INFORMATION AND NETWORK SECURITY (2170709)

LAB MANUAL

PALAK KANZARIYA.

160470107027

VVP CE SEM-7

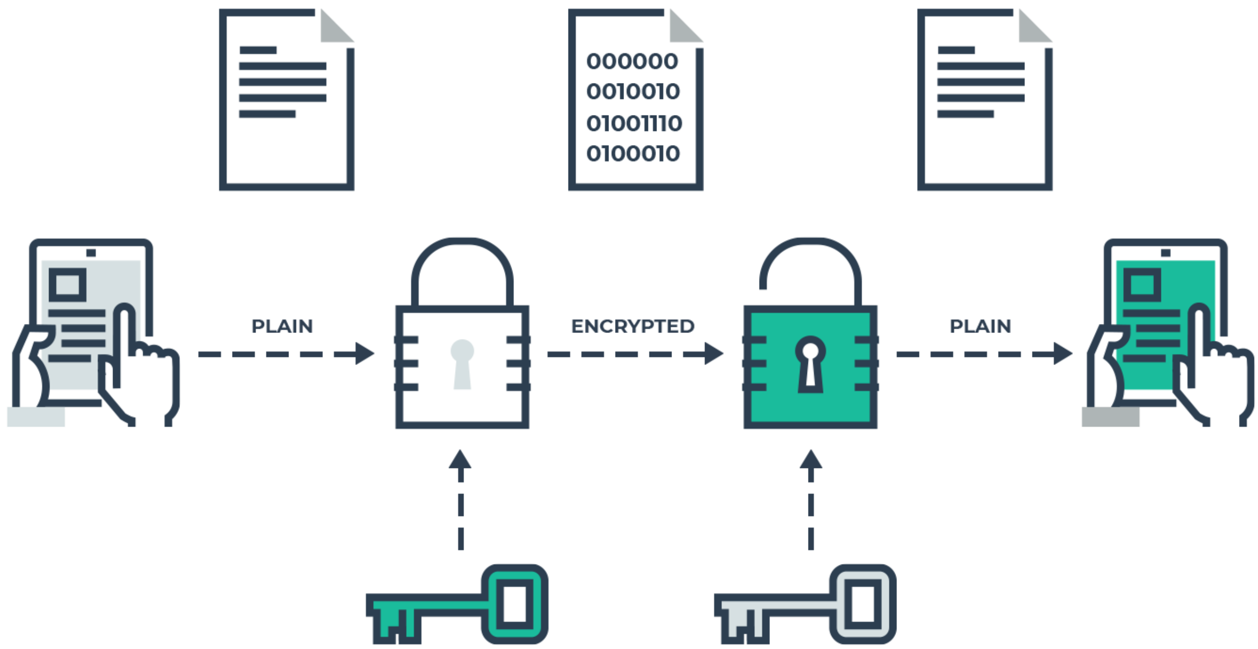
Contents

Practical – 1

Aim: Introduction to Information and Network Security.

