Contents

[Practical 1 2](#_Toc21196002)

[AIM:-Introduction to Information And Network Security 2](#_Toc21196003)

[Practical 2 6](#_Toc21196004)

[AIM:- Write a program to implement Caeser Cipher. 6](#_Toc21196005)

[Practical 3 8](#_Toc21196006)

[AIM:- Write a program to implement Playfair Cipher. 8](#_Toc21196007)

[Practical 4 14](#_Toc21196008)

[AIM:- Write a program to implement Columnar Cipher 14](#_Toc21196009)

[Practical 5 19](#_Toc21196010)

[AIM:- Write a program to implement Hill Cipher 19](#_Toc21196011)

[Practical 6 28](#_Toc21196012)

[AIM:- Write a program to implement Vigenere Cipher 28](#_Toc21196013)

[Practical 7 30](#_Toc21196014)

[AIM:- Write a program to implement RailFence Cipher 30](#_Toc21196015)

[Practical 8 33](#_Toc21196016)

[AIM:- Write a program to implement Vernum Cipher 33](#_Toc21196017)

[Practical 9 35](#_Toc21196018)

[AIM:- Write a program to implement Diffie Hellman Key Exchange 35](#_Toc21196019)

[Practical 10 37](#_Toc21196020)

[AIM: - Case study of DES Algorithm 37](#_Toc21196021)

[Practical 11 45](#_Toc21196022)

[AIM: - Case study of RSA Algorithm 45](#_Toc21196023)

# Practical 1

# AIM:-Introduction to Information And Network Security

1. What is cryptography?

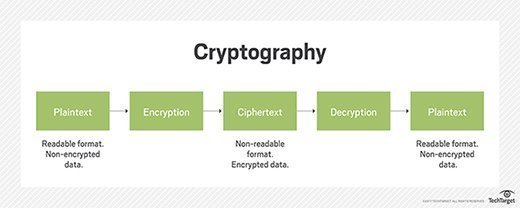
Cryptography is a method of protecting information and communications through the use of codes so that only those for whom the information is intended can read and process it. The pre-fix "crypt" means "hidden" or "vault" and the suffix "graphy" stands for "writing."[1]

Cryptography techniques

Cryptography is closely related to the disciplines of [cryptology](https://searchsecurity.techtarget.com/definition/cryptology) and [cryptanalysis](https://searchsecurity.techtarget.com/definition/cryptanalysis). It includes techniques such as microdots, merging words with images, and other ways to hide information in storage or transit. However, in today's computer-centric world, cryptography is most often associated with scrambling [plaintext](https://searchsecurity.techtarget.com/definition/plaintext) (ordinary text, sometimes referred to as cleartext) into [ciphertext](https://whatis.techtarget.com/definition/ciphertext) (a process called [encryption](https://searchsecurity.techtarget.com/definition/encryption)), then back again (known as decryption). Individuals who practice this field are known as cryptographers.[1]

Modern cryptography concerns itself with the following four objectives:[1]

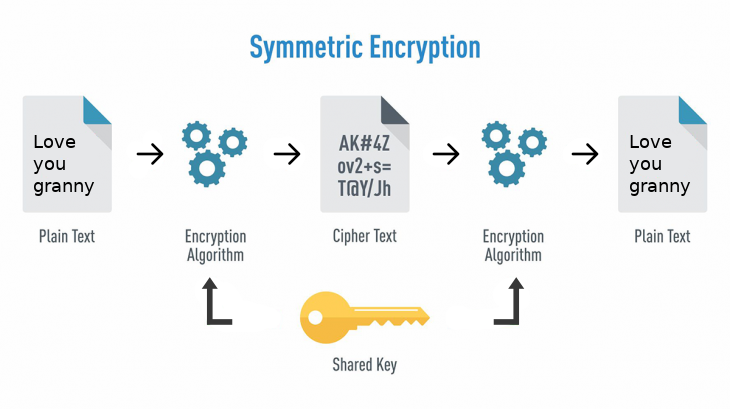
1. **Confidentiality**: the information cannot be understood by anyone for whom it was unintended
2. **Integrity:**the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected
3. **Non-repudiation**: the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information
4. **Authentication**: the sender and receiver can confirm each other's identity and the origin/destination of the information



1. What is symmetric and asymmetric key cryptography?

Symmetric encryption

In **symmetric encryption**, you use the same key for both **encryption** and **decryption** of your data or message. Taking the example I gave above, sending a secure message to your granny, both of you need to have the same key in order to **encrypt** and **decrypt** the messages that you may exchange with each other.[2]



Asymmetric encryption

**Asymmetric encryption** is quite the opposite to the **symmetric encryption** as it uses not one key but a pair of keys: a **private** one and a **public** one. One might ask:[2]

You use one to **encrypt** your data, which is called **public key,** and the other to **decrypt**the encrypted message, which is called the **private key.**

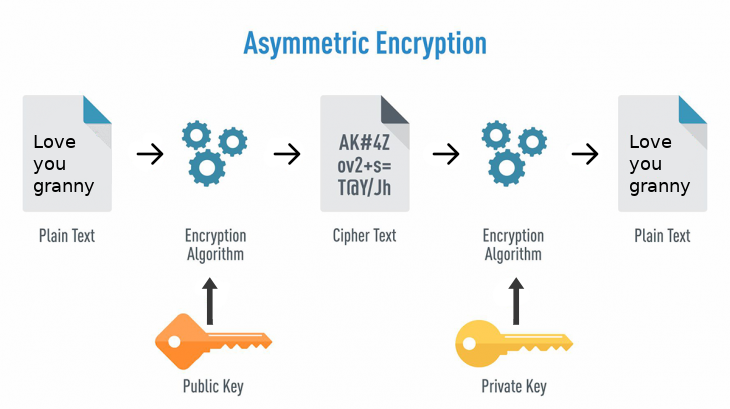
When you encrypt your message using, let’s say, your granny’s public key, that same message can only be decrypted using her private key.

#### Private keys

Your **private key**, as the name states, is yours and it must be kept private, as it’s the only key that can **decrypt** any messaged that was **encrypted** with your **public key**.

#### Public keys

**Public keys**as, yet again, the name states, are public and thus no security is required because of it should publicly available and can be passed over the internet. The **public key**is used to **encrypt**a message that can only be **decrypted** using, as I written above, its **private** counterpart.



Terms used:[3]

* **Encryption**: It is the process of locking up information using cryptography. Information that has been locked this way is encrypted.
* **Decryption**: The process of unlocking the encrypted information using cryptographic techniques.
* **Key**: A secret like a password used to encrypt and decrypt information. There are a few different types of keys used in cryptography.
* **Steganography**: It is actually the science of hiding information from people who would snoop on you. The difference between steganography and encryption is that the would-be snoopers may not be able to tell there’s any hidden information in the first place.

Difference Between Symmetric and Asymmetric Encryption

* Symmetric encryption uses a single key that needs to be shared among the people who need to receive the message while asymmetrical encryption uses a pair of public key and a private key to encrypt and decrypt messages when communicating.
* Symmetric encryption is an old technique while asymmetric encryption is relatively new.
* Asymmetric encryption was introduced to complement the inherent problem of the need to share the key in symmetrical encryption model, eliminating the need to share the key by using a pair of public-private keys.
* Asymmetric encryption takes relatively more time than the symmetric encryption.

