



**K.RAMAKRISHNAN
COLLEGE OF ENGINEERING**

Permanently Affiliated to Anna University Chennai and Approved by AICTE, New Delhi
ISO 9001:2015 Certified Institution, Accredited with 'A' grade by NAAC
Samayapuram, Trichy, Tamilnadu



(An Autonomous Institution)

IT8761 SECURITY LABORATORY

(As per the Anna University Syllabus Regulation 2017 for CSE)

LAB MANUAL

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**VII SEMESTER
(2018-2022)**

Prepared by			Verified by,
Dr R Sridevi, Professor/CSE			
Mrs.S.Saranya, Assistant Professor/CSE			

**<< SYLLABUS AS IN REGULATION
R2017 >>**

IT8761 SECURITY LABORATORY

LIST OF EXPERIMENTS

EX.NO	TITLE
1	PERFORM ENCRYPTION, DECRYPTION USING THE FOLLOWING SUBSTITUTION TECHNIQUES
1(a)	IMPLEMENTATION OF CAESAR CIPHER
1(b)	IMPLEMENTATION OF PLAY FAIR CIPHER
1(c)	IMPLEMENTATION OF HILL CIPHER
1(d)	IMPLEMENTATION OF VIGENER CIPHER
2	PERFORM ENCRYPTION AND DECRYPTION USING FOLLOWING TRANSPOSITION TECHNIQUES
2(a)	IMPLEMENTATION OF RAIL FENCE CIPHER
2(b)	IMPLEMENTATION OF ROW & COLUMN TRANSFORMATION
3	IMPLEMENTATION OF DATA ENCRYPTION STANDARD(DES)
4	IMPLEMENTATION OF AES ALGORITHM
5	IMPLEMENTATION OF RSA ALGORITHM
6	IMPLEMENTATION OF DIFFIEE HELLMAN KEY EXCHANGE ALGORITHM
7	IMPLEMENTATION OF SECURE HASH FUNCTION (SHA)
8	IMPLEMENTATION OF DIGITAL SIGNATURE SCHEME
9	DEMONSTRATE INTRUSION DETECTION SYSTEM (IDS) USING SNORT
10	AUTOMATED ATTACK AND PENETRATION TOOLS EXPLORING N-STALKER, A VULNERABILITY ASSESSMENT TOOL
11	DEFEATING MALWARE – ROOTKIT HUNTER
CONTENT BEYOND THE STLLABUS	
12	IMPLEMENTATION OF TRIPLE DES
13	IMPLEMENTATION OF BLOWFISH ALGORITHM

SYLLABUS

IT8761

SECURITY LABORATORY

L T P C

0 0 4 2

OBJECTIVES:

The student should be made to:

- To learn different cipher techniques
- To implement the algorithms DES, RSA, MD5, SHA-1
- To use network security tools and vulnerability assessment tools

LIST OF EXPERIMENTS:

1. Perform encryption, decryption using the following substitution techniques
 - i. Ceaser cipher
 - ii. Playfair cipher
 - iii. Hill Cipher
 - iv. Vigenere cipher
2. Perform encryption and decryption using following transposition techniques
Rail fence - Row & Column Transformation
3. Apply DES algorithm for practical applications.
4. Apply AES algorithm for practical applications
5. Implement RSA Algorithm using HTML and JavaScript
6. Implement the Diffie-Hellman Key Exchange algorithm for a given problem.
7. Calculate the message digest of a text using the SHA-1 algorithm
8. Implement the SIGNATURE SCHEME - Digital Signature Standard.
9. Demonstrate intrusion detection system (ids) using any tool eg. Snort or any other s/w.
10. Automated Attack and Penetration Tools Exploring N-Stalker, a Vulnerability Assessment Tool
11. Defeating Malware - Building Trojans, Rootkit Hunter

TOTAL: 60 PERIODS

LIST OF EQUIPMENT FOR A BATCH OF 30 STUDENTS:

SOFTWARE:

- C / C++ / Java or equivalent compiler
- GnuPG, Snort, N-Stalker or Equivalent

HARDWARE:

- Standalone desktops - 30 Nos. (or) Server supporting 30 terminals or more.

OUTCOMES:

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

- Develop code for classical Encryption Techniques to solve the problems.
- Build cryptosystems by applying symmetric and public key encryption algorithms.
- Construct code for authentication algorithms.
- Develop a signature scheme using Digital signature standard.
- Demonstrate the network security system using open source tools

AIM:

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

ALGORITHM:

1. Create and initialize a string ALPHABET that holds the alphabet characters. The index position of the string represents the numeric representation for the corresponding characters in the string ALPHABET.
2. Read the input plain text to be encrypted and also the Caesar cipher key an integer between 0 and 25.
3. Encrypt the plain text using the Caesar cipher key and the ALPHABET string.
 - a. For every character in the plain text
 - i. Search the ALPHABET string for the character and assign the numeric representation of the character (plainnumeric) as the index position of the character in the ALPHABET string.
 - ii. Perform encryption using
$$\text{ciphernumeric} = (\text{plainnumeric} + \text{Caesar cipher key}) \bmod 26$$
 - iii. Use ciphernumeric as the index position and get the corresponding character from the ALPHABET string as the equivalent cipher text character for the plain text character
 - b. Print the equivalent cipher text
4. Decrypt the cipher text using the Caesar cipher key and the ALPHABET string.
 - a. For every character in the cipher text
 - i. Search the ALPHABET string for the character and assign the numeric representation of the character (ciphernumeric) as the index position of the character in the ALPHABET string.
 - ii. Perform decryption using
$$\text{Plainnumeric} = (\text{ciphernumeric} - \text{Caesar cipher key}) \bmod 26,$$
if $\text{plainnumeric} < 0$, $\text{plainnumeric} = \text{plainnumeric} + 26$
 - iii. Use plainnumeric as the index position and get the corresponding character from the ALPHABET string as the equivalent plain text character for the cipher text character
 - b. Print the equivalent plain text
5. Stop

1(a) : IMPLEMENTATION OF CAESAR CIPHER

PROGRAM:

```
import java.util.*;
import java.io.*;

public class Caesercipher
{
    public static final String ALPHABET = "abcdefghijklmnopqrstuvwxyz";
    public static String encrypt(String ptext, int cserkey)
    {
        String ctext = "";
        for (int i = 0; i < ptext.length(); i++)
        {
            int plainnumeric = ALPHABET.indexOf(ptext.charAt(i));
            int ciphernumeric = (plainnumeric+cserkey) % 26;
            char cipherchar = ALPHABET.charAt(ciphernumeric);
            ctext += cipherchar;
        }
        return ctext;
    }

    public static String decrypt(String ctext, int cserkey)
    {
        String ptext = "";
        for (int i = 0; i < ctext.length(); i++)
        {
            int ciphernumeric = ALPHABET.indexOf(ctext.charAt(i));
            int plainnumeric= (ciphernumeric-cserkey) % 26;
            if (plainnumeric < 0)
            {
                plainnumeric = ALPHABET.length() + plainnumeric;
            }
            char plainchar = ALPHABET.charAt(plainnumeric);
            ptext += plainchar;
        }
        return ptext;
    }

    public static void main(String[] args)
    throws IOException
    {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter the PLAIN TEXT for Encryption: ");
        String plaintext = new String();
        String ciphertext = new String();
        String key;
        int cserkey;
```

```
plaintext = br.readLine();
System.out.println("Enter the CAESERKEY between 0 and 25:");
key = br.readLine();
cserkey = Integer.parseInt(key);

System.out.println("ENCRYPTION");
ciphertext = encrypt(plaintext,cserkey);
System.out.println("CIPHER TEXT :"+ ciphertext);

System.out.println("DECRYPTION");
plaintext = decrypt(ciphertext,cserkey);
System.out.println("PLAIN TEXT :" + plaintext);
}
}
```


OUTPUT:

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

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

Enter the PLAIN TEXT for Encryption:

information

Enter the CAESERKEY between 0 and 25:

7

ENCRYPTION

CIPHER TEXT :pumvythapvu

DECRYPTION

PLAIN TEXT :information

RESULT:

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

AIM:

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

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.
 - e. 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).
 - f. 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).
 - g. 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.
 - h. Append the letters referred from the key table using the steps 6.a, 6.b and 6.c to generate Plain text.
7. Stop

1(b). IMPLEMENTATION OF PLAY FAIR CIPHER

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

CHARACTER PAIR :H F

TRANSLATED TEXT :G O

CHARACTER PAIR :Y F

TRANSLATED TEXT :X O

CHARACTER PAIR :L D

TRANSLATED TEXT :D M

CHARACTER PAIR :R I

TRANSLATED TEXT :O R

CHARACTER PAIR :F N

TRANSLATED TEXT :N I

CHARACTER PAIR :F E

TRANSLATED TEXT :N G

TEXT :GOODMORNING

PLAIN TEXT :GOODMORNING

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

RESULT:

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

AIM

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

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. Plain Text = (Cipher Text * Key^{-1}) 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

1 (c). IMPLEMENTATION OF HILL CIPHER

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++)
0(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

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

TECHNOLOGY

HILL CIPHER ENCRYPTION

PLAIN TEXT MATRIX

19 4 2

7 13 14

11 14 6

24 23 23

CIPHER TEXT MATRIX

21 9 9

4 17 2

25 9 21

1 10 0

CIPHER TEXT :VJJERCZJVBKA

HILL CIPHER DECRYPTION

CIPHER TEXT MATRIX

21 9 9

4 17 2

25 9 21

1 10 0

PLAIN TEXT MATRIX

19 4 2

7 13 14

11 14 6

24 23 23

PLAIN TEXT :TECHNOLOGY

RESULT:

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

AIM:

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

ALGORITHM :

1. Vigenere table 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.
2. The Key is repeated so that the total length is equal to that of the plaintext. In this way, each letter in the plaintext is shifted by the alphabet number of the corresponding letter in the key.
3. At different points in the encryption process, the cipher uses a different alphabet from one of the rows used.
4. **Encryption:** The the plaintext(P) and key(K) are added modulo 26.
$$E_i = (P_i + K_i) \bmod 26$$
5. **Decryption:** Subtract the key from the Encrypted (Cipher) text and perform modulo 26 to arrive back at the original, plaintext value
$$D_i = (E_i - K_i + 26) \bmod 26$$
6. Stop

1(d) . IMPLEMENTATION OF VIGENERE CIPHER

PROGRAM:

```
import java.util.*;
import java.io.*;

public class Vigenerecipher
{
    public static String key = new String();
    public String extndkey;
    public String plaintxt, ciphertxt;
    public String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    // String plain,cipher;
    int row, flag=0, decrypt=0;

    public String keyextnsn(String ptxt,String keytxt)
    {
        int i,j=0,n;
        String nukey="";

        for(i=0;i<ptxt.length();i++)
        {
            nukey += keytxt.charAt(j);
            j++;
            if (j == keytxt.length())
                j=0;
        }
        return nukey;
    }

    public int valueofchar(char x)
    {
        int i,pos=0;

        for(i=0;i<26;i++)
        {
            if ( x == ALPHABET.charAt(i))
            {
                pos = i;
                break;
            }
        }
        return pos;
    }

    public char charofvalue(int y)
    {
        int i;
        char ch;
```

```

        ch = ALPHABET.charAt(y);
        return ch;
    }

    public String vcencryption(String txt)
    {
        int i,j,p=0,k=0,tmp1=0;
        char tmp;
        String ctxt="";

        extndkey = keyextnsn(txt,key);
        System.out.println("VIGENERE ENCRYPTION");
        System.out.println("PLAIN TEXT : " + txt);
        System.out.println("VIGENERE KEY : " + extndkey);
        for(i=0;i<txt.length();i++)
        {
            p = valueofchar(txt.charAt(i));
            k = valueofchar(extndkey.charAt(i));

            //      System.out.println("p : " + p + " k : " + k);
            tmp1 = ( p + k )%26;
            tmp = charofvalue(tmp1);
            //      System.out.println("tmp1 : " + tmp1 + "tmp : " + tmp);

            ctxt += tmp;
            //      System.out.println("CTXT : " + ctxt);
        }
        return ctxt;
    }

    public String vdecryption(String txt)
    {
        int i,c=0,k=0,tmp1=0;
        char ch;
        String ptxt="";

        System.out.println("VIGENERE DECRYPTION");
        System.out.println("CIPHER TEXT : " + txt);
        System.out.println("VIGENERE KEY : " + extndkey);
        for(i=0;i<txt.length();i++)
        {
            c = valueofchar(txt.charAt(i));
            k = valueofchar(extndkey.charAt(i));

            tmp1 = ( c - k + 26 )%26;
            ch = charofvalue(tmp1);

