**EX NO 1 i. ENCRYPTION AND DECRYPTION USING CAESAR CIPHER**

**DATE:**

**AIM:**

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

**ALGORITHM:**

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

**STEP 2:** Read the key value from the user.

**STEP 3:** The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1, Z = 25.

**STEP 4:** Encryption of a letter x by a shift n can be described mathematically as, En(x) = (x + n) mod26

**STEP 5:** Decryption is performed similarly, Dn (x)=(x - n) mod26

**STEP 6:** If the key is positive then encrypt the text by adding the key with each character in the plain text.

**STEP 7:** Else subtract the key from the plain text.

**STEP 8:** Display the cipher text obtained above.

**PROGRAM:**

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.Scanner;

public class CaesarCipher {

static Scanner sc=new Scanner(System.in);

static BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

public static void main(String[] args) throws IOException {

System.out.print("Enter any String: ");

String str = br.readLine();

System.out.print("\nEnter the Key: ");

int key = sc.nextInt();

String encrypted = encrypt(str, key);

System.out.println("\nEncrypted String is: " +encrypted);

String decrypted = decrypt(encrypted, key);

System.out.println("\nDecrypted String is: " +decrypted);

System.out.println("\n");

}

public static String encrypt(String str, int key) { String encrypted = "";

for(int i = 0; i < str.length(); i++)

{

int c = str.charAt(i);

if (Character.isUpperCase(c))

{

c = c + (key % 26);

if (c > 'Z')

c = c - 26;

}

else if (Character.isLowerCase(c)) { c = c + (key % 26);

if (c > 'z')

c = c - 26;

}

encrypted += (char) c;

}

return encrypted;

}

public static String decrypt(String str, int key) { String decrypted = "";

for(int i = 0; i < str.length(); i++) { int c = str.charAt(i);

if (Character.isUpperCase(c)) { c = c - (key % 26);

if (c < 'A')

c = c + 26;

}

else if (Character.isLowerCase(c)) { c = c - (key % 26);

if (c < 'a')

c = c + 26;

}

decrypted += (char) c;

}

return decrypted;}}

**OUTPUT :**

C:\Program Files\Java\jdk1.6.0\_20\bin>javac CaesarCipher.java

C:\Program Files\Java\jdk1.6.0\_20\bin>java CaesarCipher

Enter any String: pani

Enter the Key: 2

Encrypted String is: ocpk

Decrypted String is: pani

**RESULT:**

Thus a program for encrypting a plain text and decrypting a cipher text using Caesar Cipher (shift cipher) substitution technique was implemented and executed successfully.

**EX NO 1 ii. PLAYFAIR CIPHER**

**DATE:**

**AIM:**

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

**ALGORITHM:**

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

**STEP 2:** Read the keyword from the user.

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

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

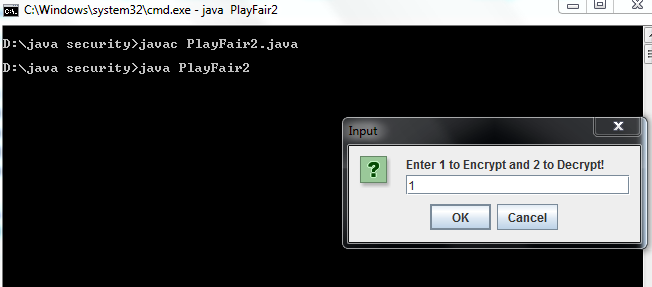
**STEP 5:** Display the obtained cipher text.

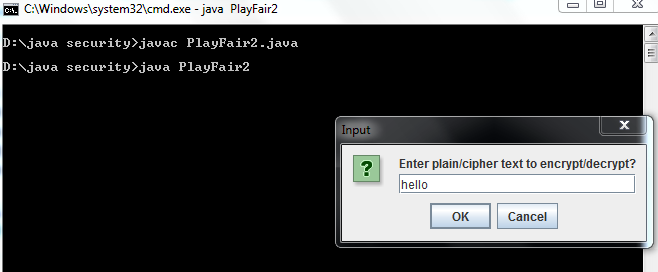
**PROGRAM:**

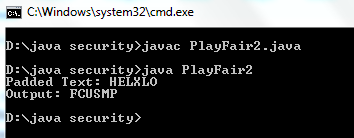
import javax.swing.JOptionPane;  
public class PlayFair2 {  
    //the key matrix  
    public static char[][] keymat = new char[][]{  
                { 'M', 'O', 'N', 'A', 'R' },  
                { 'C', 'H', 'Y', 'B', 'D' },  
                { 'E', 'F', 'G', 'I', 'K'},  
                { 'L', 'P', 'Q', 'S', 'T'},  
                { 'U', 'V', 'W', 'X', 'Z'}  
            };  
    public static String trans = "J"; //for translating J to I  
    public static char filler = 'X'; //filler letter is X  
    public static void main(String args[]){  
        String text,outtext="";  
        int ch;  
        ch = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter 1 to Encrypt and 2 to Decrypt!"));  
        text = JOptionPane.showInputDialog(null, "Enter plain/cipher text to encrypt/decrypt?");  
        text = text.toUpperCase();  
        text = text.replaceAll("\\s",""); //removing spaces  
        text = text.replace(trans, "I"); //changing J with I  
        text = text.replaceAll("([A-Z])\\1+","$1"+filler+"$1");

//handling repeated letters  
        if(text.length() % 2 !=0)  
            text+= filler;  
        char[] ptextchars = text.toCharArray();  
        System.out.println("Padded Text: "+text);  
                switch(ch){  
            case 1:  
                    for(int i=0;i< text.length(); i+=2){  
                        outtext += encrypt(ptextchars[i],ptextchars[i+1]);  
                    }  
                    break;  
            case 2:  
                    for(int i=0;i< text.length(); i+=2){  
                        outtext += decrypt(ptextchars[i],ptextchars[i+1]);  
                    }  
                    break;  
            default: System.out.println("Invalid Choice!");  
        }  
        System.out.println("Output: "+outtext);  
    }  
    private static String encrypt(char c1, char c2) {  
        int[] posa = new int[2];  
        int[] posb = new int[2];  
        String frag = "";  
        posa = search(c1);  
        posb = search(c2);  
        if(posa[0] == posb[0]){//same row  
            c1 = keymat[posa[0]][(posa[1]+1)%5];  
            c2 = keymat[posb[0]][(posb[1]+1)%5];  
            frag = (""+c1+c2+"");  
        }  
        else if(posa[1] == posb[1]){ //same column  
            c1 = keymat[(posa[0]+1)%5][posa[1]];  
            c2 = keymat[(posb[0]+1)%5][posb[1]];  
            frag = (""+c1+c2+"");  
        }  
        else{  
            c1 = keymat[posb[0]][posa[1]];  
            c2 = keymat[posa[0]][posb[1]];  
            frag = (""+c1+c2+"");  
        }  
        return frag;  
    }  
    private static String decrypt(char c1, char c2) {  
        int[] posa = new int[2];  
        int[] posb = new int[2];  
        String frag = "";  
        posa = search(c1);  
        posb = search(c2);  
        if(posa[0] == posb[0]){//same row  
            c1 = keymat[posa[0]][(posa[1]-1)%5];  
            c2 = keymat[posb[0]][(posb[1]-1)%5];  
            frag = (""+c1+c2+"");  
        }  
        else if(posa[1] == posb[1]){ //same column  
            c1 = keymat[(posa[0]-1)%5][posa[1]];  
            c2 = keymat[(posb[0]-1)%5][posb[1]];  
            frag = (""+c1+c2+"");  
        }  
        else{  
            c1 = keymat[posb[0]][posa[1]];  
            c2 = keymat[posa[0]][posb[1]];  
            frag = (""+c1+c2+"");  
        }  
        return frag;  
    }  
    private static int[] search(char c) {  
        int i,j;  
        int[] pos = new int[2];  
        for (i = 0; i < 5; i++) {  
            for (j = 0; j < 5; j++) {  
                if (keymat[i][j] == c){  
                    pos[0] = i;  
                    pos[1] = j;  
                    break;  
                }}}  
        return pos;  
    }}

**OUTPUT:**

****

****

****

**RESULT:**

Thus a program to encrypt a plain text and decrypt a cipher text using Playfair Cipher substitution technique was implemented and executed successfully.

**EX NO : 1 iii. HILL CIPHER**

**DATE:**

**AIM:**

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

**ALGORITHM:**

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

**STEP 2:** Split the plain text into groups of length three.

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

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

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

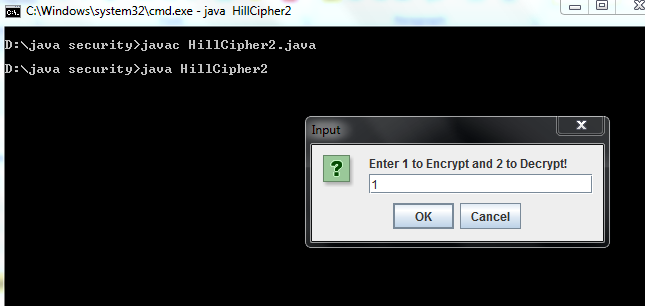
**PROGRAM:**

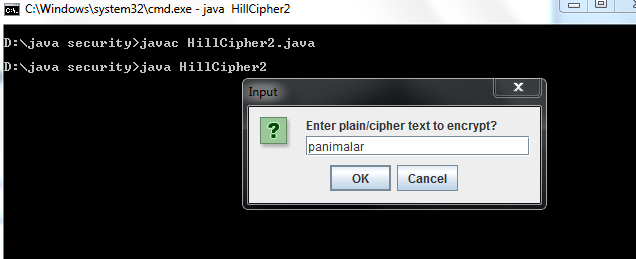
import java.io.\*;  
import java.lang.\*;  
public class hillcipher  
{  
public static void main(String []arg)throws Exception  
{  
    int k[][]={{17,17,5}, {21,18,21}, {2,2,19}};  
    int p[]=new int[300];  
    int c[]=new int[300];  
    int i=0;      
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));  
    System.out.println("enter plain text");  
    String str=br.readLine();  
      
    for( i=0;i<str.length();i++)  
    {  
        int c1=str.charAt(i);  
        //System.out.println(c1);  
        p[i]=(c1)-97;  
    }  
                i=0;int zz=0;  
        for( int b=0;b<str.length()/3;b++)  
        {  
        for(int j=0;j<3;j++)  
        {  
        for(int x=0;x<3;x++)  
        {  
        c[i]+=k[j][x]\*p[x+zz];  
        }i++;  
        }  
        zz+=3;  
        }  
                System.out.println("Encrypted Text : ");  
        for(int z=0;z<str.length();z++)  
        System.out.print((char)((c[z]%26)+97));  
        }  
}

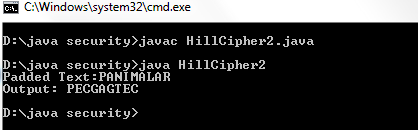
import javax.swing.JOptionPane;  
public class HillCipher {  
    //the 3x3 key matrix for 3 characters at once  
    public static int[][] keymat = new int[][]{  
                { 1, 2, 1 },  
                { 2, 3, 2 },  
                { 2, 2, 1 },  
            };  
     public static int[][] invkeymat = new int[][]{  
                { -1, 0, 1 },  
                { 2, -1, 0 },  
                { -2, 2, -1},  
            };  
    public static String key = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
    public static void main(String[] args) {