Reference:

1. <https://searchsecurity.techtarget.com/definition/cryptography>
2. <https://hackernoon.com/symmetric-and-asymmetric-encryption-5122f9ec65b1>
3. <https://www.ssl2buy.com/wiki/symmetric-vs-asymmetric-encryption-what-are-differences>

# Practical 2

# AIM:- Write a program to implement Caeser Cipher.

**Program:-**

import java.io.\*;

class cipher

{

// Encrypts text using a shift od s

public static StringBuffer encrypt(String text, int s)

{

StringBuffer result= new StringBuffer();

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

{

if (Character.isUpperCase(text.charAt(i)))

{

char ch = (char)(((int)text.charAt(i) +

s - 65) % 26 + 65);

result.append(ch);

}

else

{

char ch = (char)(((int)text.charAt(i) +

s - 97) % 26 + 97);

result.append(ch);

}

}

return result;

}

// Driver code

public static void main(String[] args)

{

String text = "ATTACKATONCE";

int s = 4;

System.out.println("Text : " + text);

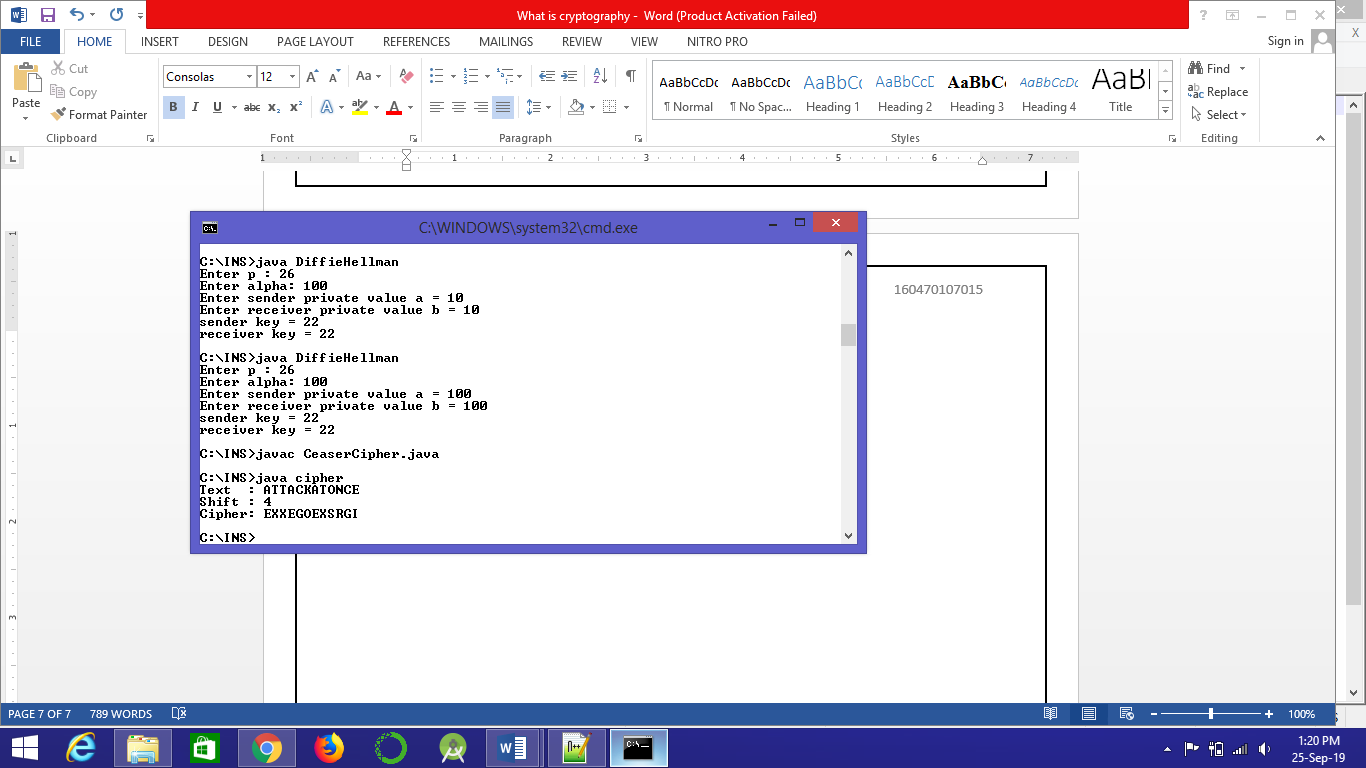
System.out.println("Shift : " + s);

System.out.println("Cipher: " + encrypt(text, s));

}

}

**OUTPUT:-**



# Practical 3

# AIM:- Write a program to implement Playfair Cipher.

**Program:-**

import java.util.\*;

class Playfair{

public static void main(String args[]){

Scanner s= new Scanner(System.in);

System.out.println("Enter Plain Text:");

String plainText=s.nextLine();

System.out.println("\nEnter Key:");

String key=s.nextLine();

char[][] matrix=new char[5][5];

int l=key.length();

int count=0;

char[] keychar=new char[l];

int p=0;

String flag="";

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

{

if(key.charAt(i)=='j'){

keychar[i]='i';

}

else{

keychar[i]=key.charAt(i);

}

}

key=String.valueOf(keychar);

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

if(flag.indexOf(key.charAt(i))==-1)

{

keychar[p]=key.charAt(i);

p++;

flag=flag+key.charAt(i);

}

}

key=String.valueOf(keychar);

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

if(count<flag.length())

{

matrix[i][j]=keychar[count];

count++;

}

}

}

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

int i=0;

int j=0;

l=flag.length();

char a='a';

i=l/5;

int q=l%5;

for(p=i;p<5;p++)

{

while(q<5)

{

if(a=='j'){

a++;

}

if(key.indexOf(a)==-1)

{

matrix[p][q]=a;

q++;

}

a++;

}

q=0;

}

for (int m =0 ; m < 5 ; m++ ) {

for (int k = 0 ; k < 5 ; k++ ) {

System.out.print(matrix[m][k]);

}

System.out.println(" ");

}

String cipherText = encrypt(matrix,plainText);

System.out.println("Cipher Text =" + cipherText);

String decPlainText = decrypt(matrix,cipherText);

System.out.println("Decrypt Plain Text =" + decPlainText);

}

public static String encrypt(char matrix[][], String pt){

int length = pt.length();

String ct ="";

int l,k,m,n;

l =k = m =n = 0;

for(int p = 0; p < length ; p = p + 2){

for (int i =0 ; i < 5 ; i++ ) {

for (int j = 0 ; j < 5 ; j++ ) {

if(pt.charAt(p)==matrix[i][j]){

l = i;

k = j;

}

if(pt.charAt(p + 1)==matrix[i][j]){

m = i;

n = j;

}

}

}

if(l == m){

ct = ct+matrix[l][(k+1) % 5]+matrix[l][(n+1) % 5];

}

else if(k == n){

ct = ct+matrix[(l+1)%5][k] + matrix[(m+1) % 5][k];

}

else{

ct = ct + matrix[l][n] + matrix[m][k];

}

}

return ct;

}

public static String decrypt(char matrix[][], String ct){

int length = ct.length();

String pt ="";

int l,k,m,n;

l =k = m =n = 0;

for(int p = 0; p < length ; p = p + 2){

for (int i =0 ; i < 5 ; i++ ) {

for (int j = 0 ; j < 5 ; j++ ) {

if(ct.charAt(p)==matrix[i][j]){

l = i;

k = j;

}

if(ct.charAt(p + 1)==matrix[i][j]){

m = i;

n = j;

}

}

}

if(l == m){

if(k == 0 )

k = 5;

if(n == 0)

n = 5;

pt = pt+matrix[l][(k-1)] + matrix[l][(n-1)];

}

else if(k == n){

if(l == 0 )

l = 5;

if(m == 0)

m = 5;

pt = pt + matrix[(l-1)][k] + matrix[(m-1)][k];

}

else{

pt = pt + matrix[l][n] + matrix[m][k];