```

```
        ptxt += ch;
    }
    return ptxt;
}

public static void main(String[] args)
{
    Vigenerecipher vc = new Vigenerecipher();
    Scanner sc = new Scanner(System.in);

    System.out.println("ENTER KEY");
    key = sc.next();
    String text = new String();
    System.out.println("Enter PLAIN TEXT");
    text = sc.next();

    String ciphertext = new String();
    ciphertext = vc.vcencryption(text);
    System.out.println();
    System.out.println("CIPHER TEXT : " + ciphertext);
    System.out.println();

    String plaintext = new String();
    plaintext = vc.vcdecryption(ciphertext);
    System.out.println();
    System.out.println("PLAIN TEXT : " + plaintext);

    sc.close();
}}
```

OUTPUT:

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

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

ENTER KEY

DECEPTIVE

Enter PLAIN TEXT

WEAREDISCOVEREDSAVEYOURSELF

VIGENERE ENCRYPTION

PLAIN TEXT : WEAREDISCOVEREDSAVEYOURSELF

VIGENERE KEY : DECEPTIVEDECEPTIVEDECEPTIVE

CIPHER TEXT :ZICVTWQNGRZGVTWAVZHCQYGLMGJ

VIGENERE DECRYPTION

CIPHER TEXT : ZICVTWQNGRZGVTWAVZHCQYGLMGJ

VIGENERE KEY : DECEPTIVEDECEPTIVEDECEPTIVE

PLAIN TEXT :WEAREDISCOVEREDSAVEYOURSELF

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

ENTER KEY

LEMON

Enter PLAIN TEXT

ATTACKATDAWN

VIGENERE ENCRYPTION

PLAIN TEXT : ATTACKATDAWN

VIGENERE KEY : LEMONLEMONLE

CIPHER TEXT :LXFOPVEFRNHR

VIGENERE DECRYPTION

CIPHER TEXT : LXFOPVEFRNHR

VIGENERE KEY : LEMONLEMONLE

PLAIN TEXT :ATTACKATDAWN

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

RESULT:

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

AIM:

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

ALGORITHM:

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.

1. Generate numerical key from the word key by the characters of the word in alphabetical order.
2. Encryption
 - i. The plain text is written in the matrix form, where the column of the matrix is number of characters in the word key and row of the matrix is to accommodate the characters of the plain text and the space left after the plain text characters in the last row is filled with any character. (eg. x or z)
 - ii. The cipher text is generated by reading the characters column by column in the order specified in the numerical key.
3. Decryption
 - i. The characters in the cipher text are filled in the matrix of same order used for encryption, but in the order specified in the key. The characters from cipher text equal to the number of rows in matrix are taken and filled in the matrix column based on the order specified in the key
 - ii. The plain text is generated from cipher text by reading the characters from the matrix row by row.
4. Stop

2 (a) . IMPLEMENTATION OF RAIL FENCE CIPHER

PROGRAM:

```
import java.util.*;
import java.io.*;

public class Railfence
{
    public static int key[] = new int[8];
    public char mat[][] = new char[10][8];
    public char pmat[][] = new char[10][8];
    public char cmat[][] = new char[10][8];
    String plain="";
    String cipher="";
    int rows=0, col;

    public String rfencryption(String text1)
    {
        int i,j,len,ch,k,p=0;
        String enctxt="";
        String text = "";
        len = text1.length();

        for(i=0;i<len;i++)
            text += text1.charAt(i);

        if (( len % 7 ) != 0)
        {
            rows = ( len / 7 ) + 1;
            ch = len % 7;
            for (i=0;i<(7-ch);i++)
                text += 'X';

        }
        else
            rows = len / 7;
        k=0;

        for(i=1;i<=rows;i++)
        {
```



```

        for(j=1;j<=7;j++)
            mat[i][j] = text.charAt(k++);
    }

    for(i=1;i<=rows;i++)
    {
        for(j=1;j<=7;j++)
            System.out.print(mat[i][j] + " ");
        System.out.println();
    }

    k = 1;
    j = 1;
    while ( k <= 7 )
    {
        for(p=0;p<7;p++)
        {
            if ( k == key[p] )
            {
                j=p+1;
                k++;
                break;
            }
        }
        for(i=1;i<=rows;i++)
            enctxt+=mat[i][j];
    }

    System.out.println(enctxt);
    return enctxt;
}

public String rfdecryption(String txt,int plength)
{
    int i,j=1,len,k=1,p,q=0;
    String dectxt="";
    String ptext="";

    while (k<=7)
    {
        for(p=0;p<7;p++)
        {
            if (key[p] == k)
            {
                j = p+1;
                k++;
            }
        }
    }

```

```

                break;
            }
        }
        for(i=1;i<=rows;i++)
            cmat[i][j] = txt.charAt(q++);
    }

    for(i=1;i<=rows;i++)
    {
        for(j=1;j<=7;j++)
            System.out.print(cmat[i][j] + " ");
        System.out.println();
    }

    for(i=1;i<=rows;i++)
    {
        for(j=1;j<=7;j++)
            dectxt += cmat[i][j];
    }

    len = dectxt.length();
    if (plength < len)
    {
        for(i=0;i<plength;i++)
            ptext += dectxt.charAt(i);
    }

    return ptext;
}

public static void main(String[] args) throws IOException
{
    int i=0;
    int k;
    String c;
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    Railfence rf = new Railfence();
    Scanner sc = new Scanner(System.in);

    System.out.println("Enter key");
    for(i=0;i<7;i++)
    {
        c = br.readLine();
        key[i] = Integer.parseInt(c);
    }
}

```

```
        for(i=0;i<7;i++)
            System.out.print(key[i] + " ");

        String plain = new String();
        System.out.println("Enter PLAIN TEXT");
        plain = sc.next();
        k=plain.length();

        System.out.println(plain);
        String ctext = new String();
        ctext = rf.rfencryption(plain);
        System.out.println();
        System.out.println("CIPHER TEXT :" + ctext);
        System.out.println();

        String plaintext = new String();
        plaintext = rf.rfdecryption(ctext,k);
        System.out.println();
        System.out.println("PLAIN TEXT :" + plaintext);

        sc.close();
    }
}
```

OUTPUT:

```
C:\Program Files\Java\jdk1.8.0_71\bin>javac Railfence.java
C:\Program Files\Java\jdk1.8.0_71\bin>java Railfence
Enter key
AUTHOR
A U T H O R
A H O R T U
KEY : NUMERICAL REPRESENTATION
0 3 4 5 2 1
Enter PLAIN TEXT ATTACKPOSTPONEDUNTILTWOAM
ATTACKPOSTPONEDUNTILTWOAM

A T T A C K
P O S T P O
N E D U N T
I L T W O A
M X X X X X
APNIMKOTAXCPNOXTOELXTSDTXATUWX
CIPHER TEXT :APNIMKOTAXCPNOXTOELXTSDTXATUWX

A T T A C K
P O S T P O
N E D U N T
I L T W O A
M X X X X X

PLAIN TEXT :ATTACKPOSTPONEDUNTILTWOAM

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

RESULT:

Thus the program for Railfence cipher was executed and verified successfully.

AIM:

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

ALGORITHM:

1. Consider the plain text hello world, and let us apply the simple columnar transposition technique as shown below

h	e	l	l
o	w	o	r
l	d		

2. The plain text characters are placed horizontally and the cipher text is created with vertical format as: **holewdlo lr**.
3. Now, the receiver has to use the same table to decrypt the cipher text to plain text.

2(b) . IMPLEMENTATION OF RAIL FENCE CIPHER

PROGRAM:

```
import java.util.*;
class TransCipher {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the plain text");
        String pl = sc.nextLine();
        sc.close();
        String s = "";
        int start = 0;
        for (int i = 0; i < pl.length(); i++) {
            if (pl.charAt(i) == ' ') {
                s = s + pl.substring(start, i);
                start = i + 1;
            }
        }
        s = s + pl.substring(start);
        System.out.print(s);
        System.out.println();

        // end of space deletion

        int k = s.length();
        int l = 0;
        int col = 4;
        int row = s.length() / col;
        char ch[][] = new char[row][col];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                if (l < k) {

                    ch[i][j] = s.charAt(l);

                    l++;
                } else {
                    ch[i][j] = '#';
                }
            }
        }
        // arranged in matrix

        char trans[][] = new char[col][row];
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                trans[j][i] = ch[i][j];
            }
        }
    }
}
```

```
for (int i = 0; i < col; i++) {  
    for (int j = 0; j < row; j++) {  
        System.out.print(trans[i][j]);  
    }  
    // display  
    System.out.println();  
}  
}
```

OUTPUT:

Enter the plain text
Security Lab
SecurityLab
Sreictuy

RESULT:

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

AIM:

To develop a program to implement Data Encryption Standard for encryption and decryption.

ALGORITHM:

1. Process the key.

i. Get a 64-bit key from the user.

ii. Calculate the key schedule.

1. Perform the following permutation on the 64-bit key. The parity bits are discarded, reducing the key to 56 bits. Bit 1 of the permuted block is bit 57 of the original key, bit 2 is bit 49, and so on with bit 56 being bit 4 of the original key.

2. Split the permuted key into two halves. The first 28 bits are called C[0] and the last 28 bits are called D[0].

3. Calculate the 16 subkeys. Start with $i = 1$.

1. Perform one or two circular left shifts on both C[i-1] and D[i-1] to get C[i] and D[i], respectively. The number of shifts per iteration are given in the table below.

Iteration # 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Left Shifts 1 1 2 2 2 2 2 2 1 2 2 2 2 2 2 1

2. Permute the concatenation C[i]D[i] as indicated below. This will yield K[i], which is 48 bits long.

3. Loop back to 1.ii.c.1 until K[16] has been calculated.

2 Process a 64-bit data block.

i. Get a 64-bit data block. If the block is shorter than 64 bits, it should be padded as appropriate for the application.

ii. Perform the initial permutation on the data block.

iii. Split the block into two halves. The first 32 bits are called L[0], and the last 32 bits are called R[0].

iv. Apply the 16 subkeys to the data block. Start with $i = 1$.

a. Expand the 32-bit R[i-1] into 48 bits according to the bit-selection function Expansion (E)

b. Exclusive-or E(R[i-1]) with K[i].

c. Break E(R[i-1]) xor K[i] into eight 6-bit blocks. Bits 1-6 are B[1], bits 7-12 are B[2], and so on with bits 43-48 being B[8].

d. Substitute the values found in the S-boxes for all B[j]. Start with $j = 1$. All values in the S-boxes should be considered 4 bits wide.

i. Take the 1st and 6th bits of B[j] together as a 2-bit value (call it m) indicating the row in S[j] to look in for the substitution.

ii. Take the 2nd through 5th bits of B[j] together as a 4-bit value (call it n) indicating the column in S[j] to find the substitution.

iii. Replace B[j] with S[j][m][n].

iv. Loop back to 2.iv.d.i until all 8 blocks have been replaced.

e. Permute the concatenation of B[1] through B[8]

f. Exclusive-or the resulting value with L[i-1]. Thus, all together, your

$R[i] = L[i-1] \text{ xor } P(S[1](B[1])...S[8](B[8]))$, where $B[j]$ is a 6-bit block of $E(R[i-1])$ xor $K[i]$. (The function for $R[i]$ is written as, $R[i] = L[i-1] \text{ xor } f(R[i-1], K[i])$.)

g. $L[i] = R[i-1]$.

h. Loop back to 2.iv.a until $K[16]$ has been applied.

v. Perform the final permutation on the block $R[16]L[16]$.

3. Decryption : Use the keys $K[i]$ in reverse order. That is, instead of applying $K[1]$ for the first iteration, apply $K[16]$, and then $K[15]$ for the second, on down to $K[1]$

3. IMPLEMENTATION OF DATA ENCRYPTION STANDARD (DES)

PROGRAM:

DES :-

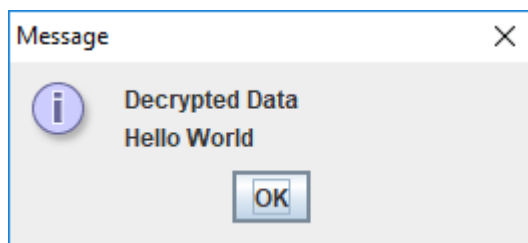
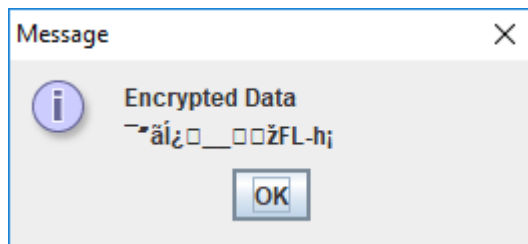
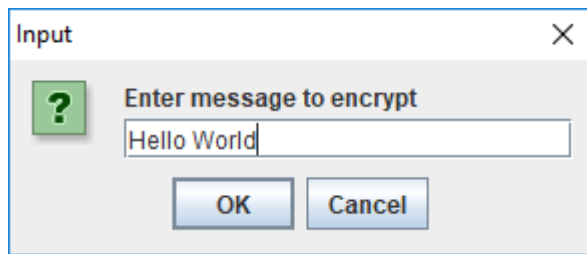
```
import javax.swing.*;
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random ;
class DES {
byte[] skey = new byte[1000];
String skeyString;
static byte[] raw;
String 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:



RESULT:

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

AIM:

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

ALGORITHM:

1. Create and initialize a string ALPHABET that holds the alphabet characters. The index position of the string represents the numeric representation for the corresponding characters in the string ALPHABET.
2. Read the input plain text to be encrypted and also the Caesar cipher key an integer between 0 and 25.
3. Encrypt the plain text using the Caesar cipher key and the ALPHABET string.
 - a. For every character in the plain text
 - i. Search the ALPHABET string for the character and assign the numeric representation of the character (plainnumeric) as the index position of the character in the ALPHABET string.
 - ii. Perform encryption using
$$\text{ciphernumeric} = (\text{plainnumeric} + \text{Caesar cipher key}) \bmod 26$$
 - iii. Use ciphernumeric as the index position and get the corresponding character from the ALPHABET string as the equivalent cipher text character for the plain text character
 - b. Print the equivalent cipher text
4. Decrypt the cipher text using the Caesar cipher key and the ALPHABET string.
 - a. For every character in the cipher text
 - i. Search the ALPHABET string for the character and assign the numeric representation of the character (ciphernumeric) as the index position of the character in the ALPHABET string.
 - ii. Perform decryption using
$$\text{Plainnumeric} = (\text{ciphernumeric} - \text{Caesar cipher key}) \bmod 26,$$
if $\text{plainnumeric} < 0$, $\text{plainnumeric} = \text{plainnumeric} + 26$
 - iii. Use plainnumeric as the index position and get the corresponding character from the ALPHABET string as the equivalent plain text character for the cipher text character
 - b. Print the equivalent plain text
5. Stop

4. IMPLEMENTATION OF AES ALGORITHM

PROGRAM:

```
package com.includehelp.stringsample;

import java.util.Base64;
import java.util.Scanner;
import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

/**
 * Program to Encrypt/Decrypt String Using AES 128 bit Encryption Algorithm
 */
public class EncryptDecryptString
{
    private static final String encryptionKey      = "ABCDEFGHJKLMNOP";
    private static final String characterEncoding   = "UTF-8";
    private static final String cipherTransformation = "AES/CBC/PKCS5PADDING";
    private static final String aesEncryptionAlgorithem = "AES";

    /**
     * Method for Encrypt Plain String Data
     * @param plainText
     * @return encryptedText
     */
    public static String encrypt(String plainText)
    {
        String encryptedText = "";
        try
        {
            Cipher cipher = Cipher.getInstance(cipherTransformation);
            byte[] key     = encryptionKey.getBytes(characterEncoding);
            SecretKeySpec secretKey = new SecretKeySpec(key, aesEncryptionAlgorithem);
            IvParameterSpec ivparameterspec = new IvParameterSpec(key);
            cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivparameterspec);
            byte[] cipherText = cipher.doFinal(plainText.getBytes("UTF8"));
            Base64.Encoder encoder = Base64.getEncoder();
            encryptedText = encoder.encodeToString(cipherText);
        } catch (Exception E)
        {
            System.err.println("Encrypt Exception : "+E.getMessage());
        }
        return encryptedText;
    }
}
```

```

public static String decrypt(String encryptedText)
{
    String decryptedText = "";
    try
    {
        Cipher cipher = Cipher.getInstance(cipherTransformation);
        byte[] key = encryptionKey.getBytes(characterEncoding);
        SecretKeySpec secretKey = new SecretKeySpec(key, aesEncryptionAlgorithem);
        IvParameterSpec ivparameterspec = new IvParameterSpec(key);
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivparameterspec);
        Base64.Decoder decoder = Base64.getDecoder();
        byte[] cipherText = decoder.decode(encryptedText.getBytes("UTF8"));
        decryptedText = new String(cipher.doFinal(cipherText), "UTF-8");
    } catch (Exception E)
    {
        System.err.println("decrypt Exception : "+E.getMessage());
    }
    return decryptedText;
}

public static void main(String[] args)
{
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter String : ");
    String plainString = sc.nextLine();

    String encryptStr = encrypt(plainString);
    String decryptStr = decrypt(encryptStr);

    System.out.println("Plain String : "+plainString);
    System.out.println("Encrypt String : "+encryptStr);
    System.out.println("Decrypt String : "+decryptStr);
} }

```


OUTPUT:

Enter String : Hello World

Plain String : Hello World

Encrypt String : IMfL/ifkuvkZwG/v2bn6Bw==

Decrypt String : Hello World

RESULT:

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

AIM:

Develop a program to implement RSA algorithm for encryption and decryption.

ALGORITHM:**1. Key Generation**

- i. Choose two distinct prime numbers p and q .
- ii. Find n such that $n = pq$, n will be used as the modulus for both the public and private keys.
- iii. Find the totient of n , $\phi(n) = (p-1)(q-1)$
- iv. Choose an e such that $1 < e < \phi(n)$, and such that e and $\phi(n)$ share no divisors other than 1 (e and $\phi(n)$ are relatively prime). e is kept as the public key exponent
- v. Determine d (using modular arithmetic) which satisfies the congruence relation

$$de \equiv 1 \pmod{\phi(n)}.$$

The public key has modulus n and the public (or encryption) exponent

e . The private key has modulus n and the private (or decryption) exponent d , which is kept secret.

2. Encryption

$$c \equiv m^e \pmod{n}.$$

3. Decryption:

$$m \equiv c^d \pmod{n}.$$

4. Stop.

5. IMPLEMENTATION OF RSA ALGORITHM

PROGRAM:

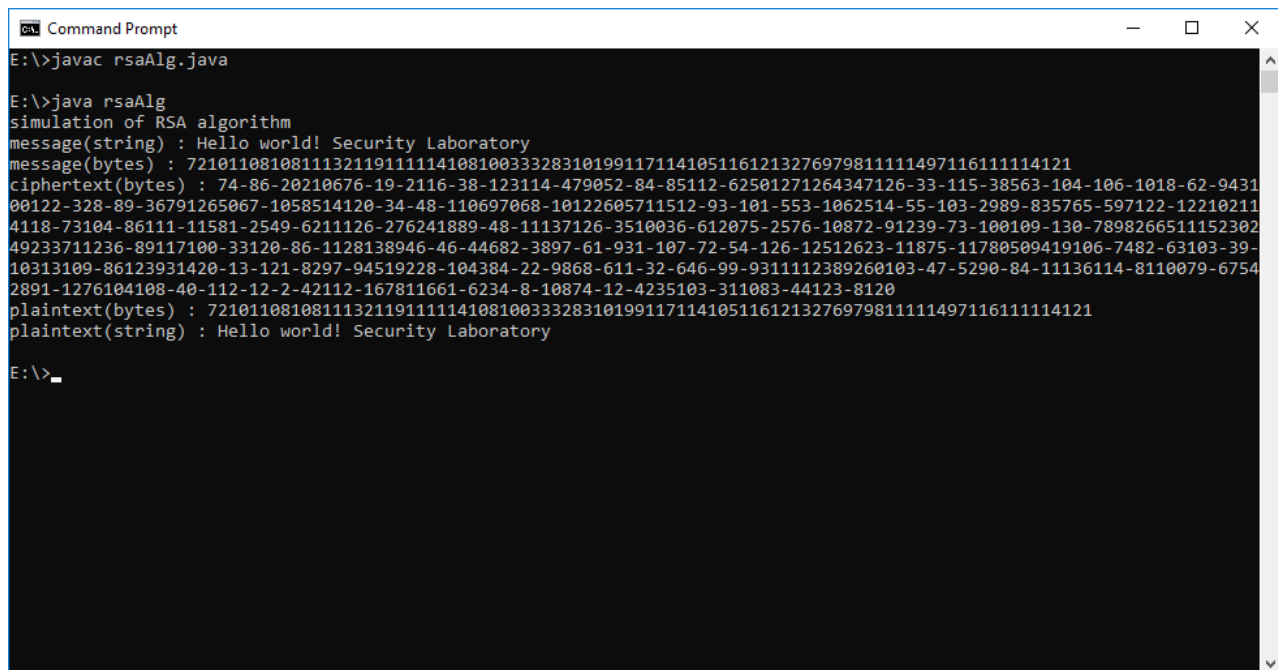
```
import java.math.BigInteger;
import java.util.Random;
import java.io.*;

class rsaAlg
{
    private BigInteger p, q, n, phi, e, d; /* public key components */
    private int bitLen = 1024;
    private int blkSz = 256; /* block size in bytes */
    private Random rand;
    /* convert bytes to string */
    private static String bytesToString(byte[] encrypted)
    {
        String str = "";
        for (byte b : encrypted)
        {
            str += Byte.toString(b);
        }
        return str;
    }
    /* encrypt message */
    public byte[] encrypt(byte[] msg)
    {
        return (new BigInteger(msg)).modPow(e, n).toByteArray();
    }
    /* decrypt message */
    public byte[] decrypt(byte[] msg)
    {
        return (new BigInteger(msg)).modPow(d, n).toByteArray();
    }
    /* calculate public key components p, q, n, phi, e, d */
    public rsaAlg()
    {
        rand = new Random();
        p = BigInteger.probablePrime(bitLen, rand);
        q = BigInteger.probablePrime(bitLen, rand);
        n = p.multiply(q);
        phi = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));
        e = BigInteger.probablePrime(bitLen/2, rand);
        while (phi.gcd(e).compareTo(BigInteger.ONE) > 0 &&
            e.compareTo(phi) < 0){
            e.add(BigInteger.ONE);}
        d = e.modInverse(phi);
    }
    public rsaAlg (BigInteger e, BigInteger d, BigInteger n) {
```

```
this.e = e;
this.d = d;
this.n = n;
}
public static void main (String[] args) throws java.lang.Exception
{
rsaAlg rsaObj = new rsaAlg();
String msg = "Hello world! Security Laboratory";
System.out.println("simulation of RSA algorithm");
System.out.println("message(string) : " + msg);
System.out.println("message(bytes) : " +
bytesToString(msg.getBytes()));
/* encrypt test message */
byte[] ciphertext = rsaObj.encrypt(msg.getBytes());
System.out.println("ciphertext(bytes) : " + bytesToString(ciphertext));
/* decrypt ciphertext */
byte[] plaintext = rsaObj.decrypt(ciphertext);

System.out.println("plaintext(bytes) : " + bytesToString(plaintext));
System.out.println("plaintext(string) : " + new String(plaintext));
}
}
```

