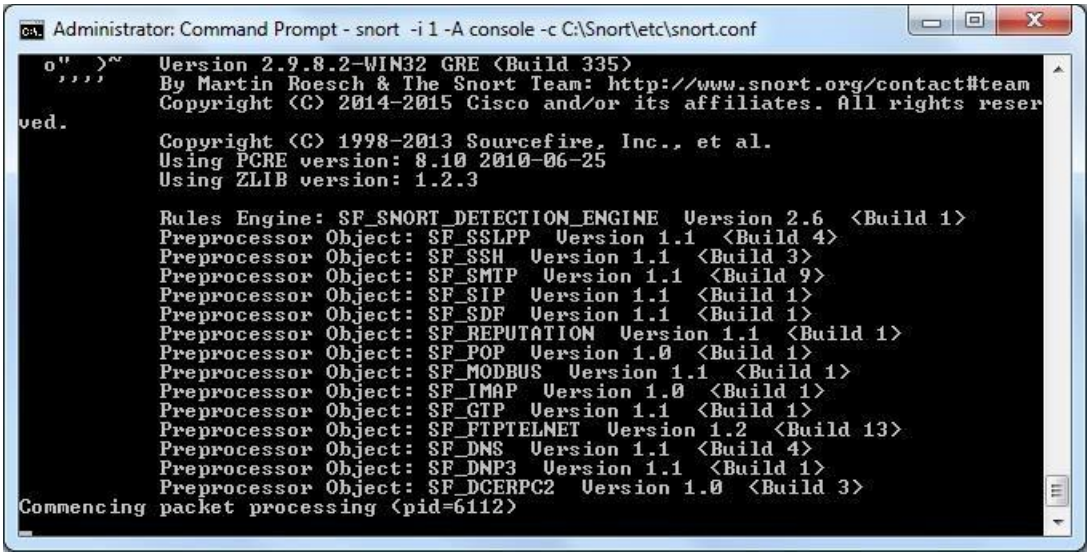
alert icmp any any -> any any (msg:"PING PING PING"; sid:1000000001;)

Step 2 : Open Command Prompt Run as Administrator

In command prompt type the following command



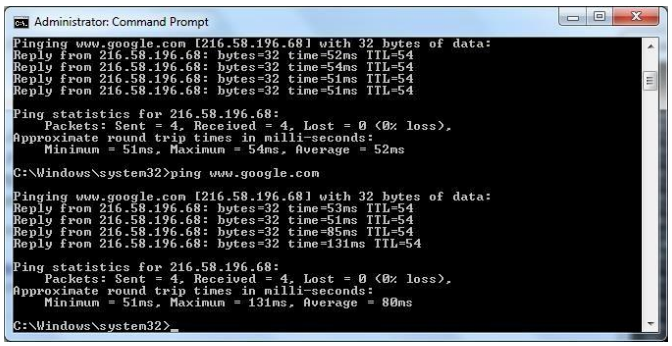
wait until you get the line **commencing packet processing**



Step 3 : Open another Command Prompt Run as Administrator

Type ping www.google.com

Now you will get the reply from google to your system ip



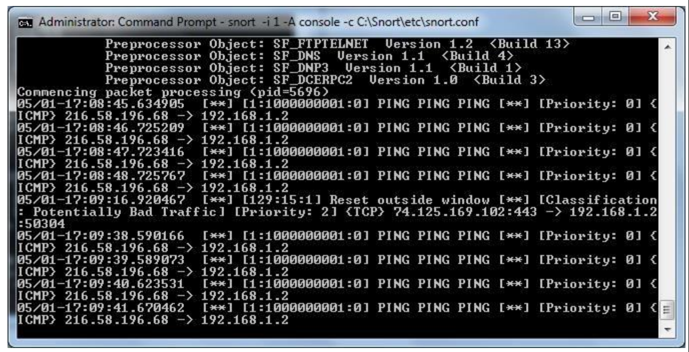


Fig. Snort Detecting Ping

c. *How to use Snort to log portscan?*

Step 1: Move to log folder in snort create a blank text file name it as say sample.txt

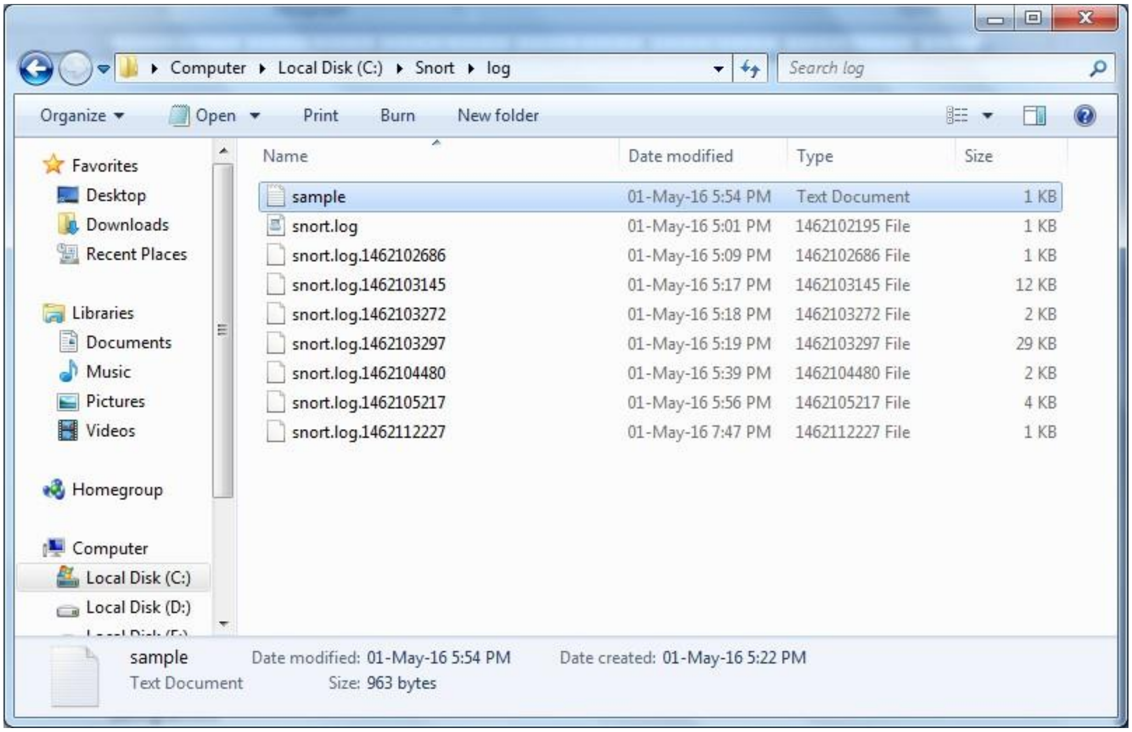


Fig. Sample.txt in log folder

Step 2: Open snort.conf using [ctrl+F] find stream5_global

check whether it as the same line

Target-Based stateful inspection/stream reassembly. For more inforation, see README.stream5

preprocessor stream5_global: track_tcp yes, \

  track_udp yes, \

  track_icmp no, \

  max_tcp 262144, \

  max_udp 131072, \

  max_active_responses 4, \

  min_response_seconds 2

Next find for portscan

Portscan detection. For more information, see README.sfportscan

preprocessor stream5_global: track_udp yes track_tcp yes

preprocessor sfportscan: proto { all } scan_type { all } sense_level { medium } logfile { **sample.txt**

}

//the file name should be same as created in log folder

Step 3: Open zenmap type the google ip obained from ping command in target

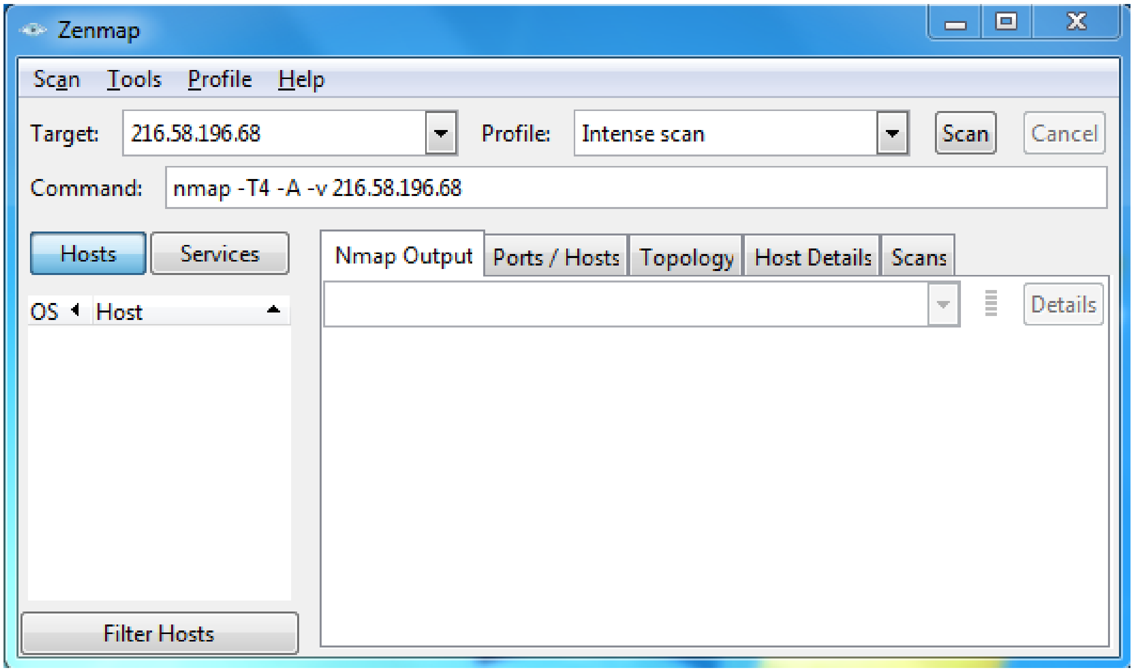


Fig. Zenmap Home Page

Step 4: Before commecning scan open the command prompt [run as administrator] type the following command