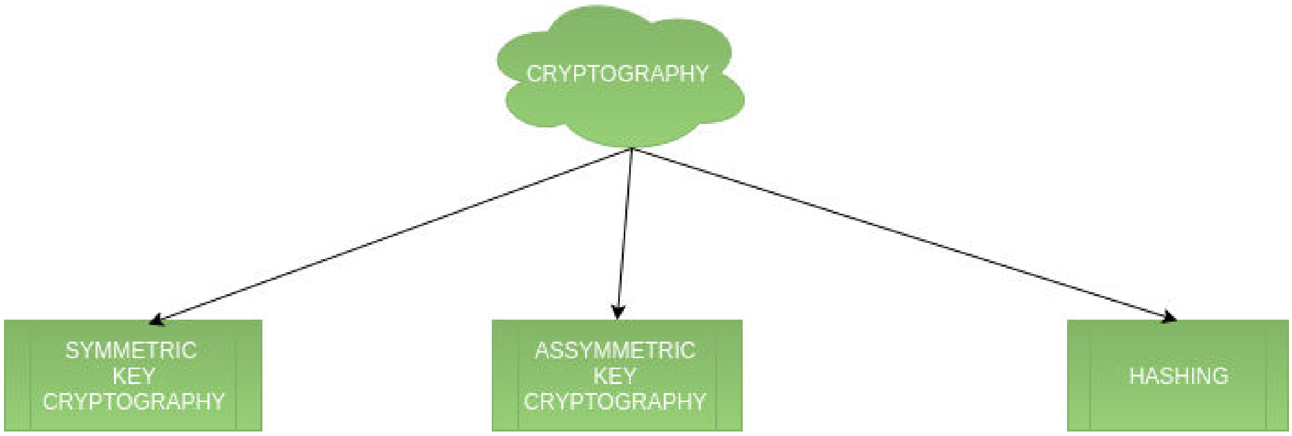
1. What is Cryptography?

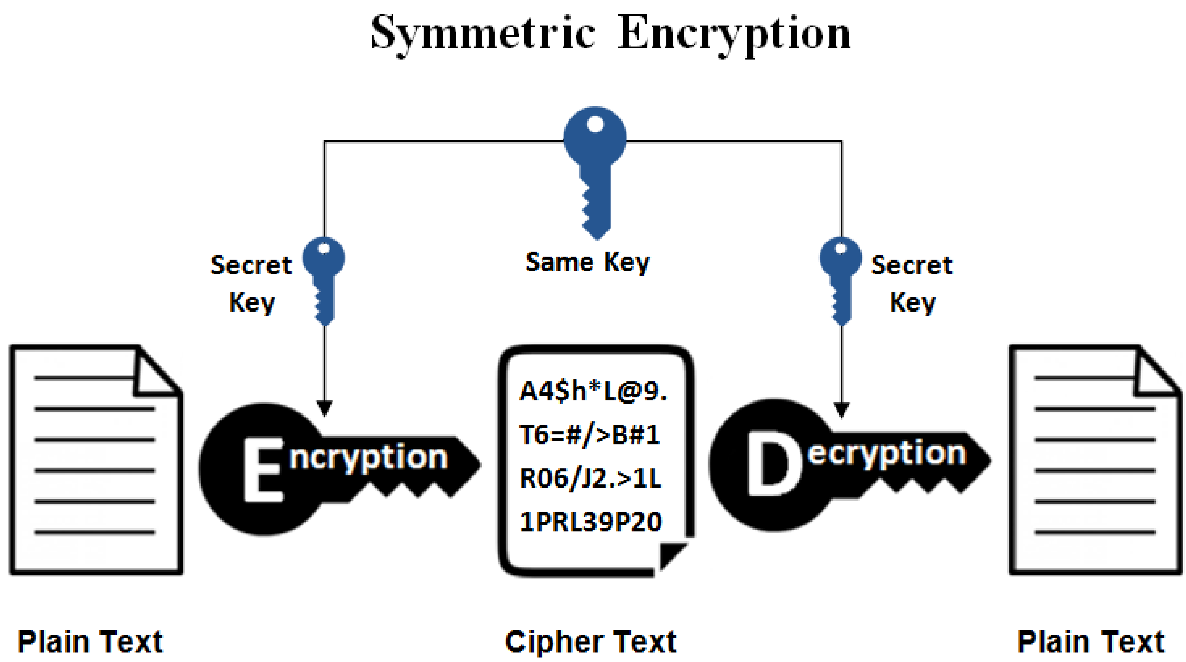
- Cryptography is mainly used in dealing with network security.
- Meaning of the word:
  crypto: secret or hidden
  graphs: writing
- Cryptography is the science of secret writing with the intention of keeping the data secret.
- The art and science of concealing the messages to introduce secrecy in information security is recognized as cryptography.
- Following figure can be seen to understand how data can be encrypted and decrypted:



- As shown above, in cryptography, plain text is sent by the sender.
- This text can encrypted to a non-readable format using a secret key.
- This encrypted text is sent to the receiver who decrypts this text to original plain text using another secret key.
- Thus, when the data is sent by the sender to the receiver, it is received securely.
- There are a lot of affecting factors but this is the main idea.