OUTPUT:



```
Command Prompt
E:\>javac rsaAlg.java

E:\>java rsaAlg
simulation of RSA algorithm
message(string) : Hello world! Security Laboratory
message(bytes) : 7210110810811132119111114108100333283101991171141051161213276979811111497116111114121
ciphertext(bytes) : 74-86-20210676-19-2116-38-123114-479052-84-85112-62501271264347126-33-115-38563-104-106-1018-62-9431
00122-328-89-36791265067-1058514120-34-48-110697068-10122605711512-93-101-553-1062514-55-103-2989-835765-597122-12210211
4118-73104-86111-11581-2549-6211126-276241889-48-11137126-3510036-612075-2576-10872-91239-73-100109-130-7898266511152302
49233711236-89117100-33120-86-1128138946-46-44682-3897-61-931-107-72-54-126-12512623-11875-11780509419106-7482-63103-39-
10313109-86123931420-13-121-8297-94519228-104384-22-9868-611-32-646-99-9311112389260103-47-5290-84-11136114-8110079-6754
2891-1276104108-40-112-12-2-42112-167811661-6234-8-10874-12-4235103-311083-44123-8120
plaintext(bytes) : 7210110810811132119111114108100333283101991171141051161213276979811111497116111114121
plaintext(string) : Hello world! Security Laboratory

E:\>
```

RESULT:

Thus the program for implementation of RSA algorithm was executed and verified successfully.

AIM:

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

ALGORITHM:

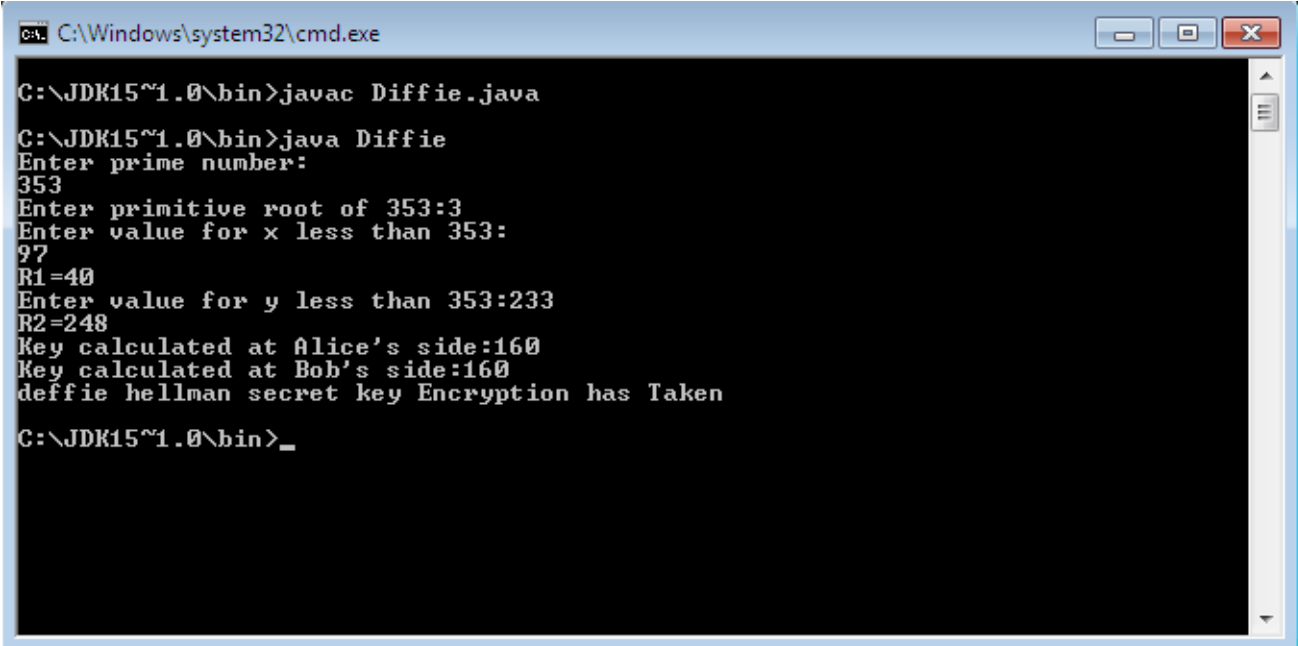
1. Global Public Elements:
Let q be a prime number and α where $\alpha < q$ and α is a primitive root of q .
2. User A Key Generation:
 - i. Select private X_A where $X_A < q$
 - ii. Calculate public Y_A where $Y_A = \alpha^{X_A} \bmod q$
3. User B Key Generation:
 - i. Select private X_B where $X_B < q$
 - ii. Calculate public Y_B where $Y_B = \alpha^{X_B} \bmod q$
4. Calculation of Secret Key by User
A: $K = (Y_B)^{X_A} \bmod q$
5. Calculation of Secret Key by User
B: $K = (Y_A)^{X_B} \bmod q$

6. IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE ALGORITHM

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 Sender's side:"+k1);
        BigInteger k2=R1.modPow(y,p);
        System.out.println("Key calculated at Receiver's side:"+k2);
        System.out.println("diffie hellman secret key Encryption has Taken");
    }
}
```

OUTPUT:



```
C:\Windows\system32\cmd.exe

C:\JDK15~1.0\bin>javac Diffie.java
C:\JDK15~1.0\bin>java Diffie
Enter prime number:
353
Enter primitive root of 353:3
Enter value for x less than 353:
97
R1=40
Enter value for y less than 353:233
R2=248
Key calculated at Alice's side:160
Key calculated at Bob's side:160
deffie hellman secret key Encryption has Taken
C:\JDK15~1.0\bin>_
```

RESULT:

Thus the program to implement Diffie-Hellman Key Exchange algorithm was developed and executed successfully.

AIM:

To develop a program to implement Secure Hash Algorithm (SHA-1)

ALGORITHM:

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 less than an even multiple of 512.
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.
3. Prepare Processing Functions: SHA1 requires 80 processing functions defined as:

$$f(t;B,C,D) = (B \text{ AND } C) \text{ OR } ((\text{NOT } B) \text{ AND } D) \quad (0 \leq t \leq 19)$$

$$f(t;B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (20 \leq t \leq 39)$$

$$f(t;B,C,D) = (B \text{ AND } C) \text{ OR } (B \text{ AND } D) \text{ OR } (C \text{ AND } D) \quad (40 \leq t \leq 59)$$

$$f(t;B,C,D) = B \text{ XOR } C \text{ XOR } D \quad (60 \leq t \leq 79)$$
4. Prepare Processing Constants: SHA1 requires 80 processing constant words defined as:

$$K(t) = 0x5A827999 \quad (0 \leq t \leq 19)$$

$$K(t) = 0x6ED9EBA1 \quad (20 \leq t \leq 39)$$

$$K(t) = 0x8F1BBCDC \quad (40 \leq t \leq 59)$$

$$K(t) = 0xCA62C1D6 \quad (60 \leq t \leq 79)$$
5. Initialize Buffers: SHA1 requires 160 bits or 5 buffers of words (32 bits):

$$H0 = 0x67452301 \quad H1 = 0xEFCDAB89$$

$$H2 = 0x98BADCFE \quad H3 = 0x10325476$$

$$H4 = 0xC3D2E1F0$$
6. Processing Message in 512-bit blocks (L blocks in total message)
 - i. This is the main task of SHA1 algorithm which loops through the padded and appended message in 512-bit blocks.
 - ii. Input and predefined functions: $M[1, 2, \dots, L]$: Blocks of the padded and appended message $f(0;B,C,D), f(1;B,C,D), \dots, f(79;B,C,D)$: 80 Processing Functions

$$K(0), K(1), \dots, K(79)$$
: 80 Processing Constant Words

$$H0, H1, H2, H3, H4, H5$$
: 5 Word buffers with initial values
7. For loop on $k = 1$ to L
 1. $(W(0), W(1), \dots, W(15)) = M[k]$ /* Divide $M[k]$ into 16 words */
8. For $t = 16$ to 79 do:

$$W(t) = (W(t-3) \text{ XOR } W(t-8) \text{ XOR } W(t-14) \text{ XOR } W(t-16)) \lll 1$$

$$A = H0, B = H1, C = H2, D = H3, E = H4$$
 For $t = 0$ to 79 do:

$$\text{TEMP} = A \lll 5 + f(t;B,C,D) + E + W(t) + K(t)$$

$$E = D, D = C, C = B \lll 30, B = A, A = \text{TEMP}$$
 End of for loop

$$H0 = H0 + A, H1 = H1 + B, H2 = H2 + C, H3 = H3 + D, H4 = H4 + E$$
 End of for loop

7. IMPLEMENTATION OF SECURE HASH FUNCTION (SHA)

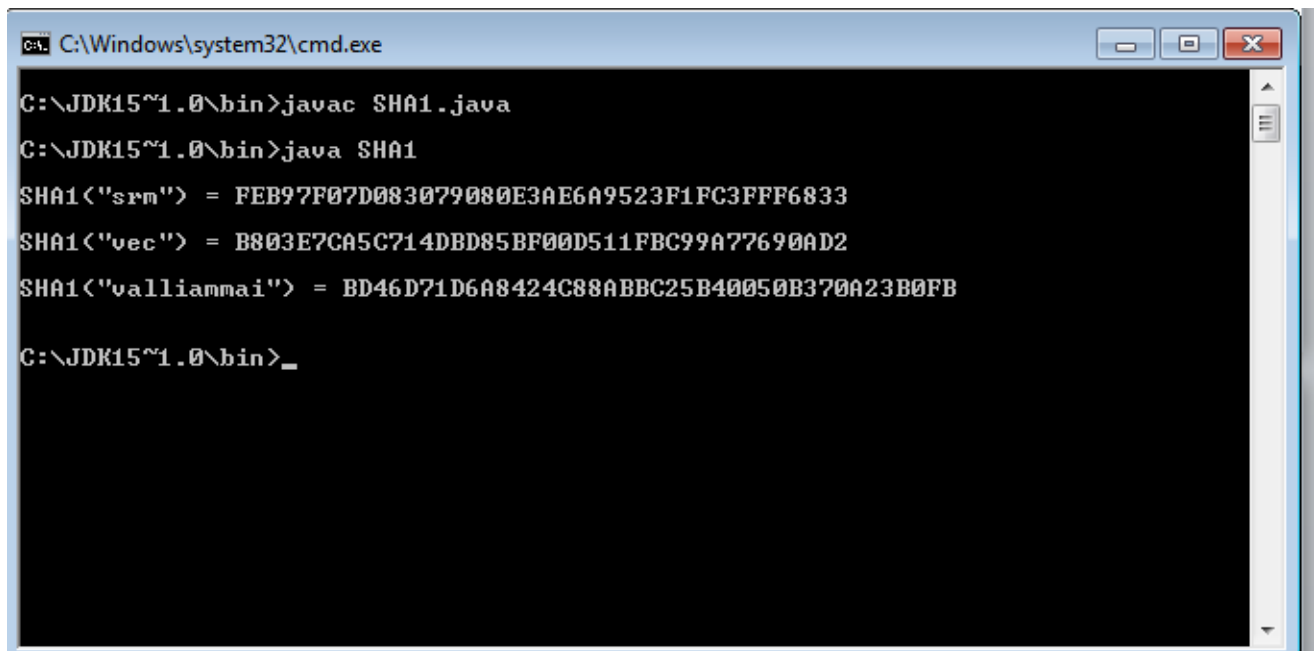
PROGRAM:

```
import java.security.*;
public class SHA1 {
public static void main(String[] a) {
try {
MessageDigest md = MessageDigest.getInstance("SHA1");
String input = "srm";
md.update(input.getBytes());
byte[] output = md.digest();
System.out.println();
System.out.println("SHA1(\""+input+"") = " +bytesToHex(output));
input = "vec";
md.update(input.getBytes());
output = md.digest();

System.out.println();
System.out.println("SHA1(\""+input+"") = " +bytesToHex(output));
input = "valliammai";
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:



```
C:\Windows\system32\cmd.exe

C:\JDK15~1.0\bin>javac SHA1.java
C:\JDK15~1.0\bin>java SHA1
SHA1("srm") = FEB97F07D083079080E3AE6A9523F1FC3FFF6833
SHA1("vec") = B803E7CA5C714DBD85BF00D511FBC99A77690AD2
SHA1("valliammai") = BD46D71D6A8424C88ABBC25B40050B370A23B0FB

C:\JDK15~1.0\bin>_
```

RESULT:

Thus the program to implement Secure Hash Algorithm was developed and executed successfully.