}

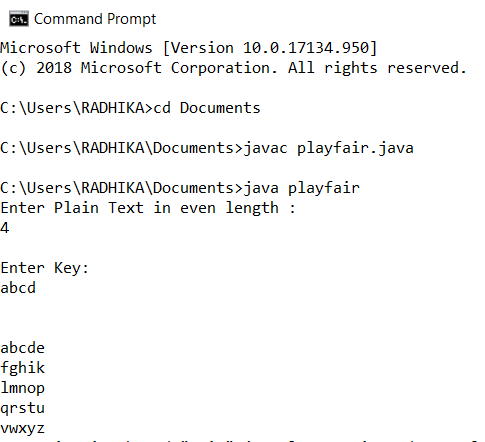
}

return pt;

}

}

**OUTPUT:-**

****

# Practical 4

# AIM:- Write a program to implement Columnar Cipher

**Program:-**

import java.io.\*;

class columnar

{

char arr[][],encrypt[][],decrypt[][],keya[],keytemp[];

public void creatematrixE(String s,String key,int row,int column)

{

arr=new char[row][column];

int k=0;

keya=key.toCharArray();

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

{

for(int j=0;j<column;j++)

{

if(k<s.length())

{

arr[i][j]=s.charAt(k);

k++;

}

else

{

arr[i][j]=' ';

}

}

}

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

{

for(int j=0;j<column;j++)

{

System.out.print(arr[i][j]+" ");

}

System.out.println();

}

}

public void createkey(String key,int column)

{

keytemp=key.toCharArray();

for(int i=0;i<column-1;i++)

{

for(int j=i+1;j<column;j++)

{

if(keytemp[i]>keytemp[j])

{

char temp=keytemp[i];

keytemp[i]=keytemp[j];

keytemp[j]=temp;

}

}

}

}

public void creatematrixD(String s,String key,int row,int column)

{

arr=new char[row][column];

int k=0;

keya=key.toCharArray();

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

{

for(int j=0;j<row;j++)

{

if(k<s.length())

{

arr[j][i]=s.charAt(k);

k++;

}

else

{

arr[j][i]=' ';

}

}

}

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

{

for(int j=0;j<column;j++)

{

System.out.print(arr[i][j]+" ");

}

System.out.println();

}

}

public void encrypt(int row,int column)

{

encrypt=new char[row][column];

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

{

for(int j=0;j<column;j++)

{

if(keya[i]==keytemp[j])

{

for(int k=0;k<row;k++)

{

encrypt[k][j]=arr[k][i];

}

keytemp[j]='?';

break;

}

}

}

}

public void decrypt(int row,int column)

{

decrypt=new char[row][column];

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

{

for(int j=0;j<column;j++)

{

if(keya[j]==keytemp[i])

{

for(int k=0;k<row;k++)

{

decrypt[k][j]=arr[k][i];

}

keya[j]='?';

break;

}

}

}

}

public void resultE(int row,int column,char arr[][])

{

System.out.println("Result:");

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

{

for(int j=0;j<row;j++)

{

System.out.print(arr[j][i]+" ");

}

}

}

public void resultD(int row,int column,char arr[][])

{

System.out.println("Result:");

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

{

for(int j=0;j<column;j++)

{

System.out.print(arr[i][j]+" ");

}

}

}

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

{

int row,column,choice;

columnar obj=new columnar();

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

System.out.println("Menu:\n1) Encryption\n2) Decryption");

choice=Integer.parseInt(in.readLine());

System.out.println("Enter the string:");

String s=in.readLine();

System.out.println("Enter the key:");

String key=in.readLine();

row=s.length()/key.length();

if(s.length()%key.length()!=0)

row++;

column=key.length();

switch(choice)

{

case 1: obj.creatematrixE(s,key,row,column);

obj.createkey(key,column);

obj.encrypt(row,column);

obj.resultE(row,column,obj.encrypt);

break;

case 2: obj.creatematrixD(s,key,row,column);

obj.createkey(key,column);

obj.decrypt(row,column);

obj.resultD(row,column,obj.decrypt);

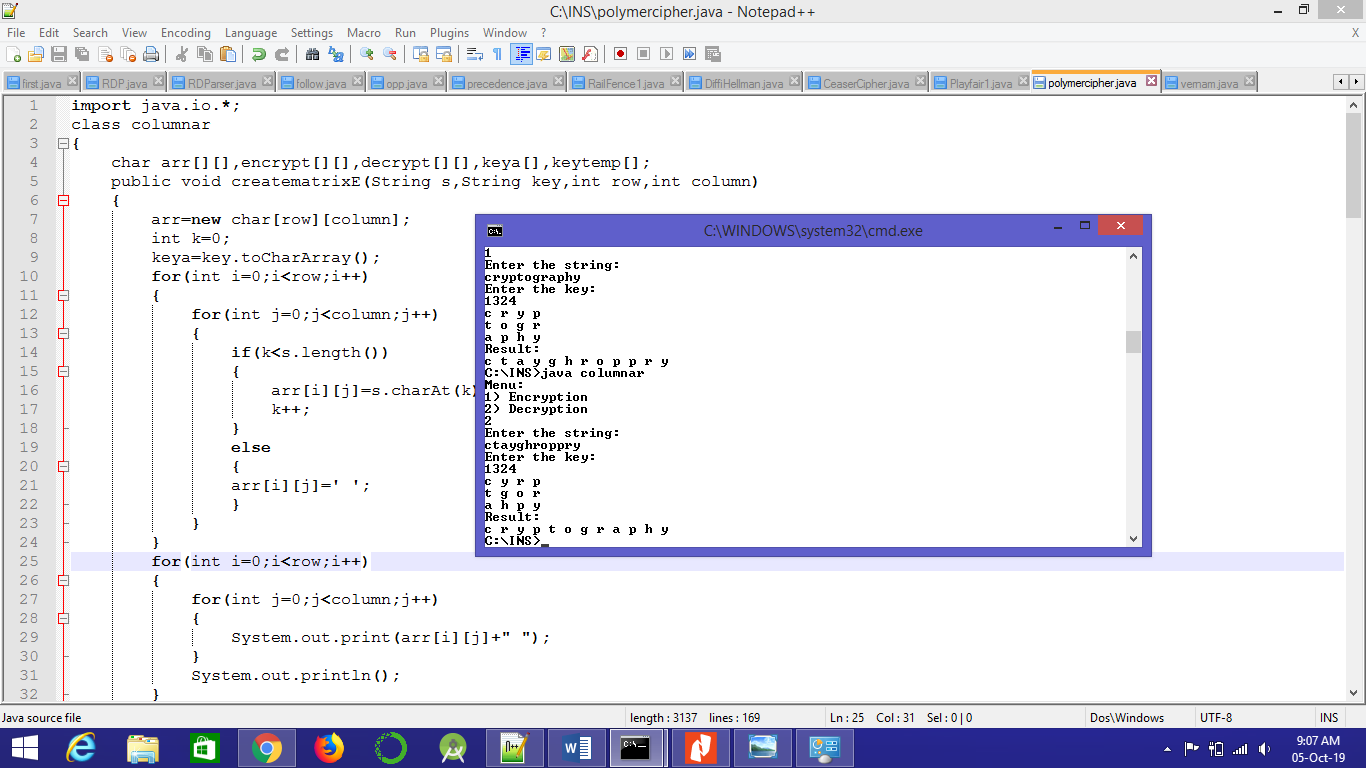
break;

}

}

}

**OUTPUT:-**



# Practical 5

# AIM:- Write a program to implement Hill Cipher

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class HillCipher

{

int keymatrix[][];

int linematrix[];

int resultmatrix[];

public void divide(String temp, int s)

{

while (temp.length() > s)

{

String sub = temp.substring(0, s);

temp = temp.substring(s, temp.length());

perform(sub);