snort –i l –A console –c C:\Snort\etc\snort.conf

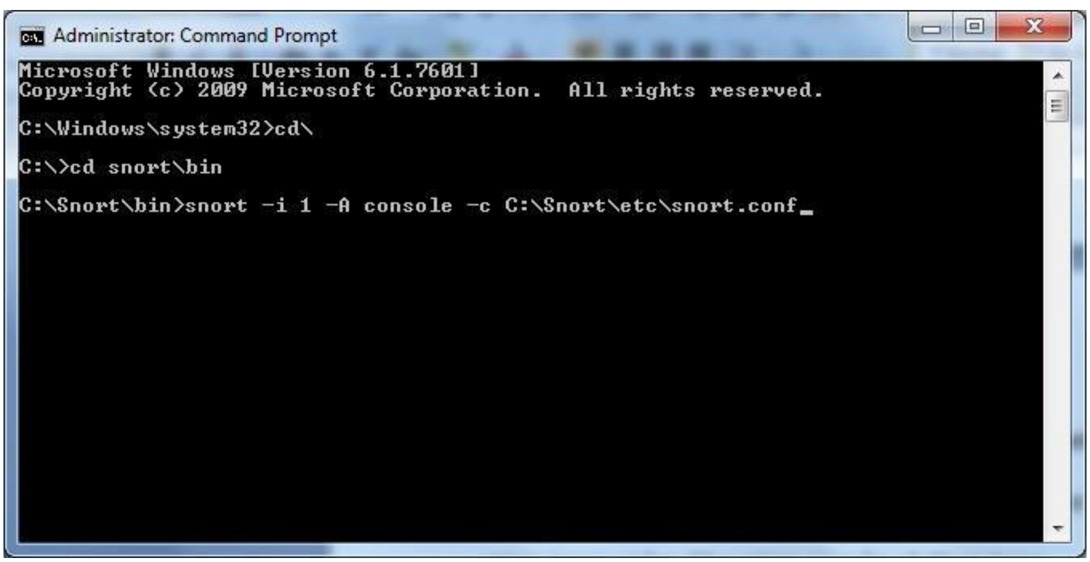


Fig. Commencing Snort

Wait for the packet processing window as shown

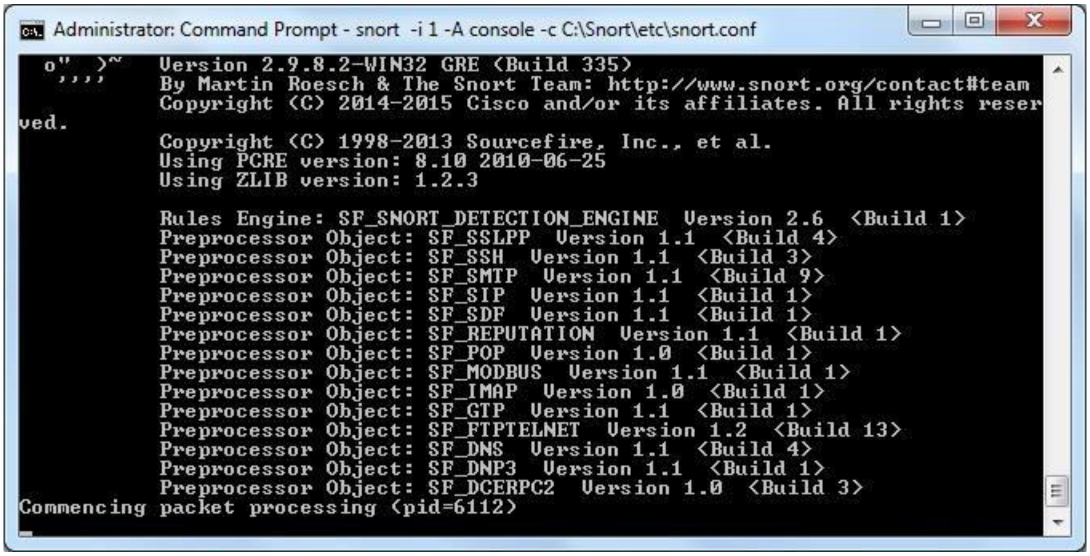


Fig. Packet Processing Window

Step 5: Click the scan check whether the profile is intense scan in Zenmap GUI

The list of scan runs in the Zenmap GUI as shown.

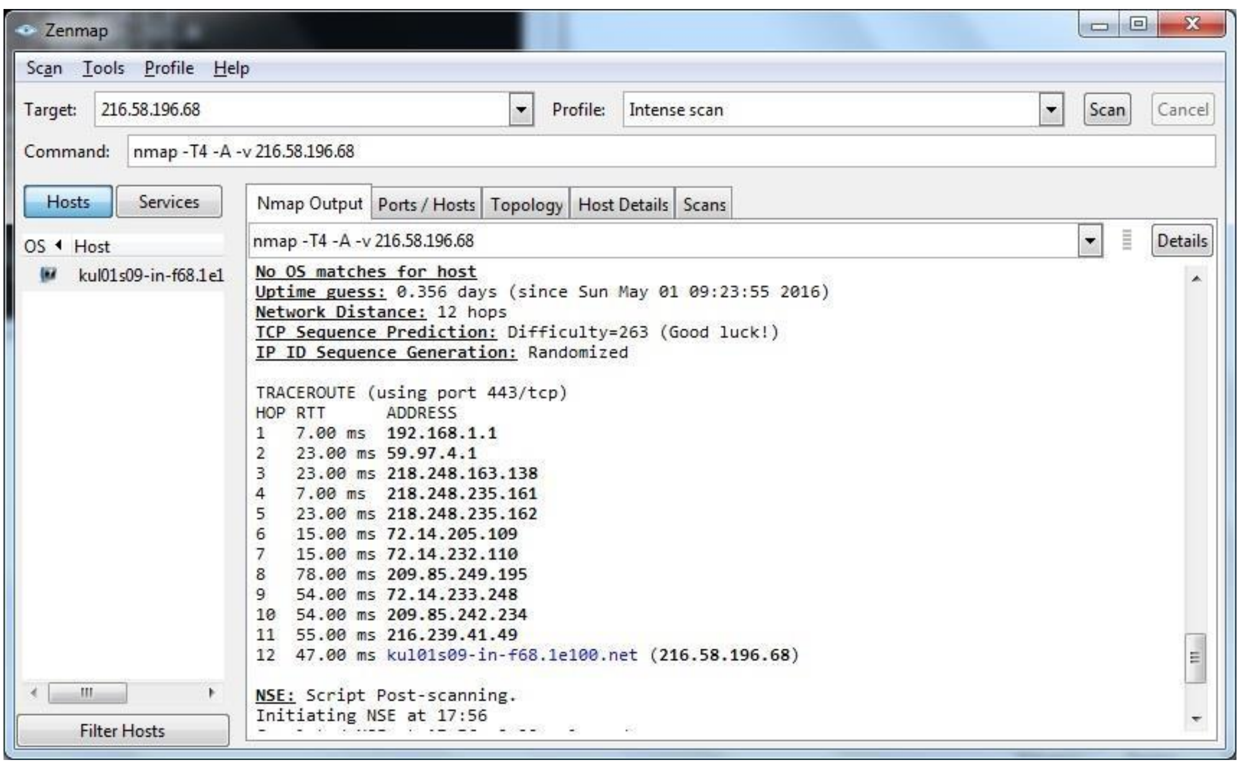


Fig. Zenmap GUI in SCAN

Step 6: The snort runs as shown

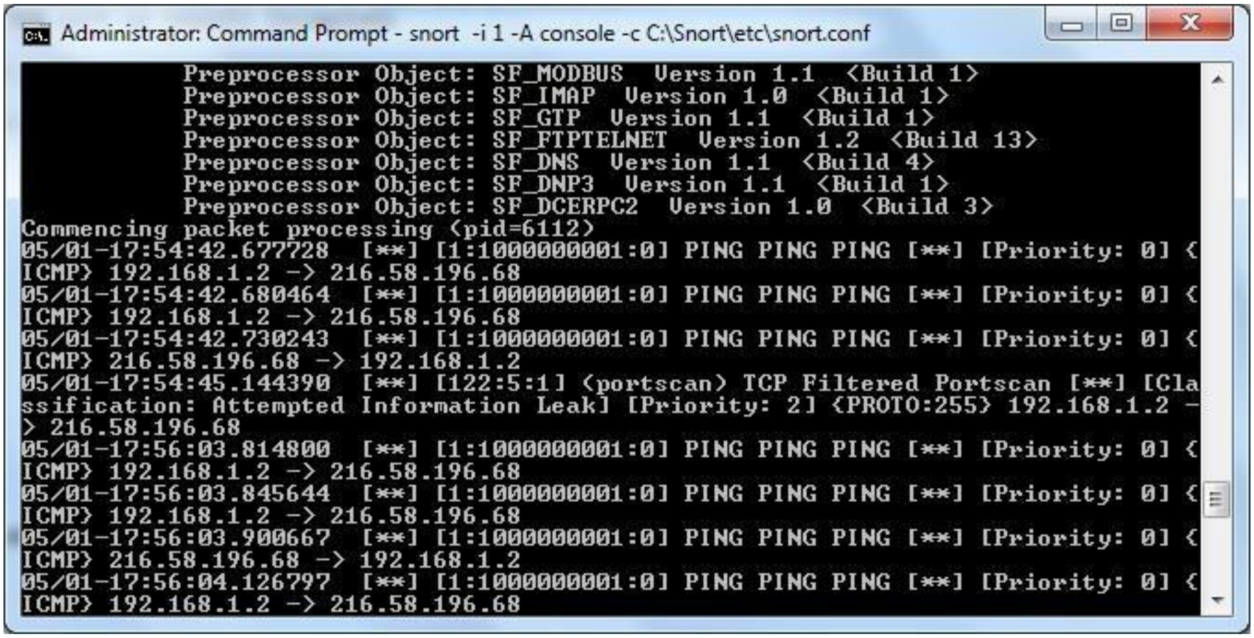


Fig. Snort runs

To stop snort type ctrl+c in snort window it shows the result as snort exiting as shown in fig

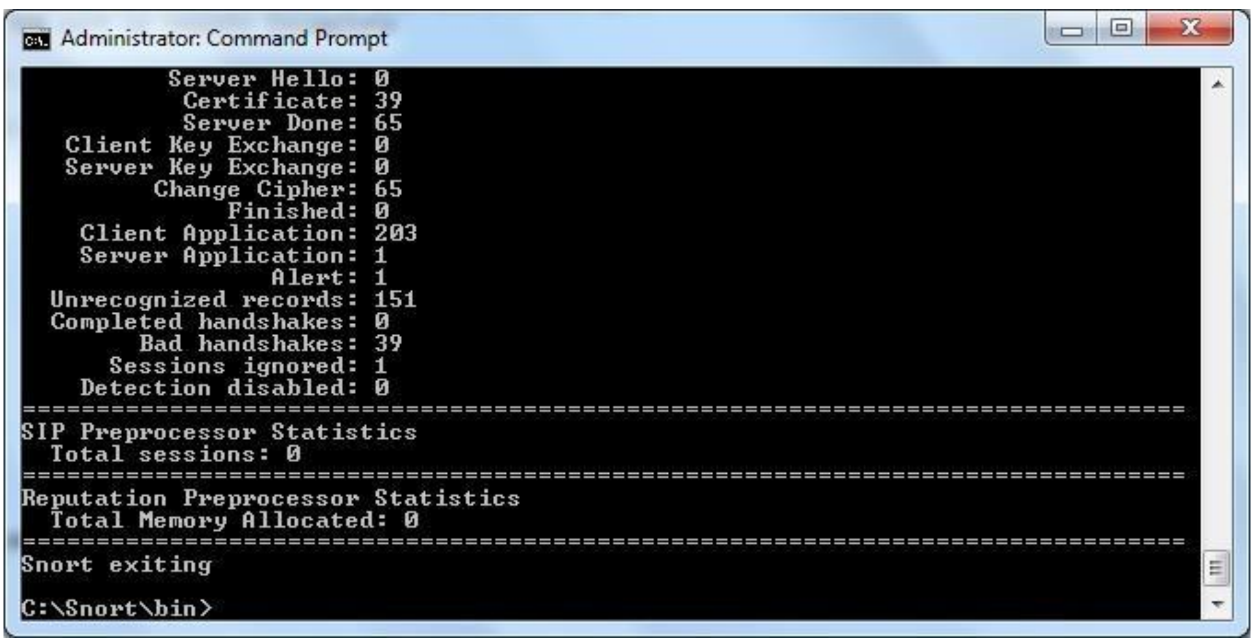


Fig. Snort Exiting

Step 7: To check the log go to log folder in snort and check sample.txt you could see the time and packets transferred

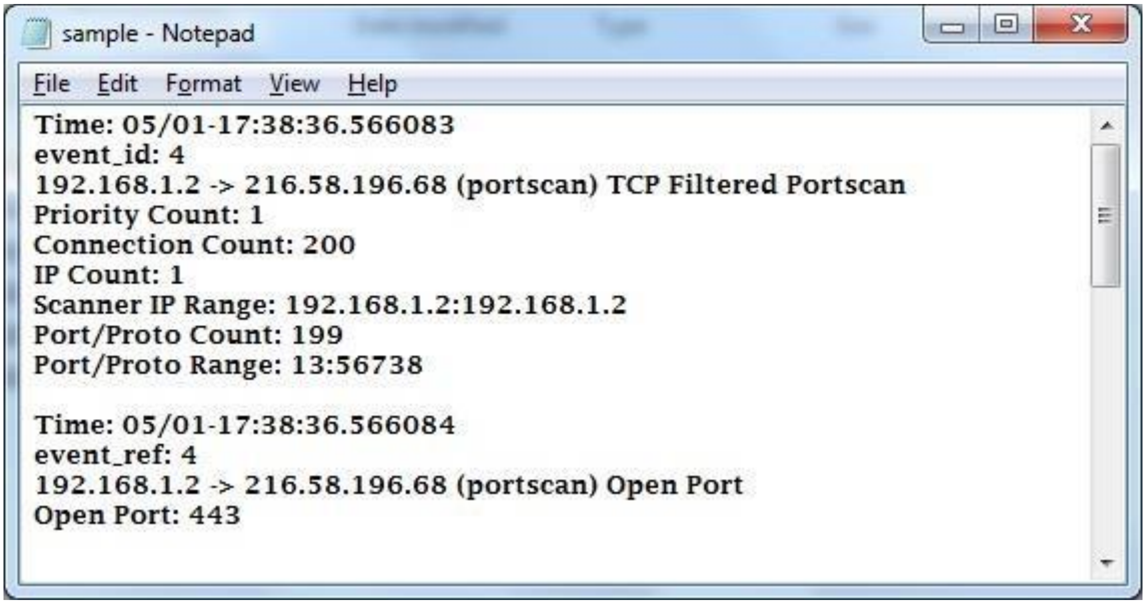


Fig. Packets Transferred

RESULT

Thus the demonstration of Instrusion Detection System (IDs) is done using Snort Tool and output is verified successfully.