AIM:

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

ALGORITHM:

1. Choose a prime number q , which is called the prime divisor.
2. Choose another prime number p , such that $p-1 \bmod q = 0$. p is called the prime modulus.
3. Choose an integer g , such that $1 < g < p$, $g^{q-1} \bmod p = 1$ and $g = h^{((p-1)/q)} \bmod p$. q is also called g 's multiplicative order modulo p .
4. Choose an integer, such that $0 < x < q$.
5. Compute y as $g^x \bmod p$.
6. Package the public key as $\{p, q, g, y\}$, $\{p, q, g, x\}$.
7. Generate the message digest h , using a hash algorithm like SHA1.
8. Generate a random number k , such that $0 < k < q$.
9. Compute r as $(g^k \bmod p) \bmod q$. If $r = 0$, select a different k .
10. Compute i , such that $k \cdot i \bmod q = 1$. i is called the modular multiplicative inverse of k modulo q .
11. Compute $s = i \cdot (h + r \cdot x) \bmod q$. If $s = 0$, select a different k .
12. Package the digital signature as $\{r, s\}$.
13. Generate the message digest h , using the same hash algorithm.
14. Compute w , such that $s \cdot w \bmod q = 1$. w is called the modular multiplicative inverse of s modulo q .
15. Compute $u_1 = h \cdot w \bmod q$. Compute $u_2 = r \cdot w \bmod q$.
16. Compute $v = (((g^{u_1}) \cdot (y^{u_2})) \bmod p) \bmod q$.
17. If $v == r$, the digital signature is valid.

8. IMPLEMENTATION OF DIGITAL SIGNATURE SCHEME

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("10600"); /* approximate prime */
BigInteger q = findQ(p.subtract(one));
BigInteger g = getGen(p,q,randObj);

/* public key components */
System.out.println("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);

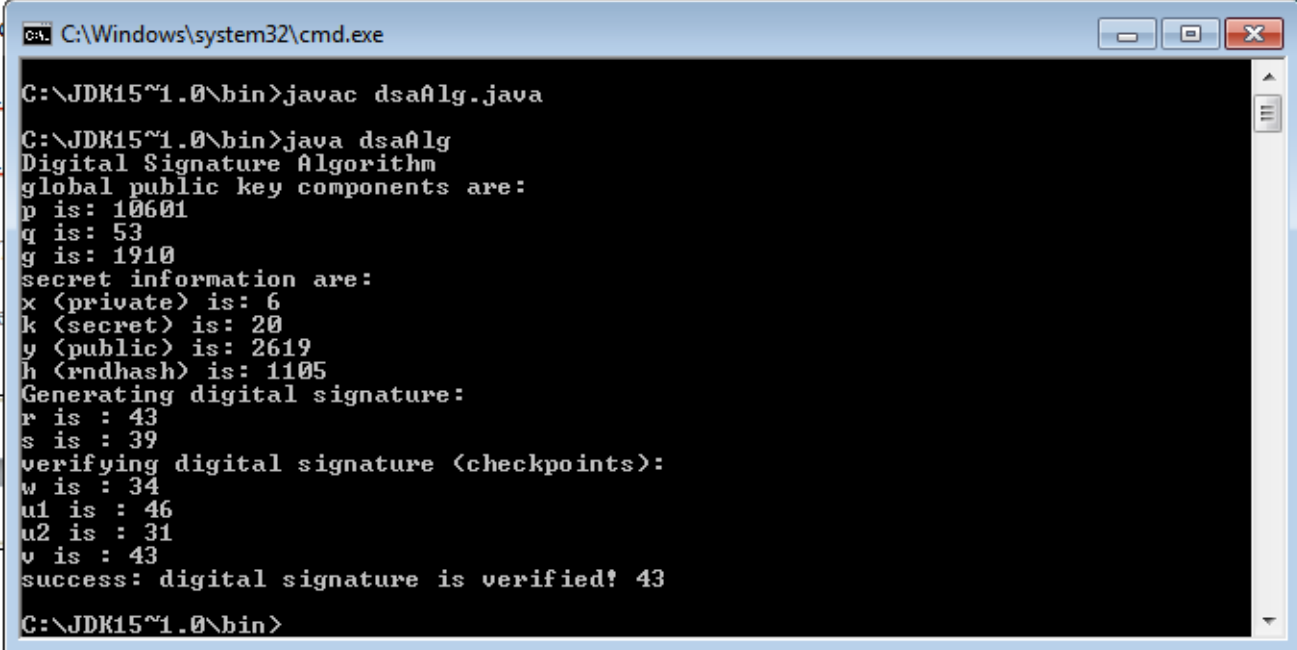
/* secret information */
System.out.println("secret information are:");
System.out.println("x (private) is: " + x);
System.out.println("k (secret) is: " + k);
System.out.println("y (public) is: " + y);
System.out.println("h (rndhash) is: " + hashVal);
System.out.println("Generating digital signature:");
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");
}
}
```

OUTPUT



```
C:\Windows\system32\cmd.exe

C:\JDK15~1.0\bin>javac dsaAlg.java

C:\JDK15~1.0\bin>java dsaAlg
Digital Signature Algorithm
global public key components are:
p is: 10601
q is: 53
g is: 1910
secret information are:
x (private) is: 6
k (secret) is: 20
y (public) is: 2619
h (rndhash) is: 1105
Generating digital signature:
r is : 43
s is : 39
verifying digital signature (checkpoints):
w is : 34
u1 is : 46
u2 is : 31
v is : 43
success: digital signature is verified! 43

C:\JDK15~1.0\bin>
```

RESULT:

Thus the program to implement Digital Signature was developed and executed successfully.

AIM:

To demonstrate intrusion detection system (ids) using the tool snort.

PROCEDURE:**1. Configure and Use Snort IDS on Windows**

Steps to configure Snort on Windows machine and how to use it for detection of attacks.

2. Steps:

1. Download Snort from "<http://www.snort.org/>" website.
2. Also download Rules from the same website. You need to sign up to get rules for registered users.
3. Click on the Snort_(version-number)_Installer.exe file to install it. By-default it will install snort in the "C:\Snort" directory.
4. Extract downloaded Rules file: snortrules-snapshot-(number).tar.gz
5. Copy all files from the "rules" directory of the extracted folder and paste them into "C:\Snort\rules" directory.
6. Copy "snort.conf" file from the "etc" directory of the extracted folder and paste it into "C:\Snort\etc" directory. Overwrite existing file if there is any.
7. Open command prompt (cmd.exe) and navigate to directory "C:\Snort\bin" directory.
8. To execute snort in sniffer mode use following command:

```
snort -dev -i 2
```

-i indicate interface number.

-dev is used to run snort to capture packets.

To check interface list use following command: snort -W

9. To execute snort in IDS mode, we need to configure a file "snort.conf" according to our network environment.

10. Set up network address we want to protect in snort.conf file. To do that look for "HOME_NET" and add your IP address.

```
var HOME_NET 10.1.1.17/8
```

11. You can also set addresses or DNS_SERVERS, if you have any. otherwise go to the next step.
12. Change RULE_PATH variable with the path of rules directory.