        String text,outtext ="";  
        int ch, n;  
        ch = Integer.parseInt(JOptionPane.showInputDialog(null, "Enter 1 to Encrypt and 2 to Decrypt!"));  
        text = JOptionPane.showInputDialog(null, "Enter plain/cipher text to encrypt?");  
        text = text.toUpperCase();  
        text = text.replaceAll("\\s",""); //removing spaces  
        n = text.length() % 3;  
        if(n!=0){  
            for(int i = 1; i<= (3-n);i++){  
                text+= 'X';  
            }  
        }  
        System.out.println("Padded Text:" + text);  
        char[] ptextchars = text.toCharArray();  
        switch(ch){  
            case 1:  
                    for(int i=0;i< text.length(); i+=3){  
                        outtext += encrypt(ptextchars[i],ptextchars[i+1],ptextchars[i+2]);  
                    }  
                    break;  
            case 2:  
                    for(int i=0;i< text.length(); i+=3){  
                        outtext += decrypt(ptextchars[i],ptextchars[i+1],ptextchars[i+2]);  
                    }  
                    break;  
            default: System.out.println("Invalid Choice!");  
        }  
        System.out.println("Output: " + outtext);  
    }  
        private static String encrypt(char a, char b, char c) {  
            String ret = "";  
            int x,y, z;  
            int posa = (int)a - 65;  
            int posb = (int)b - 65;  
            int posc = (int)c - 65;  
            x = posa \* keymat[0][0] + posb \* keymat[1][0] + posc \* keymat[2][0];  
            y = posa \* keymat[0][1] + posb \* keymat[1][1] + posc \* keymat[2][1];  
            z = posa \* keymat[0][2] + posb \* keymat[1][2] + posc \* keymat[2][2];  
            a = key.charAt(x%26);  
            b = key.charAt(y%26);  
            c = key.charAt(z%26);  
            ret = "" + a + b + c;  
            return ret;  
        }  
        private static String decrypt(char a, char b, char c) {  
            String ret = "";  
            int x,y,z;  
            int posa = (int)a - 65;  
            int posb = (int)b - 65;  
            int posc = (int)c - 65;  
            x = posa \* invkeymat[0][0]+ posb \* invkeymat[1][0] + posc \* invkeymat[2][0];  
            y = posa \* invkeymat[0][1]+ posb \* invkeymat[1][1] + posc \* invkeymat[2][1];  
            z = posa \* invkeymat[0][2]+ posb \* invkeymat[1][2] + posc \* invkeymat[2][2];  
            a = key.charAt((x%26<0)?(26+x%26):(x%26));  
            b = key.charAt((y%26<0)?(26+y%26):(y%26));  
            c = key.charAt((z%26<0)?(26+z%26):(z%26));  
            ret = "" + a + b + c;  
            return ret;  
        }     
}

**OUTPUT:**

****

****

****

**RESULT:**

Thus a program to encrypt and decrypt a text using Hill Cipher substitution technique was implemented and executed successfully.

**EX NO 1 iv. VIGENERE CIPHER**

**DATE:**

**AIM:**

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

**ALGORITHM:**

**STEP 1:** The Vigenere cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword.

**STEP 2:** It is a simple form of polyalphabetic substitution.

**STEP 3:** To encrypt, a table of alphabets can be used, termed a Vigenere square, or Vigenere table.

**STEP 4:** It consists of the alphabet written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar ciphers.

**STEP 5:** At different points in the encryption process, the cipher uses a different alphabet from one of the rows used.

**STEP 6:** The alphabet at each point depends on a repeating keyword.

**STEP 7:** By using math. Equation:

C= E(p) = (p+k*i*) mod (26)

PlaintextTHISPROCESSCANALSOBEEXPRESSED

KeywordCIPHERCIPHERCIPHERCIPHERCIPHE

CiphertextVPXZTIQKTZWTCVPSWFDMTETIGAHLH

**PROGRAM:**

import java.util.\*;

public class VigenereCipher

{

public static String encrypt(String text, final String key)

{

String res = "";

text = text.toUpperCase();

for (int i = 0, j = 0; i < text.length(); i++)

{

char c = text.charAt(i);

if (c < 'A' || c > 'Z')

continue;

res += (char) ((c + key.charAt(j) - 2 \* 'A') % 26 + 'A');

j = ++j % key.length();

}

return res;

}

public static String decrypt(String text, final String key)

{

String res = "";

text = text.toUpperCase();

for (int i = 0, j = 0; i < text.length(); i++)

{

char c = text.charAt(i);

if (c < 'A' || c > 'Z')

continue;

res += (char) ((c - key.charAt(j) + 26) % 26 + 'A');

j = ++j % key.length();

}

return res;

}

public static void main(String[] args)

{

String key = "VIGENERECIPHER";

String message = "Beware the Jabberwock, my son! The jaws that bite, the claws that catch!";

String encryptedMsg = encrypt(message, key);

System.out.println("String: " + message);

System.out.println("Encrypted message: " + encryptedMsg);

System.out.println("Decrypted message: " + decrypt(encryptedMsg, key));

}

}

**OUTPUT :**

C:\Program Files\Java\jdk1.6.0\_20\bin>javac VigenereCipher.java

C:\Program Files\Java\jdk1.6.0\_20\bin>java VigenereCipher

String: Beware the Jabberwock, my son! The jaws that bite, the claws that catch!

Encrypted message: WMCEEIKLGRPIFVMEUGXQPWQVIOIAVEYXUEKFKBTALVXTGAFXYEVKPAGY

Decrypted message: BEWARETHEJABBERWOCKMYSONTHEJAWSTHATBITETHECLAWSTHATCATCH

**RESULT:**

Thus a program for encryption and decryption using vigenere cipher substitution technique was implemented and executed successfully.

**EX NO 2 i. RAIL FENCE**

**DATE:**

**AIM:**

To implement a program for encryption and decryption using Rail Fence transposition technique.

**ALGORITHM:**

**STEP 1:** Read the Plaintext.

**STEP 2:** The key for the railfence cipher is just the number of rails.

**STEP 3:** Write message letters out diagonally over a number of rows then read off cipher row by row

**STEP 4:** We write it out in a special way on a number of rails (the key here is 3) depend the east wall of the castle

d . . . n . . . e . . . t . . . l . . . h . . . s . . .

. e .e .d .h .e .s .w .l .o .t .e .a .t .e

. . f . . . t . . . a . . . a . . . f . . . c . . . l .

**STEP 5:** The ciphertext is read off along the rows:

dnetlhseedheswloteateftaafcl

**STEP 6:**With a key of 4:

d . . . . . t . . . . . t . . . . . f . . . . . s . . .

. e . . . d . h . . . s . w . . . o . t . . . a . t . .

. . f .n . . . e . a . . . a . l . . . h . c . . . l .

. . . e . . . . . e . . . . . l . . . . . e . . . . . e

The ciphertext is again read off along the rows:

Dttfsedhswotatfneaalhcleelee

**PROGRAM:**

import java.io.\*;  
class RailFence  
{  
public static void main(String[] args) throws IOException  
{  
String plaintxt,ciphertxt;  
int i,j,k;  
System.out.println("Enter the message to be encrypted\n");  
BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));  
plaintxt=br.readLine();  
int len=plaintxt.length();  
String s1="",s2="",temp="";  
for(i=0;i<len;i++)  
{  
if(plaintxt.charAt(i)!=' ')  
temp=temp+plaintxt.charAt(i);  
}  
for(i=0;i<temp.length();i++)  
{  
if(i%2==0)  
s1=s1+temp.charAt(i);  
else  
s2=s2+temp.charAt(i);  
}  
ciphertxt=s1.concat(s2);  
System.out.println("Ciphertext : "+ciphertxt);  
plaintxt="";  
if(ciphertxt.length()%2==0)  
j=ciphertxt.length()/2;  
else  
j=ciphertxt.length()/2+1;  
k=j;  
for(i=0;i<k;i++,j++)  
{

plaintxt+=ciphertxt.charAt(i);

plaintxt+=ciphertxt.charAt(j);  
}  
System.out.println("Plaintext: "+plaintxt);  
}  
}

**OUTPUT :**

Enter the message to be encrypted  
network security  
Ciphertext : ntokeuiyewrscrt  
Plaintext : network security

**RESULT:**

Thus a program for encryption and decryption using Rail Fence transposition technique was implemented and executed successfully.

**EX NO 2 ii. ROW AND COLUMN TRANSFORMATION**

**DATE:**

**AIM:**

To implement a program for encryption and decryption by using row and

column transformation technique.

**ALGORITHM:**

**STEP 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 |  |  |

**STEP 2:**The plain text characters are placed horizontally and the cipher text is created with vertical format as: **holewdlo lr**.

**STEP 3:** Now, the receiver has to use the same table to decrypt the cipher text to plain text.

**PROGRAM:**

***TransCipher.java***

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.

**EX NO : 3 DES**

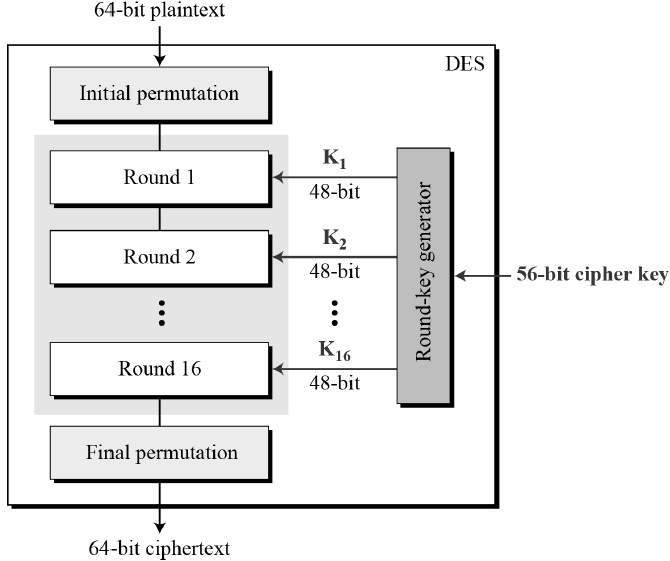
**DATE:**

**AIM:**

To write a JAVA program to implement the DES Algorithm.

**ALGORITHM :**

**STEP 1:** DES algorithm is designed to encipher and decipher blocks of data consisting of 64 bits under control of a 64-bit key



**STEP 2:** A block to be enciphered is subjected to an initial permutation IP and then to a complex key-dependent computation and finally to a permutation which is the inverse of the initial permutation IP-1.

**STEP 3:** Permutation is an operation performed by a function, which moves an element at place *j* to the place *k*.

**STEP 4:** The key-dependent computation can be simply defined in terms of a function *f*, called the *cipher function*, and a function KS, called the *key schedule*.

**STEP 5:** First, a description of the computation.

**STEP 6:** Next, the use of the algorithm for decipherment.

**STEP 7:** Finally, a definition of the cipher function *f* that is given in terms of selection function S*i* and permutation function P.