Ex. No: 10	AUTOMATED ATTACK AND PENETRATION TOOLS EXPLORING N-STALKER

AIM

To explore N-Stalker, a vulnerability Assessment Tool

DESCRIPTION

N-Stalker is a web server security-auditing tool that scans for more than 30,000 vulnerabilities. N-Stalker was created in April 2000 by Information Security Technology specialists, aiming at providing solutions to protect corporations and individuals against digital threats that affect information systems. The first product to be released was N-Stealth HTTP Security Scanner Suite, a complete set of tools to assess Web servers' security, including the capabilities of identifying vulnerabilities and providing a possible solution to mitigate the risks from critical mission business infrastructure, either on the Internet or in a corporate environment.

By permanently making use of attack signature updates, the software has aggregated the most extensive and updated database available on the market, with more than 39,000 vulnerabilities and exploits for Web environments, recursively utilized by the scanning tool.

Steps:

You need to download and install N-Stalker from www.nstalker.com

1. Start N-Stalker from a windows computer by choosing Start→Programs→N-Stalker→N-Stalker Free Edition.
2. In the web application URL field enter a host address or a range of addresses to scan.
3. Click start scan
4. After the scan is complete, the N-Stalker Report Manager prompts you to select a format for the resulting report. Choose Generate HTML.
5. Review the HTML report for vulnerabilities.

Armed with this report, your next step should be to set priorities on which services should be patched and hardened.

Fig. N-Stalker to scan for Vulnerabilities

RESULT

Thus the demonstration of Vulnerability Assessment Tool is explored using N-Stalker Assessment Tool and performance is analyzed.

Ex. No: 11a	**DEFEATING MALWARE**
	BUILDING TROJANS

AIM

To build Simple Trojans Programs and check their performance.

DESCRIPTION

Trojans are programs that seem to do something you want but actually perform another, malicious, act. Before a Trojan program can act, it must trick the user into downloading it or performing some type of action. The Trojan may be configured to do many things, such as log keystrokes, add the user's system to a botnet, or even give the attacker full access to the victim's computer. A user may think that a file looks harmless and is safe to run but, once executed, it delivers its malicious payload. Unlike a virus or worm, Trojans cannot spread themselves. They rely on the uninformed user

TASK:

Trojans and malware pose a real danger. This challenge highlights one of the ways that a hacker may distribute a Trojan. By default, older Windows systems automatically start a CD when it is inserted in the CD tray. You use this technique to distribute simulated malicious code. You need a blank CD and a CD burner for this exercise.

1. Create a text file named autorun.ini. Inside this text file, add the following contents:
   [autorun]
   Open paint.exe
   Icon=paint.exe

2. Place the autorun.ini file and a copy of paint.exe into a folder to be burned to a CD.

3. After you have burned the CD, reinsert it in the CD-ROM drive and observe the results. The CD should autostart and automatically starts the Paint program.

4. Think about the results. Although this exercise was benign, you could have easily used a Trojan program wrapped with a legitimate piece of software. Just leaving the CD lying around or giving it an attractive title, such as "pending 2006 bonuses," may lead someone to pick it up and view its contents. Anyone running the CD would then become infected. Even with AutoRun turned off, the user would only have to double-click the CD-ROM icon and the program would still run.

RESULT

Thus the Simple Trojan Programs is done and their performance is checked.

Ex. No: 11b	DEFEATING MALWARE
	ROOT KIT HUNTER

AIM

To install root kits and study about variety of options.

RootKit Description:

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

- Download Rootkit Tool from GMER website. www.gmer.net
- This displays the Processes, Modules, Services, Files, Registry, RootKit/Malwares, Autostart, CMD of local host.
- Select Processes menu and kill any unwanted process if any. Modules menu displays the various system files like .sys, .dll
- Services menu displays the complete services running with Autostart, Enable, Disable, System, and Boot.
- Files menu displays full files on Hard-Disk volumes.
- Registry displays Hkey_Current_user and Hkey_Local_Machine. Rootkits/Malawares scans the local drives selected.
- Autostart displays the registry base Autostart applications.

CMD allows the user to interact with command line utilities or Registry

OUTPUT

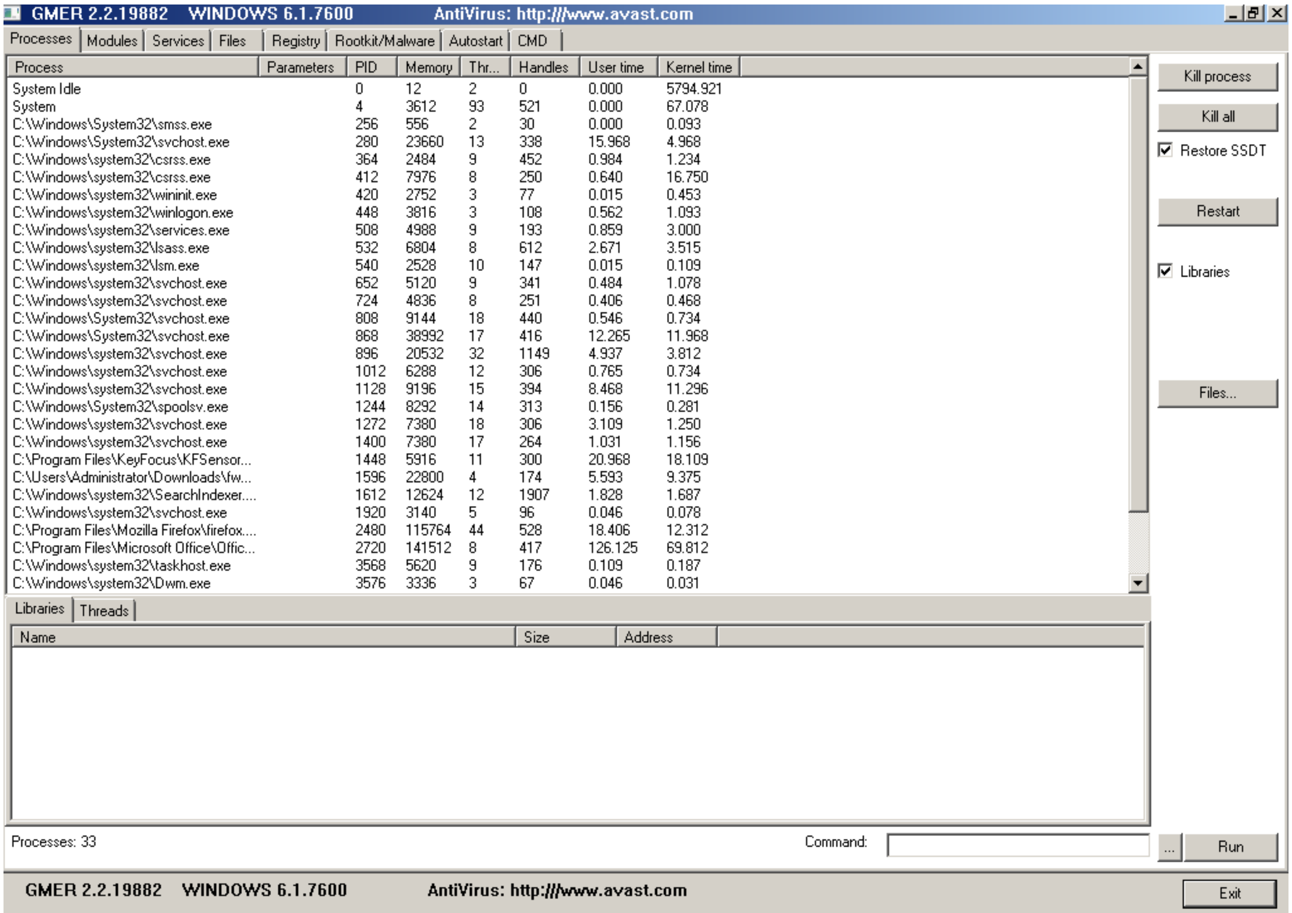
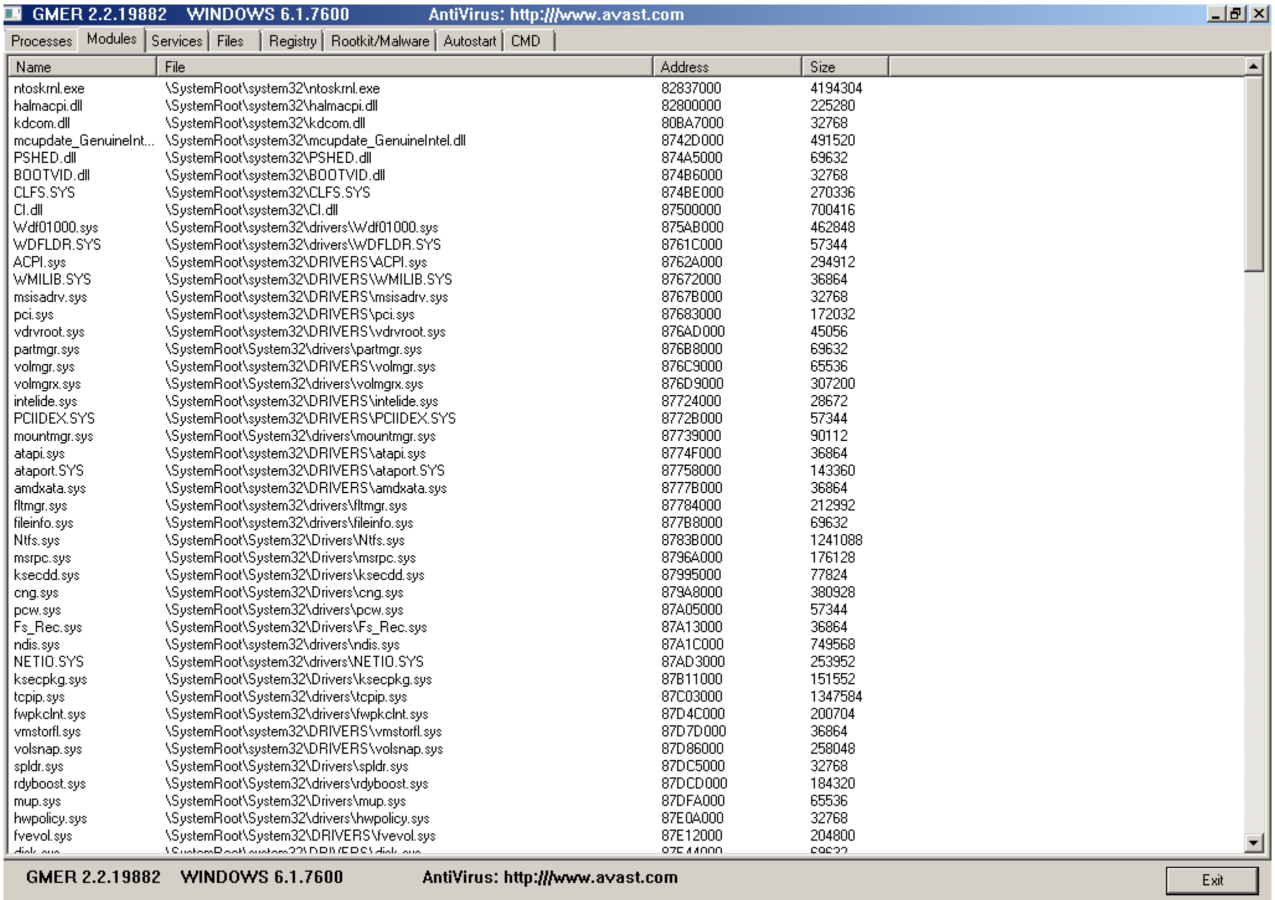