}

if (temp.length() == s)

perform(temp);

else if (temp.length() < s)

{

for (int i = temp.length(); i < s; i++)

temp = temp + 'x';

perform(temp);

}

}

public void perform(String line)

{

linetomatrix(line);

linemultiplykey(line.length());

result(line.length());

}

public void keytomatrix(String key, int len)

{

keymatrix = new int[len][len];

int c = 0;

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

{

for (int j = 0; j < len; j++)

{

keymatrix[i][j] = ((int) key.charAt(c)) - 97;

c++;

}

}

}

public void linetomatrix(String line)

{

linematrix = new int[line.length()];

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

{

linematrix[i] = ((int) line.charAt(i)) - 97;

}

}

public void linemultiplykey(int len)

{

resultmatrix = new int[len];

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

{

for (int j = 0; j < len; j++)

{

resultmatrix[i] += keymatrix[i][j] \* linematrix[j];

}

resultmatrix[i] %= 26;

}

}

public void result(int len)

{

String result = "";

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

{

result += (char) (resultmatrix[i] + 97);

}

System.out.print(result);

}

public boolean check(String key, int len)

{

keytomatrix(key, len);

int d = determinant(keymatrix, len);

d = d % 26;

if (d == 0)

{

System.out

.println("Invalid key!!! Key is not invertible because determinant=0...");

return false;

}

else if (d % 2 == 0 || d % 13 == 0)

{

System.out

.println("Invalid key!!! Key is not invertible because determinant has common factor with 26...");

return false;

}

else

{

return true;

}

}

public int determinant(int A[][], int N)

{

int res;

if (N == 1)

res = A[0][0];

else if (N == 2)

{

res = A[0][0] \* A[1][1] - A[1][0] \* A[0][1];

}

else

{

res = 0;

for (int j1 = 0; j1 < N; j1++)

{

int m[][] = new int[N - 1][N - 1];

for (int i = 1; i < N; i++)

{

int j2 = 0;

for (int j = 0; j < N; j++)

{

if (j == j1)

continue;

m[i - 1][j2] = A[i][j];

j2++;

}

}

res += Math.pow(-1.0, 1.0 + j1 + 1.0) \* A[0][j1]

\* determinant(m, N - 1);

}

}

return res;

}

public void cofact(int num[][], int f)

{

int b[][], fac[][];

b = new int[f][f];

fac = new int[f][f];

int p, q, m, n, i, j;

for (q = 0; q < f; q++)

{

for (p = 0; p < f; p++)

{

m = 0;

n = 0;

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

{

for (j = 0; j < f; j++)

{

b[i][j] = 0;

if (i != q && j != p)

{

b[m][n] = num[i][j];

if (n < (f - 2))

n++;

else

{

n = 0;

m++;

}

}

}

}

fac[q][p] = (int) Math.pow(-1, q + p) \* determinant(b, f - 1);

}

}

trans(fac, f);

}

void trans(int fac[][], int r)

{

int i, j;

int b[][], inv[][];

b = new int[r][r];

inv = new int[r][r];

int d = determinant(keymatrix, r);

int mi = mi(d % 26);

mi %= 26;

if (mi < 0)

mi += 26;

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

{

for (j = 0; j < r; j++)

{

b[i][j] = fac[j][i];

}

}

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

{

for (j = 0; j < r; j++)

{

inv[i][j] = b[i][j] % 26;

if (inv[i][j] < 0)

inv[i][j] += 26;

inv[i][j] \*= mi;

inv[i][j] %= 26;

}

}

System.out.println("\nInverse key:");

matrixtoinvkey(inv, r);

}

public int mi(int d)

{

int q, r1, r2, r, t1, t2, t;

r1 = 26;

r2 = d;

t1 = 0;

t2 = 1;

while (r1 != 1 && r2 != 0)

{

q = r1 / r2;

r = r1 % r2;

t = t1 - (t2 \* q);

r1 = r2;

r2 = r;

t1 = t2;

t2 = t;

}

return (t1 + t2);

}

public void matrixtoinvkey(int inv[][], int n)

{

String invkey = "";

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

{

for (int j = 0; j < n; j++)

{

invkey += (char) (inv[i][j] + 97);

}

}

System.out.print(invkey);

}

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

{

HillCipher obj = new HillCipher();

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

int choice;

System.out.println("Menu:\n1: Encryption\n2: Decryption");

choice = Integer.parseInt(in.readLine());

System.out.println("Enter the line: ");

String line = in.readLine();

System.out.println("Enter the key: ");

String key = in.readLine();

double sq = Math.sqrt(key.length());

if (sq != (long) sq)

System.out

.println("Invalid key length!!! Does not form a square matrix...");

else

{

int s = (int) sq;

if (obj.check(key, s))

{

System.out.println("Result:");

obj.divide(line, s);

obj.cofact(obj.keymatrix, s);

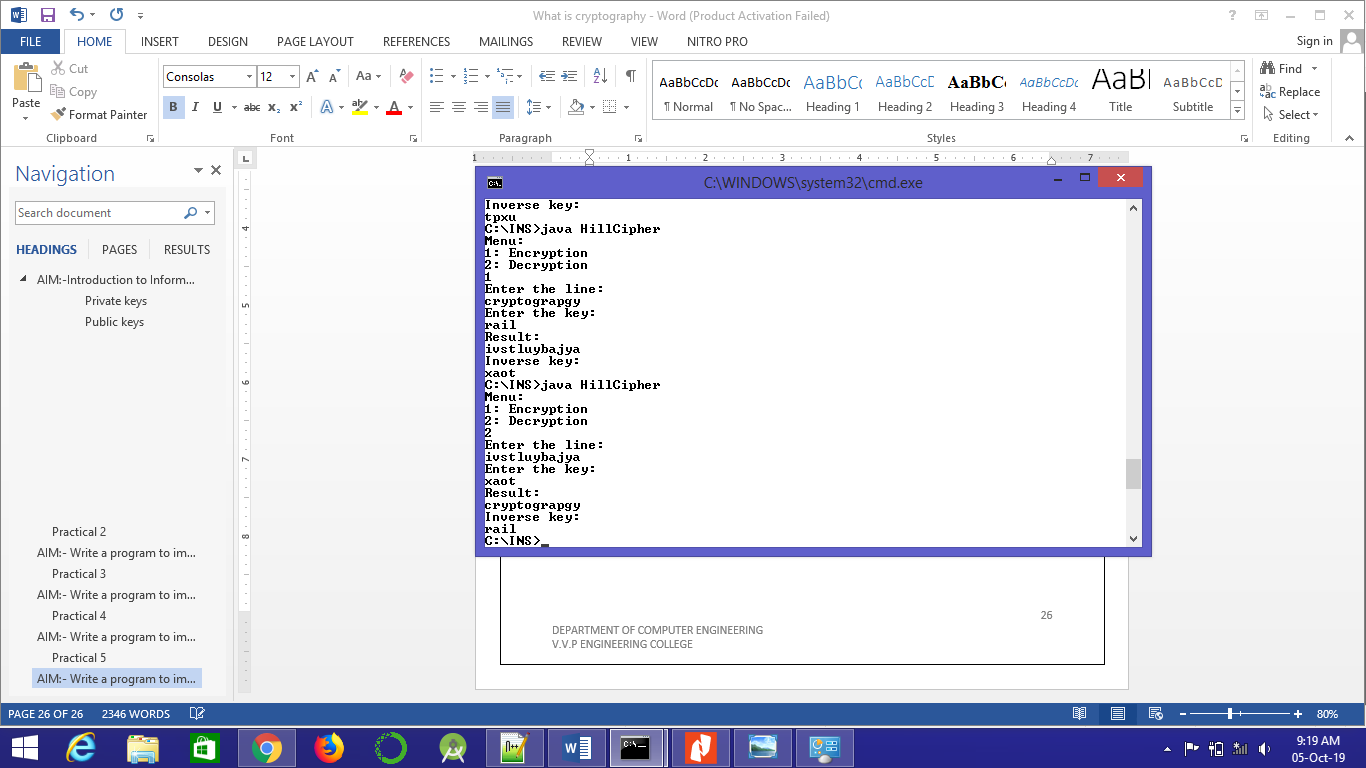
}

}

}

}

**OUTPUT:-**



# Practical 6

# AIM:- Write a program to implement Vigenere Cipher

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)