**STEP 8:** LR denotes the block consisting of the bits of L followed by the bits of R.

**PROGRAM:**

import java.io.\*;

import java.lang.\*;

class SDES

{

public int K1, K2;

public static final int P10[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

public static final int P10max = 10;

public static final int P8[] = { 6, 3, 7, 4, 8, 5, 10, 9};

public static final int P8max = 10;

public static final int P4[] = { 2, 4, 3, 1};

public static final int P4max = 4;

public static final int IP[] = { 2, 6, 3, 1, 4, 8, 5, 7};

public static final int IPmax = 8;

public static final int IPI[] = { 4, 1, 3, 5, 7, 2, 8, 6};

public static final int IPImax = 8;

public static final int EP[] = { 4, 1, 2, 3, 2, 3, 4, 1};

public static final int EPmax = 4;

public static final int S0[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,

3},{ 3, 1, 3, 2}};

public static final int S1[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,

2},{ 2, 1, 0, 3}};

public static int permute( int x, int p[], int pmax)

{

int y = 0;

for( int i = 0; i < p.length; ++i)

{

y <<= 1;

y |= (x >> (pmax - p[i])) & 1;

}

return y;

}

public static int F( int R, int K)

{

int t = permute( R, EP, EPmax) ^ K;

int t0 = (t >> 4) & 0xF;

int t1 = t & 0xF;

t0 = S0[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];

t1 = S1[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];

t = permute( (t0 << 2) | t1, P4, P4max);

return t;

}

public static int fK( int m, int K)

{

int L = (m >> 4) & 0xF;

int R = m & 0xF;

return ((L ^ F(R,K)) << 4) | R;

}

public static int SW( int x) {

return ((x & 0xF) << 4) | ((x >> 4) & 0xF);

}

public byte encrypt( int m)

{

System.out.println("\nEncryption Process Starts........\n\n");

m = permute( m, IP, IPmax);

System.out.print("\nAfter Permutation : ");

printData( m, 8);

m = fK( m, K1);

System.out.print("\nbefore Swap : ");

printData( m, 8);

m = SW( m);

System.out.print("\nAfter Swap : ");

printData( m, 8);

m = fK( m, K2);

System.out.print("\nbefore IP inverse : ");

printData( m, 8);

m = permute( m, IPI, IPImax);

return (byte) m;

}

public byte decrypt( int m) {

System.out.println("\nDecryption Process Starts........\n\n");

printData( m, 8);

m = permute( m, IP, IPmax);

System.out.print("\nAfter Permutation : ");

printData( m, 8);

m = fK( m, K2);

System.out.print("\nbefore Swap : ");

printData( m, 8);

m = SW( m);

System.out.print("\nAfter Swap : ");

printData( m, 8);

m = fK( m, K1);

System.out.print("\nBefore Extraction Permutation : ");

printData( m, 4);

m = permute( m, IPI, IPImax);

System.out.print("\nAfter Extraction Permutation : ");

printData( m, 8);

return (byte) m;

}

public static void printData( int x, int n)

{

int mask = 1 << (n-1);

while( mask > 0)

{

System.out.print( ((x & mask) == 0) ? '0' : '1');

mask >>= 1;

} }

public SDES( int K) {

K = permute( K, P10, P10max);

int t1 = (K >> 5) & 0x1F;

int t2 = K & 0x1F;

t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);

t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);

K1 = permute( (t1 << 5)| t2, P8, P8max);

t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);

t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);

K2 = permute( (t1 << 5)| t2, P8, P8max);

}}

public class SimplifiedDES

{

public static void main( String args[]) throws Exception

{

DataInputStream inp=new DataInputStream(System.in);

System.out.println("Enter the 10 Bit Key :");

int K = Integer.parseInt(inp.readLine(),2);

SDES A = new SDES( K);

System.out.println("Enter the 8 Bit message To be Encrypt : ");

int m = Integer.parseInt(inp.readLine(),2);

System.out.print("\nKey K1: ");

SDES.printData( A.K1, 8);

System.out.print("\nKey K2: ");

SDES.printData( A.K2, 8);

m = A.encrypt( m);

System.out.print("\nEncrypted Message: ");

SDES.printData( m, 8);

m = A.decrypt( m);

System.out.print("\nDecrypted Message: ");

SDES.printData( m, 8);

}}  
  
**OUTPUT :**

D:\javapgm>java SimplifiedDES

Enter the 10 Bit Key :

1011011010

Enter the 8 Bit message To be Encrypt :

10110110

Key K1: 11110101

Key K2: 01100011

Encryption Process Starts........

After Permutation : 01111001

before Swap : 00001001

After Swap : 10010000

before IP inverse : 10000000

Encrypted Message: 01000000

Decryption Process Starts........

01000000

After Permutation : 10000000

before Swap : 10010000

After Swap : 00001001

Before Extraction Permutation : 1001

After Extraction Permutation : 10110110

Decrypted Message: 10110110

**RESULT:**

Thus the Data Encryption Standard algorithm has been implemented and executed successfully.

**EX NO :4 AES**

**DATE:**

**AIM**:

To Write a JAVA program to implement the AES Algorithm.

**ALGORITHM:**

**STEP 1**: Initialize the variable

**STEP 2:** Encryption Process a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below



### STEP 3: Byte Substitution (SubBytes)

### The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

### STEP 4: Shift rows

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows

* First row is not shifted.
* Second row is shifted one (byte) position to the left.
* Third row is shifted two positions to the left.
* Fourth row is shifted three positions to the left.
* The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

### STEP 5: Mix Columns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

### STEP 6: Add roundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

## STEP 7: Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order

* Add round key
* Mix columns
* Shift rows
* Byte substitution

**PROGRAM:**

***AES.java***

import java.io.UnsupportedEncodingException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import java.util.Arrays;

import java.util.Base64;

import javax.crypto.Cipher;

import javax.crypto.spec.SecretKeySpec;

public class AES {

private static SecretKeySpec secretKey;

private static byte[] key;

public static void setKey(String myKey) {

MessageDigest sha = null;

try {

key = myKey.getBytes("UTF-8");

sha = MessageDigest.getInstance("SHA-1");

key = sha.digest(key);

key = Arrays.copyOf(key, 16);

secretKey = new SecretKeySpec(key, "AES");

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

public static String encrypt(String strToEncrypt, String secret) {

try {

setKey(secret);

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));

} catch (Exception e) {

System.out.println("Error while encrypting: " + e.toString());

}

return null;

}

public static String decrypt(String strToDecrypt, String secret) {

try {

setKey(secret);

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");

cipher.init(Cipher.DECRYPT\_MODE, secretKey);

return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

} catch (Exception e) {

System.out.println("Error while decrypting: " + e.toString());

}

return null;

}

public static void main(String[] args) {

final String secretKey = "annaUniversity";

String originalString = "www.annauniv.edu";

String encryptedString = AES.encrypt(originalString, secretKey);

String decryptedString = AES.decrypt(encryptedString, secretKey);

System.out.println("URL Encryption Using AES Algorithm\n------------");

System.out.println("Original URL : " + originalString);

System.out.println("Encrypted URL : " + encryptedString);

System.out.println("Decrypted URL : " + decryptedString);

}

}

**OUTPUT:**

URL Encryption Using AES Algorithm

-------------------------------------------------

Original URL : www.annauniv.edu

Encrypted URL : vibpFJW6Cvs5Y+L7t4N6YWWe07+JzS1d3CU2h3mEvEg=

Decrypted URL : www.annauniv.edu

**RESULT:**

Thus the program has been successfully executed and verified.

**EX NO 5 IMPLEMENT RSA ALGORITHM USING HTML AND JAVASCRIPT**

**DATE:**

**AIM:**

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

and Javascript.

**ALGORITHM:**

1. Choose two prime number p and q
2. Compute the value of n and p
3. Find the value of ***e*** (public key)
4. Compute the value of ***d*** (private key) using gcd()
5. Do the encryption and decryption
   1. Encryption is given as,

***c = te mod n***

* 1. Decryption is given as,

***t = cd mod n***

**PROGRAM:**

***rsa.html***

<html>

<head>

<title>RSA Encryption</title>

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

</head>

<body>

<center>

<h1>RSA Algorithm</h1>

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

<hr>

<table>

<tr>

<td>Enter First Prime Number:</td>

<td><input type="number" value="53" id="p"></td>

</tr>

<tr>

<td>Enter Second Prime Number:</td>

<td><input type="number" value="59" id="q"></p>

</td>

</tr>

<tr>

<td>Enter the Message(cipher text):<br>[A=1, B=2,...]</td>

<td><input type="number" value="89" id="msg"></p>

</td>

</tr>

<tr>

<td>Public Key:</td>

<td>

<p id="publickey"></p>

</td>

</tr>

<tr>

<td>Exponent:</td>

<td>

<p id="exponent"></p>

</td>

</tr>

<tr>

<td>Private Key:</td>

<td>

<p id="privatekey"></p>

</td>

</tr>

<tr>

<td>Cipher Text:</td>

<td>

<p id="ciphertext"></p>

</td>

</tr>

<tr>

<td><button onclick="RSA();">Apply RSA</button></td>

</tr>

</table>

</center>

</body>

<script type="text/javascript"> function RSA() {

var gcd, p, q, no, n, t, e, i, x;

gcd = function (a, b) { return (!b) ? a : gcd(b, a % b); }; p = document.getElementById('p').value;

q = document.getElementById('q').value;

no = document.getElementById('msg').value; n = p \* q;

t = (p - 1) \* (q - 1);

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

break;

}

}

for (i = 0; i < 10; i++) { x = 1 + i \* t

if (x % e == 0)

{ d = x / e; break;

}

}

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

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

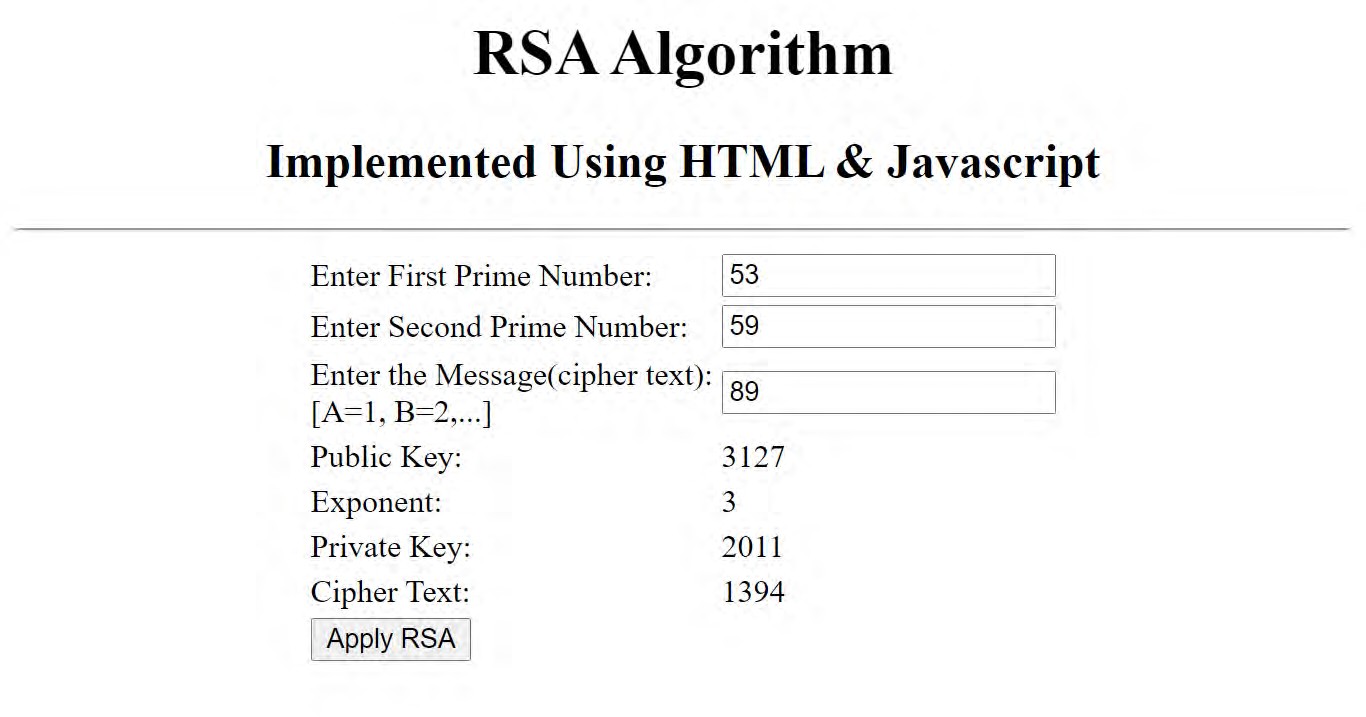
document.getElementById('publickey').innerHTML = n; document.getElementById('exponent').innerHTML = e; document.getElementById('privatekey').innerHTML = d; document.getElementById('ciphertext').innerHTML = ct;

}

</script>

</html>

**OUTPUT:**



**RESULT:**

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

**EXNO 6 DIFFIEE –HELLMAN KEY EXCHANGE ALGORITHM**

**DATE:**

**AIM:**

To implement the Diffie-Hellman Key Exchange algorithm using JAVA language

**ALGORITHM:**

**STEP 1:** Alice and Bob exchange their public keys PA and PB.

**STEP 2:** Alice computes F(SA , PB)

**STEP 3:** Bob computes F(SB, PA)

**STEP 4:** The special property of the public key cipher system, and the choice of the function F, are such that F(SA , PB) = F(SB, PA). If this is the case then Alice and Bob now share a secret.