Fig. Processes Tab



GMER 2.2.19882 WINDOWS 6.1.7600 AntiVirus: http:///www.avast.com

Processes | Modules | Services | Files | Registry | Rootkit/Malware | Autostart | CMD

Name	File	Address	Size
ntoskrnl.exe	\SystemRoot\system32\ntoskrnl.exe	82837000	4194304
halmacpi.dll	\SystemRoot\system32\halmacpi.dll	82800000	225280
kdcom.dll	\SystemRoot\system32\kdcom.dll	80BA7000	32768
mcupdate_GenuineInt...	\SystemRoot\system32\mcupdate_GenuineIntel.dll	8742D000	491520
PSHED.dll	\SystemRoot\system32\PSHED.dll	874A5000	69632
BOOTVID.dll	\SystemRoot\system32\BOOTVID.dll	874B6000	32768
CLFS.SYS	\SystemRoot\system32\CLFS.SYS	874BE000	270336
CI.dll	\SystemRoot\system32\CI.dll	87500000	700416
Wdf01000.sys	\SystemRoot\system32\drivers\Wdf01000.sys	875AB000	462848
WDFLDR.SYS	\SystemRoot\system32\drivers\WDFLDR.SYS	8761C000	57344
ACPI.sys	\SystemRoot\system32\DRIVERS\ACPI.sys	8762A000	294912
WMILIB.SYS	\SystemRoot\system32\DRIVERS\WMILIB.SYS	87672000	36864
msisadrv.sys	\SystemRoot\system32\DRIVERS\msisadrv.sys	8767B000	32768
pci.sys	\SystemRoot\system32\DRIVERS\pci.sys	87683000	172032
vdrvroot.sys	\SystemRoot\system32\DRIVERS\vdrvroot.sys	876AD000	45056
partmgr.sys	\SystemRoot\System32\drivers\partmgr.sys	876B8000	69632
volmgr.sys	\SystemRoot\system32\DRIVERS\volmgr.sys	876C9000	65536
volmgrx.sys	\SystemRoot\System32\drivers\volmgrx.sys	876D9000	307200
intelide.sys	\SystemRoot\system32\DRIVERS\intelide.sys	87724000	28672
PCIIDEX.SYS	\SystemRoot\system32\DRIVERS\PCIIDEX.SYS	8772B000	57344
mountmgr.sys	\SystemRoot\System32\drivers\mountmgr.sys	87739000	90112
atapi.sys	\SystemRoot\system32\DRIVERS\atapi.sys	8774F000	36864
ataport.SYS	\SystemRoot\system32\DRIVERS\ataport.SYS	87758000	143360
amdxata.sys	\SystemRoot\system32\DRIVERS\amdxata.sys	8777B000	36864
fltmgr.sys	\SystemRoot\system32\drivers\fltmgr.sys	87784000	212992
fileinfo.sys	\SystemRoot\system32\drivers\fileinfo.sys	877B8000	69632
Ntfs.sys	\SystemRoot\System32\Drivers\Ntfs.sys	8783B000	1241088
msrpc.sys	\SystemRoot\System32\Drivers\msrpc.sys	8796A000	176128
ksecdd.sys	\SystemRoot\System32\Drivers\ksecdd.sys	87995000	77824
cng.sys	\SystemRoot\System32\Drivers\cng.sys	879A8000	380928
pcw.sys	\SystemRoot\System32\drivers\pcw.sys	87A05000	57344
Fs_Rec.sys	\SystemRoot\System32\Drivers\Fs_Rec.sys	87A13000	36864
ndis.sys	\SystemRoot\system32\drivers\ndis.sys	87A1C000	749568
NETIO.SYS	\SystemRoot\system32\drivers\NETIO.SYS	87AD3000	253952
ksecpkg.sys	\SystemRoot\System32\Drivers\ksecpkg.sys	87B11000	151552
tcpip.sys	\SystemRoot\System32\drivers\tcpip.sys	87C03000	1347584
fwpkclnt.sys	\SystemRoot\System32\drivers\fwpkclnt.sys	87D4C000	200704
vmstorfl.sys	\SystemRoot\system32\DRIVERS\vmstorfl.sys	87D7D000	36864
volsnap.sys	\SystemRoot\system32\DRIVERS\volsnap.sys	87D86000	258048
spldr.sys	\SystemRoot\System32\Drivers\spldr.sys	87DC5000	32768
rdyboost.sys	\SystemRoot\System32\drivers\rdyboost.sys	87DCD000	184320
mup.sys	\SystemRoot\System32\Drivers\mup.sys	87DFA000	65536
hwpolicy.sys	\SystemRoot\System32\drivers\hwpolicy.sys	87E0A000	32768
fvevol.sys	\SystemRoot\System32\DRIVERS\fvevol.sys	87E12000	204800