```
var RULE_PATH c:\snort\rules
```

13. Change the path of all libraries with the name and path on your system. or change path of snort dynamic preprocess or variable.

or file C:\Snort\lib\snort_dynamicccpreprocessor\sf_dcerpc.dll

You need to do this to all library files in the "C:\Snort\lib" directory. The old path might be something like: "/usr/local/lib/...". you need to replace that path with you system path.

14. Change path of the "dynamicengine" variable value in the "snort.conf" file with the path of your system. Such as:

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

15 Add complete path for "include classification.config" and "include reference.config" files.

include c:\snort\etc\classification.config

include c:\snort\etc\reference.config

16. Remove the comment on the line to allow **ICMP** rules, if it is alredy commented.

include \$RULE_PATH/icmp.rules

17. Similary, remove the comment of ICMP-info rules comment, if it is already commented.

include \$RULE_PATH/icmp-info.rules

18 To add log file to store alerts generated by snort, search for "output log" test and add following line:

output alert_fast: snort-alerts.ids

19. Comment whitelist \$WHITE_LIST_PATH/white_list.rules and blacklist

\$BLACK_LIST_PATH/black_list.rules lines. Also ensure that you add change the line above \$WHITE_LIST_PATH

Change nested_ip inner , \ to nested_ip inner #, \

20. Comment following lines:

#preprocessor normalize_ip4

#preprocessor normalize_tcp: ips ecn stream

#preprocessor normalize_icmp4

#preprocessor normalize_ip6

#preprocessor normalize_icmp6

21. Save the "snort.conf" file and close it.

22. Go to the "C:\Snort\log" directory and create a file: snort-alerts.ids

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

snort -c c:\snort\etc\snort.conf -l c:\snort\log -i 2

Above command will generate log file that will not be readable without using a tool. To read it use following command:

C:\Snort\Bin> snort -r ../log/log-filename

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

snort -A console -i2 -c c:\Snort\etc\snort.conf -l c:\Snort\log -K ascii

24. Scan the computer running snort from another computer using PING or launch attack. Then check snort-alerts.ids file the log folder.

RESULT:

Thus the intrusion detection system (ids) using the tool snort program was demonstrated and verified successfully.

AIM:

To explore automated and penetration tools on network (KF Sensor)

PROCEDURE:**HONEYPOTS**

When it comes to computer security, honeypots are all the rage. Honeypots can detect unauthorized activities that might never be picked up by a traditional intrusion detection system. Furthermore, since almost all access to a honeypot is unauthorized, nearly everything in a honeypot's logs is worth paying attention to. Honeypots can act as a decoy to keep hackers away from your production servers. At the same time though, a honeypot can be a little tricky to deploy. In this article, I will walk you through the process of deploying a honeypot.

INTRODUCTION

There are many different types of honeypot systems. Honeypots can be hardware appliances or they can be software based. Software based firewalls can reside on top of a variety of operating systems. For the most part though, honeypots fall into two basic categories; real and virtual.

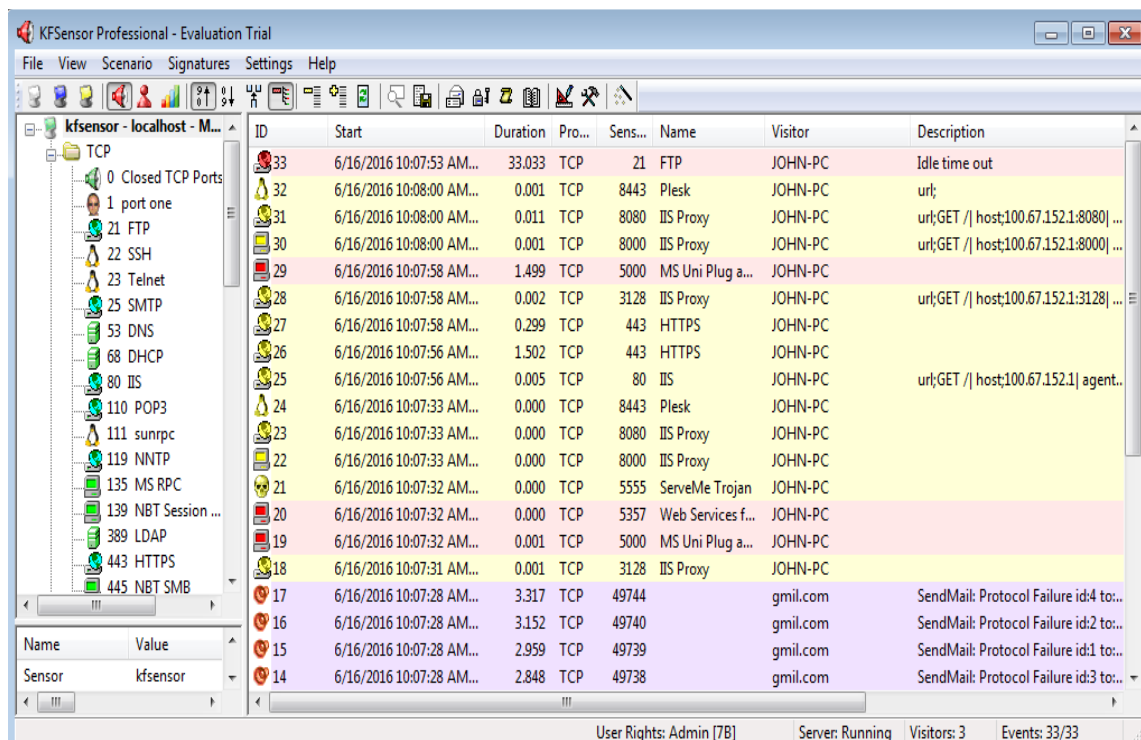
A virtual honeypot is essentially an emulated server. There are both hardware and software implementations of virtual honeypots. For example, if a network administrator was concerned that someone might try to exploit an FTP server, the administrator might deploy a honeypot appliance that emulates an FTP server.

Downloading and installing KF Sensor

- The KF Sensor download consists of a 1.7 MB self-extracting executable file.
- Download the file and copy it into an empty folder on your computer.
- When you double click on the file, it will launch a very basic Setup program.
- The only thing special that you need to know about the Setup process is that it will require a reboot

Using KFSensor

Step1: You will see the main KFSensor screen shown

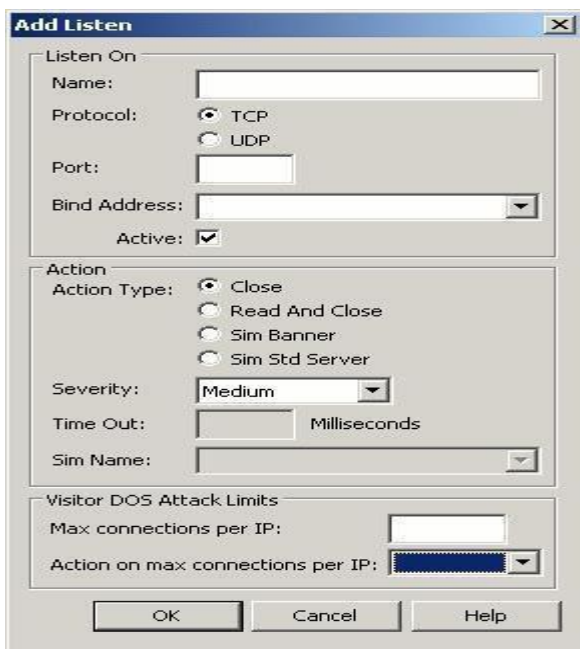


- As you can see, the column on the left contains a list of port numbers and what the port is typically used for.
- If the icon to the left of a port listing is green, it means that KFSensor is actively monitoring that port for attacks.
- If the icon is blue, it means that there has been an error and KFSensor is not watching for exploits aimed at that particular port.

Testing the software

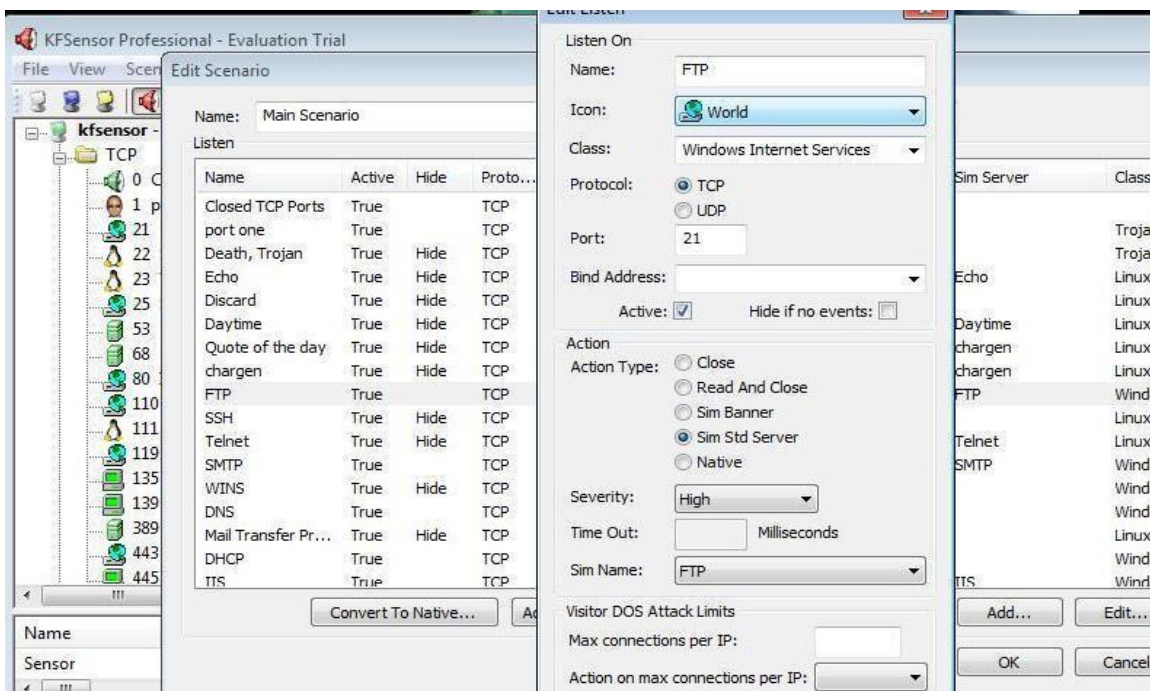
- Once you've got the software up and running, one of the best things that you can do is to test the software by launching a port scan against the machine that's running KFSensor.
 - For the port scan, we using the HostScan.
- It simply scans a block of IP addresses, looking for open ports. Figure B shows how the KFSensor reacts to a partial port scan.
- If you look at Figure B, you will notice that the icons next to ports that were scanned turn red to indicate recent activity.

Click on Add Button



The 'Add Listen' dialog box is shown. It has three main sections: 'Listen On', 'Action', and 'Visitor DOS Attack Limits'. In the 'Listen On' section, 'Name' is empty, 'Protocol' has 'TCP' selected, 'Port' is empty, and 'Bind Address' is empty. 'Active' is checked. In the 'Action' section, 'Action Type' has 'Close' selected, 'Severity' is 'Medium', 'Time Out' is empty, and 'Sim Name' is empty. In the 'Visitor DOS Attack Limits' section, 'Max connections per IP' is empty and 'Action on max connections per IP' is empty. At the bottom are 'OK', 'Cancel', and 'Help' buttons.

Click on Edit Button



The KFSensor Professional - Evaluation Trial interface is shown. The 'Edit Scenario' dialog box is open, showing a list of sensors on the left and a table of sensors in the center. The 'Listen' section is selected, and the 'FTP' sensor is highlighted. The 'Listen On' section is open, showing 'Name' as 'FTP', 'Icon' as 'World', 'Class' as 'Windows Internet Services', 'Protocol' as 'TCP', 'Port' as '21', and 'Bind Address' as empty. 'Active' is checked and 'Hide if no events' is unchecked. In the 'Action' section, 'Action Type' has 'Sim Std Server' selected, 'Severity' is 'High', 'Time Out' is empty, and 'Sim Name' is 'FTP'. In the 'Visitor DOS Attack Limits' section, 'Max connections per IP' is empty and 'Action on max connections per IP' is empty. At the bottom are 'Add...', 'Edit...', 'OK', and 'Cancel' buttons.

Name	Active	Hide	Proto...
Closed TCP Ports	True		TCP
port one	True		TCP
Death, Trojan	True	Hide	TCP
Echo	True	Hide	TCP
Discard	True	Hide	TCP
Daytime	True	Hide	TCP
Quote of the day	True	Hide	TCP
chargen	True	Hide	TCP
FTP	True		TCP
SSH	True	Hide	TCP
Telnet	True	Hide	TCP
SMTP	True		TCP
WINS	True	Hide	TCP
DNS	True		TCP
Mail Transfer Pr...	True	Hide	TCP
DHCP	True		TCP
ITS	True		TCP

- The next few fields are protocol, port, and Bind Address. These fields allow you to choose what the rule is listening for. For example, you could configure the rule to listen to TCP port 1023 on IP address 192.168.1.100. The bind address portion of the rule is optional though. If you leave the bind address blank, the rule will listen across all of the machine's NICs.
- Now that you have defined the listener, it's time to configure the action that the rule takes when traffic is detected on the specified port. Your options are close, read and close, Sim Banner, and SimStd Server.
- The close option tells the rule to just terminate the connection. Read and close logs the information and then terminates the connection. The SimStd Server and Sim Banner options

pertain to server emulation. The Sim Banner option allows you to perform a very simple server emulation, such as what you might use to emulate an FTP server.

- The Sim STD Server option allows you to emulate a more complex server, such as an IIS server.
- If you choose to use one of the sim options, you will have to fill in the simulator's name just below the Time Out field.
- The other part of the Action section that's worth mentioning is the severity section. KFSensor treated some events as severe and other events as a more moderate threat. The dialog box's Severity drop down list allows you to determine what level of severity should be associated with the event that you are logging.
- The final portion of the Add Listen dialog box is the Visitor DOS Attack Limits section. This section allows you to prevent denial of service attacks against KFSensor. You can determine the maximum number of connections to the machine per IP address (remember that this applies on a per rule basis).
- If your threshold is exceeded, you can choose to either ignore the excessive connections or you can lock out the offending IP address.
- Now that you have configured the new rule, select the Active Button to Enable/Disable. The new rule should now be in effect.

RESULT:

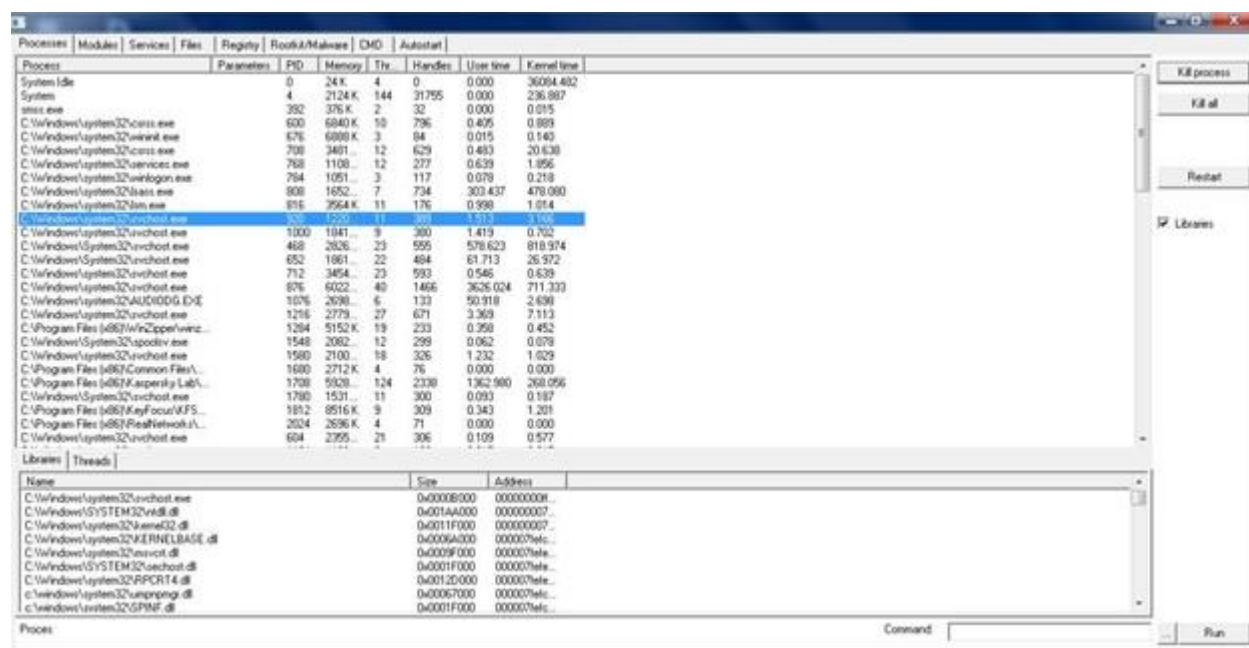
Thus the program to explore automated attack and penetration tools has been completed successfully

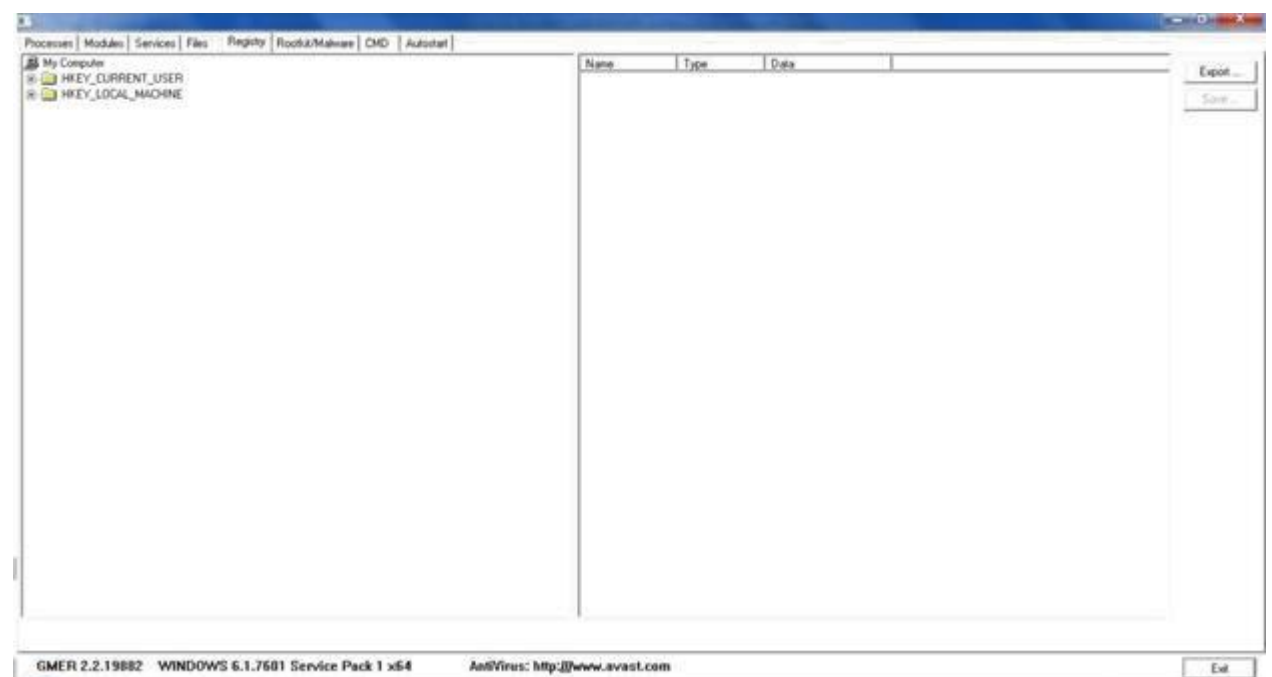
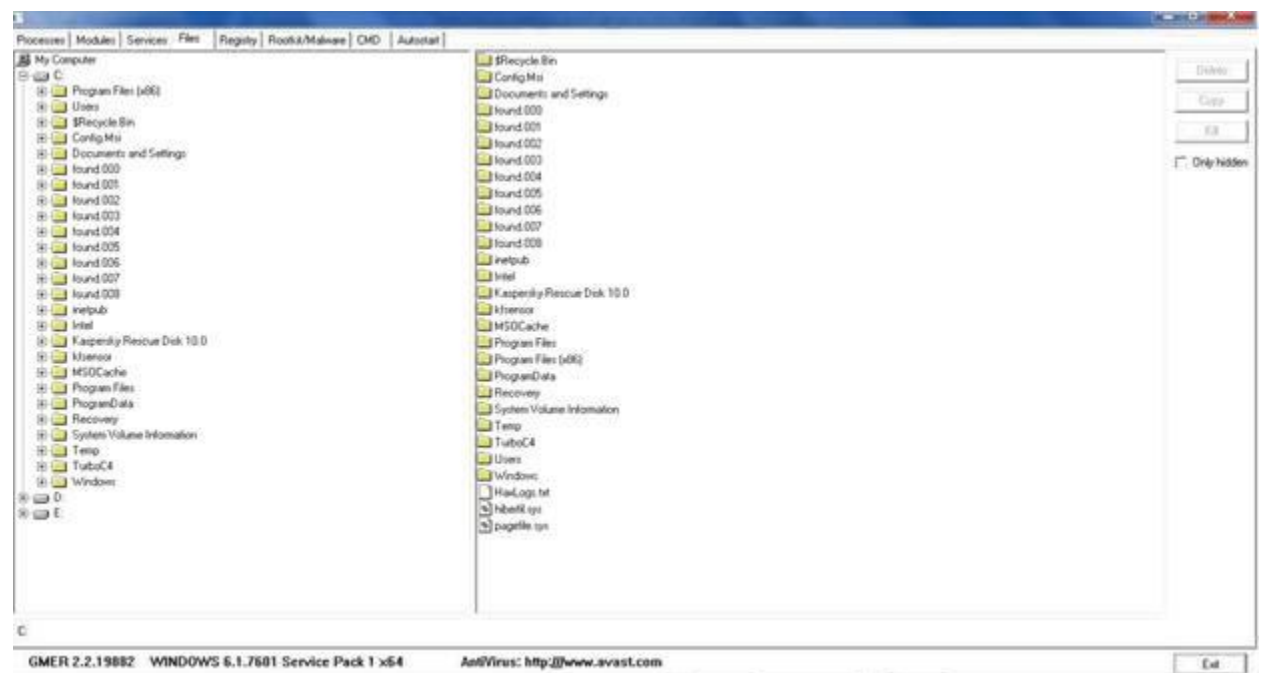
AIM

To install a rootkit hunter and find the malwares in 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, Boot.
- Files menu displays full files on Hard-Disk volumes.
- Registry displays **Hkey_Current_user** and **Hkey_Local_Machine**. Rootkits/Malwares scans the local drives selected.
- **Autostart** displays the registry base Autostart applications.
- **CMD** allows the user to interact with command line utilities or Registry.





AIM:

To implement the TRIPLE DES in java.

ALGORITHM:

1. Start the program.
2. Encrypt the plaintext blocks using single DES with key K_1 .
3. Now decrypt the output of step 1 using single DES with key K_2 .
4. Finally, encrypt the output of step 2 using single DES with key K_3 .
5. The output of step 3 is the ciphertext.
6. Decryption of a ciphertext is a reverse process. User first decrypt using K_3 , then encrypt with K_2 , and finally decrypt with K_1 .
7. Stop the program.

12. IMPLEMENTATION OF TRIPLE DES

PROGRAM:

```
import java.util.Arrays;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;
public class TripleDESTest
{
    public static void main(String[] args) throws Exception
    {
        String text = "textToEncrypt";
        String codedtext = new TripleDESTest()._encrypt(text,"SecretKey");
        String decodedtext = new TripleDESTest()._decrypt(codedtext,"SecretKey");
        System.out.println(codedtext + " ---> " + decodedtext);
    }
    private String _encrypt(String message, String secretKey) throws Exception
    {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] digestOfPassword = md.digest(secretKey.getBytes("utf-8"));
        byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24);

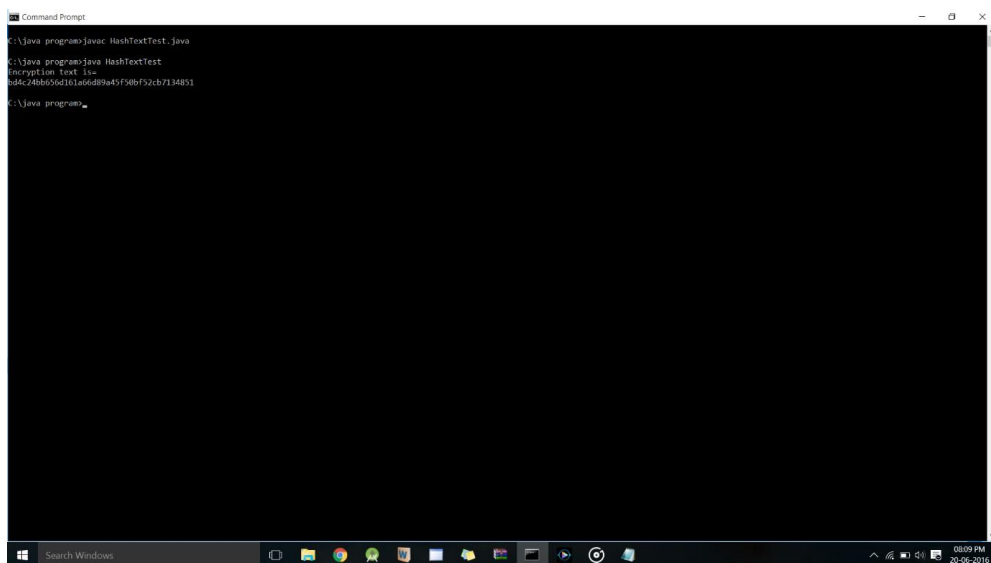
        SecretKey key = new SecretKeySpec(keyBytes, "DESede");
        Cipher cipher = Cipher.getInstance("DESede");
        cipher.init(Cipher.ENCRYPT_MODE, key);

        byte[] plainTextBytes = message.getBytes("utf-8");
        byte[] buf = cipher.doFinal(plainTextBytes);
        byte [] base64Bytes = Base64.encodeBase64(buf);
        String base64EncryptedString = new String(base64Bytes);

        return base64EncryptedString;
    }
    private String _decrypt(String encryptedText, String secretKey) throws Exception {
        byte[] message = Base64.decodeBase64(encryptedText.getBytes("utf-8"));
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] digestOfPassword = md.digest(secretKey.getBytes("utf-8"));
        byte[] keyBytes = Arrays.copyOf(digestOfPassword, 24); SecretKey
        key = new SecretKeySpec(keyBytes, "DESede");