Classification:

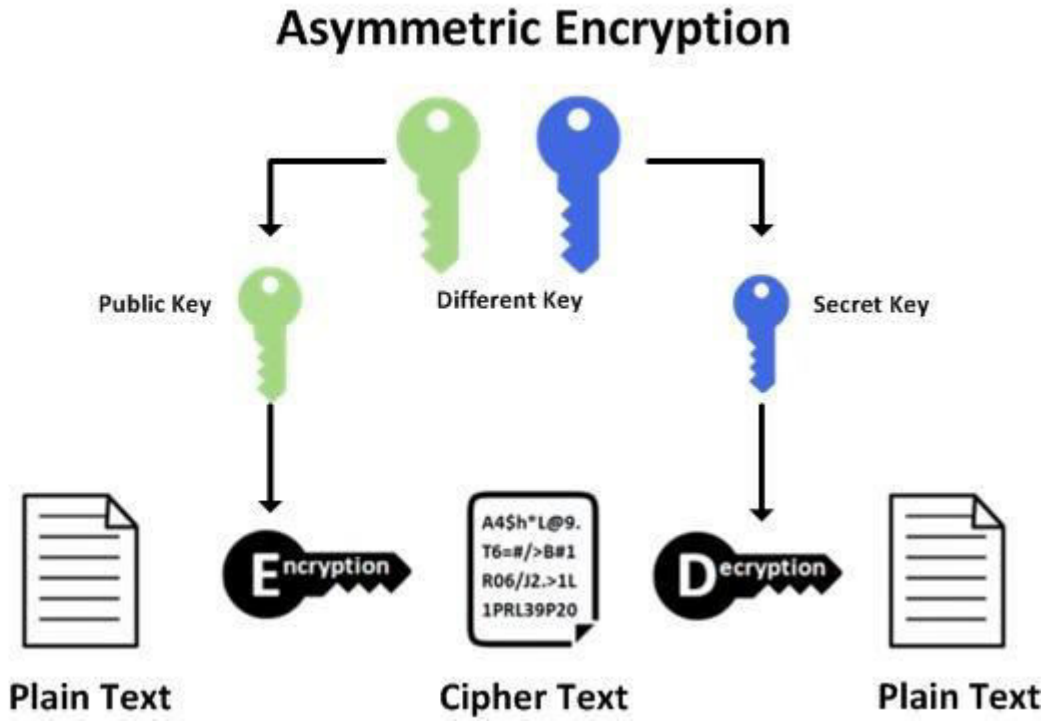




1. Symmetric Key Cryptography:

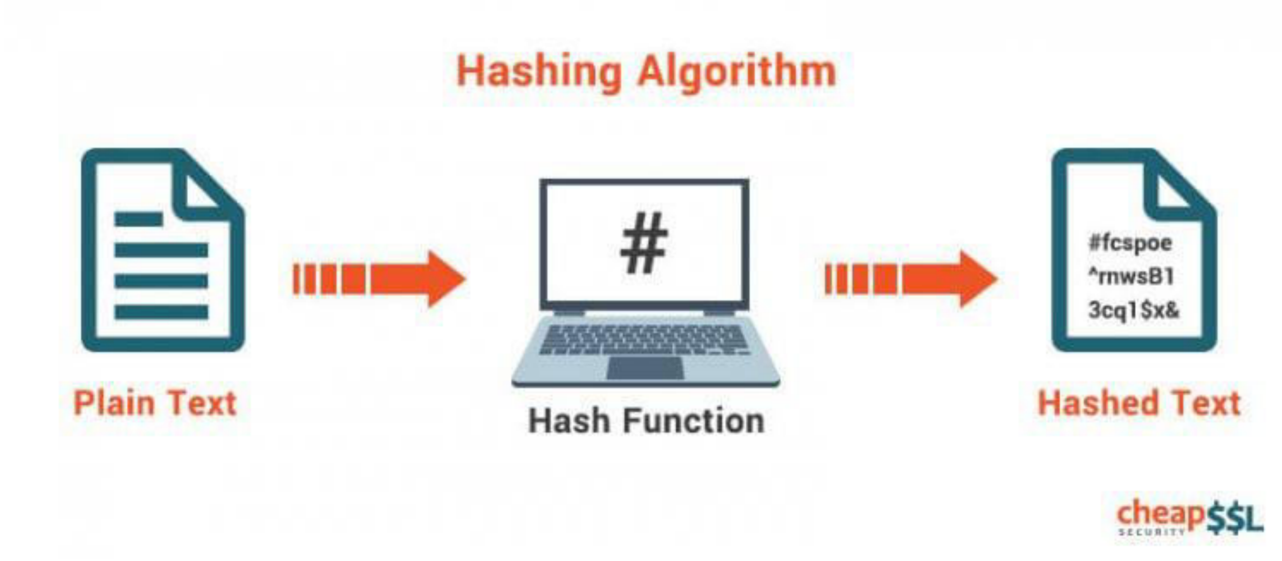
- As shown in above figure, symmetric key cryptography can be easily understood.
- In this type, in order to encrypt and decrypt data, a secret key is used that is shared.
- This key remains same in both, encryption and decryption.
- It is relatively faster than asymmetric key cryptography.
- There arises a key distribution problem as the key has to be transferred from the sender to receiver through a secure channel.

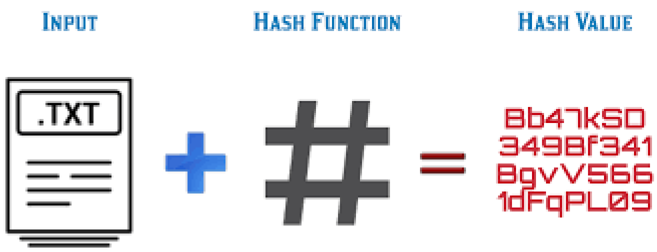
2. Asymmetric Key Cryptography:



- As shown in above figure, asymmetric cryptography can be easily understood.