GMER 2.2.19882 WINDOWS 6.1.7600 AntiVirus: http:///www.avast.com

Exit

Fig. Modules Tab

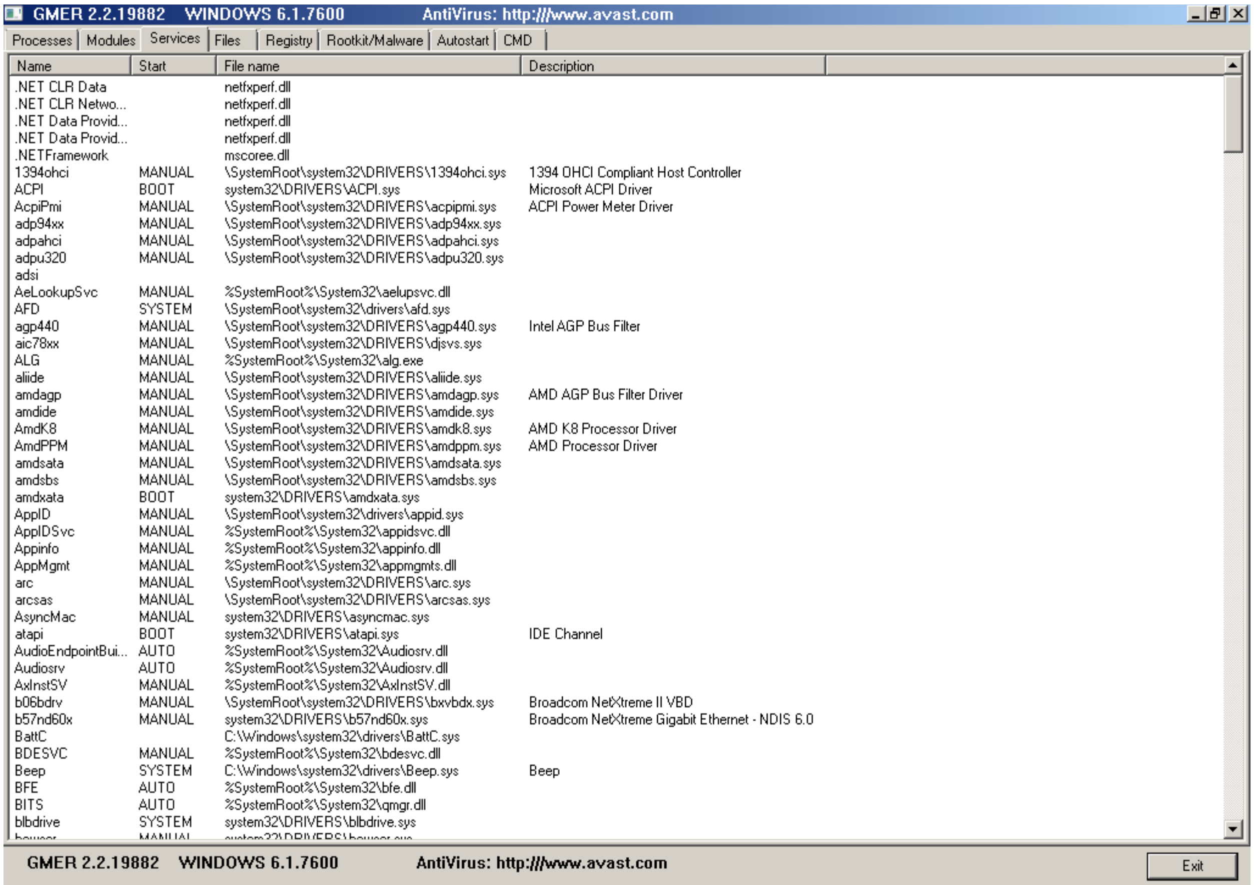


Fig. Services Tab

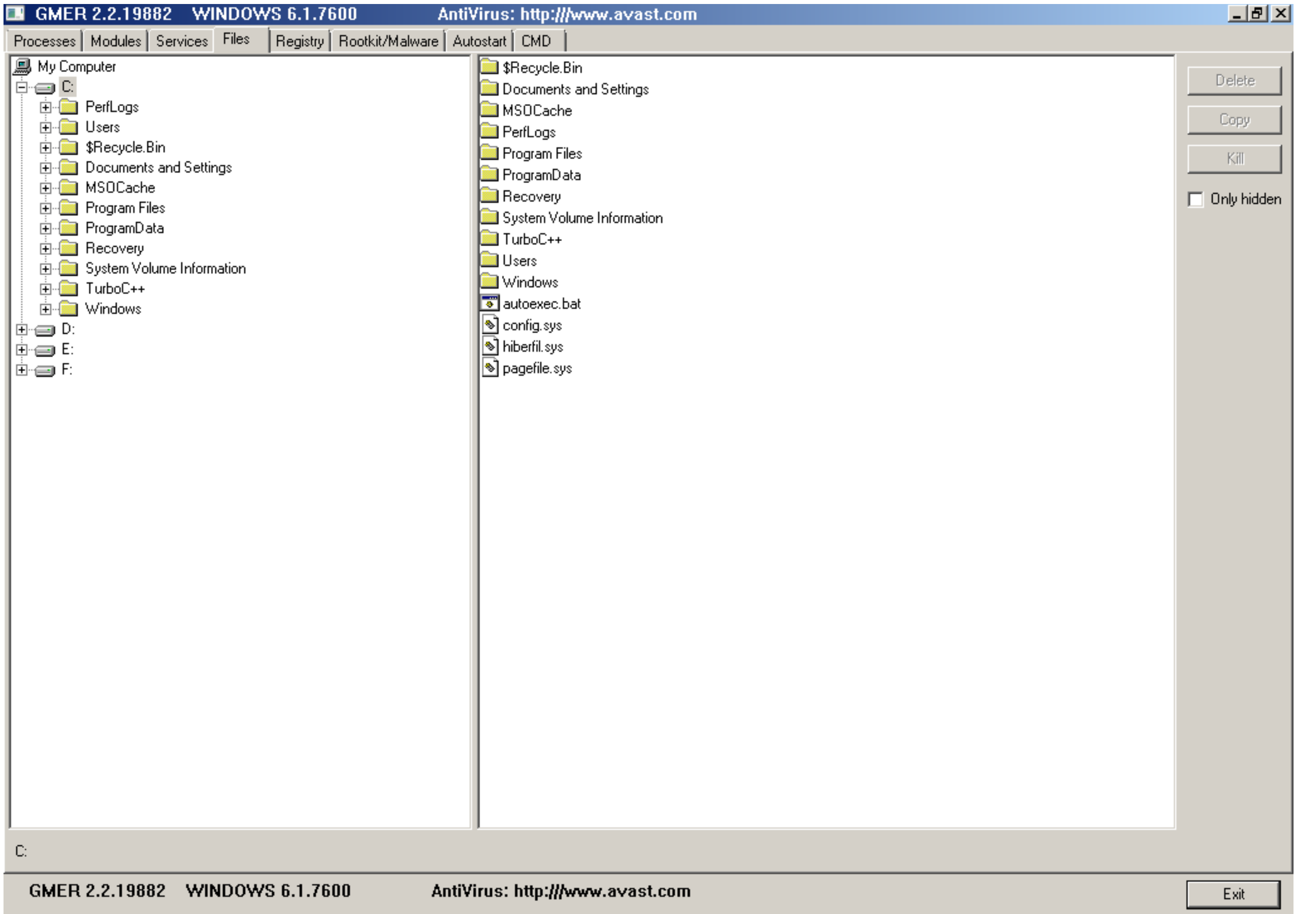


Fig. Files Tab

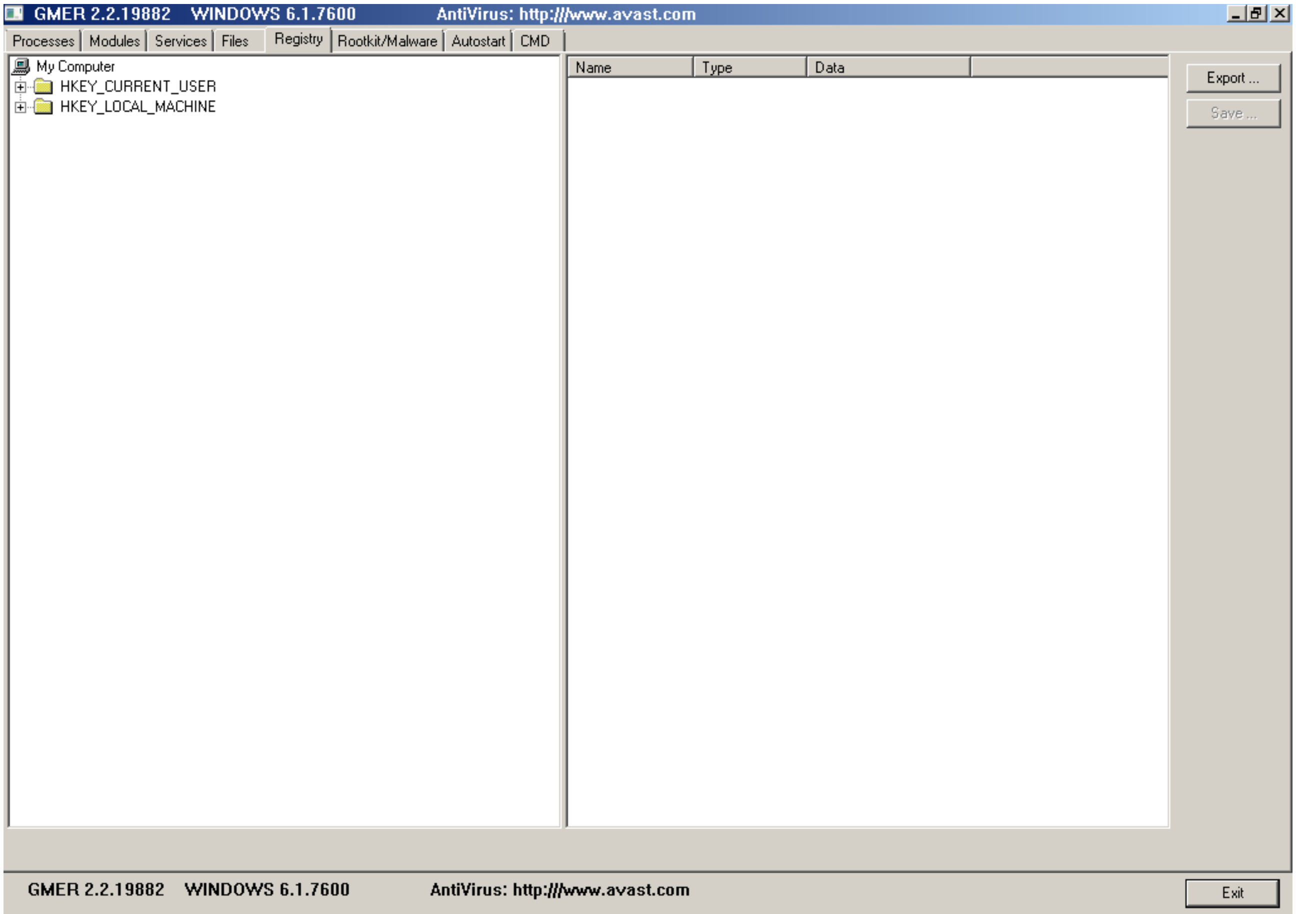


Fig. Registry Tab

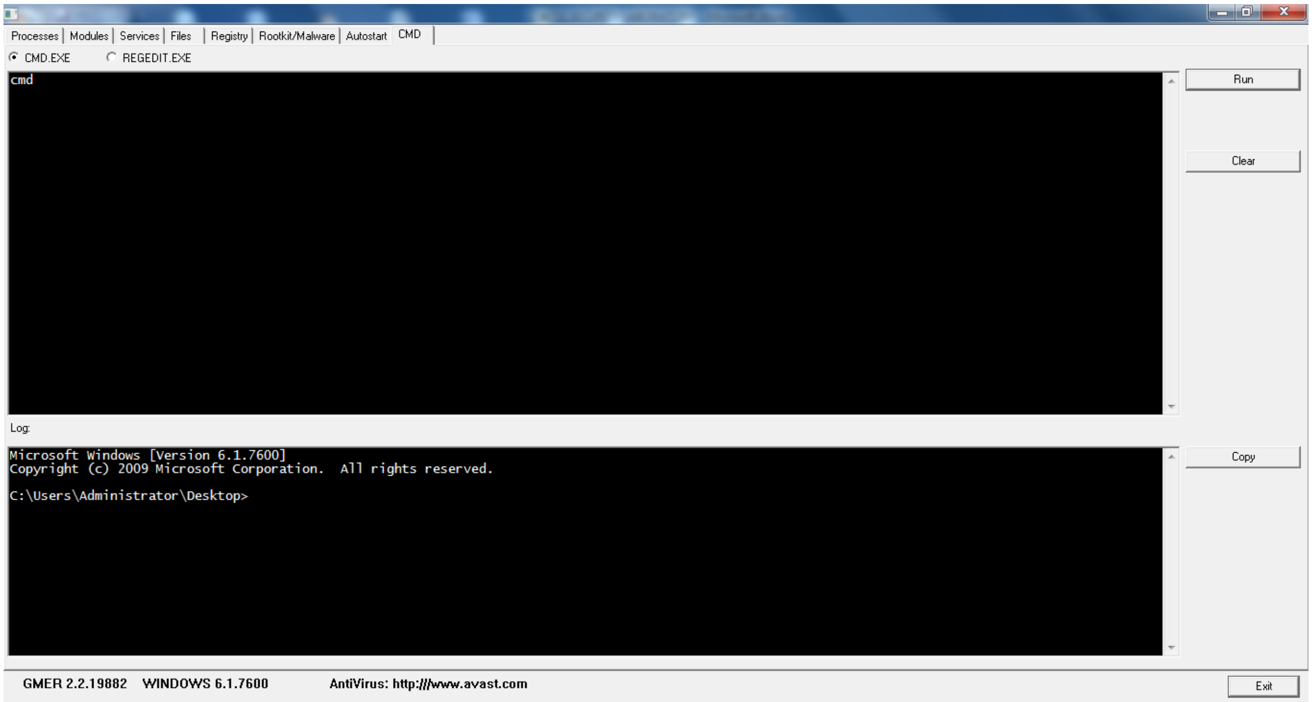


Fig. CMD Tab

RESULT

Thus the installation of Root Kits and study about variety of options is done.

Ex. No: 12	ADDITIONAL EXPERIMENTS
	IMPLEMENT MESSAGE DIGEST (MD5) ALGORITHM

AIM

Develop a program to implement Message Digest Algorithm.

ALGORITHM DESCRIPTION

The MD5 **message-digest algorithm** is a widely used **cryptographic hash function** producing a 128-bit (16-byte) **hash value**, typically expressed in text format as a 32-digit hexadecimal number. MD5 has been utilized in a wide variety of cryptographic applications and is also commonly used to verify data integrity.

MD5 Algorithm:

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of eight, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1}

The following five steps are performed to compute the message digest of the message.

Step 1. Append Padding Bits:

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

Step 2. Append Length:

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

Step 3. Initialize MD Buffer:

A four-word buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32-bit register. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Step 4. Process Message in 16-Word Blocks:

We first define four auxiliary functions that each take as input three 32-bit wordsand produce as output one 32-bit word.

F(X,Y,Z) = XY v not(X) Z

G(X,Y,Z) = XZ v Y not(Z)

H(X,Y,Z) = X xor Y xor Z

I(X,Y,Z) = Y xor (X v not(Z)) In each bit position F acts as a conditional: if X then Y else Z. The function F could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) It is interesting to note that if the bits of X, Y, and Z are independent and unbiased, the each bit of F(X,Y,Z) will be independent and unbiased.

The functions G, H, and I are similar to the function F, in that they act in "bitwise parallel" to produce their output from the bits of X, Y, and Z, in such a manner that if the corresponding bits of X, Y, and Z are independent and unbiased, then each bit of G(X,Y,Z), H(X,Y,Z), and I(X,Y,Z) will be independent and unbiased. Note that the function H is the bit-wise "xor" or "parity" function of its inputs.

This step uses a 64-element table T[1 ... 64] constructed from the sine function. Let T[i] denote the i-th element of the table, which is equal to the integer part of 4294967296 times abs(sin(i)), where i is in radians. The elements of the table are given in the appendix.

Example:

```
/* Process each 16-word block. */
For i = 0 to N/16-1 do
  /* Copy block i into X. */
  For j = 0 to 15 do
    Set X[j] to M[i*16+j].
  end /* of loop on j */
  /* Save A as AA, B as BB, C as CC, and D as DD. */
              AA = A
              BB = B
              CC = C
              DD = D
/* Round 1. */
  /* Let [abcd k s i] denote the operation
      a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0  7  1]  [DABC  1 12 2]  [CDAB  2 17  3]  [BCDA   3 22 4]
  [ABCD  4  7  5]  [DABC  5 12 6]  [CDAB  6 17  7]  [BCDA   7 22 8]
  [ABCD  8  7  9]  [DABC  9 12 10]  [CDAB 10 17 11]  [BCDA 11 22 12]
  [ABCD 12 7 13]  [DABC 13 12 14]  [CDAB 14 17 15]  [BCDA 15 22 16]
  /* Round 2. */
  /* Let [abcd k s i] denote the operation
      a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  1  5 17]  [DABC  6 9 18]  [CDAB 11 14 19]  [BCDA  0 20 20]
  [ABCD  5  5 21]  [DABC 10  9 22]  [CDAB 15 14 23]  [BCDA  4 20 24]
  [ABCD  9  5 25]  [DABC 14  9 26]  [CDAB  3 14 27]  [BCDA  8 20 28]
  [ABCD 13 5 29]  [DABC  2  9 30]  [CDAB  7 14 31]  [BCDA 12 20 32]
  /* Round 3. */
  /* Let [abcd k s t] denote the operation
```

```
    a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  5 4 33]  [DABC  8 11 34]  [CDAB 11 16 35]  [BCDA 14 23 36]
  [ABCD  1  4 37]  [DABC  4 11 38]  [CDAB  7 16 39]  [BCDA 10 23 40]
  [ABCD 13  4 41]  [DABC  0 11 42]  [CDAB  3 16 43]  [BCDA  6 23 44]
  [ABCD 9  4 45]  [DABC 12 11 46]  [CDAB 15 16 47]  [BCDA  2 23 48]
  /* Round 4. */
  /* Let [abcd k s t] denote the operation
    a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
  /* Do the following 16 operations. */
  [ABCD  0 6 49]  [DABC  7 10 50]  [CDAB 14 15 51]  [BCDA  5 21 52]
  [ABCD 12  6 53]  [DABC  3 10 54]  [CDAB 10 15 55]  [BCDA  1 21 56]
  [ABCD  8  6 57]  [DABC 15 10 58]  [CDAB  6 15 59]  [BCDA 13 21 60]
  [ABCD 4  6 61]  [DABC 11 10 62]  [CDAB  2 15 63]  [BCDA  9 21 64]
  /* Then perform the following additions. (That is increment each
     of the four registers by the value it had before this block
     was started.) */
  A = A + AA
  B = B + BB
  C = C + CC
  D = D + DD
 end /* of loop on i */
```