        Cipher decipher = Cipher.getInstance("DESede");
        decipher.init(Cipher.DECRYPT_MODE, key);
        byte[] plainText = decipher.doFinal(message);
        return new String(plainText, "UTF-8");
    }
}
```

OUTPUT:



```
Command Prompt
C:\java_program>javac HashTextTest.java
C:\java_program>java HashTextTest
Encryption text is=
8d4c248bb0d101e06d8943f5bf32cb7134801
C:\java_program>
```

The screenshot shows a Windows Command Prompt window with a black background and white text. The window title is "Command Prompt". The command prompt shows the following sequence of commands and output: 1. The user is at the C:\java_program directory. 2. The command 'javac HashTextTest.java' is executed. 3. The command 'java HashTextTest' is executed, resulting in the output 'Encryption text is=' followed by a new line and the hexadecimal string '8d4c248bb0d101e06d8943f5bf32cb7134801'. 4. The prompt returns to C:\java_program>. The Windows taskbar is visible at the bottom, showing the Start button, search bar, and several application icons. The system clock in the bottom right corner indicates 08:09 PM on 20-06-2016.

RESULT:

Thus the program to implement the TRIPLE DES in java has been executed and the output was verified successfully.

AIM:

To implement the blowfish algorithm by using Java .

ALGORITHM:

- 1.Blowfish has a 64-bit block size and a variable key length from 30bits up to 48bits.
- 2.It is a 16-round Feistel cipher and uses large key dependent s-boxes.
- 3.There are 5 sub key-arrays. One 18-entry p-array and four 256-entry s-boxes.
- 4.Every round r consists of 4 actions.
 - a)XOR the left half of the data with the 'r'th p-array entry.
 - b)Use the XORED data as input for Blowfish algorithm.
 - c)F-function's output with the right half (R) of the data.
 - d)Swap L and R.
- 5.The F-function splits the 32bits into four 8-bits quarters and uses the quarters as input to the s-boxes.
- 6.The s-boxes accept 8-bit input and produce 32-bit output.The outputs are added modulo 2 power 32 and XORED to produce the final 32-bit output.

13. IMPLEMENTATION OF BLOWFISH ALGORITHM

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

public BlowfishNew() {
try {
generateSymmetricKey();
inputMessage=JOptionPane.showInputDialog(null,"Enter message to encrypt");
byte[] iblete = inputMessage.getBytes();
byte[] ebyte=encrypt(raw, iblete);
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("Blowfish Symmetric key = "+skeyString);
}
catch(Exception e) {
System.out.println(e);
}
}
```

```

private static byte[] getRawKey(byte[] seed) throws Exception {
    KeyGenerator kgen = KeyGenerator.getInstance("Blowfish");
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
    sr.setSeed(seed);
    kgen.init(128, sr); // 128, 256 and 448 bits may not be available

    SecretKey skey = kgen.generateKey();
    raw = skey.getEncoded();
    return raw;
}

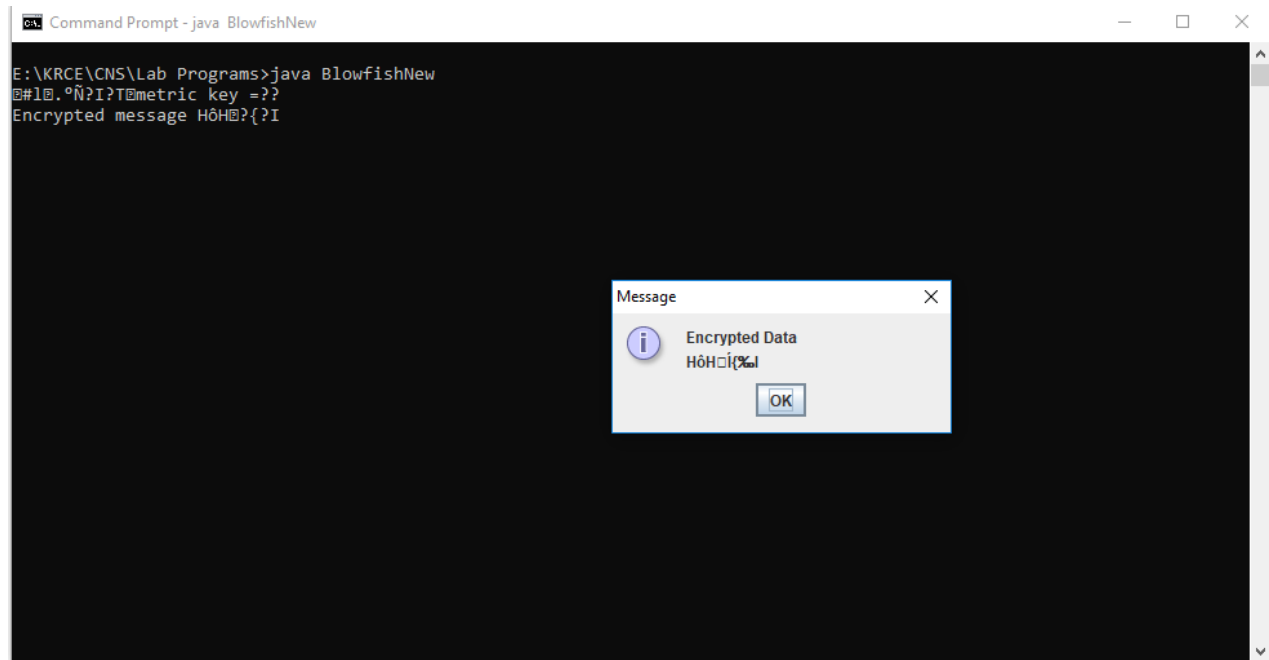
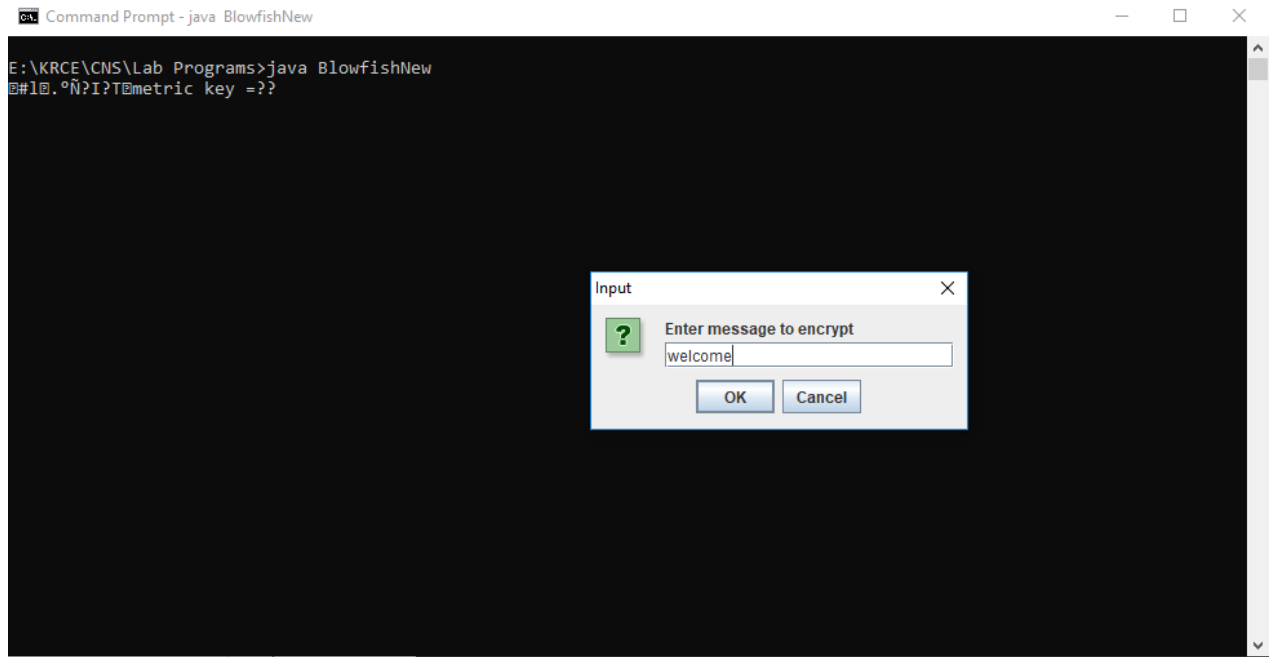
private static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
    SecretKeySpec skeySpec = new SecretKeySpec(raw, "Blowfish");
    Cipher cipher = Cipher.getInstance("Blowfish");
    cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
    byte[] encrypted = cipher.doFinal(clear);
    return encrypted;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

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

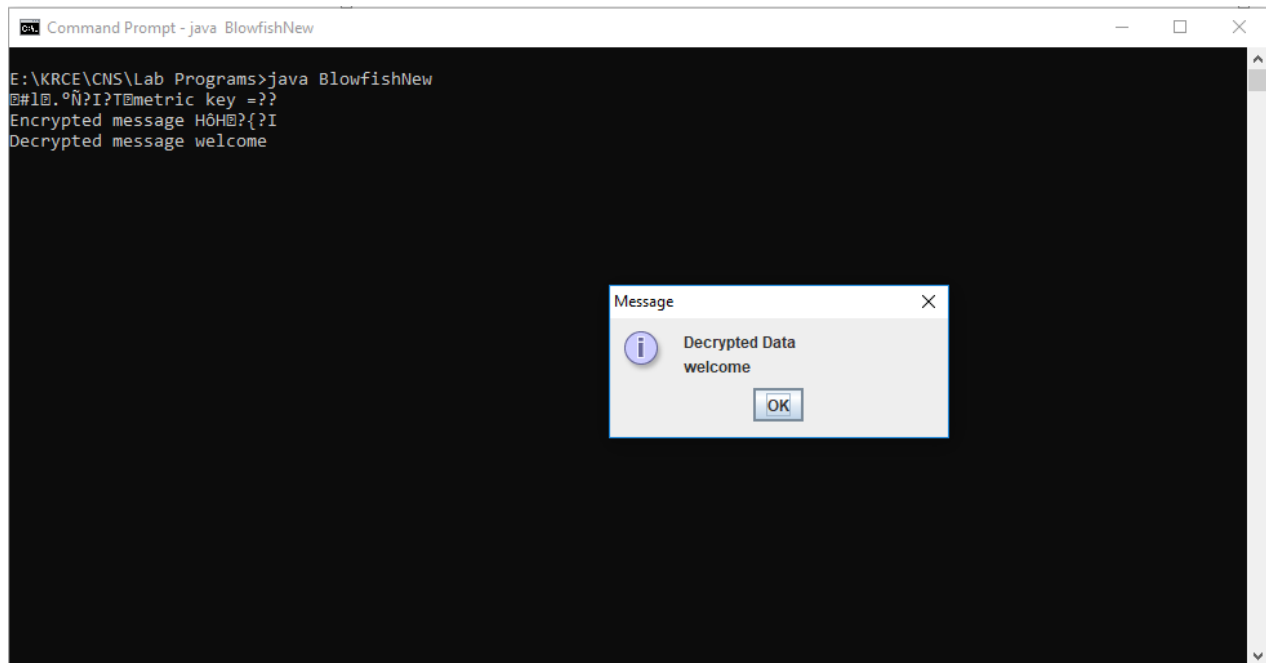
public static void main(String args[]) {
    BlowfishNew bf = new BlowfishNew();
}
}

```

OUTPUT:



```
Command Prompt - java BlowfishNew
E:\KRCE\CNS\Lab Programs>java BlowfishNew
@#1@.°Ñ?I?T@metric key =??
Encrypted message HôH@?{?I
Decrypted message welcome
```



RESULT:

Thus the Blowfish algorithm has been implemented and the output has been verified successfully.

