

Universal College Of Engineering, Vasai (E).



**Department
Of
Information Technology**

Cryptography and Network Security Lab manual

Universal College Of Engineering, Vasai (E).

DEPARTMENT OF INFORMATION TECHNOLOGY

Lab Manual for the Academic Year 2022-23

(In accordance with Mumbai university syllabus)

SUBJECT : Security Lab
SUBJECT CODE : ITL502
SEMESTER : V
STREAM : Information Technology

Subject In-charge

Mr. Akshay Agrawal

Lab Objectives: Students will try:

1. To be able to apply the knowledge of symmetric cryptography to implement simple ciphers
2. To be able to analyze and implement public key algorithms hashing and digital signature algorithms.
3. To explore the different network reconnaissance tools to gather information about networks
4. To explore and use tools like sniffers, port scanners and other related tools for analyzing packets in a network.
5. To Scan the network for vulnerabilities and simulate attacks
6. To be able to set up intrusion detection systems using open source technologies and to explore email security.

Lab Outcome: Students will learn to:

1. Illustrate symmetric cryptography by implementing classical ciphers
2. Demonstrate Key management, distribution and user authentication
3. Explore the different network reconnaissance tools to gather information about networks
4. Use tools like sniffers, port scanners and other related tools for analyzing packets in a network.
5. Use open-source tools to scan the network for vulnerabilities and simulate attacks
6. Demonstrate the network security system using open source tools

Hardware Requirements

PC With following Configuration

1. Intel Core i3/i5/i7 Processor
2. 4 GB RAM
3. 500 GB Hard disk

Software Requirements

1. Windows or Linux Desktop OS
2. Wire shark
3. ARPWATCH

INTRODUCTION:

This manual is organized in such a way that the students can directly use it in the laboratory. Each laboratory exercise comprises of

1. Statement of the problem (Aim)

2. Theory

3. Program

4. Output

5. Conclusion

LIST OF LAB EXERCISES:

Sr No	Name of the Experiment
1	Design and Implementation of a product cipher using Substitution and Transposition ciphers
2	To implement a program to encrypt a plain text and decrypt a cipher text using play fair Cipher substitution technique.
3	To develop a program to encrypt and decrypt using the Hill cipher substitution technique
4	Implementation and analysis of RSA cryptosystem
5	Implementation of Digital signature scheme using Digital Signature Algorithm
6	Study the use of network reconnaissance tools like WHOIS, dig,traceroute, nslookup to gather information about networks and domain registrars
7	Download and install nmap. Use it with different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan, udp port scan, etc.
8	Study of packet sniffer tools wireshark
9	Set up Snort and study the logs.

Experiment 1

AIM: Design and Implementation of a product cipher using Substitution and Transposition ciphers

THEORY:

Substitution cipher is a method of encryption by which units of plaintext are replaced with ciphertext according to a regular system; the "units" may be single letters (the most common), pairs of letters, triplets of letters, mixtures of the above, and so forth. The receiver deciphers the text by performing an inverse substitution.

Transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed.

PROGRAM:

```
import java.util.*;
class ProductCipher {
public static void main(String args[]) {
System.out.println("Enter the input to be encrypted:");
String substitutionInput = new Scanner(System.in).nextLine();
System.out.println("Enter a number of rows for transposition matrix:");
int n = new Scanner(System.in).nextInt();

// Substitution encryption
StringBuffer substitutionOutput = new StringBuffer();
for(int i=0 ; i<substitutionInput.length() ; i++) {
char c = substitutionInput.charAt(i);
substitutionOutput.append((char) (c+5));
}
System.out.println("\nSubstituted text:");
System.out.println(substitutionOutput);
```

```
// Transposition encryption
String transpositionInput = substitutionInput.toString();
int modulus;
if((modulus = transpositionInput.length()%n) != 0) {
    modulus = n-modulus;
// 'modulus' is now the number of blanks/padding (X) to be appended
for( ; modulus!=0 ; modulus--) {
    transpositionInput += "/";
}}
StringBuffer transpositionOutput = new StringBuffer();
System.out.println("\nTransposition Matrix:");
for(int i=0 ; i<n ; i++) {
    for(int j=0 ; j<transpositionInput.length()/n ; j++) {
        char c = transpositionInput.charAt(i+(j*n));
        System.out.print(c);
        transpositionOutput.append(c);
    }
    System.out.println();
}
System.out.println("\nFinal encrypted text:");
System.out.println(transpositionOutput);

// Transposition decryption
n = transpositionOutput.length()/n;
StringBuffer transpositionPlaintext = new StringBuffer();
for(int i=0 ; i<n ; i++) {
    for(int j=0 ; j<transpositionOutput.length()/n ; j++) {
        char c = transpositionOutput.charAt(i+(j*n));
        transpositionPlaintext.append(c);
    }
}
System.out.println("\nPlaintext of Transposition:");
System.out.println(transpositionPlaintext);

// Substitution decryption
StringBuffer plaintext = new StringBuffer();
for(int i=0 ; i<substitutionOutput.length() ; i++) {
```

```
char c = substitutionOutput.charAt(i);
plaintext.append((char) (c-5));
}
System.out.println("\nPlaintext of Substitution:");
System.out.println(plaintext);
}}
```

OUTPUT:

D:\>javac ProductCipher.java

D:\>java ProductCipher

Enter the input to be encrypted:

hello

Enter a number of rows for transposition matrix:

2

Substituted text:

mjqqt

Transposition Matrix:

hlo

el/

Final encrypted text:

hloel/

Plaintext of Transposition:

hello/

Plaintext of Substitution:

hello

D:\>

CONCLUSION: Thus, we have studied and successfully executed product cipher.

Experiment No : 2

AIM:

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

THEORY:

The Playfair cipher or Playfair square is a manual symmetric encryption technique and was the first literal digraph substitution cipher. Playfair cipher is a multi- alphabet letter encryption cipher, which deals with letters in plaintext as single units and renders these units into Ciphertext letters. The Playfair algorithm is based on the use of a 5X5 matrix of letters built using a keyword. The playfair cipher starts with creating a key table. The key table is a 5×5 grid of letters that will act as the key for encrypting your plaintext. Each of the 25 letters must be unique and one letter of the alphabet (usually Q) is omitted or treat I and J as the same alphabet from the table (as there are 25 spots and 26 letters in the alphabet). Let's say we wanted to use the phrase "Hello World" as our key. The first characters (going left to right) in the table will be the phrase, with duplicate letters removed. The rest of the table will be filled with the remaining letters of the alphabet, in order. The text can only contain alphabets (i.e. no spaces or punctuation). Also this cipher is case insensitive. We start by removing spaces from the text and duplicate letters from the key then converting them into uppercase. If any double letters occurring in the plaintext, an 'x' is inserted between the occurrences of the letters. In a playfair cipher the message is split into digraphs, pairs of two letters. If there is an odd number of letters, a Z or X is added to the last letter. Now for the actual encryption process, the Playfair cipher uses a few simple rules relating to where the letters of each digraph are in relation to each other. Performing this quick encryption process for each digraph in the message eventually results in the entire plaintext being encrypted. Decrypting the Playfair cipher (assuming you have the key) is as simple as doing the same process in reverse. Assuming you have the same key you will always be able to create the same key table, and then decrypt any messages made using that key. The Playfair cipher was used mainly to protect important, yet non-critical secrets, as it is quick to use and requires no special equipment. By the time enemy cryptanalysts could break the code the information it was protecting would often no longer be relevant.

ALGORITHM:

1. Generate the (5 x 5) key table or matrix using the key.
 - a. Fill the spaces in the table with the letters of the key without any duplication of letters.