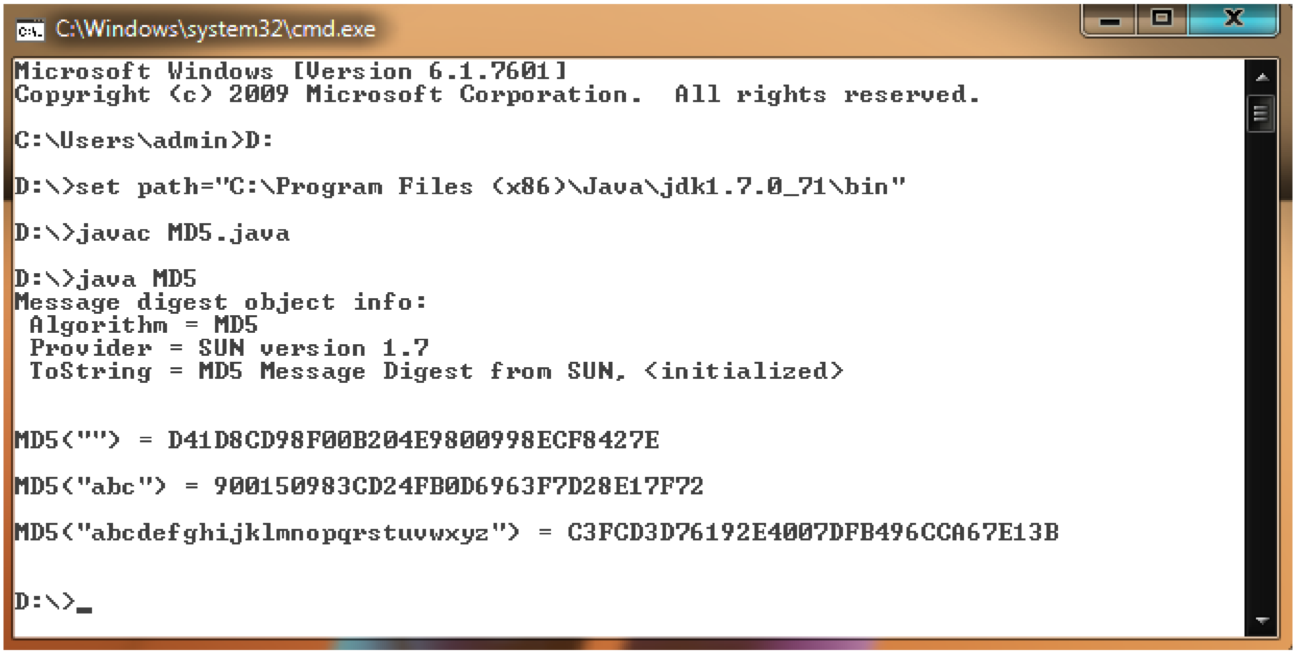
Step 5. Output: The message digest produced as output is A, B, C, D. That is, we begin with the low-order byte of A, and end with the high-order byte of D.

PROGRAM

```
import java.security.*;
public class MD5 {
public static void main(String[] a) {
try {
MessageDigest md = MessageDigest.getInstance("MD5");
System.out.println("Message digest object info: ");
System.out.println(" Algorithm = " +md.getAlgorithm());
System.out.println(" Provider = " +md.getProvider());
System.out.println(" ToString = " +md.toString());
String input = "";
md.update(input.getBytes());
byte[] output = md.digest();
System.out.println();
System.out.println("MD5(\""+input+"\") = " +bytesToHex(output));
input = "abc";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("MD5(\""+input+"\") = " +bytesToHex(output));
input = "abcdefghijklmnopqrstuvwxyz";
md.update(input.getBytes());
output = md.digest();
System.out.println();
System.out.println("MD5(\"" +input+"\") = " +bytesToHex(output));
System.out.println(""); }
catch (Exception e) {
System.out.println("Exception: " +e); } }
public static String bytesToHex(byte[] b) {
 char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
```

```
StringBuffer buf = new StringBuffer();
for (int j=0; j<b.length; j++){
buf.append(hexDigit[(b[j] >> 4) & 0x0f]);
buf.append(hexDigit[b[j] & 0x0f]); }
return buf.toString();
} }
```

OUTPUT



RESULT

Thus the program for implementing Message Digest (MD5) algorithm is done using java and output is verified successfully.

Ex. No: 13	ADDITIONAL EXPERIMENTS
	DEMONSTRATE HOW TO PROVIDE SECURE DATA STORAGE, SECURE DATA TRANSMISSION AND FOR CREATING DIGITAL SIGNATURE (GnuPG)

AIM

To write a program to demonstrate how to provide secure data storage, secure data transmission and how to create digital signatures.

GnuPG Description:

GnuPG is a complete and free implementation of the OpenPGP standard as defined by RFC4880 GnuPG allows to encrypt and sign our data and communication, features a versatile key management system as well as access modules for all kinds of public key directories. GnuPG, also known asGPG, is a command line tool with features for easy integration with other applications.

PROCEDURE

i) Generate the Key

1. Open GPA (GNU Privacy Assistant) from Start→GPA.
2. Open Key Manager, by selecting Window→Keyring Manager.
3. Select New Key, by selecting Keys→New key.
4. Generate key by Selecting Algorithm, Key Size and specify Name, Email also check Expires if you want to specify key expiry date and Click Ok.
5. Enter '**passphrase'** a secret key to protect your keys. (**ex**: **cnslab**)
6. Re-enter '**passphrase'** to confirm.
7. If the '**passphrase**' is not strength, a dialog will be shown.Click "Take this one anyway" if you do not want to change phrase key. Otherwise if you want to change the "passphrase", click "Enter new passphrase".
8. Repeat steps 1 to 8 to create keys for another user. (Ex:receiver@gmail.com)

ii) Encrypt and Sign Text

1. Open GPA (GNU Privacy Assistant) from Start→GPA.
2. Type the message to encrypt and sign in Clipboard.
3. Click Encrypt, in the tool bar,
4. Select the public key of the receiver to Encrypt and for sign select the sender private key. and click Ok.

5. Enter the ‘passphrase’ keyword of the sender.
6. The Encrypted and signed message will be shown,
7. Copy and save the encrypted message in text file.(message.txt)

iii) Decrypt and verify Message received.

1. Open GPA (GNU Privacy Assistant) from Start→GPA.
2. Under Clipboard paste the content of the message.txt.
3. Click **Encrypt** menu in tool bar,
4. Enter the receivers “passphrase” to decrypt the message.
5. The Decrypted message will be shown in, GNU Privacy assistant – Clipboard.

iv) Encrypt and Sign a File

1. Create a folder SEND and copy the file to be encrypted in it(**Ex:**Input.txt)
2. Open GPA (GNU Privacy Assistant) from Start→GPA.
3. Open the file manager by selecting, menu “Files” in toolbar,
4. Open the file “Input.txt” by clicking “Open” menu in Tool bar,
5. The select the file will be loaded in file Manager.
6. Select the file in File manager window and click Encrypt.
7. Select the Public Key of the receiver and select the sign key.
8. Enter the “passpharse” of the sender to Sign.
9. The new encrypted file(Input.txt.gpg) will be generated in the same folder contains the extension .gpg.
10. The file Input.txt.gpg is the Encrypted and digitally signed.

v) Decrypt and Verify Encrypted Signed File

1. Copy the Encrypted file Input.txt.gpg in new folder “Receive”.
2. Open File manager, and select the file Input.txt.gpg in folder “Receive”.
3. Select the File, Input.txt.gpg in the file manager, and click Decrypt.
4. Enter the “passphrase” of the receiver, and click Ok.
5. The sender Signature will be verified, and the status is shown as valid.
6. The Decrypted file “**Input.txt**” will be in the folder “Receive”.