- In this type, 2 different keys are used to encrypt and decrypt data.
- Here, the encryption is done with a public key and decryption is done with a secret private key.
- It solves the problem of key distribution as both parties uses different keys for encryption/decryption.
- It is not feasible to use for decrypting bulk messages as it is very slow compared to symmetric key cryptography.

3. Hashing:

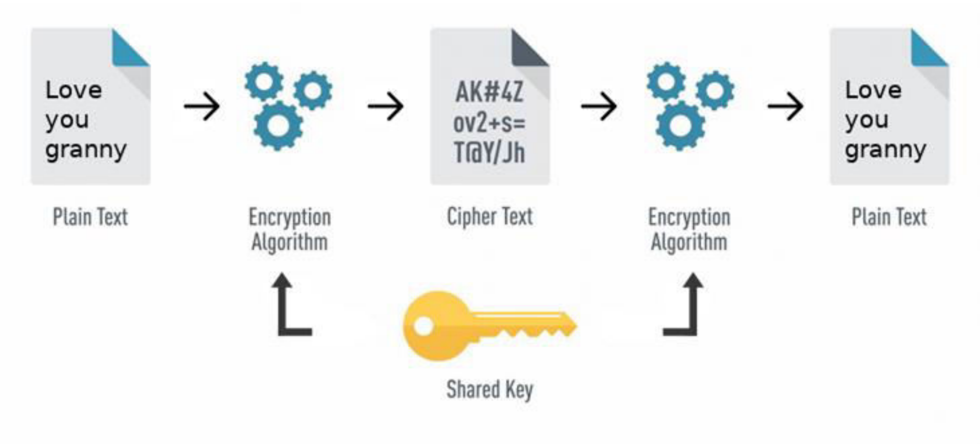




- It involves taking the plain-text and converting it to a hash value of fixed size by a hash function.
- This process ensures integrity of the message as the hash value on both, sender\'s and receiver\'s side should match if the message is unaltered.