-
- b. Fill the remaining spaces with the rest of the letters of the alphabet in order by having 'I' & 'J' in the same space or omitting 'Q' to reduce the alphabet to fit.
 2. Remove any punctuation or characters from the plain text that are not present in the key square.
 3. Identify any double letters in the plaintext and insert 'X' between the two occurrences.
 4. Split the plain text into digraphs (groups of 2 letters)
 5. Encryption: Locate the digraph letters in the key table
 - a. If the letters appear on the same row of the 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).
 - b. If the letters appear on the same column of the 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).
 - c. If the letters are in different rows and columns, replace the pair 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 encrypted letter of the pair is the one that lies on the same row as the first plaintext letter.
 - d. Append the letters referred from the key table using the steps 5.a, 5.b and 5.c to generate Cipher text.
 6. Decryption: Split the cipher text into digraphs and locate the digraph letters in the key table.
 - a. If the letters appear on the same row of the table, replace them with the letters to their immediate left respectively (wrapping around to the right side of the row if a letter in the original pair was on the left side of the row).
 - e. If the letters appear on the same column of the table, replace them with the letters immediately above respectively (wrapping around to the bottom side of the column if a letter in the original pair was on the top side of the column).
 - f. If the letters are in different rows and columns, replace the pair 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 encrypted letter of the pair is the one that lies on the same row as the first plaintext letter.
 - g. Append the letters referred from the key table using the steps 6.a, 6.b and 6.c to generate Plain text.
-

7. Stop

Program:

```
import java.util.*;
import java.io.*;
public class Playfair
{
    private char pfmatrix[][] = new char[5][5];
    public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String plain,cipher;
    int jflg=0,xpad=0;
    int row,col;
    public void matrixgen(String key)
    {
        char keychar;
        int count=0;
        int alphacount=0;
        int p,k,flg=1;
        for(int i=0; i<5; i++)
        {
            for(int j=0;j<5;j++)
            {
                if (count<key.length())
                {
                    keychar=key.charAt(count);
                    if (keychar == 'J')
                        keychar='I';
                    p=0;
                    while (p<count)
                    {
                        flg=1;
                        if (keychar==key.charAt(p))
                        {
                            count++;
                            if (count == key.length())
```

```
{
    flg=0;
    break;
}
keychar=key.charAt(count);
p=0;
}
else
    p++;
}
if (flg!=0)
{
    pfmatrix[i][j]=keychar;
    count++;
}
if ((count==key.length()) && (flg==0))
{
    if(alphacount<26)
    {
        keychar=ALPHABET.charAt(alphacount);
        k=0;
        while (k<key.length())
        {
            if ((keychar==key.charAt(k)) || (keychar=='J'))
            {
                alphacount++;
                keychar=ALPHABET.charAt(alphacount);
                k=0;
            }
            else
                k++;
        }
        //if (keychar!='J' && k==key.length())
        pfmatrix[i][j]=keychar;
```

```
alphacount++;
}
}
}
else
{
if(alphacount<26)
{
keychar=ALPHABET.charAt(alphacount);
k=0;
while (k<key.length())
{
if ((keychar==key.charAt(k)) || (keychar=='J'))
{
alphacount++;
keychar=ALPHABET.charAt(alphacount);
k=0;
}
else
k++;
}
pfmatrix[i][j]=keychar;
alphacount++;
}
}
}
}
}

public void matrixdisplay()
{
for(int i=0;i<5;i++)
{
for(int j=0;j<5;j++)
System.out.print(pfmatrix[i][j] + " ");
```

```
System.out.println();
}
}
public String pfencryption(String txt)
{
int i,j,k;
int ch1row,ch2row,ch1col,ch2col;
char ch1,ch2,tmp1,tmp2;
String nutext="";
String text="";
i=0;
while (i<(txt.length()-1))
{
text += txt.charAt(i);
if (txt.charAt(i) == txt.charAt(i+1))
{
text += 'X';
xpad++;
}
i++;
}
text += txt.charAt(txt.length()-1);
System.out.println("TEXT : " + text);
if (text.length()%2 != 0)
{
text += 'X';
xpad++;
}
System.out.println("FINAL TEXT : "+ text);
for(k=0;k<text.length();k=k+2)
{
ch1=text.charAt(k);
ch2=text.charAt(k+1);
System.out.println("CHARACTER PAIR : " + ch1 + " " + ch2);
```

```
matsearch(ch1);
ch1row=row;
ch1col=col;
matsearch(ch2);
ch2row=row;
ch2col=col;
// System.out.println("ch1row:" + ch1row + "ch1col:" + ch1col);
// System.out.println("ch2row:" + ch2row + "ch2col:" + ch2col);
if (ch1row==ch2row)
{
tmp1=pfmatrix[ch1row][(ch1col+1)%5];
tmp2=pfmatrix[ch2row][(ch2col+1)%5];
}
else if (ch1col==ch2col)
{
tmp1=pfmatrix[(ch1row+1)%5][ch1col];
tmp2=pfmatrix[(ch2row+1)%5][ch2col];
}
else
{
tmp1=pfmatrix[ch1row][ch2col];
tmp2=pfmatrix[ch2row][ch1col];
}
nutext += tmp1;
nutext += tmp2;
System.out.println("TRANSLATED TEXT :" + tmp1 + " " + tmp2);
}
return nutext;
}
public String pfdecryption(String text)
{
int i,j,k;
int ch1row,ch2row,ch1col,ch2col;
char ch1,ch2,tmp1,tmp2;
```

```
String nutext="";
String txt="";
for(k=0;k<text.length();k=k+2)
{
ch1=text.charAt(k);
ch2=text.charAt(k+1);
System.out.println("CHARACTER PAIR : " + ch1 + " " + ch2);
matsearch(ch1);
ch1row=row;
ch1col=col;
matsearch(ch2);
ch2row=row;
ch2col=col;
// System.out.println("ch1row:" + ch1row + "ch1col:" + ch1col);
// System.out.println("ch2row:" + ch2row + "ch2col:" + ch2col);
if (ch1row==ch2row)
{ int c1,c2;
c1 = ch1col-1;
if (c1<0)
c1 = c1+5;
c2 = ch2col-1;
if (c2<0)
c2=c2+5;
tmp1=pfmatrix[ch1row][c1];
tmp2=pfmatrix[ch2row][c2];
}
else if (ch1col==ch2col)
{
int r1,r2;
r1=ch1row-1;
r2=ch2row-1;
if (r1<0)
r1=r1+5;
if (r2<0)
```

```
r2=r2+5;
tmp1=pfmatrix[r1][ch1col];
tmp2=pfmatrix[r2][ch2col];
}
else
{
tmp1=pfmatrix[ch1row][ch2col];
tmp2=pfmatrix[ch2row][ch1col];
}
nutext += tmp1;
nutext += tmp2;
System.out.println("TRANSLATED TEXT :" + tmp1 + " " + tmp2);
}
if (xpad != 0)
{
i=0;
while (i<nutext.length())
{
if (nutext.charAt(i) == 'X')
{
i++;
continue;
}
txt += nutext.charAt(i);
i++;
}
System.out.println("TEXT :" + txt);
return txt;
}
else
{
System.out.println("TEXT : " + nutext);
return nutext;
}
```

```
}  
public void matsearch(char ch)  
{  
    int i,j;  
    if (ch=='J')  
    {  
        ch='I';  
        jflg=1;  
    }  
    for(i=0;i<5;i++)  
    {  
        for(j=0;j<5;j++)  
        {  
            if (pfmatrix[i][j] == ch)  
            {  
                row=i;  
                col=j;  
            }  
        }  
    }  
}  
public static void main(String[] args)  
{  
    Playfair pf = new Playfair();  
    Scanner sc = new Scanner(System.in);  
    System.out.println("Enter the PLAYFAIR KEY: ");  
    String pfkey = new String();  
    pfkey = sc.next();  
    System.out.println("PLAYFAIR MATRIX");  
    pf.matrixgen(pfkey);  
    pf.matrixdisplay();  
    String ptext = new String();  
    System.out.println("Enter PLAIN TEXT");  
    ptext = sc.next();
```

```
String ctext = new String();
ctext = pf.pfencryption(ptext);
System.out.println();
System.out.println("CIPHER TEXT :" + ctext);
System.out.println();
String plaintext = new String();
plaintext = pf.pfdecryption(ctext);
System.out.println();
System.out.println("PLAIN TEXT :" + plaintext);
sc.close();
} }
```

Output:

C:\Program Files\Java\jdk1.8.0_71\bin>javac Playfair.java

C:\Program Files\Java\jdk1.8.0_71\bin>java Playfair

Enter the PLAYFAIR KEY:

INFORMATION

PLAYFAIR MATRIX

I N F O R

M A T B C

D E G H K

L P Q S U

V W X Y Z

Enter PLAIN TEXT

GOODMORNING

TEXT : GOXODMORNING

FINAL TEXT : GOXODMORNING

CHARACTER PAIR :G O

TRANSLATED TEXT :H F

CHARACTER PAIR :X O

TRANSLATED TEXT :Y F

CHARACTER PAIR :D M

TRANSLATED TEXT :L D

CHARACTER PAIR :O R

TRANSLATED TEXT :R I

CHARACTER PAIR : N I

TRANSLATED TEXT :F N

CHARACTER PAIR : N G

TRANSLATED TEXT :F E

CIPHER TEXT :HFYFLDRIFNFE

Conclusion:

Thus the program to implement Play Fair Cipher technique was developed and executed successfully

Experiment No : 3**AIM:**

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

THEORY:

The Hill cipher is a substitution cipher invented by Lester S. Hill in 1929. Hill's major contribution was the use of mathematics to design and analyse cryptosystems. The **Hill cipher** is a polygraphic substitution **cipher** based on linear algebra. Hill ciphers are applications of linear algebra because a Hill cipher is simply a linear transformation represented by a matrix with respect to the standard basis. Groups of letters are represented by vectors. The domain of the linear transformation is all plaintext vectors, while the codomain is made up of all ciphertext vectors. Matrix multiplication is involved in the encoding and decoding process. And when trying to find the inverse key, we will use elementary row operations to row reduce the key matrix in order to find its inverse in the standard manner. 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.