**STEP 5:** This shared secret can easily be converted by some public means into a bitstring suitable for use as, for example, a DES key.

**STEP 6:** The system parameters (which are public) are:a large prime number p – typically 1024 bits in length**, a** primitive element g

**STEP 7:** Meanwhile Bob generates a private random value b, calculates gb (mod p) and sends it to Alice.

Alice takes gb and her private random value a to compute (gb)a = gab(mod p).

Bob takes ga and his private random value b to compute (ga)b = gab (mod p).

Alice and Bob adopt gab (mod p) as the shared secret.

**PROGRAM:**

import java.util.\*;

class Diffie\_Hellman

{

public static void main(String args[])

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter modulo(p)");

int p=sc.nextInt();

System.out.println("Enter primitive root of "+p);

int g=sc.nextInt();

System.out.println("Choose 1st secret no(Alice)");

int a=sc.nextInt();

System.out.println("Choose 2nd secret no(BOB)");

int b=sc.nextInt();

int A = (int)Math.pow(g,a)%p;

int B = (int)Math.pow(g,b)%p;

int S\_A = (int)Math.pow(B,a)%p;

int S\_B =(int)Math.pow(A,b)%p;

if(S\_A==S\_B)

{

System.out.println("ALice and Bob can communicate with each other!!!");

System.out.println("They share a secret no = "+S\_A);

}

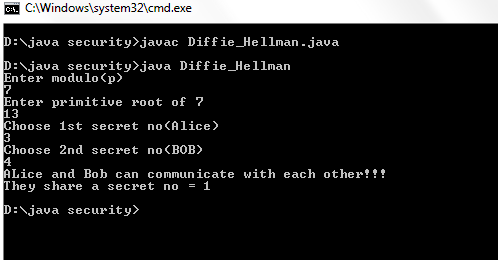
else

{

System.out.println("ALice and Bob cannot communicate with each other!!!");

}}}

**OUTPUT :**



**RESULT:**

Thus the Diffie-Hellman key exchange algorithm has been executed successfully.

**EX NO:7 SHA -1**

**DATE:**

**AIM:**

To implement the SHA-1 algorithm using JAVA program.

**ALGORITHM :**

**STEP 1**: Appending Padding Bits. The original message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. The padding rules are:

The original message is always padded with one bit "1" first. Then zero or more bits "0" are padded to bring the length of the message up to 64 bits fewer than a multiple of 512.

**STEP 2:**Appending Length. 64 bits are appended to the end of the padded message to indicate the length of the original message in bytes. The rules of appending length are: The length of the original message in bytes is converted to its binary format of 64 bits. If overflow happens, only the low-order 64 bits are used.

Break the 64-bit length into 2 words (32 bits each). The low-order word is appended first and followed by the high-order word.

**STEP 3:**Preparing Processing Functions. SHA1 requires 80 processing functions defined as:

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

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

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

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

**STEP 4**: Preparing Processing Constants. SHA1 requires 80 processing constant words defined as:

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

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

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

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

**STEP 5:** Initializing Buffers. SHA1 algorithm requires 5 word buffers with the following initial values:

H0 = 0x67452301

H1 = 0xEFCDAB89

H2 = 0x98BADCFE

H3 = 0x10325476

H4 = 0xC3D2E1F0

**STEP 6**: Processing Message in 512-bit Blocks. This is the main task of SHA1 algorithm, which loops through the padded and appended message in blocks of 512 bits each. For each input block, a number of operations are performed.

**PROGRAM:**

import java.security.\*;

public class SHA1 {

public static void main(String[] a) {

try {

MessageDigest md = MessageDigest.getInstance("SHA1");

System.out.println("Message digest object info: ");

System.out.println(" Algorithm = " +md.getAlgorithm());

System.out.println(" Provider = " +md.getProvider());

System.out.println(" ToString = " +md.toString());

String input = "";

md.update(input.getBytes());

byte[] output = md.digest();

System.out.println();

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

input = "abc";

md.update(input.getBytes());

output = md.digest();

System.out.println();

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

input = "abcdefghijklmnopqrstuvwxyz";

md.update(input.getBytes());

output = md.digest();

System.out.println();

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

System.out.println("");

}

catch (Exception e) {

System.out.println("Exception: " +e);

} }

public static String bytesToHex(byte[] b) {

char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};

StringBuffer buf = new StringBuffer();

for (int j=0; j<b.length; j++) {

buf.append(hexDigit[(b[j] >> 4) & 0x0f]);

buf.append(hexDigit[b[j] & 0x0f]); }

return buf.toString(); } }

**OUTPUT :**

C:\Program Files\Java\jdk1.6.0\_20\bin>javac SHA1.java

C:\Program Files\Java\jdk1.6.0\_20\bin>java SHA1

Message digest object info:

Algorithm = SHA1

Provider = SUN version 1.6

ToString = SHA1 Message Digest from SUN, <initialized>

SHA1("") = DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

SHA1("abc") = A9993E364706816ABA3E25717850C26C9CD0D89D

SHA1("abcdefghijklmnopqrstuvwxyz") = 32D10C7B8CF96570CA04CE37F2A19D84240D3A89

**RESULT:**

Thus the program for implementing SHA – 1 algorithm was written and executed successfully.

**EX NO:8 IMPLEMENTATION OF SIGNATURE SCHEME - DIGITAL DATE: SIGNATURE STANDARD**

**AIM:**

To write a java program to implement the signature scheme named Digital Signature Standard (Euclidean Algorithm).

**ALGORITHM:**

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

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

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

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

**STEP5:** With these values, he applies it to the elliptic curve cryptographic equation to obtain y

**STEP 6:** Comparing this ‘y’ with actual y’s document, Alice concludes that y is a forgery.

**PROGRAM:**

import java.security.KeyPair;

import java.security.KeyPairGenerator;

import java.security.PrivateKey;

import java.security.PublicKey;

class GenerateKeyPairsDSADHRSA {

public static void main(String[] args) {

try {

// Generate a 1024-bit Digital Signature Algorithm (DSA) key pair

KeyPairGenerator keyGen = KeyPairGenerator.getInstance("DSA");

keyGen.initialize(1024);

KeyPair keypair = keyGen.genKeyPair();

PrivateKey privateKey = keypair.getPrivate();

PublicKey publicKey = keypair.getPublic();

System.out.println(privateKey + "n" + publicKey);

// Generate a 576-bit DH key pair

keyGen = KeyPairGenerator.getInstance("DH");

keyGen.initialize(576);

keypair = keyGen.genKeyPair();

privateKey = keypair.getPrivate();

publicKey = keypair.getPublic();

System.out.println(privateKey + "n" + publicKey);

// Generate a 1024-bit RSA key pair

keyGen = KeyPairGenerator.getInstance("RSA");

keyGen.initialize(1024);

keypair = keyGen.genKeyPair();

privateKey = keypair.getPrivate();

publicKey = keypair.getPublic();

System.out.println(privateKey + "n" + publicKey);

}

catch (java.security.NoSuchAlgorithmException e) {

} }}

**OUTPUT:**

C:\Program Files\Java\jdk1.6.0\_20\bin>java GenerateKeyPairsDSADHRSA

Sun DSA Private Key

parameters:

p:

fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669

455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7

6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb

83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7

q:

9760508f 15230bcc b292b982 a2eb840b f0581cf5

g:

f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267

5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1

3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b

cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a

x: 3670ea3a 315796f4 6c3e4162 b8d95b11 3b6f2103

nSun DSA Public Key

Parameters:

p:

fd7f5381 1d751229 52df4a9c 2eece4e7 f611b752 3cef4400 c31e3f80 b6512669

455d4022 51fb593d 8d58fabf c5f5ba30 f6cb9b55 6cd7813b 801d346f f26660b7

6b9950a5 a49f9fe8 047b1022 c24fbba9 d7feb7c6 1bf83b57 e7c6a8a6 150f04fb

83f6d3c5 1ec30235 54135a16 9132f675 f3ae2b61 d72aeff2 2203199d d14801c7

q:

9760508f 15230bcc b292b982 a2eb840b f0581cf5

g:

f7e1a085 d69b3dde cbbcab5c 36b857b9 7994afbb fa3aea82 f9574c0b 3d078267

5159578e bad4594f e6710710 8180b449 167123e8 4c281613 b7cf0932 8cc8a6e1

3c167a8b 547c8d28 e0a3ae1e 2bb3a675 916ea37f 0bfa2135 62f1fb62 7a01243b

cca4f1be a8519089 a883dfe1 5ae59f06 928b665e 807b5525 64014c3b fecf492a

y:

baa058c9 8eb5aeac 9765d9fe 69d899b0 abf75247 6771a3c5 eda49f86 2ec63b66

f633627d 8c48d701 fceb6c66 f7def93c f37eeaff 363a8358 e2441f99 5901766b

2ab9f753 4ed138db dad7bb47 d03e453f 6f3e00f6 d3993f20 402791fd ffcb9be6

e376ed72 e3a14062 09681a97 5177d679 c67db65e 308f6265 80fa0b66 13be04d5

SunJCE Diffie-Hellman Private Key:

x:

5754f83a 7c2a1a8f 723a6725 403ec23b ad848f06 9dd6ce19 a949ba3f f68709db

28c50f19 3bba10db e97c80d4 f090373f

p:

d6a640b1 3ce12e5f e1accc6e ff3883bf 635f47ff 3b07e599 c632d579 951a1131

dae81e18 2443e068 aff6cc02 e862e00c 5ad649a6 e5036b38 eb3e48d1 d78c5d39

352e4345 79e1bd11

g:

b4ef16c9 14c2f66f 18ee2117 db1d4da0 851705a2 ff241f03 35e2c1f6 5b2be728

2689c2d3 dc8254ae c1b6ef36 fc687f5f 41f0279d 106a6807 3a5f7555 709a29b3

b7aba640 1a24388d

l:

384nSunJCE Diffie-Hellman Public Key:

y:

b8c9ac95 8bcbbd64 75d5811a de4d2ce8 9b51c96c bb8e4ca3 2437c439 0eefb87c

5411a909 df79d76e 5a1ac8b3 3e15c5dc fbb82b48 7cad52fe 2672704e eb78a542

9913c979 4dee6680

p:

d6a640b1 3ce12e5f e1accc6e ff3883bf 635f47ff 3b07e599 c632d579 951a1131

dae81e18 2443e068 aff6cc02 e862e00c 5ad649a6 e5036b38 eb3e48d1 d78c5d39

352e4345 79e1bd11

g:

b4ef16c9 14c2f66f 18ee2117 db1d4da0 851705a2 ff241f03 35e2c1f6 5b2be728

2689c2d3 dc8254ae c1b6ef36 fc687f5f 41f0279d 106a6807 3a5f7555 709a29b3

b7aba640 1a24388d

l:

384

Sun RSA private CRT key, 1024 bits

modulus: 115153566492780149823791597948385406618805766082270821869506

22709726281231665780853507026949235755319597905233063196521944472588574442168108

52454481666588291267364364228652838670077265898395994985030743420382621482322775

80304965897704086458974995595681113936450815393132089715040989605182921764891539

355803919

public exponent: 65537

private exponent: 651875630084096549805860549595663302494422070227848008202798

57566084049269084710875552847981544245595172541334916854748331466657935487501232

32076730675865160895123437408244675713248882600092937677778357612326995328541950

03041276338343315712981512900422881419349785331108038517387463573579889492157565

131041

prime p: 129294277070314059708537367360791501153343461020919990387678

77243211817889987613038777518665249265111861542531000606338725725078766824571365

290882157779167

prime q: 890631581706870348247089384980476972921628943580183462114508

63945358339327095896841184329626619011385180890744700329544386042607029898727240

00844944674257

prime exponent p: 159386066565614429770247858600154803823467937438090331010277

67558464421125960896247964290903848026738900072036857637458315933428652085953287

48417488635395

prime exponent q: 315948145980179573773541375275204985633381926381977895092851

53972040771335515090798490844553294557207288562627577367923118713163111496032921

90604454258385

crt coefficient: 695331880513233145593849753856206361120693205989954251077961

39207985190595896421692022549784185872138352480217253264474616074468724265980438

79515139563912nSun RSA public key, 1024 bits

modulus: 115153566492780149823791597948385406618805766082270821869506227097262

81231665780853507026949235755319597905233063196521944472588574442168108524544816

66588291267364364228652838670077265898395994985030743420382621482322775803049658

97704086458974995595681113936450815393132089715040989605182921764891539355803919

public exponent: 65537

**RESULT:**

Thus the program for implementing Digital Signature Standard was written and executed successfully.

**EX.NO: 9 DEMONSTRATE INTRUSION DETECTION SYSTEM DATE: (IDS) USING ANY TOOL**

**AIM:**

To demonstrate the working of Intrusion Detection System (IDS) using the tool snort.

**ALGORITHM:**

SNORT can be configured to run in three modes:

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

|  |  |
| --- | --- |
| 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 transit. |
| 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. |
| Network Intrusion Detection System mode | snort –d c:\log –h ipaddress/24 –c snort.confThis 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 cnfigure snort to run in its most basic NIDS form, logging packets that trigger rules specifies in the snort.conf |

* Download SNORT from snort.org
* Install snort with or without database support.
* Select all the components and Click Next.
* Install and Close.
* Skip the WinPcap driver installation
* Add the path variable in windows environment variable by selecting new classpath.
* Create a path variable and point it at snort.exe variable name🡪path and variable value🡪c:\snort\bin.
* Click OK button and then close all dialog boxes

**STEPS ON CONFIGURING AND INTRUSION DETECTION:**

**1**. Download Snort from the Snort.org website. (http://www.snort.org/snort-downloads)

**2**. Download Rules(https://www.snort.org/snort-rules). You must register to get the rules. (You should download these often)

**3**. Double click on the .exe to install snort. This will install snort in the “C:\Snort” folder.It is important to have WinPcap (https://www.winpcap.org/install/) installed

**4**. Extract the Rules file. You will need WinRAR for the .gz file.

**5**. Copy all files from the “rules” folder of the extracted folder. Now paste the rules into *“C:\Snort\rules”* folder.

**6**. Copy “snort.conf” file from the “etc” folder of the extracted folder. You must paste it into “C:\Snort\etc” folder. Overwrite any existing file. Remember if you modify your snort.conf file and download a new file, you must modify it for Snort to work.

7. Open a command prompt (cmd.exe) and navigate to folder “C:\Snort\bin” folder. ( at the Prompt, type cd\snort\bin)

8. To start (execute) snort in sniffer mode use following command:

snort -dev -i 3

-i indicates the interface number. You must pick the correct interface number. In my case, it is 3.

-dev is used to run snort to capture packets on your network.

To check the interface list, use following command:

snort -W



Finding an interface

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

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

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

var HOME\_NET 192.168.1.0/24 (You will normally see any here)

11. You may also want to set the addresses of DNS\_SERVERS, if you have some on your network.

Example:

example snort

12. Change the RULE\_PATH variable to the path of rules folder.

var RULE\_PATH c:\snort\rules

path to rules

13. Change the path of all library files with the name and path on your system. and you must change the path of snort\_dynamicpreprocessorvariable.

C:\Snort\lib\snort\_dynamiccpreprocessor

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

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

Example:

dynamicengine C:\Snort\lib\snort\_dynamicengine\sf\_engine.dll

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

include c:\snort\etc\classification.config

include c:\snort\etc\reference.config

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

include $RULE\_PATH/icmp.rules

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

include $RULE\_PATH/icmp-info.rules

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

output alert\_fast: snort-alerts.ids

19. Comment (add a #) the whitelist $WHITE\_LIST\_PATH/white\_list.rules and the blacklist

Change the nested\_ip inner , \ to nested\_ip inner #, \

20. Comment out (#) following lines:

#preprocessor normalize\_ip4

#preprocessor normalize\_tcp: ips ecn stream

#preprocessor normalize\_icmp4

#preprocessor normalize\_ip6

#preprocessor normalize\_icmp6

21. Save the “snort.conf” file.

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

snort -c c:\snort\etc\snort.conf -l c:\snort\log -i 3

(Note: 3 is used for my interface card)

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

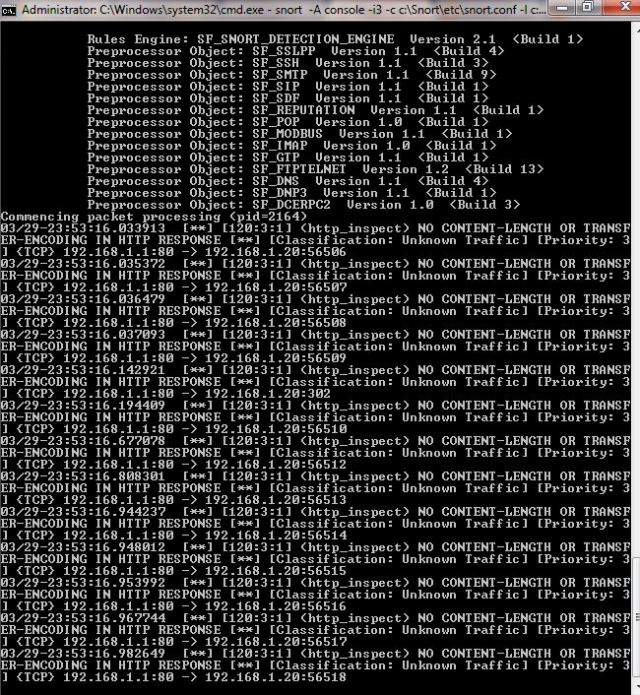
To generate Log files in ASCII mode, you can use following command while running snort in IDS mode:

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

23. Scan the computer that is running snort from another computer by using PING or NMap (ZenMap).

After scanning or during the scan you can check the snort-alerts.ids file in the log folder to insure it is logging properly. You will see IP address folders appear.

Snort monitoring traffic –



**RESULT:**

Thus the working of Intrusion Detection System (IDS) using the tool snort was demonstrated.

**EX.NO: 10. EXPLORING N-STALKER,A VULNERABILITY ASSESSMENT TOOL**

**DATE:**

**AIM:**

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

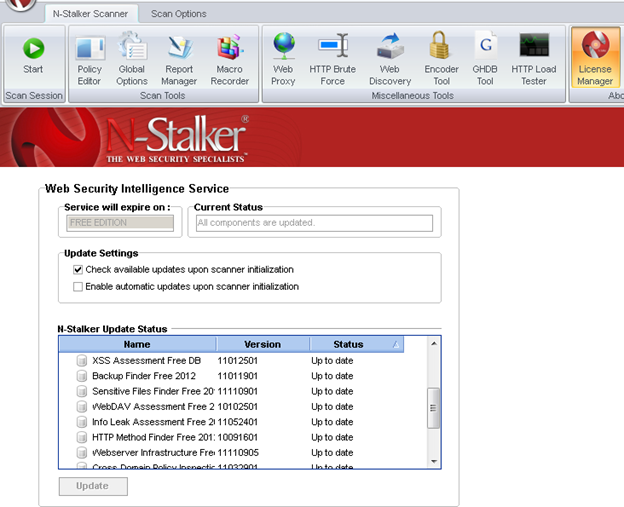
**ALGORITHM:**

* Start N-Stalker from a Windows computer. The program is installed under Start ➪ Programs ➪ N-Stalker ➪ N-Stalker Free Edition.
* Enter a host address or a range of addresses to scan.
* Click Start Scan.
* After the scan completes, the N-Stalker Report Manager will prompt
* you to select a format for the resulting report as choose Generate HTML.
* Review the HTML report for vulnerabilities.

**EXPLORING N-STALKER:**

* N-Stalker Web Application Security Scanner is a Web security assessment tool.
* It incorporates with a well-known N-Stealth HTTP Security Scanner and 35,000 Web attack signature database.
* This tool also comes in both free and paid version.
* Before scanning the target, go to “License Manager” tab, perform the update.
* Once update, you will note the status as up to date.
* You need to download and install N-Stalker from [www.nstalker.com](http://www.nstalker.com).

1. Start N-Stalker from a Windows computer. The program is installed under Start ➪ Programs ➪ N-Stalker ➪ N-Stalker Free Edition.
2. Enter a host address or a range of addresses to scan.
3. Click Start Scan.
4. After the scan completes, the N-Stalker Report Manager will prompt
5. you to select a format for the resulting report as choose Generate HTML.
6. Review the HTML report for vulnerabilities.



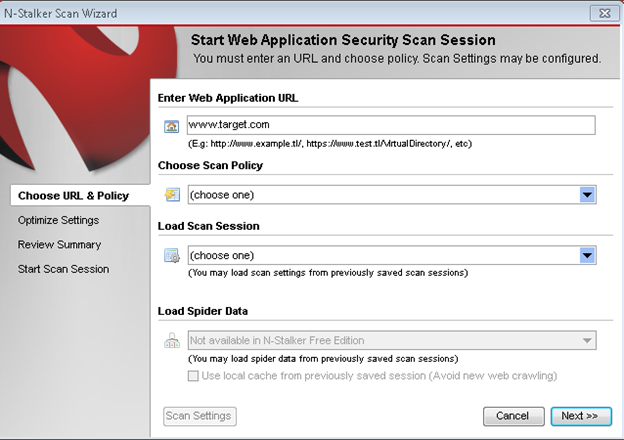
Now goto “Scan Session”, enter the target URL.