2. Explain private key cryptography & public key cryptography?

A) Private Key



Cryptography:

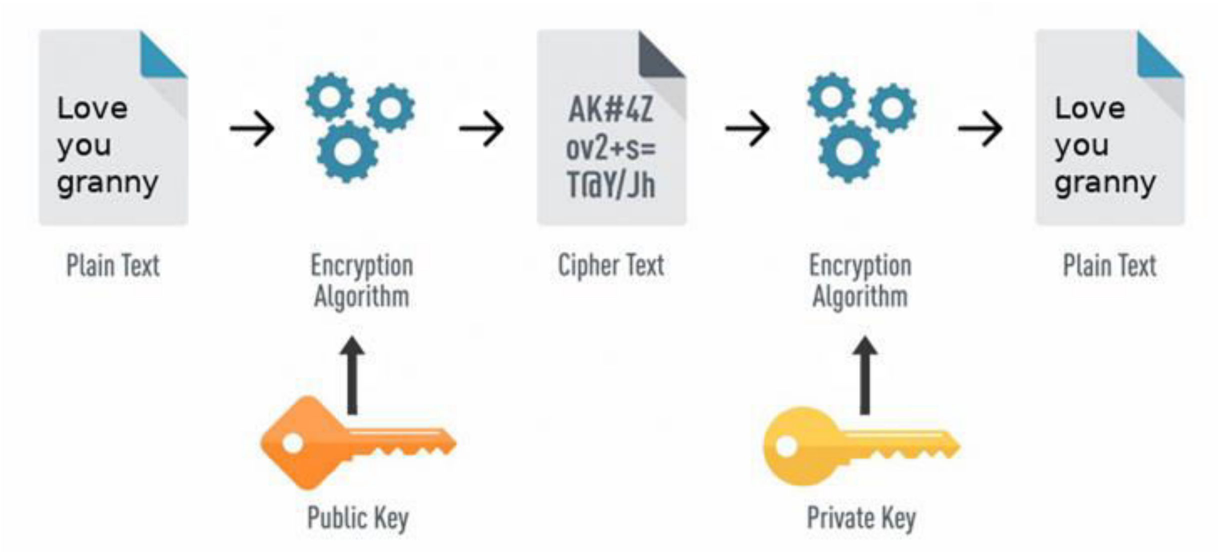
- A private key, also known as a secret key, is a variable in cryptography that is used with an algorithm to encrypt and decrypt code.
- Secret keys are only shared with the key's generator, making it highly secure.
- It is a fast process since it uses a single key.
- However, protecting one key creates a key management issue when everyone is using private keys.
- The private key may be stolen or leaked.
- Key management requires prevention of these risks and necessitates changing the encryption key often, and appropriately distributing the key.

Challenges of private key encryption:

- While private key encryption does ensure a high level of security, the following challenges must be considered:

1. Encryption key management can become too complex if each user has their own private key.
2. Private keys need to be changed frequently to avoid being leaked or stolen.
3. If the private key is forgotten or lost, the system is broken and messages stay encrypted.
4. Significant computing resources are required to create long, strong private keys.

B) Public Key Cryptography:



- Public-key cryptography, or asymmetric cryptography, is an encryption scheme that uses two mathematically related, but not identical, keys - a public key and a private key.
- Unlike symmetric key algorithms that rely on one key to both encrypt and decrypt, each key performs a unique function.
- The public key is used to encrypt and the private key is used to decrypt.
- It is computationally infeasible to compute the private key based on the public key.