Encryption : Cipher text = (Plain text * Key) mod 26

Decryption : Plain text = (Cipher text * Key-1) mod 26

ALGORITHM:

1. Get the n -by- n key matrix with vectors of length n .
2. Separate the plaintext from left to right into some number k of groups of n letters each. If you run out of letters when forming the final group, repeat the last plaintext letter or 'X' as many times as needed to fill out the final group of n letters.
3. Replace each letter by the corresponding number of its position (from 0 through $m-1$) in the alphabet to get k groups of n integers each.
4. Encryption:
 - a. Reshape each of the k groups of integers into an n -row column vector called Plain text matrix
 - b. Cipher Text Matrix = (Plain Text Matrix * Key Matrix) mod 26.
 - c. Using step 4.b, perform matrix multiplication of Plain text matrix and Key matrix and modulus 26 to yield Cipher text matrix.

d. Translate the Cipher text matrix to string representation by replacing each of the entries with the corresponding letter of the alphabet, after arranging all k of the resulting column vectors in order into a single vector of length k x n

5.Decryption:

a. Find the inverse of the key matrix

i. Find the determinant of the key matrix

ii. Transpose the key matrix

iii. Find minor matrix and then Cofactor of the key matrix

iv. $\text{Key-1} = [[\text{Det}(\text{key matrix})]^{-1} * \text{Cofactor}] \text{ mod } 26$ using modulus arithmetic

b. $\text{Plain Text} = (\text{Cipher Text} * \text{Key-1}) \text{ mod } 26$

c. Using step 5.b, perform matrix multiplication of Cipher text matrix and Key inverse matrix and modulus 26 to yield Plain text matrix.

d. Translate the Plain text matrix to string representation by replacing each of the entries with the corresponding letter of the alphabet, after arranging all k of the resulting column vectors and removing any letters padded in order into a single vector of length k x n

6. Stop

Program:

```
import java.util.*;
import java.io.*;
public class Hillcipher
{
    public int keyinverse[][] = new int[3][3];
    public int key[][] = { { 17, 17, 5 }, { 21, 18, 21 }, { 2, 2, 19 } };
    public int plainmat[][] = new int[8][3];
    public int ciphermat[][] = new int[8][3];
    public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    String plain,cipher;
    int row, flag=0, decrypt=0;
    public void matdisplay(int mat[][])
    {
        int i, j;
        for(i=0;i<row;i++)
```

```
{
for(j=0;j<3;j++)
System.out.print(mat[i][j] + " ");
System.out.println();
}
}

public void keydisplay(int mat[][])
{
int i, j;
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
System.out.print(mat[i][j] + " ");
System.out.println();
}
}

public void inverse()
{
int dtrmnt = 0;
int mulinvdtrmnt = 0;
int x, y, z, i, j, a, tmp, p, q;
int transkey[][] = new int[3][3];
int minormat[][] = new int[3][3];
int temp[][] = new int[2][2];
System.out.println("HILL CIPHER KEY");
keydisplay(key);
x = key[0][0] * ((key[1][1] * key[2][2]) - (key[1][2] * key[2][1]));
y = key[0][1] * ((key[1][0] * key[2][2]) - (key[1][2] * key[2][0]));
z = key[0][2] * ((key[1][0] * key[2][1]) - (key[1][1] * key[2][0]));
dtrmnt = (x-y+z)%26;
if ( dtrmnt < 0 )
dtrmnt = dtrmnt + 26;
System.out.println("DETERMINANT :" + dtrmnt);
a = dtrmnt;
```

```
for(i=0;i<25;i++)
{
tmp = ( a * i ) % 26;
if ( tmp == 1 )
{
mulinvdtrmnt = i;
break;
}
}
System.out.println("MULTIPLICATIVE INVERSE OF DETERMINANT:" +
mulinvdtrmnt);
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
transkey[i][j] = key[j][i];
}
// keydisplay(transkey);
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
p=0;
q=0;
for(x=0;x<3;x++)
{
for(y=0;y<3;y++)
{
if ((x!=i) && (y!=j))
{
temp[p][q] = transkey[x][y];
q++;
if ( q == 2)
{
q=0;
```

```

p++;
}
}
}
}

minormat[i][j]=(temp[0][0]*temp[1][1])-(temp[0][1]*temp[1][0]);
minormat[i][j] = minormat[i][j] * (int)Math.pow(-1,(i+j));
}
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
keyinverse[i][j]=(mulinvdtrmnt * minormat[i][j])%26;
if (keyinverse[i][j] < 0)
keyinverse[i][j] = keyinverse[i][j] + 26;
}
}
System.out.println("KEY INVERSE");
keydisplay(keyinverse);
} p
ublic void str2matrix(String text)
{
int k, p, n;
if ((text.length() % 3) == 1)
{
n=text.length();
text += 'X';
text += 'X';
flag = 2;
}
if ((text.length() % 3) == 2)
{
text += 'X';

```

```
flag = 1;
}
row = (text.length()) / 3;
k = 0;
for(int i=0; i<row; i++)
{
for(int j=0;j<3;j++)
{
for (p=0;p<26;p++)
{
if (text.charAt(k) == ALPHABET.charAt(p))
{
plainmat[i][j] = p;
k++;
break;
}
}
}
}
// System.out.println("PLAIN TEXT MATRIX");
matdisplay(plainmat);
System.out.println();
}
public String matrix2str(int mat[][] )
{
int i, j, k;
String txt="";
String tmp="";
for(i=0;i<row;i++)
{
for(j=0;j<3;j++)
{
k = mat[i][j];
txt += ALPHABET.charAt(k);
```

```
}  
}  
if (decrypt == 1)  
{  
    if ( flag == 1 )  
    {  
        tmp = txt.substring(0, (txt.length()-1));  
        return tmp;  
    }  
    if ( flag == 2 )  
    {  
        tmp = txt.substring(0, (txt.length()-2));  
        return tmp;  
    }  
}  
return txt;  
}  
public String hcencryption(String ptxt)  
{  
    int i,j,k;  
    int sum=0;  
    String ctxt="";  
    decrypt=0;  
    System.out.println("HILL CIPHER ENCRYPTION");  
    System.out.println("PLAIN TEXT MATRIX");  
    str2matrix(ptxt);  
    for(i=0;i<row;i++)  
    {  
        for(j=0;j<3;j++)  
        {  
            for(k=0;k<3;k++)  
            sum += plainmat[i][k] * key[k][j];  
            ciphermat[i][j] = sum % 26;  
            sum = 0;  
        }  
    }  
}
```

```

}
}
System.out.println("CIPHER TEXT MATRIX");
matdisplay(ciphermat);
ctxt = matrix2str(ciphermat);
return ctxt;
}
public String hcdecryption(String ctxt)
{
    int i,j,k;
    int sum=0;
    String ptxt="";
    decrypt=1;
    System.out.println("HILL CIPHER DECRYPTION");
    System.out.println("CIPHER TEXT MATRIX");
    str2matrix(ctxt);
    for(i=0;i<row;i++)
    {
        for(j=0;j<3;j++)
        {
            for(k=0;k<3;k++)
            sum += ciphermat[i][k] * keyinverse[k][j];
            plainmat[i][j] = sum % 26;
            sum = 0;
        }
    }
    System.out.println("PLAIN TEXT MATRIX");
    matdisplay(plainmat);
    ptxt = matrix2str(plainmat);
    return ptxt;
}
public static void main(String[] args)
{

```

```
Hillcipher hc = new Hillcipher();
Scanner sc = new Scanner(System.in);
hc.inverse();
String ptext = new String();
System.out.println("Enter PLAIN TEXT");
ptext = sc.next();
String ctext = new String();
ctext = hc.hcencryption(ptext);
System.out.println();
System.out.println("CIPHER TEXT :" + ctext);
System.out.println();
String plaintext = new String();
plaintext = hc.hcdecryption(ctext);
System.out.println();
System.out.println("PLAIN TEXT :" + plaintext);
sc.close();
}
}
```

Output:

C:\Program Files\Java\jdk1.8.0_71\bin>javac Hillcipher.java

C:\Program Files\Java\jdk1.8.0_71\bin>java Hillcipher

HILL CIPHER KEY

17 17 5

21 18 21

2 2 19

DETERMINANT : 23

MULTIPLICATIVE INVERSE OF DETERMINANT : 17

KEY INVERSE

4 9 15

15 17 6

24 0 17

Enter PLAIN TEXT

PAYMOREMONEY

HILL CIPHER ENCRYPTION

PLAIN TEXT MATRIX

15 0 24

12 14 17

4 12 14

13 4 24

CIPHER TEXT MATRIX

17 17 11

12 22 1

10 0 18

15 3 7

CIPHER TEXT :RRLMWBKASPDH

HILL CIPHER DECRYPTION

CIPHER TEXT MATRIX

17 17 11

12 22 1

10 0 18

15 3 7

PLAIN TEXT MATRIX

15 0 24

12 14 17

4 12 14

13 4 24

PLAIN TEXT :PAYMOREMONEY

Conclusion:

Thus the program to implement Hill cipher encryption technique was developed and executed successfully.