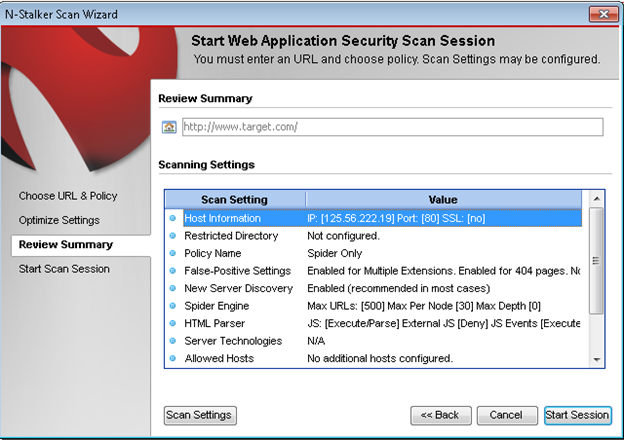
In scan policy, you can select from the four options,

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

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

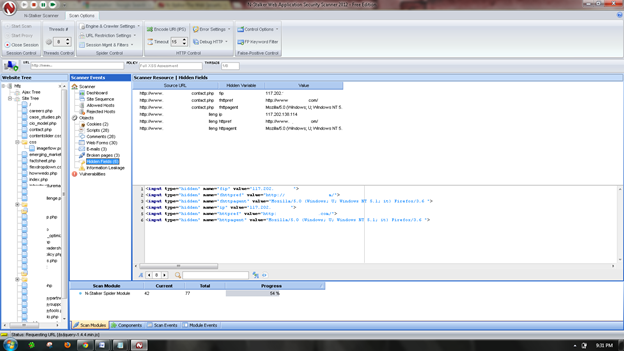
In review option, you can get all the information like host information, technologies used, policy name, etc.



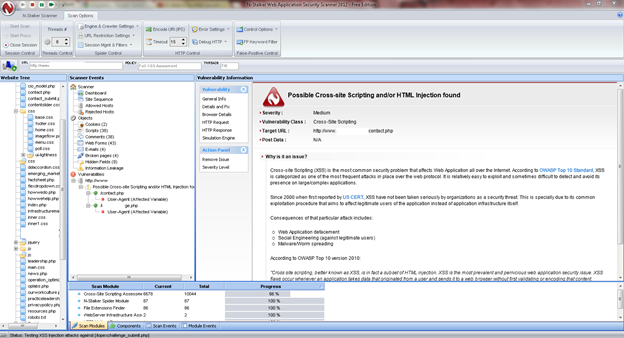


Once done, start the session and start the scan.

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



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



**RESULT:**

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

**EX.NO: 11 i. DEFEATING MALWARE - BUILDING TROJANS**

**DATE:**

**AIM:**

To build a Trojan and know the harmness of the trojan malwares in a computer system.

**ALGORITHM:**

1. Create a simple trojan by using Windows Batch File (***.bat***)
2. Type these below code in notepad and save it as **Trojan.bat**
3. Double click on ***Trojan.bat***file.
4. When the trojan code executes, it will open MS-Paint, Notepad, Command Prompt, Explorer, etc., infinitely.
5. Restart the computer to stop the execution of this trojan.

**TROJAN:**

* In computing, a Trojan horse,or trojan, is any malware which misleads users of its true intent.
* Trojans are generally spread by some form of social engineering, for example where a user is duped into executing an email attachment disguised to appear not suspicious, (e.g., a routine form to be filled in), or by clicking on some fake advertisement on social media or anywhere else.
* Although their payload can be anything, many modern forms act as a backdoor, contacting a controller which can then have unauthorized access to the affected computer.
* Trojans may allow an attacker to access users' personal information such as banking information, passwords, or personal identity.
* ***Example:*** *Ransomware* attacks are often carried out using a *trojan*.

**CODE:**

***Trojan.bat***

@echo off

:x

start mspaint

start notepad

start cmd

start explorer

start control

start calc

goto x

**OUTPUT:**

(MS-Paint, Notepad, Command Prompt, Explorer will open infinitely)

**RESULT:**

Thus a trojan has been built and the harmness of the trojan viruses has been explored.

**EX.NO: 11 ii. DEFEATING MALWARE - ROOTKIT HUNTER**

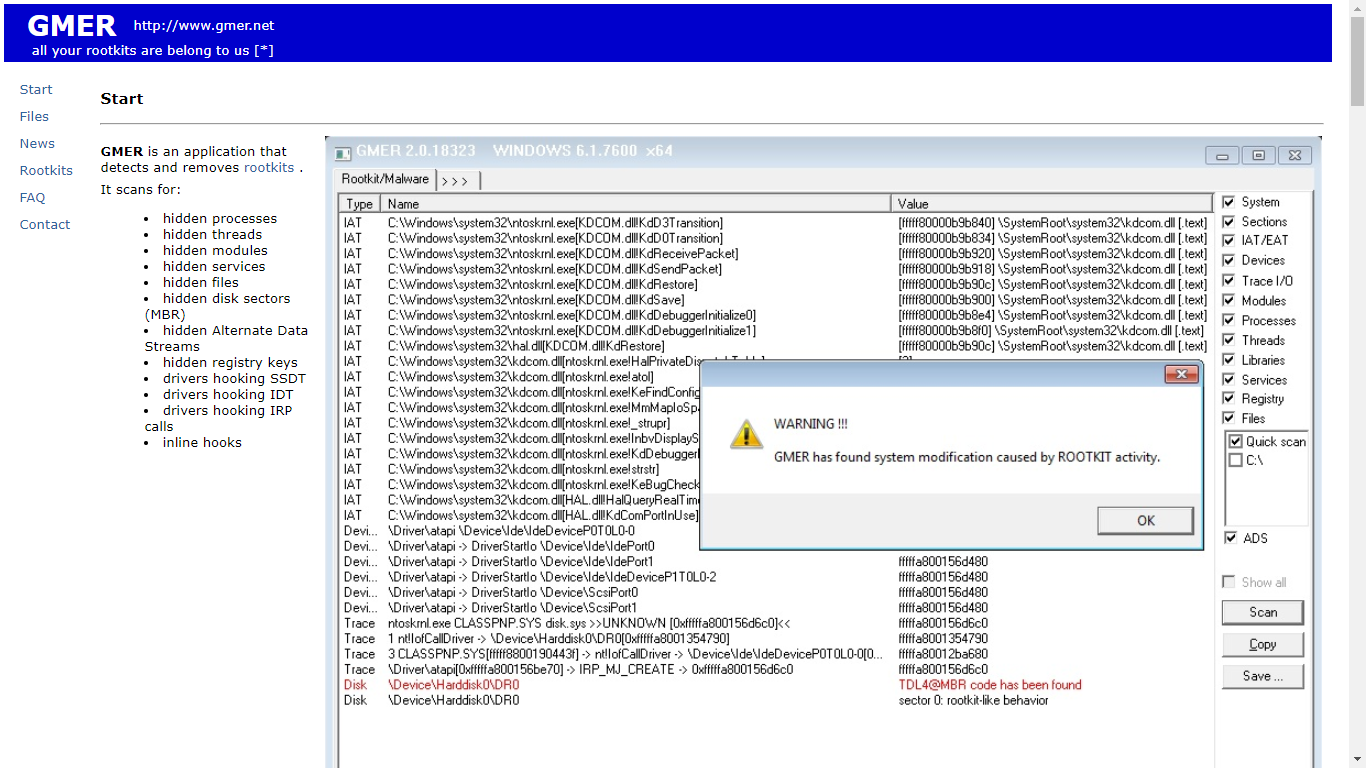
**DATE:**

**AIM:**

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

**ALGORITHM:**

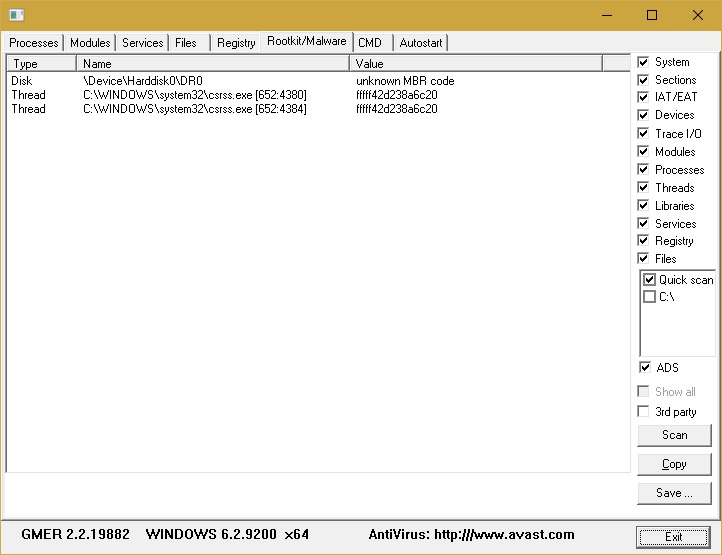
**Step 1**



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

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

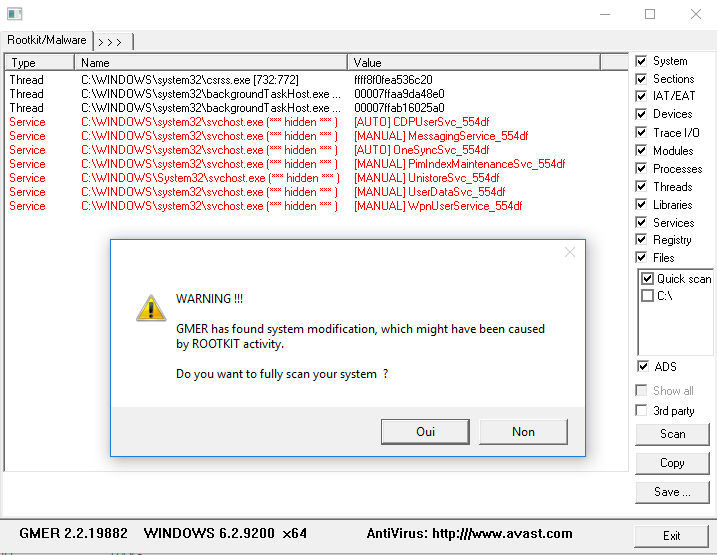
**Step 2**



Double-click the icon for the program.

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

**Step 3**



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

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

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

**ROOTKIT HUNTER:**

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

**GMER ROOTKIT TOOL:**

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

**RESULT:**

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

**ADDITIONAL EXPERIMENTS**

**EX.NO: 12 IMPLEMENTATION OF AFFINE CIPHER**

**DATE:**

**AIM**:

To Write a JAVA program to generate Affine cipher.

**ALGORITHM:**

**STEP 1**: In the affine cipher the letters of an alphabet of size *m* is first mapped to the integers in the range 0 … *m* − 1. It then uses [modular arithmetic](https://en.wikipedia.org/wiki/Modular_arithmetic" \o "Modular arithmetic) to transform the integer that each plaintext letter corresponds to into another integer that corresponds to a ciphertext letter. The encryption function for a single letter is

\mbox{E}(x)=(ax+b)\mod{m},

where modulus *m* is the size of the alphabet and *a* and *b* are the key of the cipher.

**STEP 2**:The value *a* must be chosen such that *a* and *m* are [coprime](https://en.wikipedia.org/wiki/Coprime" \o "Coprime). The decryption function is

\mbox{D}(x)=a^{-1}(x-b)\mod{m},

where *a*−1 is the [modular multiplicative inverse](https://en.wikipedia.org/wiki/Modular_multiplicative_inverse" \o "Modular multiplicative inverse) of *a* [modulo](https://en.wikipedia.org/wiki/Modular_arithmetic" \o "Modular arithmetic) *m*. i.e., it satisfies the equation

1 = a a^{-1}\mod{m}.