{

Scanner sc=new Scanner(System.in);

System.out.println("enter key");

String key = sc.nextLine();

System.out.println("enter text");

String message = sc.nextLine();

String encryptedMsg = encrypt(message, key);

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

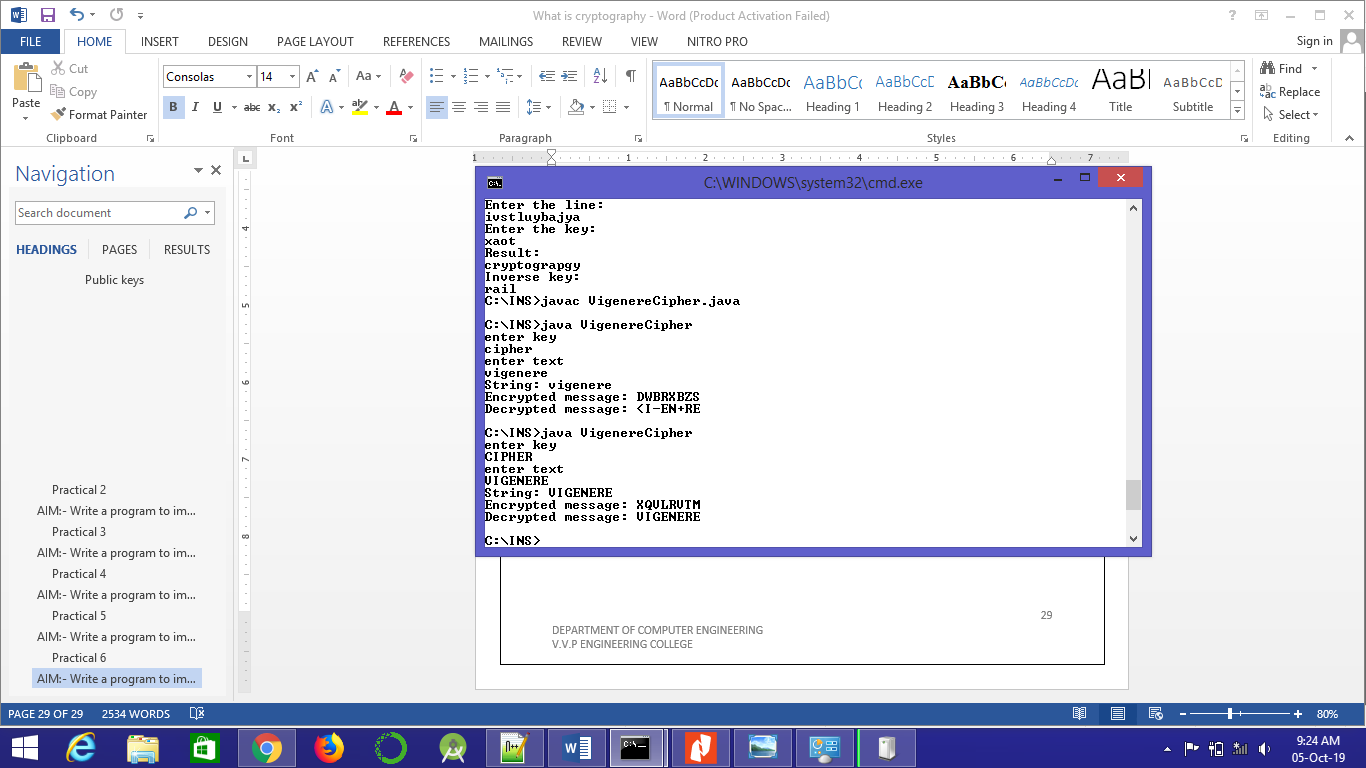
System.out.println("Encrypted message: " + encryptedMsg);

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

}

}

**OUTPUT:-**



# Practical 7

# AIM:- Write a program to implement RailFence Cipher

import java.util.\*;

class RailFenceBasic{

int depth;

String Encryption(String plainText,int depth)throws Exception

{

int r=depth,len=plainText.length();

int c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String cipherText="";

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

{

for(int j=0;j< r;j++)

{

if(k!=len)

mat[j][i]=plainText.charAt(k++);

else

mat[j][i]='X';

}

}

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

{

for(int j=0;j< c;j++)

{

cipherText+=mat[i][j];

}

}

return cipherText;

}

String Decryption(String cipherText,int depth)throws Exception

{

int r=depth,len=cipherText.length();

int c=len/depth;

char mat[][]=new char[r][c];

int k=0;

String plainText="";

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

{

for(int j=0;j< c;j++)

{

mat[i][j]=cipherText.charAt(k++);

}

}

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

{

for(int j=0;j< r;j++)

{

plainText+=mat[j][i];

}

}

return plainText;

}

}

class RailFence{

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

{

RailFenceBasic rf=new RailFenceBasic();

Scanner scn=new Scanner(System.in);

int depth;

String plainText,cipherText,decryptedText;

System.out.println("Enter plain text:");

plainText=scn.nextLine();

System.out.println("Enter depth for Encryption:");

depth=scn.nextInt();

cipherText=rf.Encryption(plainText,depth);

System.out.println("Encrypted text is:\n"+cipherText);

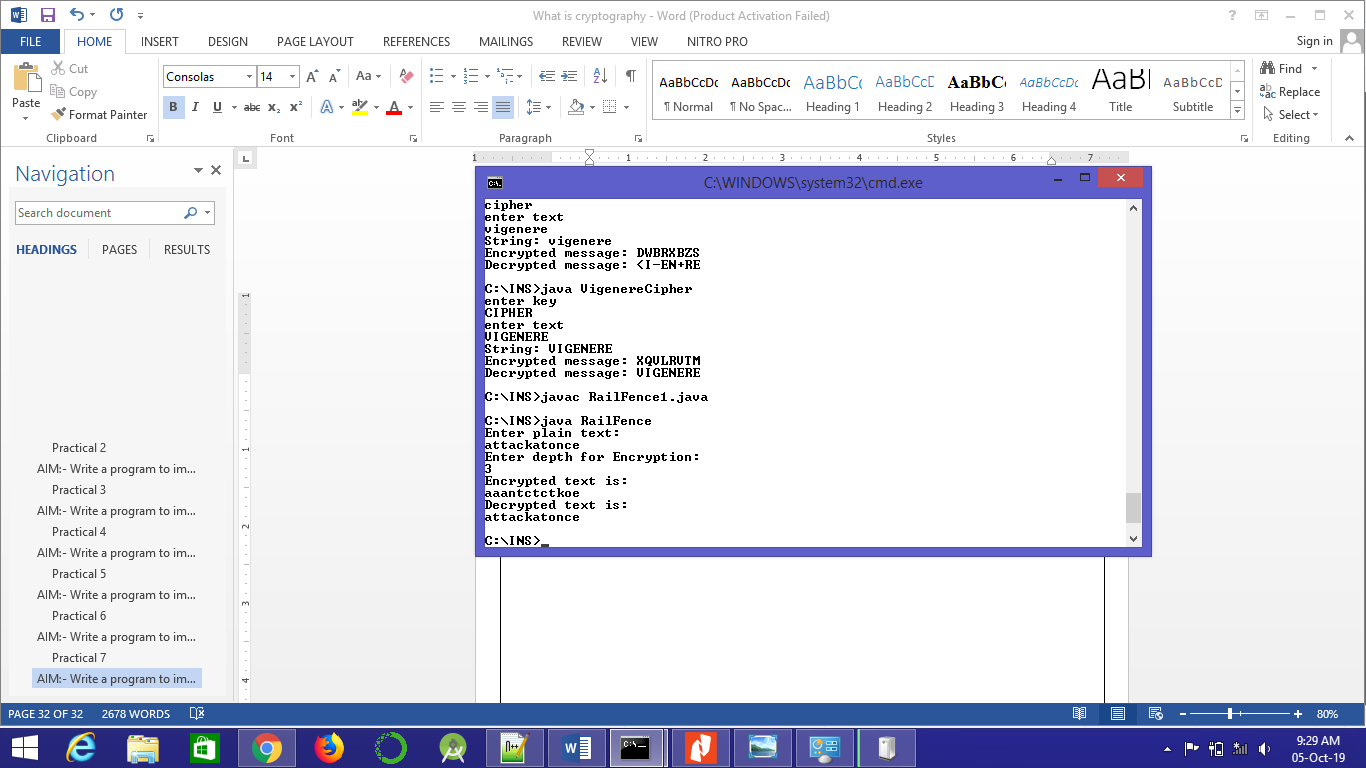
decryptedText=rf.Decryption(cipherText, depth);