Experiment No : 4

AIM: Implementation of RSA cryptosystem

THEORY:

RSA Cryptosystem

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. We will see two aspects of the RSA cryptosystem, firstly generation of key pair and secondly encryption-decryption algorithms.

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 –

$$ed = 1 \text{ mod } (p - 1)(q - 1)$$

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.

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 – $C = Pe \text{ mod } n$
- 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 .

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 .

$$\text{Plaintext} = Cd \text{ mod } n$$

PROGRAM:

```
import java.util.*;
class Exp1
{
public static void main(String args[])
```

```
{
Scanner sc=new Scanner(System.in);
int d=0;
System.out.println("Enter two prime numbers");
int p=sc.nextInt();
int q=sc.nextInt();
int n=p*q;
System.out.println("n="+n);
int e=0;
int pn=(p-1)*(q-1);
search:
{
for(int i=2;i<=pn;i++)
{
int r;
int j=i;
int k=pn;
while(k != j)
{
if(k > j)
k = k-j;
else
j = j-k;
}
if(k==1)
{
e=i;
break search;
}
}
}
System.out.println("e="+e);
go:{
for(int i=1;i<=pn;i++)
```

```
{
int x=(e*i)%pn;
if(x==1)
{
System.out.println("d="+i);
System.out.println("The private key is (d) "+i);
d=i;
break go;
} } }
System.out.println("The public key is (n,e) "+n+", "+e);
String t;
int c;
System.out.println("Enter plaintext");
t=sc.next();
int m = 0;
for (int i = 0; i < t.length(); i++){
m += (int)t.charAt(i);
}
c=((m)^e)%n;
System.out.println("The Encryted message is "+m);
m=(c^d)%n;
System.out.println("The decrypted message is "+t);
} }
```

OUTPUT: .

C:\Users\UCOE\Desktop>java Exp1

Enter two prime numbers

7

11

n=77

e=7

d=43

The private key is (d) 43

The public key is (n,e) 77, 7

Enter plaintext

hello123

The Encrypted message is 682

The decrypted message is hello123

C:\Users\UCOE\Desktop>

CONCLUSION: Thus, we have studied and successfully executed RSA algorithm.

Experiment 5:**AIM:** Implementation of Digital signature scheme using Digital Signature Algorithm**THEORY:**

A digital signature is represented in a computer as a string of binary digits.

Global Public-Key Components

- p** prime number where $2^{L-1} < p < 2^L$
for $512 \leq L \leq 1024$ and L a multiple of 64;
i.e., bit length of between 512 and 1024 bits
in increments of 64 bits
- q** prime divisor of $(p - 1)$, where $2^{N-1} < q < 2^N$
i.e., bit length of N bits
- g** $= h(p - 1)/q \bmod p$,
where h is any integer with $1 < h < (p - 1)$
such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

- x** random or pseudorandom integer with $0 < x < q$

User's Public Key

- y** $= g^x \bmod p$

User's Per-Message Secret Number

- k** random or pseudorandom integer with $0 < k < q$

Signing

- $r = (g^k \bmod p) \bmod q$
 $s = [k^{-1} (H(M) + xr)] \bmod q$
 Signature = (r, s)

Verifying

- $w = (s')^{-1} \bmod q$
 $u_1 = [H(M')w] \bmod q$
 $u_2 = (r')w \bmod q$
 $v = [(g^{u_1} y^{u_2}) \bmod p] \bmod q$
 TEST: $v = r'$

M = message to be signed

$H(M)$ = hash of M using SHA-1

M', r', s' = received versions of M, r, s

PROGRAM:

```
import java.util.*;
import java.math.BigInteger;
class dsaAlg
{
    final static BigInteger one = new BigInteger("1");
    final static BigInteger zero = new BigInteger("0");
    /* incrementally tries for next prime */
    public static BigInteger getNextPrime(String ans)
    {
        BigInteger test = new BigInteger(ans);
```

```
        while (!test.isProbablePrime(99))
        {
            test = test.add(one);
        }
        return test;
    }

    /* finds largest prime factor of n */
    public static BigInteger findQ(BigInteger n)
    {
        BigInteger start = new BigInteger("2");
        while (!n.isProbablePrime(99))
        {
            while (!(n.mod(start)).equals(zero)))
            {
                start = start.add(one);
            }
            n = n.divide(start);
        }
        return n;
    }

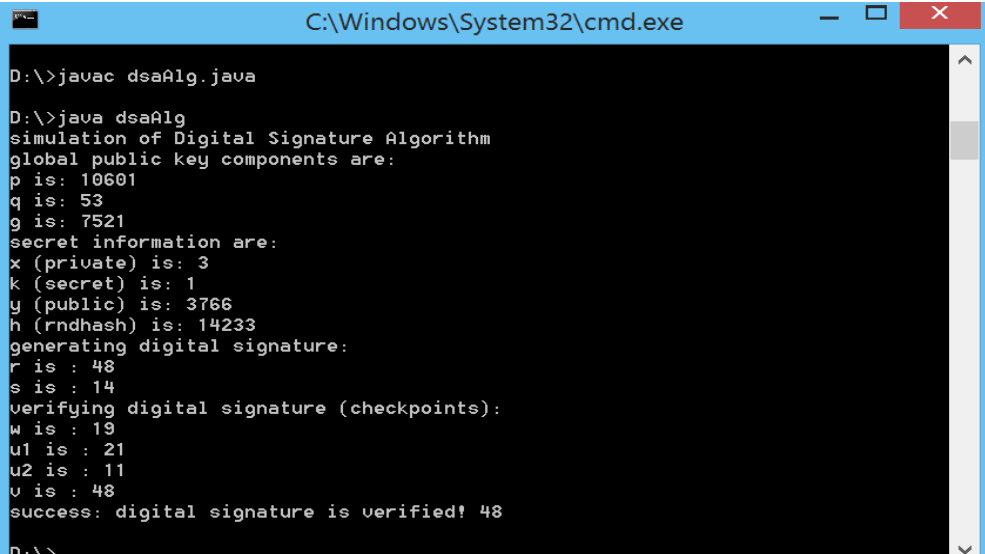
    /* finds a generator mod p */
    public static BigInteger getGen(BigInteger p, BigInteger q, Random r)
    {
        BigInteger h = new BigInteger(p.bitLength(), r);
        h = h.mod(p);
        return h.modPow((p.subtract(one)).divide(q), p);
    }

    public static void main (String[] args) throws java.lang.Exception
    {
        Random randObj = new Random();

        /* establish the global public key components */
        BigInteger p = getNextPrime("10601");
        /* approximate prime */
        BigInteger q = findQ(p.subtract(one));
```

```
BigInteger g = getGen(p,q,randObj);
/* public key components */
System.out.println("simulation of Digital Signature Algorithm");
System.out.println("global public key components are:");
System.out.println("p is: " + p);
System.out.println("q is: " + q);
System.out.println("g is: " + g);
/* find the private key */
BigInteger x = new BigInteger(q.bitLength(), randObj);
x = x.mod(q);
/* corresponding public key */
BigInteger y = g.modPow(x,p);
/* random value message */
BigInteger k = new BigInteger(q.bitLength(), randObj);
k = k.mod(q);
/* randomly generated hash value and digital signature */
BigInteger r = (g.modPow(k,p)).mod(q);
BigInteger hashVal = new BigInteger(p.bitLength(), randObj);
BigInteger kInv = k.modInverse(q);
BigInteger s = kInv.multiply(hashVal.add(x.multiply(r)));
s = s.mod(q);
/* secret information */
System.out.println("secret information are:");
System.out.println("x (private) is: " + x);
System.out.println("k (secret) is: " + k);
System.out.println("y (public) is: " + y);
System.out.println("h (rndhash) is: " + hashVal);
System.out.println("generating digital signature:");
System.out.println("r is : " + r);
System.out.println("s is : " + s);
/* verify the digital signature */
BigInteger w = s.modInverse(q);
BigInteger u1 = (hashVal.multiply(w)).mod(q);
BigInteger u2 = (r.multiply(w)).mod(q);
```

```
BigInteger v = (g.modPow(u1,p)).multiply(y.modPow(u2,p));
v = (v.mod(p)).mod(q);
System.out.println("verifying digital signature (checkpoints):");
System.out.println("w is : " + w);
System.out.println("u1 is : " + u1);
System.out.println("u2 is : " + u2);
System.out.println("v is : " + v);
if (v.equals(r))
{
    System.out.println("success: digital signature is verified! " + r);
}
else
{
    System.out.println("error: incorrect digital signature");
}
}
```



```
C:\Windows\System32\cmd.exe
D:\>javac dsaAlg.java
D:\>java dsaAlg
simulation of Digital Signature Algorithm
global public key components are:
p is: 10601
q is: 53
g is: 7521
secret information are:
x (private) is: 3
k (secret) is: 1
y (public) is: 3766
h (rndhash) is: 14233
generating digital signature:
r is : 48
s is : 14
verifying digital signature (checkpoints):
w is : 19
u1 is : 21
u2 is : 11
v is : 48
success: digital signature is verified! 48
D:\>
```

CONCLUSION: Thus we have successfully implemented Digital Signature scheme using DSA

Experiment 6:

AIM: Study the use of network reconnaissance tools like WHOIS, dig, traceroute, nslookup to gather information about networks and domain registrars

THEORY:

❖ **WHOIS**

Whois and fwhois are extremely simple but useful tools that query particular “whois” databases for information about a domain name or an IP address.