- Because of this, public keys can be freely shared, allowing users an easy and convenient method for encrypting content and verifying digital signatures, and private keys can be kept secret, ensuring only the owners of the private keys can decrypt content and create digital signatures.
- Since public keys need to be shared but are too big to be easily remembered, they are stored on digital certificates for secure transport and sharing.

- Since private keys are not shared, they are simply stored in the software or operating system you use, or on hardware (e.g., USB token, hardware security module) containing drivers that allow it to be used with your software or operating system.

References

- https://www.geeksforgeeks.org/cryptography-introduction-to-crypto-terminologies/
- https://whatismyipaddress.com/protection-from-malware
- https://searchsecurity.techtarget.com/definition/private-key
- https://koolspan.com/private-key-encryption/
- https://hackernoon.com/symmetric-and-asymmetric-encryption-5122f9ec65b1
- https://www.globalsign.com/en-in/ssl-information-center/what-is-public-key-cryptography/

Practical: 2

Aim: Implement Caesar Cipher.

```
import java.util.*;


class Caesar
{
	public static void main(String a[])
	{
		int k=Integer.parseInt(a[0]);
		StringBuffer ans=new StringBuffer();
		System.out.println("Enter 1 for encryption\nEnter 2 for decryption");
		Scanner s=new Scanner(System.in);
		int c=s.nextInt();
		char tmp;
		int m;
		if(c==1)
		{
		for(int i=0;i<a[0].length();i++)
		{
				if(Character.isUpperCase(a[0].charAt(i)))
					m=65;
				else
					m=97;
				tmp=(char)((((int)a[0].charAt(i)+k-m)%26)+m);
				ans.append(tmp);
		}
		System.out.println("Encrypted Text: "+ans);
		}
		else
		{
			for(int i=0;i<a[0].length();i++)
		{
			if(Character.isUpperCase(a[0].charAt(i)))
```

```
                        m=65;
                else
                        m=97;
        if((int)a[0].charAt(i)-k-m>=0)
                tmp=(char)((((int)a[0].charAt(i)-k-m)%26)+m);
        else
                tmp=(char)((int)a[0].charAt(i)-k+26);
                ans.append(tmp);
    }
        System.out.println("Decrypted Text: "+ans);
    }
    }
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>javac Caesar.java

C:\Users\Palak\Documents\Documents\INS>java Caesar winteriscoming 3
Enter 1 for encryption
Enter 2 for decryption
1
Encrypted Text: zlqwhulvfrplqj

C:\Users\Palak\Documents\Documents\INS>java Caesar zlqwhulvfrplqj 3
Enter 1 for encryption
Enter 2 for decryption
2
Decrypted Text: winteriscoming
```

Practical: 3

Aim: Implement Columnar Transposition Cipher.

```
import java.io.*;
public 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]=' ';
}
}
}
}
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]=' ';
}
}
}
}
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:

```
C:\Users\Palak\Documents\Documents\INS>java Columnar
Menu:
1) Encryption
2) Decryption
1
Enter the string:
winter is coming
Enter the key:
itskey
Result:
ec w mt gnsniiiro
C:\Users\Palak\Documents\Documents\INS>java Columnar
Menu:
1) Encryption
2) Decryption
2
Enter the string:
ec w mt gnsniiiro
Enter the key:
itskey
Result:
winter is coming
```

Practical: 4

Aim: Implement PlayFair Cipher

```
import java.util.Scanner;

public class PlayFair

{

private String KeyWord = new String(); private String Key = new String(); private char
matrix_arr[][] = new char[5][5]; static String text = new String();

private static Scanner sc;

public void setKey(String k)

{

String K_adjust = new String();

boolean flag = false;