**STEP 3**: The multiplicative inverse of *a* only exists if *a* and *m* are coprime. Hence without the restriction on *a* decryption might not be possible. It can be shown as follows that decryption function is the inverse of the encryption function,


\begin{align}
\mbox{D}(\mbox{E}(x)) &= a^{-1}(\mbox{E}(x)-b)\mod{m}\\
 &= a^{-1}(((ax+b)\mod{m})-b)\mod{m} \\
 &= a^{-1}(ax+b-b)\mod{m} \\
 &= a^{-1}ax \mod{m}\\
 &= x\mod{m}.
\end{align}

**PROGRAM**:

class GFG

{

static int a = 17;

static int b = 20;

static String encryptMessage(char[] msg)

{

/// Cipher Text initially empty

String cipher = "";

for (int i = 0; i < msg.length; i++)

{

// Avoid space to be encrypted

/\* applying encryption formula ( a x + b ) mod m

{

here x is msg[i] and m is 26} and added 'A' to

bring it in range of ascii alphabet[ 65-90 | A-Z ] \*/

if (msg[i] != ' ')

{

cipher = cipher + (char) ((((a \* (msg[i] - 'A')) + b) % 26) + 'A');

}

else // else simply append space character

{

cipher += msg[i];

}

}

return cipher;

}

static String decryptCipher(String cipher)

{

String msg = "";

int a\_inv = 0;

int flag = 0;

//Find a^-1 (the multiplicative inverse of a

//in the group of integers modulo m.)

for (int i = 0; i < 26; i++)

{

flag = (a \* i) % 26;

// Check if (a\*i)%26 == 1,

// then i will be the multiplicative inverse of a

if (flag == 1)

{

a\_inv = i;

}

}

for (int i = 0; i < cipher.length(); i++)

{

/\*Applying decryption formula a^-1 ( x - b ) mod m

{

here x is cipher[i] and m is 26} and added 'A' to bring it in range of ASCII alphabet[ 65-90 | A-Z ] \*/

if (cipher.charAt(i) != ' ')

{

msg = msg + (char) (((a\_inv \* ((cipher.charAt(i) + 'A' - b)) % 26)) + 'A');

}

else //else simply append space characte

{

msg += cipher.charAt(i);

}

}

return msg;

}

public static void main(String[] args) {

String msg = "AFFINE CIPHER";

// Calling encryption function

String cipherText = encryptMessage(msg.toCharArray());

System.out.println("Encrypted Message is : " + cipherText);

// Calling Decryption function

System.out.println("Decrypted Message is: " + decryptCipher(cipherText));

}

}

**OUTPUT:**

Encrypted Message is : UBBAHK CAPJKX

Decrypted Message is : AFFINE CIPHER

**RESULT:**

Thus the program has been successfully executed and verified.

**EX.NO: 13 BLOW FISH ALGORITHM LOGIC**

**DATE:**

**AIM**:

To Write a JAVA program to implement the BlowFish Algorithm.

**ALGORITHM:**

**STEP 1**: Initialize the variable

**STEP 2**: **Fast:** It encrypts data on large 32-bit microprocessors at a rate of 26 clock cycles per byte.

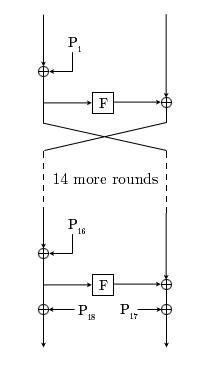
**STEP 3**: **Compact:** It can run in less than 5K of memory.

**STEP 4**: **Simple:** It uses addition, XOR, lookup table with 32-bit operands.

**STEP 5**: **Secure:** The key length is variable, it can be in the range of 32~448 bits: default 128 bits key length.

**STEP 6**: It is suitable for applications where the key does not change often, like communication link or an automatic file encryptor.

**STEP 7**: Unpatented and royalty-free.



**Fig 1: The Feistel structure of Blowfish**

**STEP 8**: Blowfish symmetric block cipher algorithm encrypts block data of 64-bits at a time.it will follow the feistel network and this algorithm is divided into two parts.

                                              1. Key-expansion

                                    2. Data Encryption

**STEP 9**:**Key-expansion:**

It will convert a key of at most 448 bits into several subkey arrays totaling 4168 bytes. Blowfish uses large number of subkeys. These keys are used to generate any data encryption or decryption.

The p-array consists of 18, 32-bit subkeys:

                                          P1,P2,………….,P18

                           Four 32-bit S-Boxes consist of 256 entries each:

                                          S1,0, S1,1,………. S1,255

                                          S2,0, S2,1,……….. S2,255

                                          S3,0, S3,1,……….. S3,255

                                           S4,0, S4,1,..............S4,255

**STEP 10**:**Data Encryption:**

Data encryption consists of a function to iterate 16 times of network. Each round consists of key-dependent permutation and a key and data-dependent substitution. All operations are XORs and additions on 32-bit words. The only additional operations are four indexed array data lookup tables for each round.

                ----------------------------------------------------

                  Algorithm:Blowfish Encryption

                     ------------------------------------------------------------------

                Divide x into two 32-bit halves: xL, xR

For i = 1to 16:

                                        xL = XL XOR Pi

                                        xR = F(XL) XOR xR

                                        Swap XL and xR

                                        Swap XL and xR (Undo the last swap.)

                                        xR = xR XOR P17

                             xL = xL XOR P18

                            Recombine xL and xR

**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 Blowfish {

byte[] skey = new byte[1000];

String skeyString;

static byte[] raw;

String inputMessage,encryptedData,decryptedMessage;

public Blowfish() {

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("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;

}

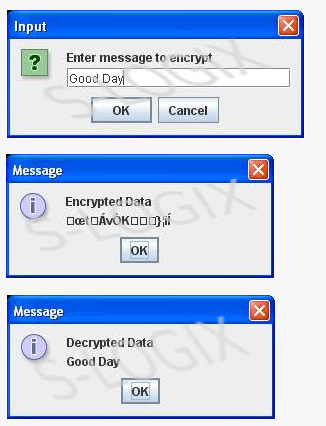
public static void main(String args[]) {

Blowfish bf = new Blowfish();

}

}

**OUTPUT:**



**RESULT:**

Thus the program has been successfully executed and verified.

**EX NO 14 RC5 ALGORITHM**

**DATE:**

**AIM**:

To write a JAVA program to implement the RC5 Algorithm.

**ALGORITHM:**

**STEP 1**: Initialize the variable

**STEP 2**: RC5 encryption and decryption both expand the random key into 2(r+1) words that will be used sequentially (and only once each) during the encryption and decryption processes

### STEP 3: Key expansion

* b - The length of the key in bytes.
* K - The key, considered as an array of bytes (using 0-based indexing).
* w - The length of a word in bits. Typical values of this in RC5 are 16, 32, and 64.

Note that a "block" is two words long.

* u - The length of a word in bytes.
* r - The number of rounds to use when encrypting data.
* S - The expanded list of words derived from the key, of length 2(r+1), with each

element being a word.

* L - A convenience to encapsulate K as an array of word-sized values rather than byte-

sized.

* Pw - The first magic constant, defined as Odd((e - 2) * 2^w), where Odd is the nearest odd integer (rounded up) for the given input, where *e* is the base of the natural

logarithm, and *w* is defined above. For common values of *w*, the associated values of

Pw are given here in hexadecimal:

* + - For *w* = 16: 0xB7E1
    - For *w* = 32: 0xB7E15163
    - For *w* = 64: 0xB7E151628AED2A6D
* Qw - The second magic constant, defined as Odd((\phi - 1) * 2^w), where Odd is the

nearest odd integer (rounded up) for the given input, where \phi is the golden ratio, and *w* is defined above. For common values of *w*, the associated values of Qw are given here in hexadecimal:

* + - For *w* = 16: 0x9E37
    - For *w* = 32: 0x9E3779B9
    - For *w* = 64: 0x9E3779B97F4A7C15

### STEP 4: Encryption

Encryption involved several rounds of a simple function. 12 or 20 rounds seem to be recommended, depending on security needs and time considerations. Beyond the variables used above, the following variables are used in this algorithm:

* A, B - The two words composing the block of plaintext to be encrypted.

A = A + S[0]

B = B + S[1]

**for** i = 1 to r do:

A = ((A ^ B) <<< B) + S[2 \* i]

B = ((B ^ A) <<< A) + S[2 \* i + 1]

*# The ciphertext block consists of the two-word wide block composed of A and B, in that order.*

**return** A, B

### STEP 5: Decryption

Decryption is a fairly straight-forward reversal of the encryption process

**for** i = r down to 1 do:

B = ((B - S[2 \* i + 1]) >>> A) ^ A

A = ((A - S[2 \* i]) >>> B) ^ B

B = B - S[1]

A = A - S[0]

**return** A, B

**PROGRAM:**

import java.io.\*;

class RC5Enc {

public void encrypt() throws Exception{

KeyExp ke = new KeyExp();

String s[] = ke.compute();

System.out.println("Enter Input");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("a = ");

String a = fullfill0(Long.toBinaryString(Long.parseLong(br.readLine(), 16)));

System.out.print("b = ");

String b = fullfill0(Long.toBinaryString(Long.parseLong(br.readLine(), 16)));

a = add(a, fullfill0(s[0]));

b = add(b, fullfill0(s[1]));

int tmp = 0;

for (int i = 1; i <= 12; i++) {

tmp = Integer.parseInt(""+Long.parseLong(b,2)%32);

a = xor(a, b);

a = a.substring(tmp)+a.substring(0,tmp);

a = add(a, fullfill0(s[2 \* i]));

tmp = Integer.parseInt(""+Long.parseLong(a,2)%32);

b = xor(b, a);

b = b.substring(tmp)+b.substring(0,tmp);

b = add(b, fullfill0(s[(2 \* i)+1]));

System.out.println(i+" iteration = "+(Long.toHexString(Long.parseLong(a,2)))+(Long.toHexString(Long.parseLong(b,2))));

}

String out = a+b;

System.out.println("Output = "+(Long.toHexString(Long.parseLong((out.substring(0,32)),2)))+(Long.toHexString(Long.parseLong((out.substring(32)),2))));

}

public String fullfill0(String x) {

return (get0(32-x.length())+ x);

}

public String xor(String x, String y) {

String result = "";

for (int i = 0; i < x.length(); i++) {

if (x.charAt(i) == y.charAt(i)) {

result += "0";

} else {

result += "1";

}

}

return result;

}

public String get0(int len) {

String result = "";

for (int i = 0; i < len; i++) {

result += "0";

}

return result;

}

public String add(String x, String y) {

String result = "";

boolean carry = false;

for (int i = x.length()- 1; i >= 0; i--) {

if((x.charAt(i) == y.charAt(i) && carry == false)|| (x.charAt(i) != y.charAt(i) && carry == true)){

result = "0"+result;

}else{

result = "1"+result;

}

if((x.charAt(i) == '1' && y.charAt(i) == '1') ||

(x.charAt(i) == '1' && y.charAt(i) == '1' && carry == true) ||

(x.charAt(i) != y.charAt(i) && carry == true)){carry = true;}

else{carry = false;}

}

return result;

}

}

class RC5Dec {

public void decrypt() throws Exception{

KeyExp ke = new KeyExp();

String s[] = ke.compute();

System.out.println("Enter Input");

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.print("a = ");

String a = fullfill0(Long.toBinaryString(Long.parseLong(br.readLine(), 16)));

System.out.print("b = ");