Whois servers are databases that are maintained by domain name authorities around the world. A whois database can contain a plethora of information, but typically it contains such information as location, contact information, and IP address ranges for every domain name under its authority.

Whois tools are usually installed by default on most Unix distributions. For Windows systems, many of the port scanning tools discussed in Chapter 4 include whois querying capabilities, such as NetScanTools and SuperScan.

❖ **DIG**

The command dig is a tool for querying DNS nameservers for information about host addresses, mail exchanges, nameservers, and related information. This tool can be used from any Linux (Unix) or Macintosh OS X operating system. The most typical use of dig is to simply query a single host.

Dig displays a the request and an what the DNS server sends in response to the request. In this case, we used the default options for dig, which simply looks up the A record for a domain.

SYNTAX

\$ Dig domain_name

❖ **Traceroute**

The `tracert` command is a Command Prompt command that's used to show several details about the path that a packet takes from the computer or device you're on to whatever destination you specify.

You might also sometimes see the `tracert` command referred to as the *trace route command* or *traceroute command*.

Syntax

tracert [-d] [-h *MaxHops*] [-w *TimeOut*] [-4] [-6] *target* [/?]

Traceroute Command Options

Item	Description
-d	This option prevents <code>tracert</code> from resolving IP addresses to hostnames, often resulting in much faster results.
-h <i>MaxHops</i>	This <code>tracert</code> option specifies the maximum number of hops in the search for the <i>target</i> . If you do not specify <i>MaxHops</i> , and a <i>target</i> has not been found by 30 hops, <code>tracert</code> will stop looking.
-w <i>TimeOut</i>	You can specify the time, in milliseconds, to allow each reply before timeout using this <code>tracert</code> option.
-4	This option forces <code>tracert</code> to use IPv4 only.
-6	This option forces <code>tracert</code> to use IPv6 only.
<i>target</i>	This is the destination, either an IP address or hostname.
/?	Use the <code>help</code> switch with the <code>tracert</code> command to show detailed help about the command's several options.

❖ NSLOOKUP

. Specifies one or more **Nslookup** subcommands as a command-line option. Looks up information for Host using the current default **DNS** server, if no other server is specified. To look up a computer not in the current **DNS** domain, append a period to the name.

Nslookup Syntax

The syntax for `Nslookup` (in the command prompt mode) is as follows:

nslookup [-opt ...] [{Host| [Server]}]

Parameter	Description
-opt	Specifies one or more Nslookup subcommands as a command-line option.
Host	Looks up information for Host using the current default DNS server, if no other server is specified. To look up a computer not in the current DNS domain, append a period to the name.
Server	Specifies to use this server as the DNS name server. If you don't specify a server, the default DNS server is used.

CONCLUSION:

Thus, we have studied different network reconnaissance tools.

Experiment 7:

AIM: Download and install nmap. Use it with different options to scan open ports, perform OS fingerprinting, do a ping scan, tcp port scan and udp port scan.

THEORY:

Nmap (Network Mapper) is a security scanner originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host and then analyzes the responses. Unlike many simple port scanners that just send packets at some predefined constant rate, Nmap accounts for the network conditions (latency fluctuations, network congestion, the target interference with the scan) during the run. Also, owing to the large and active user community providing feedback and contributing to its features, Nmap has been able to extend its discovery capabilities beyond simply figuring out whether a host is up or down and which ports are open and closed; it can determine the operating system of the target, names and versions of the listening services, estimated uptime, type of device, and presence of a firewall.

Nmap features include:

- Host Discovery – Identifying hosts on a network. For example, listing the hosts which respond to pings or have a particular port open.
- Port Scanning – Enumerating the open ports on one or more target hosts.
- Version Detection – Interrogating listening network services listening on remote devices to determine the application name and version number.
- OS Detection – Remotely determining the operating system and some hardware characteristics of network devices.

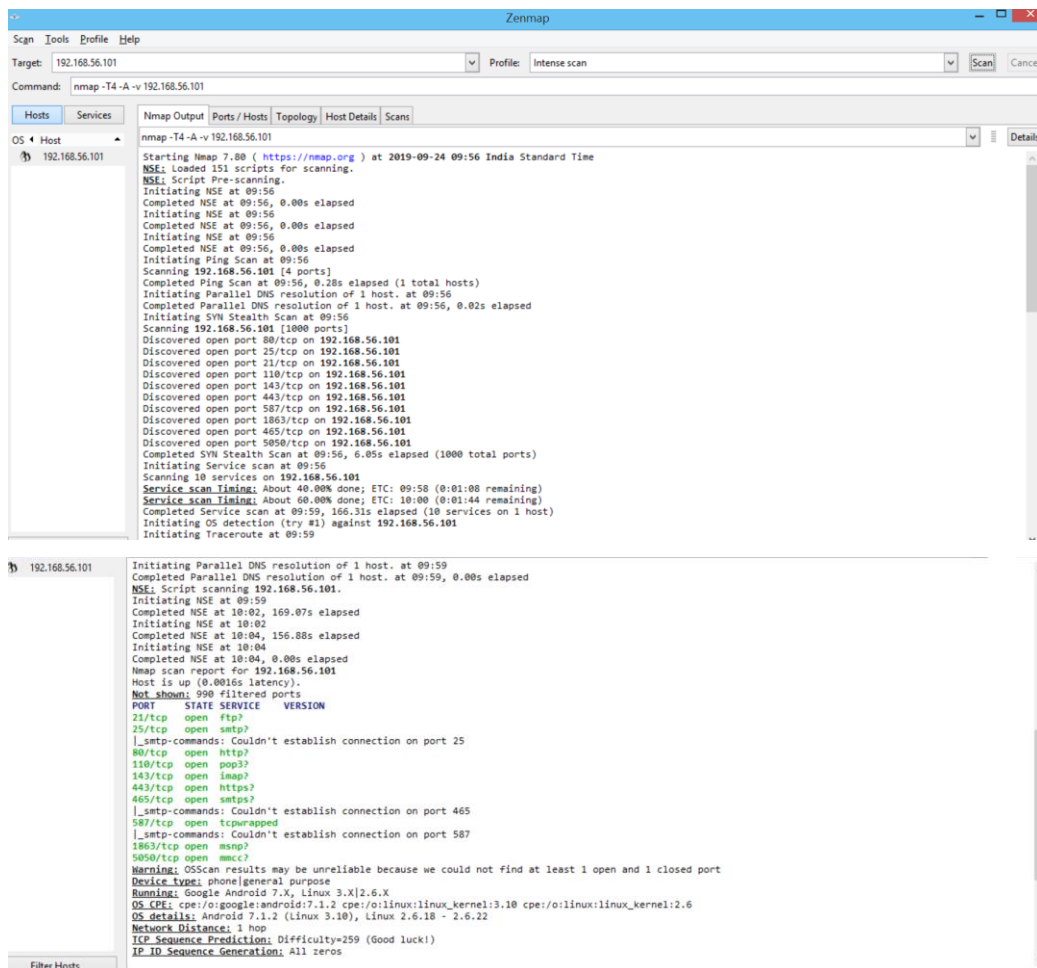
Basic commands working in Nmap:

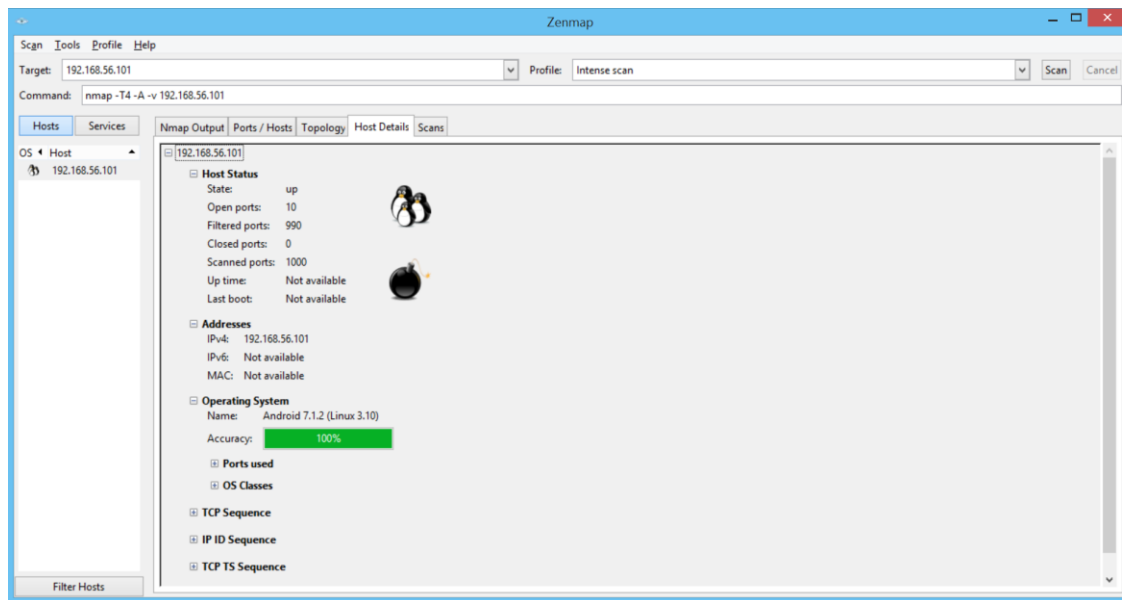
- For target specifications: nmap <target's URL or IP with spaces between them>
- For OS detection: nmap -O <target-host's URL or IP>
- For version detection: nmap -sV <target-host's URL or IP>

SYN scan is the default and most popular scan option for good reasons. It can be performed quickly, scanning thousands of ports per second on a fast network not hampered by restrictive firewalls. It is also relatively unobtrusive and stealthy since it never completes TCP connections

Algorithm\Implementation Steps\Installation Steps:

1. Download Nmap from www.nmap.org and install the Nmap Software with WinPcap Driver utility.
2. Execute the Nmap-Zenmap GUI tool from Program Menu or Desktop Icon
3. Type the Target Machine IP Address(ie.Guest OS or any website Address)
4. Perform the profiles shown in the utility.



[illegible]

Thus, we have studied different options to scan ports in Nmap

Experiment 8:

AIM: Study of packet sniffer tools wireshark

Objectives: To observe the performance in promiscuous & non-promiscuous mode & to find the packets based on different filters.

Outcomes: The learner will be able to:-

- Identify different packets moving in/out of network using packet sniffer for network analysis.
- Understand professional, ethical, legal, security and social issues and responsibilities. Also will be able to analyze the local and global impact of computing on individuals, organizations, and society.
- Match the industry requirements in the domains of Database management, Programming and Networking with the required management skills.

Hardware / Software Required: Wireshark, Ethereal and tcpdump.

Theory:

Wireshark, a network analysis tool formerly known as Ethereal, captures packets in real time and display them in human-readable format. Wireshark includes filters, color-coding and other features that let you dig deep into network traffic and inspect individual packets.

Applications:

- Network administrators use it to troubleshoot network problems
- Network security engineers use it to examine security problems
- Developers use it to debug protocol implementations
- People use it to learn network protocol internals beside these examples can be helpful in many other situations too.

Features:

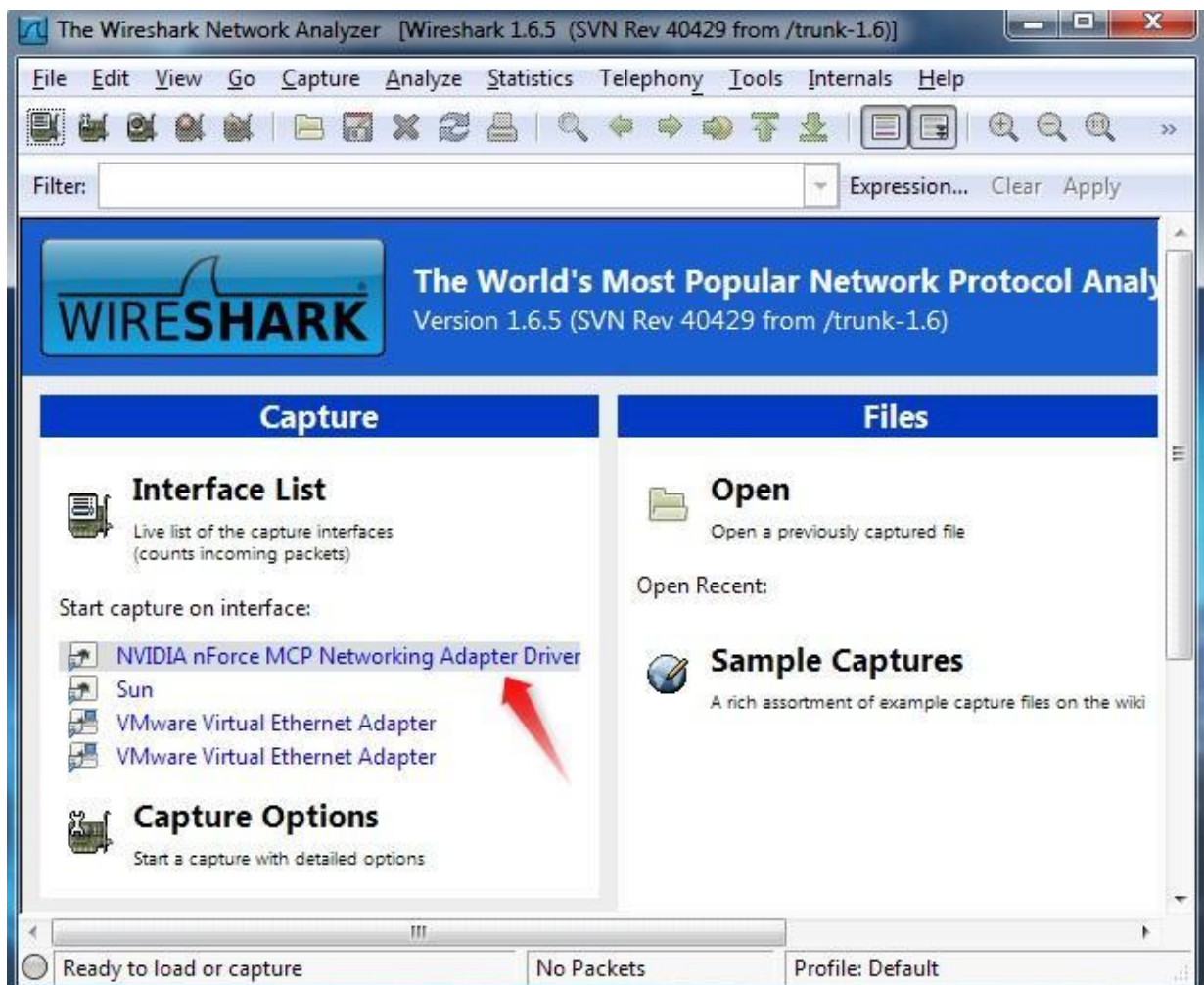
The following are some of the many features wireshark provides:

- Available for UNIX and Windows.
- Capture live packet data from a network interface.
- Open files containing packet data captured with tcpdump/WinDump, Wireshark, and a number of other packet capture programs.
- Import packets from text files containing hex dumps of packet data.
- Display packets with very detailed protocol information.
- Export some or all packets in a number of capture file formats.