OUTPUT

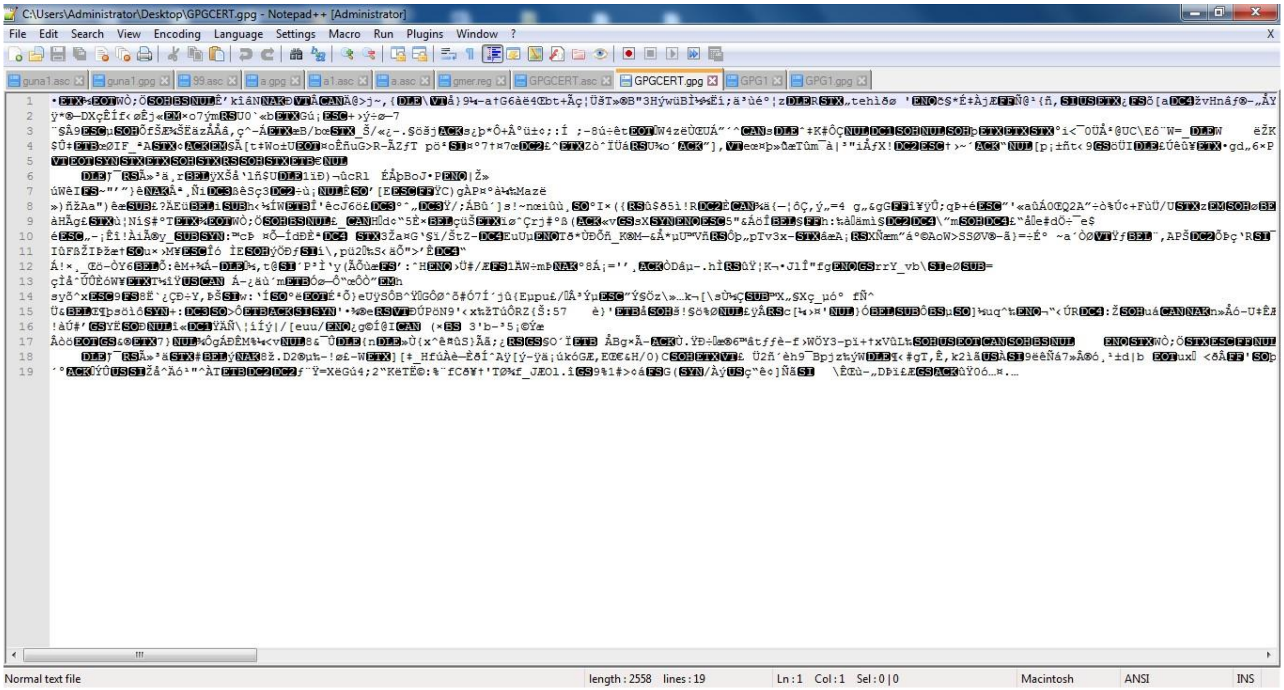


Fig. Generation of Keys

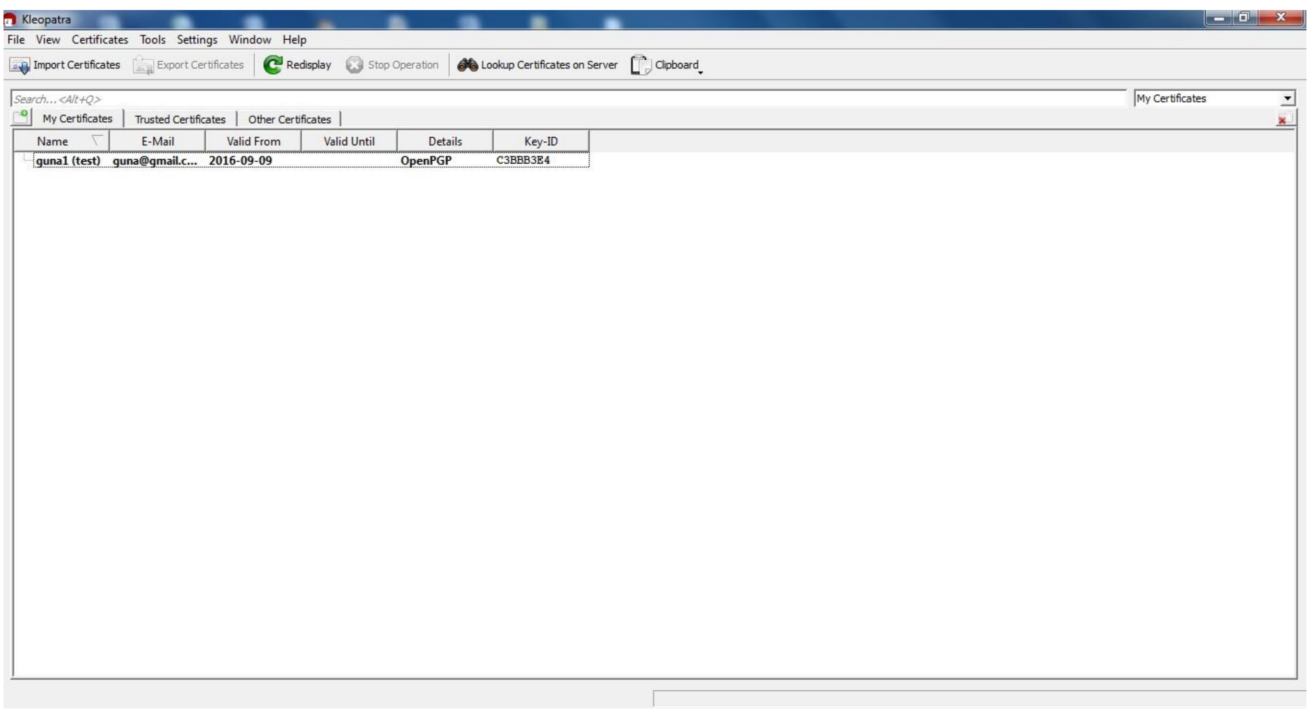


Fig. Generation of Certificate

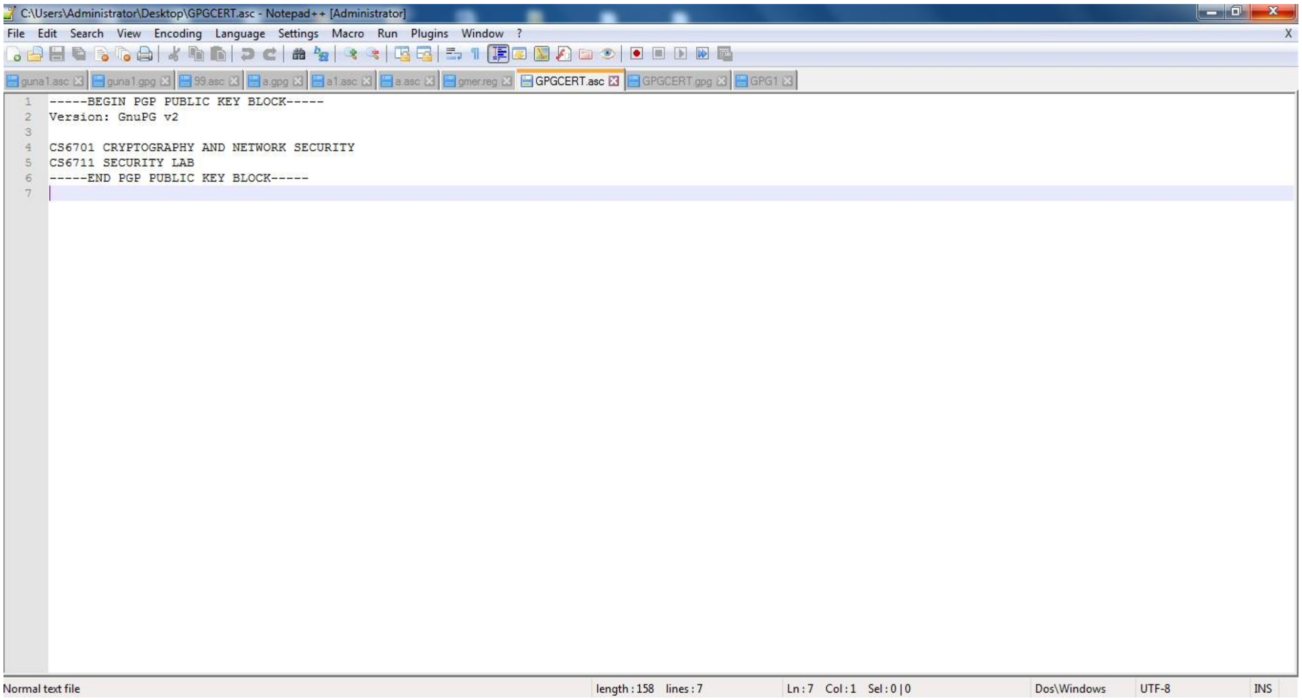


Fig. Plain Text

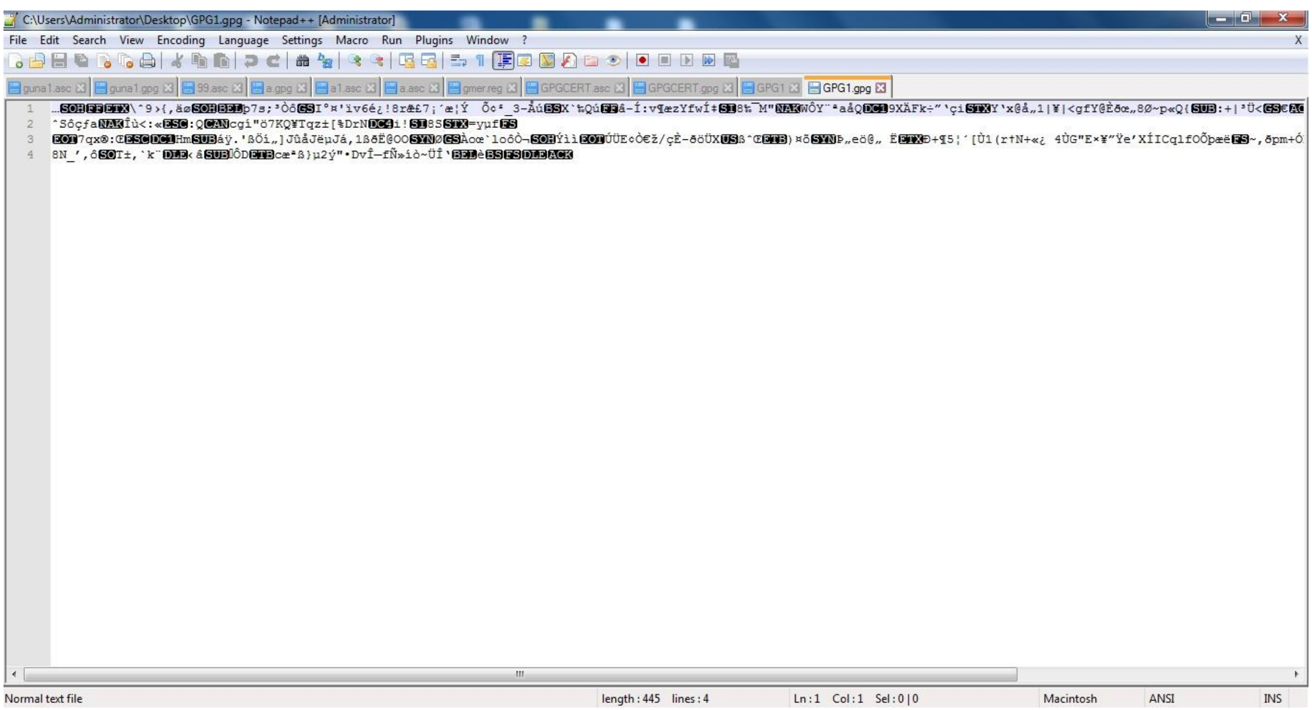


Fig. Cipher Text

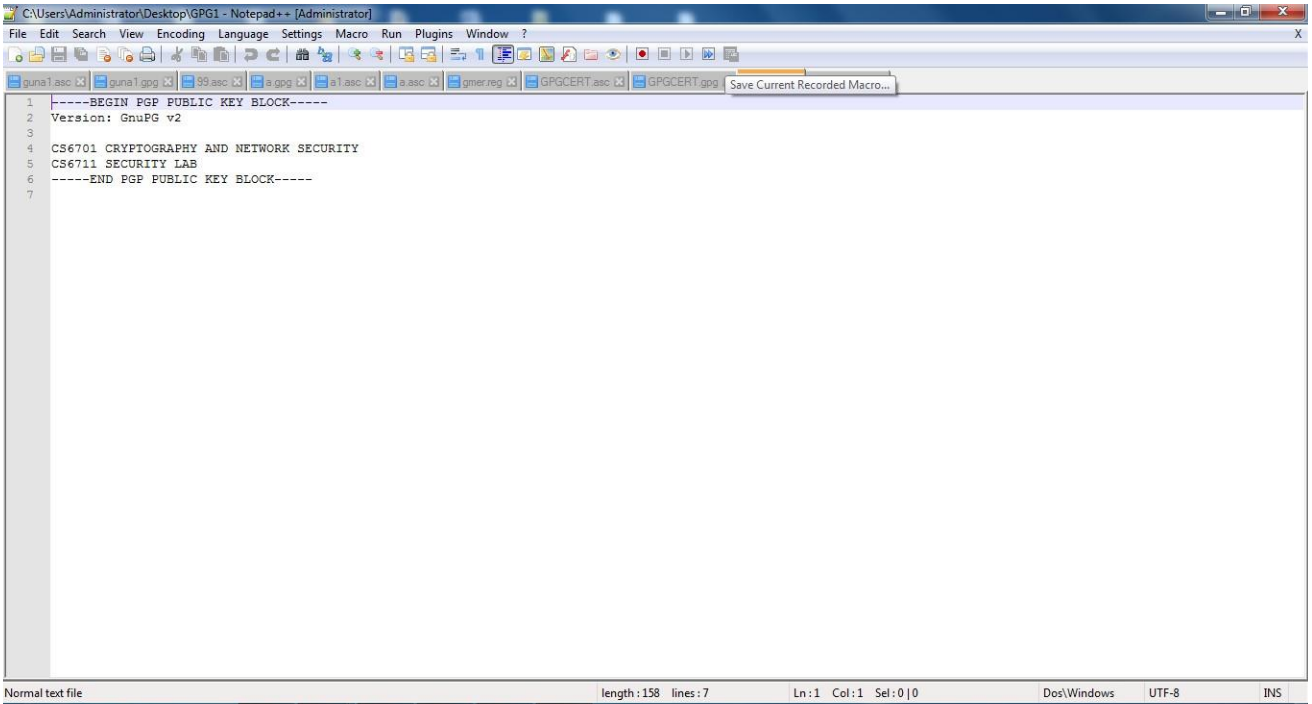


Fig. Decrypted Plain Text

RESULT

Thus the program to demonstrate how to provide secure data storage, secure data transmission, digital signatures is done and output is verified successfully.

Ex. No: 14	ADDITIONAL EXPERIMENTS
	SETUP A HONEY POT AND MONITOR THE HONEY POT ON NETWORK

AIM

To setup a honey pot and to monitor the honey pot on network.

HoneyPot Description:

Honey Pot is a device placed on Computer Network specifically designed to capture malicious network traffic.

KF Sensor is the tool to setup as honeypot when KF Sensor is running it places a siren icon in the windows system tray in the bottom right of the screen. If there are no alerts then green icon is displayed.