System.out.println("Decrypted text is:\n"+decryptedText);

}

}

**OUTPUT:-**



# Practical 8

# AIM:- Write a program to implement Vernum Cipher

import java.io.\*;

class Vernam

{

public static int getCharValue(char x)

{

int y=(int)'a';

return ((int)x-y);

}

public static char getNumberValue(int x)

{

int z=x+(int)'a';

return ((char)z);

}

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

{

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

System.out.println("Enter your plain text");

String accept=br.readLine();

System.out.println("\nEnter your one time pad text");

String pad=br.readLine();

int aval[]=new int[accept.length()];

int pval[]=new int[pad.length()];

int initval[]=new int[pad.length()];

if(pad.length()!=accept.length())

{

System.out.println("Invalid one time pad. Application terminates.");

return;

}

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

{

int k=getCharValue(accept.charAt(i));

aval[i]=k;

}

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

{

int k=getCharValue(pad.charAt(i));

pval[i]=k;

}

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

{

initval[i]=aval[i]+pval[i];

if(initval[i]>25)

initval[i]-=26;

}

System.out.println("\nCipher text is : ");

String cipher="";

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

{

cipher+=getNumberValue(initval[i]);

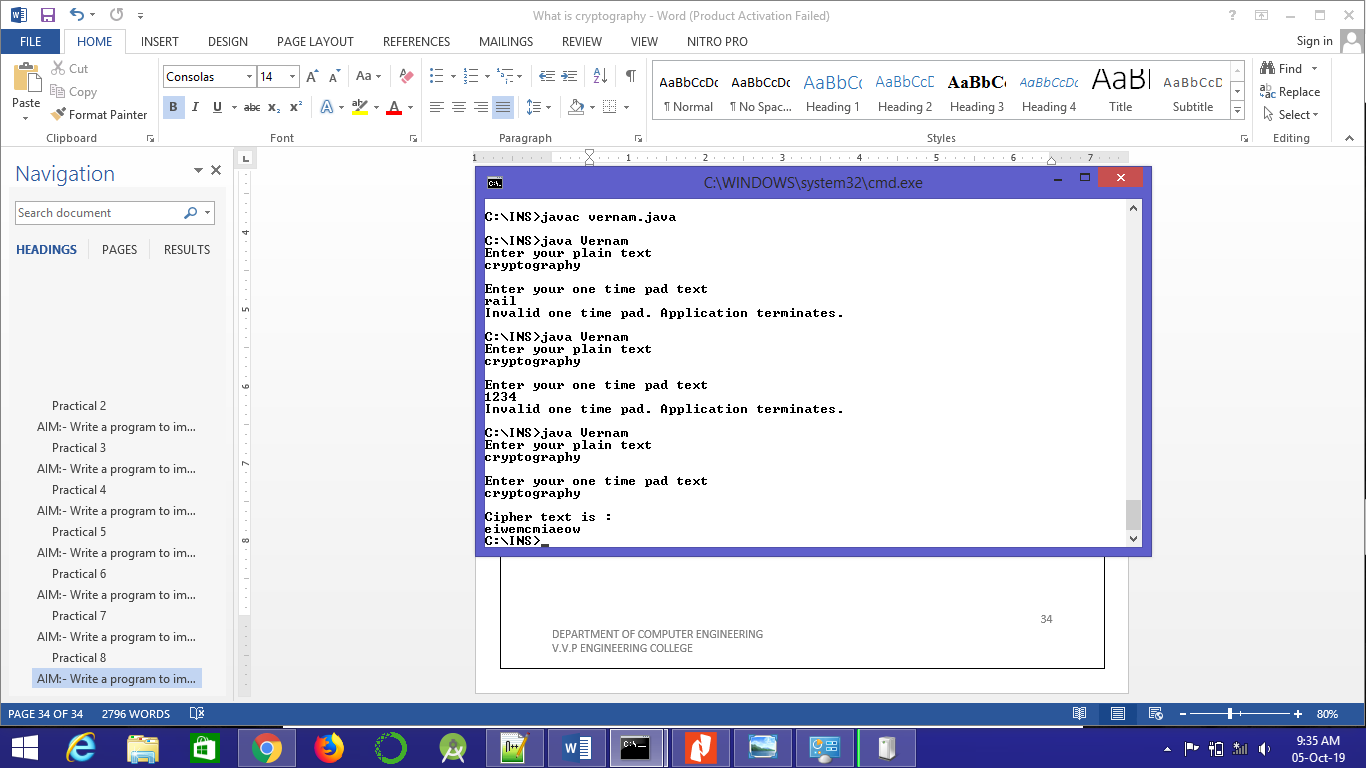
}

System.out.print(cipher);

}

}

**OUTPUT:-**



# Practical 9

# AIM:- Write a program to implement Diffie Hellman Key Exchange

import java.util.Scanner;

class DiffieHellman{

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

public static void main(String[] args) {

System.out.print("Enter p : ");

int p = s.nextInt();

System.out.print("Enter alpha: ");

int alpha = s.nextInt();

System.out.print("Enter sender private value a = ");

int a = s.nextInt();

System.out.print("Enter receiver private value b = ");

int b = s.nextInt();

keyExchange(p, alpha,a,b);

}

public static void keyExchange(int p, int alpha,int a,int b){

int alice = sender(p, alpha,a);

int bob = receiver(p, alpha,b);

int keyA = sender(p, bob,a);

int keyB = sender(p, alice,b);

System.out.println("sender key = "+keyA );

System.out.println("receiver key = "+keyB );

}

public static int sender(int p, int alpha,int a){

int alice = 1;

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

alice = (alice \* alpha) %p;

}

return alice;

}

public static int receiver(int p, int alpha,int b){

int bob = 1;

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

bob = (bob \* alpha) %p;