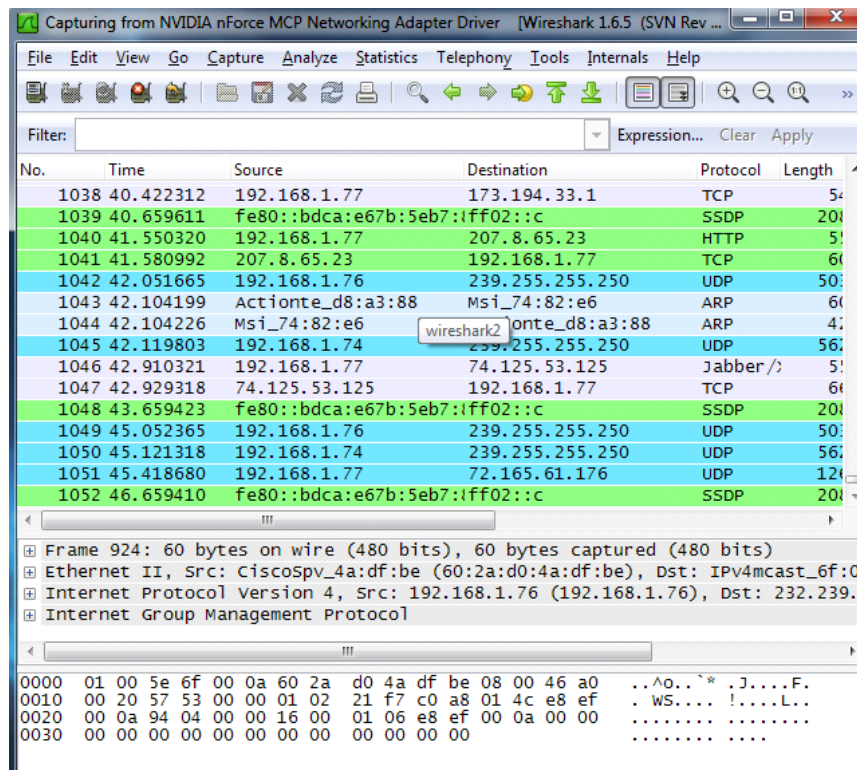
- Filter packets on many criteria.
- Search for packets on many criteria.
- Colorize packet display based on filters.
- Create various statistics.

Capturing Packets:

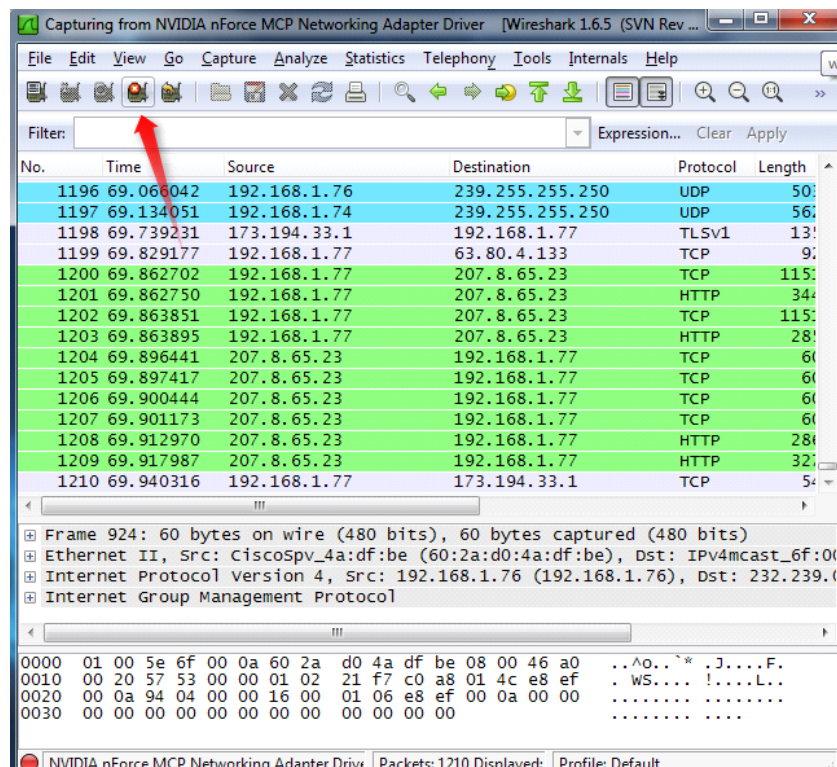
After downloading and installing wireshark, you can launch it and click the name of an interface under Interface List to start capturing packets on that interface. For example, if you want to capture traffic on the wireless network, click your wireless interface. You can configure advanced features by clicking Capture Options.



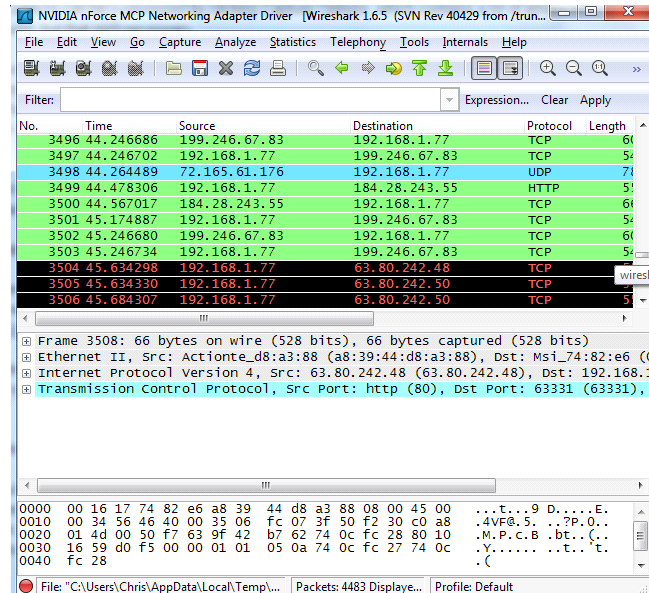
As soon as you click the interface's name, you'll see the packets start to appear in real time. Wireshark captures each packet sent to or from your system. If you're capturing on a wireless interface and have promiscuous mode enabled in your capture options, you'll also see other the other packets on the network.



Click the stop capture button near the top left corner of the window when you want to stop capturing traffic.