String b = fullfill0(Long.toBinaryString(Long.parseLong(br.readLine(), 16)));

int tmp = 0;

for (int i = 12; i >= 1; i--) {

b = fullfill0(Long.toBinaryString((Long.parseLong(b, 2) - Long.parseLong(s[(2\*i)+1], 2))));

b = b.substring(b.length()-32);

tmp = Integer.parseInt(""+Long.parseLong(a,2)%32);

b = b.substring(b.length()-tmp) + b.substring(0,b.length()-tmp);

b = xor(b, a);

a = fullfill0(Long.toBinaryString((Long.parseLong(a, 2) - Long.parseLong(s[2\*i], 2) )));

a = a.substring(a.length()-32);

tmp = Integer.parseInt(""+Long.parseLong(b,2)%32);

a = a.substring(a.length()-tmp) + a.substring(0,a.length()-tmp);

a = xor(a, b);

System.out.println(i+ " iteration = "+(Long.toHexString(Long.parseLong(a,2)))+(Long.toHexString(Long.parseLong(b,2))));

}

a = fullfill0(Long.toBinaryString((Long.parseLong(a, 2) - Long.parseLong(s[0], 2))));

b = fullfill0(Long.toBinaryString((Long.parseLong(b, 2) - Long.parseLong(s[1], 2))));

String output = a+b;

output = output.substring(output.length()-64);

System.out.println("Output = "+(Long.toHexString(Long.parseLong(output.substring(0,32),2)))+(Long.toHexString(Long.parseLong(output.substring(32),2))));

}

public String fullfill0(String x) {

return (get0(32-x.length())+ x);

}

public String xor(String x, String y) {

String result = "";

for (int i = 0; i < x.length(); i++) {

if (x.charAt(i) == y.charAt(i)) {

result += "0";

} else {

result += "1";

}

}

return result;

}

public String get0(int len) {

String result = "";

for (int i = 0; i < len; i++) {

result += "0";

}

return result;

}

}

public class RC5 {

public static void main(String[] args) throws Exception{

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

System.out.println("Encryption");

RC5Enc enc = new RC5Enc();

enc.encrypt();

System.out.println("Encryption");

RC5Dec dec = new RC5Dec();

dec.decrypt();

}

}

**OUTPUT:**

>javac RC5.java

>java RC5

Encryption

Enter uKey = 123

l = 123

l = 0

l = 0

l = 0

Enter Input

a = 111

b = 123

1 iteration = 3a8e2be0fa44f45f

2 iteration = 585f644077eee7c2

3 iteration = cb19c548be8e6d6e

4 iteration = fd4dc4939e36a5bd

5 iteration = 680e24328f8c32bb

6 iteration = 73b2a94ebe2b0631

7 iteration = c21f388f8cc2f647

8 iteration = c0e220d585ded068

9 iteration = 7b1bd4eca0360a74

10 iteration = dd7c9c941196a162

11 iteration = 6d7af34f1de7a1dc

12 iteration = 9ab6e605d2773155

Output = 9ab6e605d2773155

Encryption

Enter uKey = 123

l = 123

l = 0

l = 0

l = 0

Enter Input

a = 111

b = 123

12 iteration = ffb123b1e57f0ac9

11 iteration = 1b17a746ec00dbe6

10 iteration = 64190cb3ae366f7a

9 iteration = 9fa04737287477a

8 iteration = 8b22c57369f93ad

7 iteration = a229b9b27f052dc5

6 iteration = 7b7a06054b816047

5 iteration = 605c4d49d6bae36

4 iteration = 6d0a44092b03c2da

3 iteration = 227deb49793b8ff9

2 iteration = 169b1a0192c5f55

1 iteration = f188e3bcb06894ff

Output = eb908651abf6fd3b

**RESULT:**

Thus the program has been successfully executed and verified.

**EX NO : 15 HONEY POT SETUP AND MONITOR THE**

**DATE: HONEYPOT ON NETWORK USING KF SENSOR**

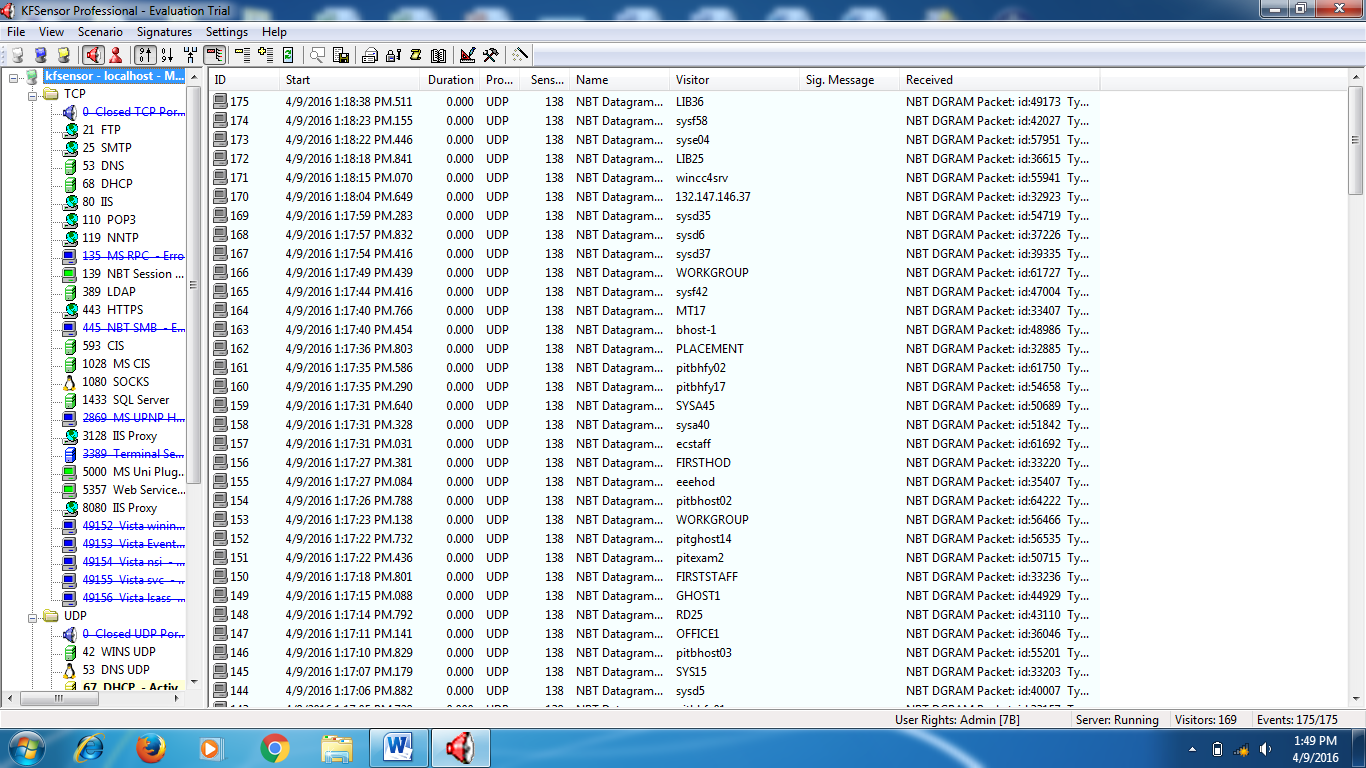
**AIM:**

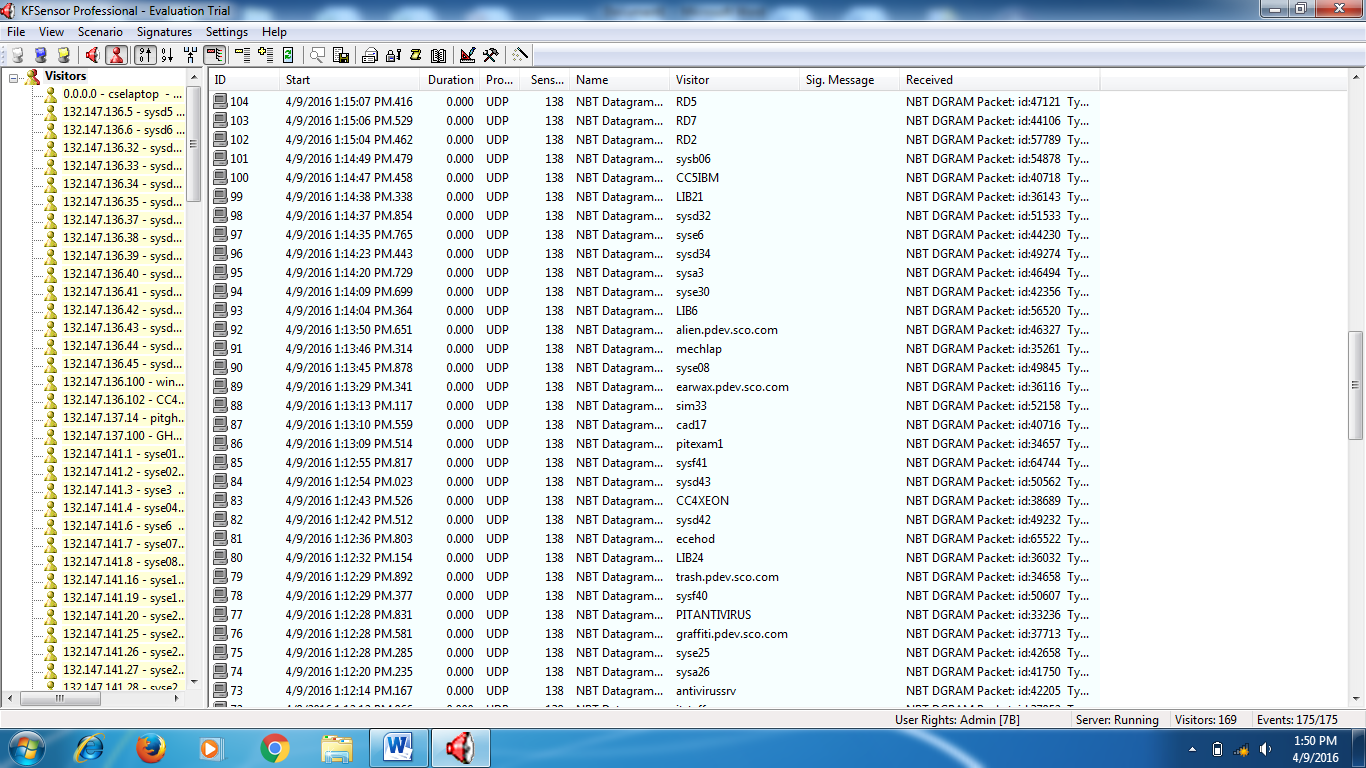
To setup Honey Pot and monitor the Honey Pot on network.

**ALGORITHM:**

Honey Pot is a device placed on Computer Network specifically designed to capture malicious network traffic. KF Sensor is the tool to setup as honeypot when KF Sensor is running it places a siren icon in the windows system tray in the bottom right of the screen. If there are no alerts then green icon is displayed.

* Honey Pot is a device placed on Computer Network specifically designed to capture malicious network traffic.
* KF Sensor is the tool to setup as honeypot when KF Sensor is running it places a siren icon in the windows system tray in the bottom right of the screen. If there are no alerts then green icon is displayed.
* Download KF Sensor Evaluation Set File from KF Sensor Website.
* Install with License Agreement and appropriate directory path. Reboot the Computer now.
* The KF Sensor automatically starts during windows boot Click Next to setup wizard. Select all port classes to include and Click Next.
* Send the email and Send from email enter the ID and Click Next.
* Select the options such as Denial of Service[DOS], Port Activity, Proxy Emulsion, Network Port Analyzer, Click Next.
* Select Install as System service and Click Next. Click finish.





**RESULT:**

Thus the Honey Pot was created and monitored on the network.