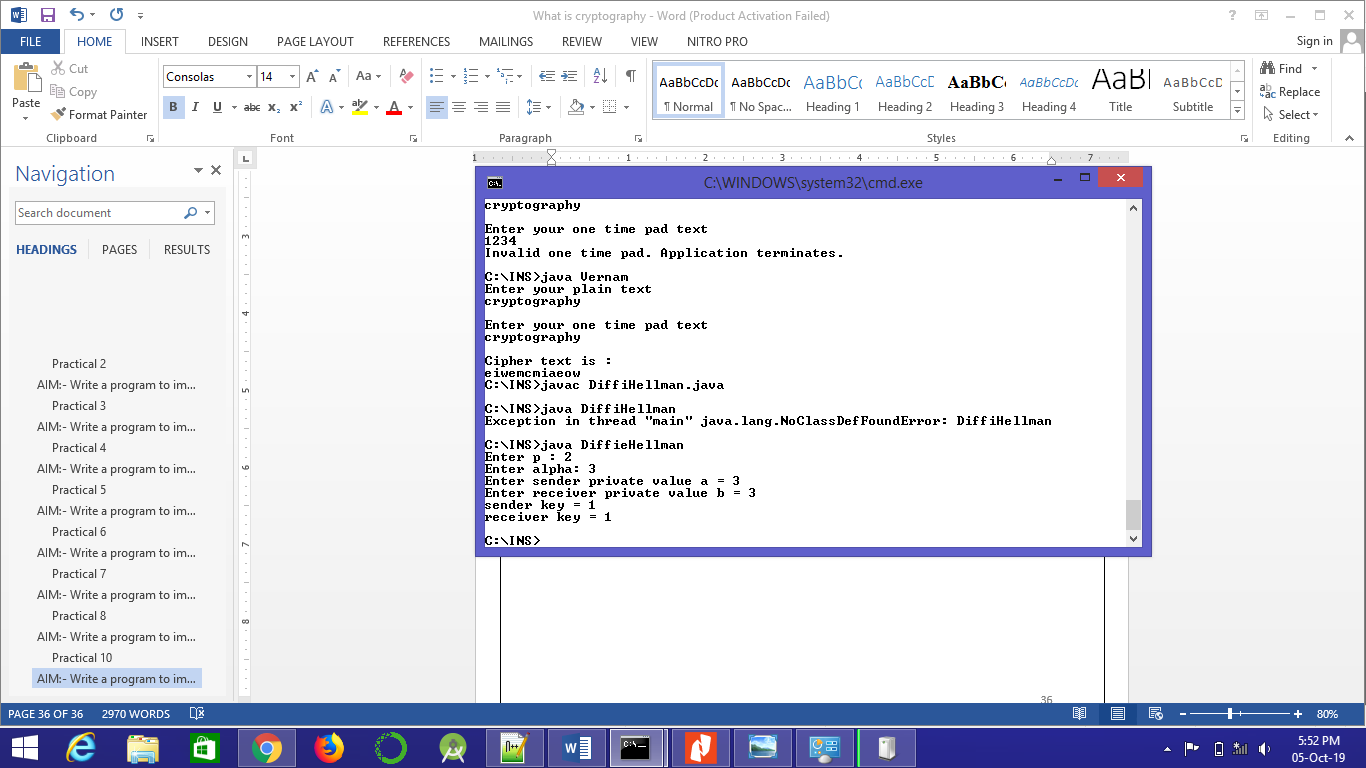
}

return bob;

}

}

**OUTPUT:-**



# Practical 10

# AIM: - Case study of DES Algorithm

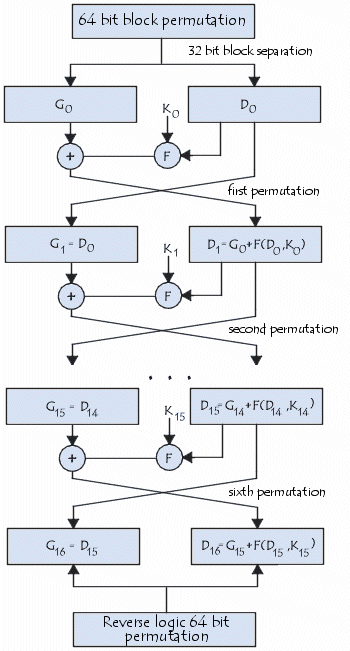
Case study of DES algorithm.

DES Algorithm :

* have a high security level related to a small key used for encryption and decryption
* be easily understood
* not depend on the algorithm's confidentiality
* be adaptable and economical
* be efficient and exportable

In late 1974, IBM proposed "Lucifer", which, thanks to the NSA (National Security Agency), was modified on 23 November 1976 to become the **DES** (*Data Encryption Standard*). The DES was approved by the NBS in 1978. The DES was standardized by the *ANSI* (*American National Standard Institute*) under the name of *ANSI X3.92*, better known as *DEA* (*Data Encryption Algorithm*).

Principle of the DES

It is a symmetric encryption system that uses 64-bit blocks, 8 bits (one octet) of which are used for parity checks (to verify the key's integrity). Each of the key's parity bits (1 every 8 bits) is used to check one of the key's octets by odd parity, that is, each of the parity bits is adjusted to have an odd number of '1's in the octet it belongs to. The key therefore has a "useful" length of 56 bits, which means that only 56 bits are actually used in the algorithm.

The algorithm involves carrying out combinations, substitutions and permutations between the text to be encrypted and the key, while making sure the operations can be performed in both directions (for decryption). The combination of substitutions and permutations is called a **product cipher**.

The key is ciphered on 64 bits and made of 16 blocks of 4 bits, generally denoted *k1* to *k16*. Given that "only" 56 bits are actually used for encrypting, there can be 256 (or 7.2\*1016) different keys!

The DES algorithm

The main parts of the algorithm are as follows:

* Fractioning of the text into 64-bit (8 octet) blocks;
* Initial permutation of blocks;
* Breakdown of the blocks into two parts: left and right, named *L* and *R*;
* Permutation and substitution steps repeated 16 times (called **rounds**);
* Re-joining of the left and right parts then inverse initial permutation.

Fractioning of the text

Initial permutation

Firstly, each bit of a block is subject to initial permutation, which can be represented by the following initial permutation (*IP*) table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IP** | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 | | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 | | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 | | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 | |

This permutation table shows, when reading the table from left to right then from top to bottom, that the 58th bit of the 64-bit block is in first position, the 50th in second position and so forth.

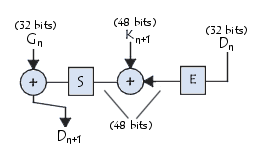
Division into 32-bit blocks

Once the initial permutation is completed, the 64-bit block is divided into two 32-bit blocks, respectively denoted **L** and **R** (for left and right). The initial status of these two blocks is denoted **L**0 and **R**0:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **L**0 | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 | | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 | | 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 | | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 | |

|  |
| --- |
|  |
| **R**0 | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 | |

It is interesting to note that **L**0 contains all bits having an even position in the initial message, whereas **R**0 contains bits with an odd position.

Rounds

The **L**n and **R**n blocks are subject to a set of repeated transformations called *rounds*, shown in this diagram, and the details of which are given below:

Expansion function

The 32 bits of the **R**0 block are expanded to 48 bits thanks to a table called an *expansion table* (denoted **E**), in which the 48 bits are mixed together and 16 of them are duplicated:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **E** | |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 32 | 1 | 2 | 3 | 4 | 5 | | 4 | 5 | 6 | 7 | 8 | 9 | | 8 | 9 | 10 | 11 | 12 | 13 | | 12 | 13 | 14 | 15 | 16 | 17 | | 16 | 17 | 18 | 19 | 20 | 21 | | 20 | 21 | 22 | 23 | 24 | 25 | | 24 | 25 | 26 | 27 | 28 | 29 | | 28 | 29 | 30 | 31 | 32 | 1 | |

As such, the last bit of **R**0 (that is, the 7th bit of the original block) becomes the first, the first becomes the second, etc.   
In addition, the bits 1,4,5,8,9,12,13,16,17,20,21,24,25,28 and 29 of **R**0 (respectively 57, 33, 25, l, 59, 35, 27, 3, 6l, 37, 29, 5, 63, 39, 31 and 7 of the original block) are duplicated and scattered in the table.

exclusive OR with the key

The resulting 48-bit table is called **R'**0 or **E**[**R**0]. The DES algorithm then *exclusive ORs* the first key **K**1 with **E**[**R**0]. The result of this *exclusive OR* is a 48-bit table we will call **R**0 out of convenience (it is not the starting **R**0!).