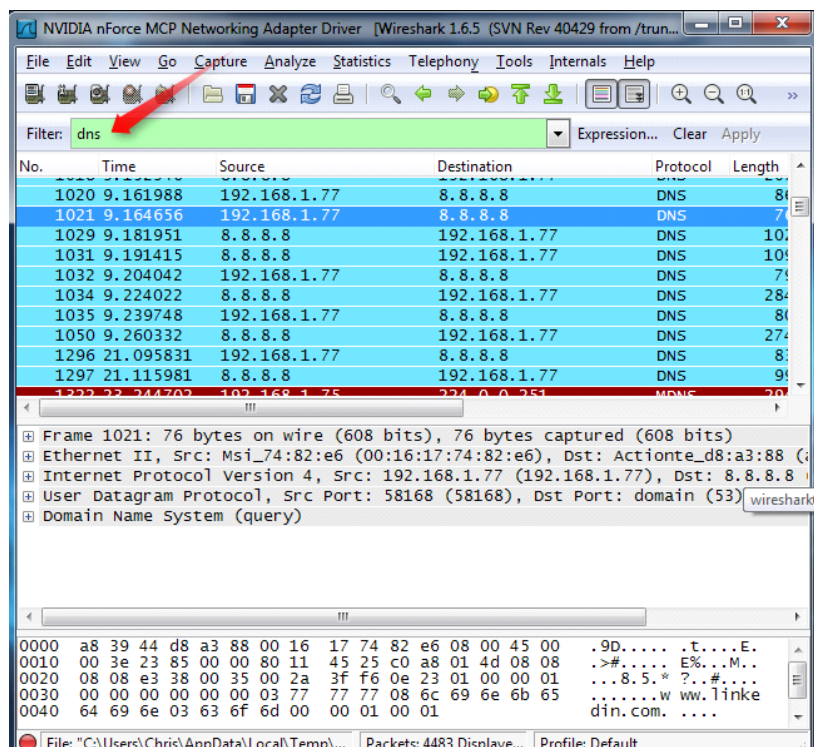


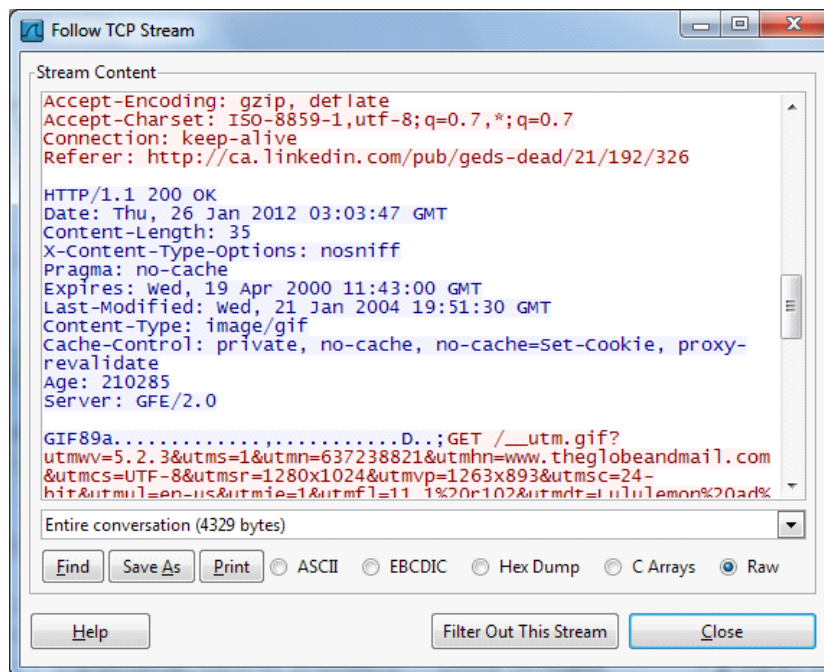
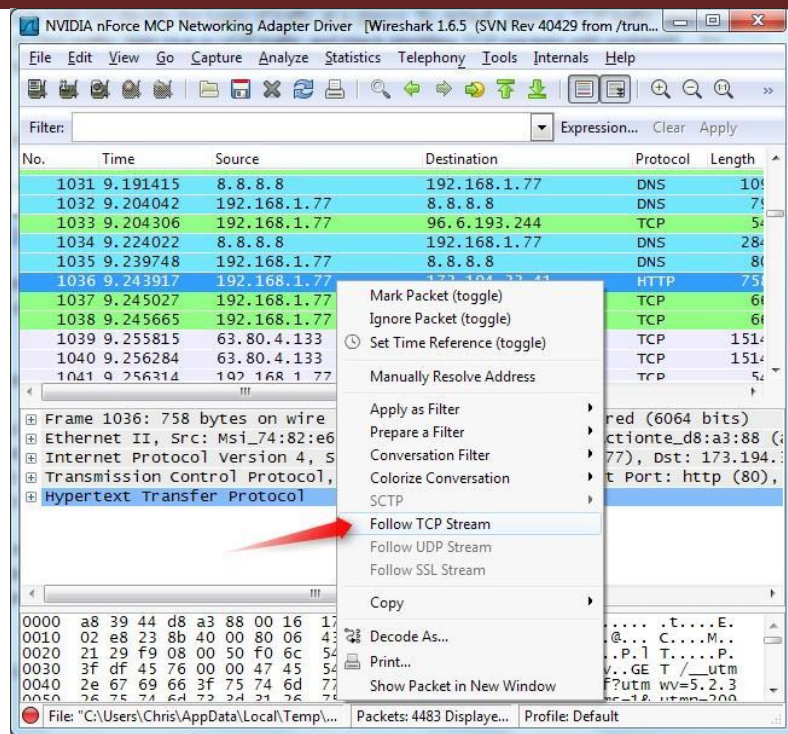
Wireshark uses colors to help you identify the types of traffic at a glance. By default, green is TCP traffic, dark blue is DNS traffic, light blue is UDP traffic, and black identifies TCP packets with problems — for example, they could have been delivered out-of-order.



Filtering Packets:

If you're trying to inspect something specific, such as the traffic a program sends when phoning home, it helps to close down all other applications using the network so you can narrow down the traffic. Still, you'll likely have a large amount of packets to sift through.





Conclusion:

In this experiment we analyze various packet sniffing tools that monitor network traffic transmitted between legitimate users or in the network. The packet sniffer is network monitoring tool. It is opted for network monitoring, traffic analysis, troubleshooting, Packet grapping, message, protocol analysis, penetration testing and many other purposes

Experiment 9:

AIM: Set up Snort and study the logs.

THEORY:

Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intrusion detection systems fall into two basic categories:

- Signature-based intrusion detection systems
- Anomaly detection systems.

Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts. Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS. Usually an intrusion detection system captures data from the network and applies its rules to that data or detects anomalies in it. Snort is primarily a rule-based IDS, however input plug-ins are present to detect anomalies in protocol headers.

SNORT TOOL:

Snort is based on libpcap (for library packet capture), a tool that is widely used in TCP/IP traffic sniffers and analyzers. Through protocol analysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealth port scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window. Snort is currently the most popular free network intrusion detection software. The advantages of Snort are numerous. According to the snort web site, "It can perform protocol analysis, content searching/matching, and can be used to detect a variety of attacks and probes, such as buffer overflow, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more" (Caswell). One of the advantages of Snort is its ease of configuration. Rules are very flexible, easily written, and easily inserted into the rule base. If a new exploit or attack is found a rule for the attack can be added to the rule base in a matter of seconds. Another advantage of snort is that it allows for raw packet data analysis.

SNORT can be configured to run in three modes:

1. Sniffer mode
2. Packet Logger mode
3. Network Intrusion Detection System mode

1. Sniffer mode

Snort -v Print out the TCP/IP packets header on the screen

Snort -vd show the TCP/IP ICMP header with application data in transmit

2. Packet Logger mode

snort -dev -l c:\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:\log -h ipaddress/24: This rule tells 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:\log -b This is binary mode logs everything into a single file.

3. Network Intrusion Detection System mode

snort -d c:\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:\log -c snort.conf This will configure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf.

PROCEDURE:

STEP-1: Sniffer mode snort -v Print out the TCP/IP packets header on the screen.

STEP-2: Snort -vd Show the TCP/IP ICMP header with application data in transit.

STEP-3: Packet Logger mode snort -dev -l c:\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.

STEP-4: `snort -dev -l c:\log -h ipaddress/24` This rule tells snort that you want to print out the data link and TCP/IP headers as well as application data into the log directory.

STEP-5: `snort -l c:\log -b` this binary mode logs everything into a single file.

STEP-6: Network Intrusion Detection System mode `snort -d c:\log -h ipaddress/24 -c snort.conf` This is a configuration file that applies rule to each packet to decide it an action based upon the rule type in the file.

STEP-7: `snort -d -h ip address/24 -l c:\log -c snort.conf` This will configure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf.

STEP-8: Download SNORT from snort.org. Install snort with or without database support.

STEP-9: Select all the components and Click Next. Install and Close.

STEP-10: Skip the WinPcap driver installation.

STEP-11: Add the path variable in windows environment variable by selecting new classpath.

STEP-12: Create a path variable and point it at snort.exe variable name path and variable value `c:\snort\bin`.

STEP-13: Click OK button and then close all dialog boxes. Open command prompt and type the following commands:

```

Administrator: C:\Windows\system32\cmd.exe
=====
Run time for packet processing was 703.909000 seconds
Snort processed 1409 packets.
Snort ran for 0 days 0 hours 11 minutes 43 seconds
Pkts/min:      128
Pkts/sec:       2
=====
Packet I/O Totals:
Received:      1411
Analyzed:      1409 ( 99.858%)
Dropped:       0 ( 0.000%)
Filtered:      0 ( 0.000%)
Outstanding:   2 ( 0.142%)
Injected:      0
=====
Breakdown by protocol (includes rebuilt packets):
Eth:          1409 (100.000%)
  VLAN:       0 ( 0.000%)
  IP4:        927 ( 65.791%)
  Frag:       0 ( 0.000%)
  ICMP:       0 ( 0.000%)
  UDP:       892 ( 63.307%)
  TCP:       473 ( 33.570%)
  IP6:       0 ( 0.000%)
  IP6 Ext:    0 ( 0.000%)
  IP6 Opts:   0 ( 0.000%)
  Frag6:     0 ( 0.000%)
  ICMP6:     0 ( 0.000%)
  UDP6:     0 ( 0.000%)
  TCP6:     0 ( 0.000%)
  Teredo:    0 ( 0.000%)
  ICMP-IP:   0 ( 0.000%)
  EAPOL:     0 ( 0.000%)
  IP4/IP4:   0 ( 0.000%)
  IP4/IP6:   0 ( 0.000%)
  IP6/IP4:   0 ( 0.000%)
  IP6/IP6:   0 ( 0.000%)
  GRE:       0 ( 0.000%)
  GRE Eth:   0 ( 0.000%)
  GRE VLAN:  0 ( 0.000%)
  GRE IP4:   0 ( 0.000%)
  GRE IP6:   0 ( 0.000%)
  GRE IP6 Ext: 0 ( 0.000%)
  GRE PPTP:  0 ( 0.000%)
  GRE ARP:   0 ( 0.000%)
  GRE IPX:   0 ( 0.000%)
  GRE Loop:  0 ( 0.000%)
  MPLS:     0 ( 0.000%)
  ARP:       9 ( 0.639%)
  IPX:       0 ( 0.000%)
  Eth Loop:  0 ( 0.000%)
  Eth Disc:  0 ( 0.000%)
  IP4 Disc:  0 ( 0.000%)
  IP6 Disc:  0 ( 0.000%)
  TCP Disc:  0 ( 0.000%)
  UDP Disc:  0 ( 0.000%)
  ICMP Disc: 0 ( 0.000%)
  All Discard: 0 ( 0.000%)
  Other:     35 ( 2.484%)
  Bad Chk Sum: 0 ( 0.000%)
  Bad TTL:   0 ( 0.000%)
  S5 G 1:   0 ( 0.000%)
  S5 G 2:   0 ( 0.000%)
  Total:    1409
=====
Snort exiting
C:\Snort\bin>

```

Conclusion: The IDS tool Snort was successfully set up and the logs were studied.