K_adjust = K_adjust + k.charAt(0); for (int i = 1; i < k.length(); i++) {

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

{

if (k.charAt(i) == K_adjust.charAt(j))

{

flag = true;

}

}

if (flag == false)

K_adjust = K_adjust + k.charAt(i); flag = false;

}

KeyWord = K_adjust;

}

public void KeyGen()

{

boolean flag = true;

char current;

Key = KeyWord;

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

{

current = (char) (i + 97);
```

```
if (current == 'j')
continue;
for (int j = 0; j < KeyWord.length(); j++)
{
if (current == KeyWord.charAt(j))
{
flag = false;
break;
}
}
if (flag)
Key = Key + current;
flag = true;
}
System.out.println(Key);
matrix();
}
private void matrix()
{
int counter = 0;
for (int i = 0; i < 5; i++)
{
for (int j = 0; j < 5; j++)
{
matrix_arr[i][j] = Key.charAt(counter); System.out.print(matrix_arr[i][j] + " "); counter++;
}
System.out.println();
}
}
private String format(String old_text)
{
int i = 0;
```

```
int len = 0;
len = old_text.length();
for (int tmp = 0; tmp < len; tmp++)
{
if (old_text.charAt(tmp) == 'j')
{
text = text + 'i';
}
else
text = text + old_text.charAt(tmp);
}
len = text.length();
for (i = 0; i < len; i = i + 2)
{
if (text.charAt(i + 1) == text.charAt(i))
{
text = text.substring(0, i + 1) + 'x' + text.substring(i + 1);
}
}
return text;
}
private String[] Divid2Pairs(String new_string)
{
String Original = format(new_string); int size = Original.length(); if (size % 2 != 0)
{
size++;
Original = Original + 'x';
}
String x[] = new String[size / 2];
int counter = 0;
for (int i = 0; i < size / 2; i++)
{
```

```
x[i] = Original.substring(counter, counter + 2); counter = counter + 2;
}
return x;
}
public int[] GetDiminsions(char letter)
{
int[] key = new int[2];
if (letter == 'j')
letter = 'i';
for (int i = 0; i < 5; i++)
{
for (int j = 0; j < 5; j++)
{
if (matrix_arr[i][j] == letter)
{
key[0] = i;
key[1] = j;
break;
}
}
}
return key;
}
public String encryptMessage(String Source)
{
String src_arr[] = Divid2Pairs(Source); String Code = new String(); char one;
char two;
int part1[] = new int[2];
int part2[] = new int[2];
for (int i = 0; i < src_arr.length; i++)
{
one = src_arr[i].charAt(0);
```

```
two = src_arr[i].charAt(1);
part1 = GetDiminsions(one);
part2 = GetDiminsions(two);
if (part1[0] == part2[0])
{
if (part1[1] < 4)
part1[1]++;
else
part1[1] = 0;
if (part2[1] < 4)
part2[1]++;
else
part2[1] = 0;
}
else if (part1[1] == part2[1])
{
if (part1[0] < 4)
part1[0]++;
else
part1[0] = 0;
if (part2[0] < 4)
part2[0]++;
else
part2[0] = 0;
}
else
{
int temp = part1[1];
part1[1] = part2[1];
part2[1] = temp;
}
Code = Code + matrix_arr[part1[0]][part1[1]] + matrix_arr[part2[0]][part2[1]];
```

```
}
return Code;
}
public String decryptMessage(String Code)
{
String Original = new String();
String src_arr[] = Divid2Pairs(Code);
char one;
char two;
int part1[] = new int[2];
int part2[] = new int[2];
for (int i = 0; i < src_arr.length; i++)
{
one = src_arr[i].charAt(0);
two = src_arr[i].charAt(1);
part1 = GetDiminsions(one);
part2 = GetDiminsions(two);
if (part1[0] == part2[0])
{
if (part1[1] > 0)
part1[1]--;
else
part1[1] = 4;
if (part2[1] > 0)
part2[1]--;
else
part2[1] = 4;
}
else if (part1[1] == part2[1])
{
if (part1[0] > 0)
part1[0]--;
```