Substitution function

**R**0 is then divided into 8 6-bit blocks, denoted **R**0i. Each of these blocks is processed by **selection functions** (sometimes called *substitution boxes* or *compression functions*), generally denoted **S**i.   
The first and last bits of each **R**0i determine (in binary value) the line of the selection function; the other bits (respectively 2, 3, 4 and 5) determine the column. As the selection of the line is based on two bits, there are 4 possibilities (0,1,2,3). As the selection of the column is based on 4 bits, there are 16 possibilities (0 to 15). Thanks to this information, the selection function "selects" a ciphered value of 4 bits.

Here is the first substitution function, represented by a 4-by-16 table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**1 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 | | **1** | 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 | | **2** | 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 | | **3** | 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 | |

Let **R**01 equal *101110*. The first and last bits give *10*, that is, 2 in binary value. The bits 2,3,4 and 5 give *0111*, or 7 in binary value. The result of the selection function is therefore the value located on line no. 2, in column no. 7. It is the value *11*, or *111* binary.

Each of the 8 6-bit blocks is passed through the corresponding selection function, which gives an output of 8 values with 4 bits each. Here are the other selection functions:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**2 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 | | **1** | 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 | | **2** | 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 | | **3** | 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**3 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 | | **1** | 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 | | **2** | 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 | | **3** | 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**4 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 | | **1** | 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 | | **2** | 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 | | **3** | 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**5 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 | | **1** | 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 | | **2** | 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 | | **3** | 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**6 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 | | **1** | 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 | | **2** | 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 | | **3** | 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**7 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 | | **1** | 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 | | **2** | 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 | | **3** | 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S**8 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | | **0** | 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 | | **1** | 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 | | **1** | 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 | | **1** | 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 | |

Each 6-bit block is therefore substituted in a 4-bit block. These bits are combined to form a 32-bit block.

Permutation

The obtained 32-bit block is then subject to a permutation **P** here is the table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **P** | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 | | 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 | | 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 | | 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 | |

Exclusive OR

All of these results output from **P** are subject to an *Exclusive OR* with the starting **L**0 (as shown on the first diagram) to give R1, whereas the initial **R**0 gives **L**1.

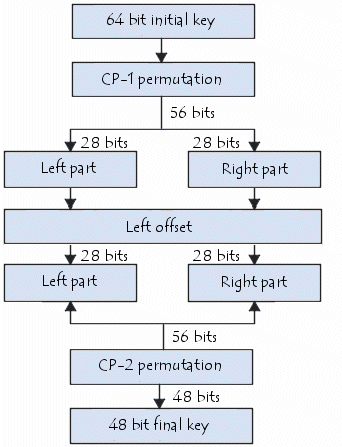
Iteration

All of the previous steps (*rounds*) are repeated 16 times.

Inverse initial permutation

At the end of the iterations, the two blocks **L**16 and **R**16 are re-joined, then subject to inverse initial permutation:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **IP-1** | |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 | | 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 | | 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 | | 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 | | 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 | | 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 | | 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 | | 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 | |

The output result is a 64-bit ciphertext!

Generation of keys

Given that the DES algorithm presented above is public, security is based on the complexity of encryption keys.

The algorithm below shows how to obtain, from a 64-bit key (made of any 64 alphanumeric characters), 8 different 48-bit keys each used in the DES algorithm:

Firstly, the key's parity bits are eliminated so as to obtain a key with a useful length of 56 bits.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PC-1** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 57 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 58 | 50 | 42 | 34 | 26 | 18 | | 10 | 2 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 | 60 | 52 | 44 | 36 | | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 | 62 | 54 | 46 | 38 | 30 | 22 | | 14 | 6 | 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | 28 | 20 | 12 | 4 | |

The first step is a permutation denoted **PC-1** whose table is presented below:

This table may be written in the form of two tables **L**i and **R**i (for left and right) each made of 28 bits:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **L**i | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 57 | 49 | 41 | 33 | 25 | 17 | 9 | | 1 | 58 | 50 | 42 | 34 | 26 | 18 | | 10 | 2 | 59 | 51 | 43 | 35 | 27 | | 19 | 11 | 3 | 60 | 52 | 44 | 36 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **R**i | |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 63 | 55 | 47 | 39 | 31 | 23 | 15 | | 7 | 62 | 54 | 46 | 38 | 30 | 22 | | 14 | 6 | 61 | 53 | 45 | 37 | 29 | | 21 | 13 | 5 | 28 | 20 | 12 | 4 | |

The result of this first permutation is denoted **L**0 and **R**0.

These two blocks are then rotated to the left, such that the bits in second position take the first position, those in third position take the second, etc.   
The bits in first position move to last position.

The 2 28-bit blocks are then grouped into one 56-bit block. This passes through a permutation, denoted **PC-2**, giving a 48-bit block as output, representing the key **K**i.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **PC-2** | |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 | 15 | 6 | 21 | 10 | | 23 | 19 | 12 | 4 | 26 | 8 | 16 | 7 | 27 | 20 | 13 | 2 | | 41 | 52 | 31 | 37 | 47 | 55 | 30 | 40 | 51 | 45 | 33 | 48 | | 44 | 49 | 39 | 56 | 34 | 53 | 46 | 42 | 50 | 36 | 29 | 32 | |

Repeating the algorithm makes it possible to give the 16 keys **K**1 to **K**16 used in the DES algorithm.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **LS** | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 15 | 17 | 19 | 21 | 23 | 25 | 27 | 28 | |

# Practical 11

# AIM: - Case study of RSA Algorithm

**RSA : Key Generation Algorithm**

This is the original algorithm.

1. Generate two large random primes, *p* and *q*, of approximately equal size such that their product n = pq is of the required bit length, e.g. 1024 bits.
2. Compute n = pq and (phi) φ = (p-1)(q-1).
3. Choose an integer *e*, 1 < e < phi, such that gcd(e, phi) = 1.
4. Compute the secret exponent *d*, 1 < d < phi, such that ed ≡ 1 (mod phi).
5. The public key is (n, e) and the private key (d, p, q). Keep all the values d, p, q and phi secret. [We prefer sometimes to write the private key as (n, d) because you need the value of n when using d.]

* n is known as the *modulus*.
* e is known as the *public exponent* or *encryption exponent* or just the *exponent*.
* d is known as the *secret exponent* or *decryption exponent*.

Encryption

Sender A does the following:-

1. Obtains the recipient B's public key (n, e).
2. Represents the plaintext message as a positive integer *m*, 1 < m < n.
3. Computes the ciphertext c = me mod n.
4. Sends the ciphertext *c* to B.

Decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute m = cd mod n.
2. Extracts the plaintext from the message representative *m*.

Digital signing

Sender A does the following:-

1. Creates a *message digest* of the information to be sent.
2. Represents this digest as an integer *m* between 1 and *n*-1.
3. Uses her *private* key (n, d) to compute the signature s = md mod n.
4. Sends this signature *s* to the recipient, B.

Signature verification

Recipient B does the following:-

1. Uses sender A's public key (n, e) to compute integer v = se mod n.
2. Extracts the message digest from this integer.
3. Independently computes the message digest of the information that has been signed.
4. If both message digests are identical, the signature is valid.