PROCEDURE

- Download KF Sensor Evaluation Set File from KF Sensor Website. Install with License Agreement and appropriate directory path. Reboot the Computer now.
- The KF Sensor automatically starts during windows boot Click Next to setup wizard. Select all port classes to include and Click Next.
- Send the email and Send from email enter the ID and Click Next.
- Select the options such as Denial of Service [DOS], Port Activity, Proxy Emulsion, Network Port Analyzer, Click Next.
- Select Install as System service and Click Next.
- Click finish.

OUTPUT

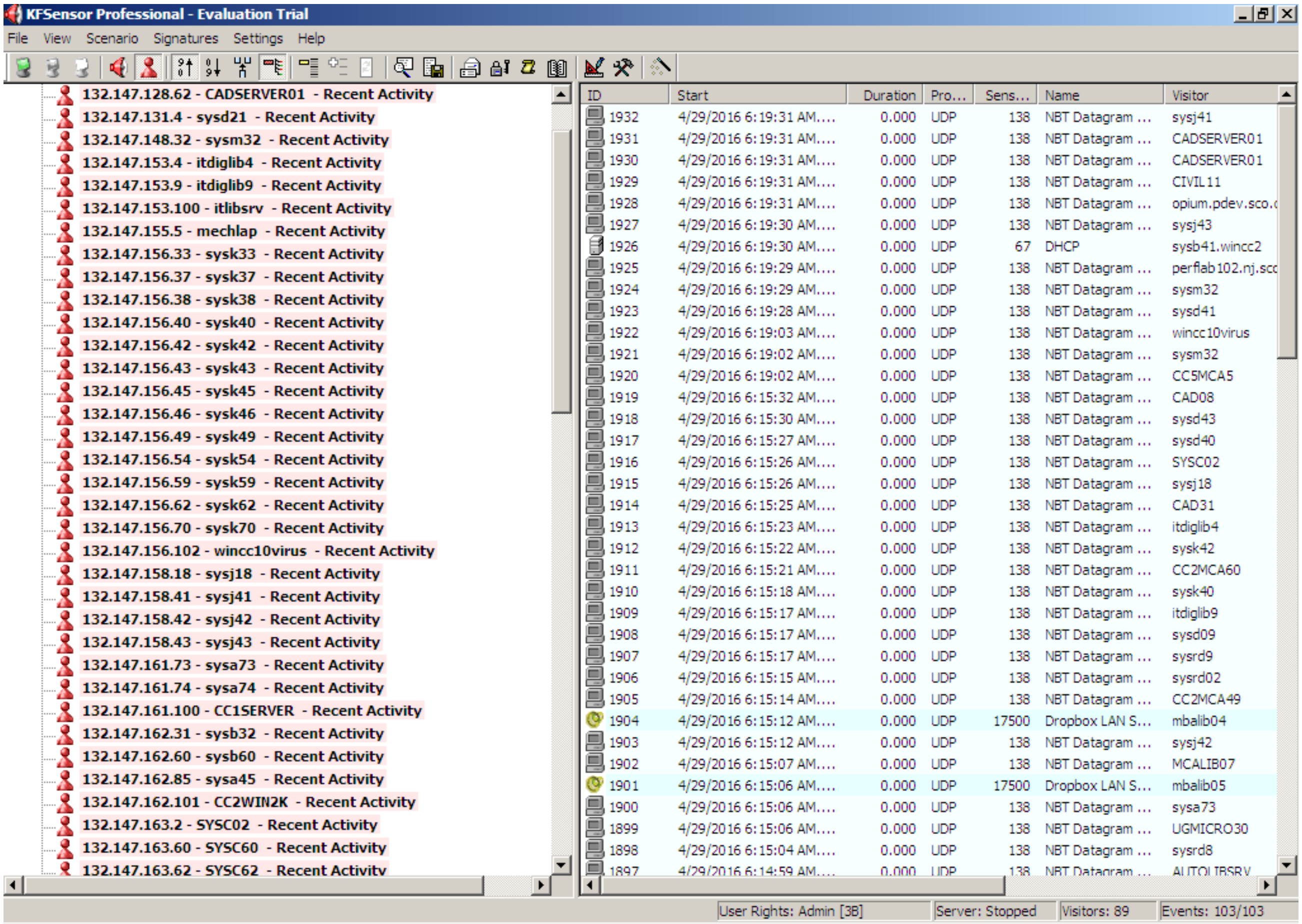


Fig. Recent Activity in System

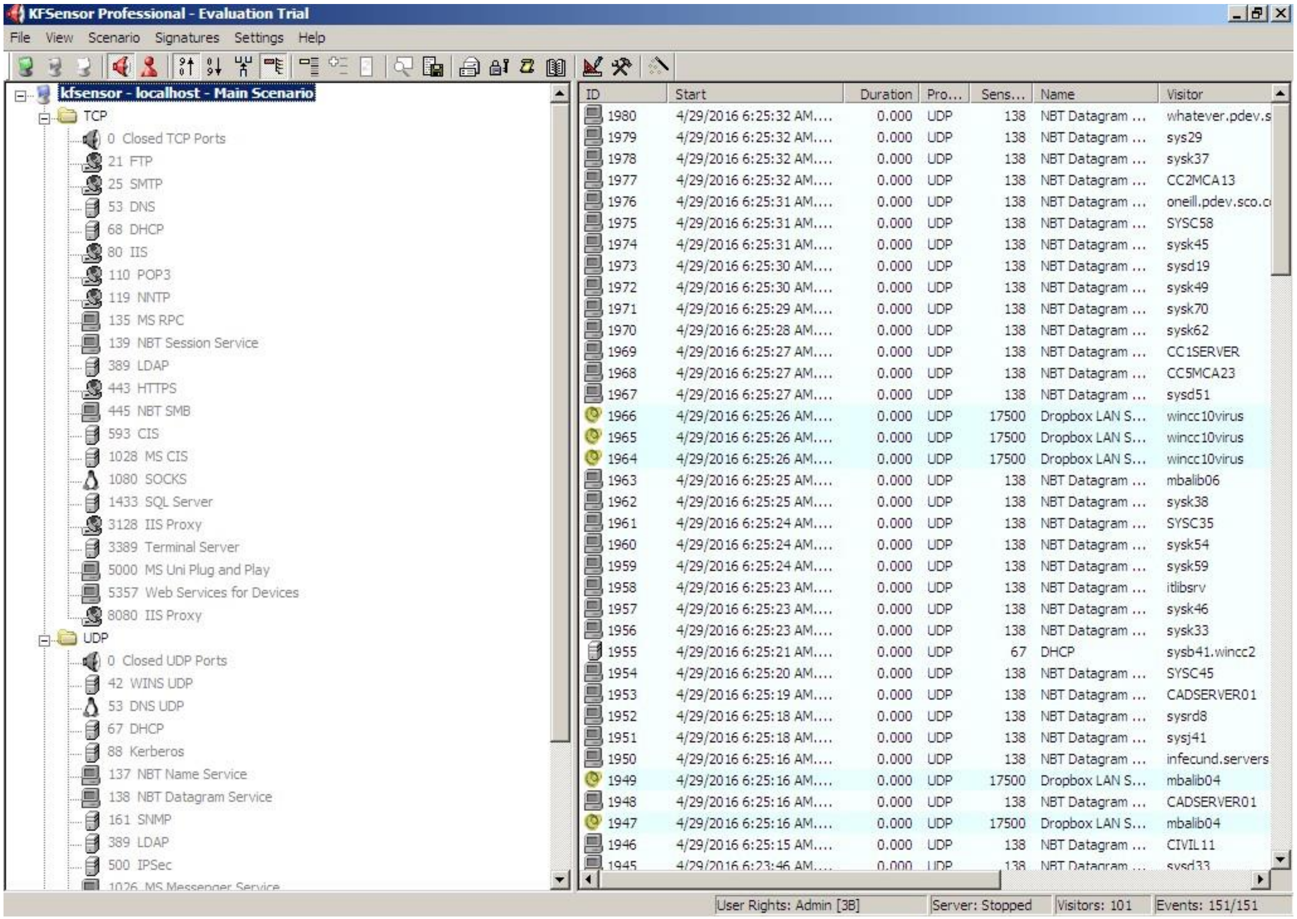


Fig. Local Host Scenario

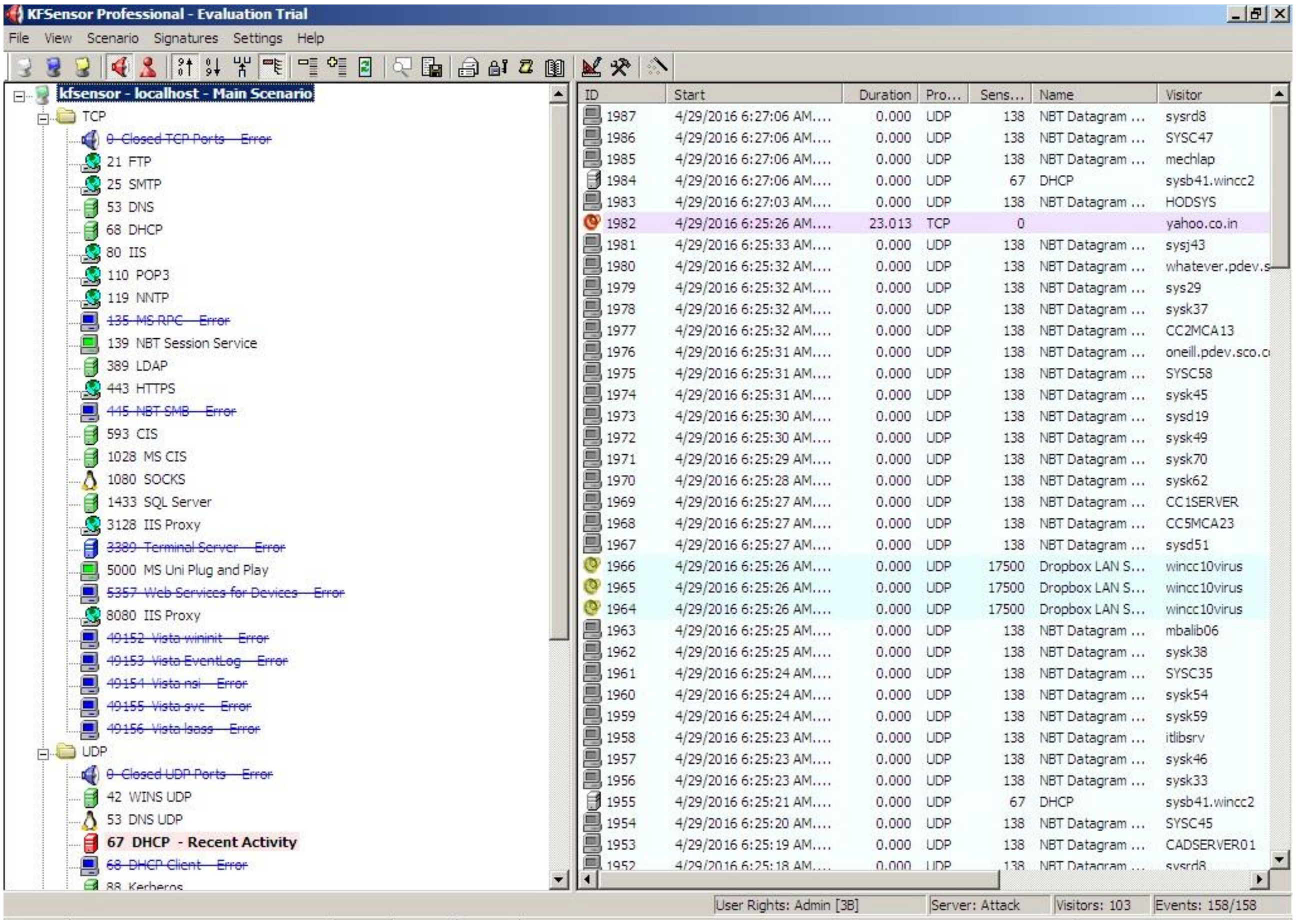


Fig. Evaluation of Local Host under Attack

RESULT

Thus the Honey pot is installed and monitored the Honey pot in a host system on network.