```
else
part1[0] = 4;
if (part2[0] > 0)
part2[0]--;
else
part2[0] = 4;
}
else
{
int temp = part1[1];
part1[1] = part2[1];
part2[1] = temp;
}
Original = Original + matrix_arr[part1[0]][part1[1]] + matrix_arr[part2[0]][part2[1]];
}
return Original;
}
public static void main(String[] args)
{
PlayFair x = new PlayFair();
System.out.println("PLAYFAIR CIPHER");
while(true)
{
sc = new Scanner(System.in);
System.out.println("Enter your Choice: \n 1.Encryption \n 2.Decryption \n 3.Exit \n");
int choice = sc.nextInt();
switch(choice)
{
case 1: System.out.println("Enter a keyword:"); String keyword = sc.next(); x.setKey(keyword);

x.KeyGen();
System.out.println("Enter word to encrypt:"); String key_input = sc.next();
```

```
if(text.length() % 2 == 0)
{
System.out.println("Encryption: " +
x.encryptMessage(key_input));
}
else
{
String ax = key_input + 'x'; System.out.println(ax); System.out.println("Encryption: " +
x.encryptMessage(ax));
}
break;
case 2: System.out.println("Enter a keyword:"); String keyword1 = sc.next(); x.setKey(keyword1);
x.KeyGen();
System.out.println("Enter word to decrypt:"); String key_input1 = sc.next();
if (text.length() % 2 == 0)
{
System.out.println("Decryption: "+ x.decryptMessage(key_input1));
}
break;
case 3:System.exit(0);
break;
}
}
}
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents

C:\Users\Palak\Documents\Documents
PLAYFAIR CIPHER
Enter your Choice:
 1.Encryption
 2.Decryption
 3.Exit


1
Enter a keyword:
itskey
itskeyabcdfghlmnopqruvwxz
i t s k e
y a b c d
f g h l m
n o p q r
u v w x z
Enter word to encrypt:
winteriscoming
Encryption: usoidztkaqfeof
```

```
C:\Users\Palak\Documents\Document
PLAYFAIR CIPHER
Enter your Choice:
 1.Encryption
 2.Decryption
 3.Exit

2
Enter a keyword:
itskey
itskeyabcdfghlmnopqruvwxz
i t s k e
y a b c d
f g h l m
n o p q r
u v w x z
Enter word to decrypt:
usoidztkaqfeof
Decryption: winteriscoming
```

Practical: 5

Aim: Implement Hill Cipher

```
import java.util.*;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Hill{
    int[] lm;
    int[][] km;
    int[] rm;
    static int choice;
    int [][] invK;

    public void performDivision(String temp, int s)
    {
        while (temp.length() > s)
        {
            String line = temp.substring(0, s);
            temp = temp.substring(s, temp.length());
            calLineMatrix(line);
            if(choice ==1){
                multiplyLineByKey(line.length());
            }else{
                multiplyLineByInvKey(line.length());
            }
            showResult(line.length());
        }
        if (temp.length() == s){

            if(choice ==1){
            calLineMatrix(temp);
            multiplyLineByKey(temp.length());
```

```
            showResult(temp.length());
            }
            else{
                calLineMatrix(temp);
                this.multiplyLineByInvKey(temp.length());
                showResult(temp.length());
            }
        }
        else if (temp.length() < s)
        {
            for (int i = temp.length(); i < s; i++)
                temp = temp + 'x';
            if(choice ==1){
            calLineMatrix(temp);
            multiplyLineByKey(temp.length());
            showResult(temp.length());
            }
            else{
                calLineMatrix(temp);
                multiplyLineByInvKey(temp.length());
                showResult(temp.length());
            }
        }
    }


    public void calKeyMatrix(String key, int len)
    {
        km = new int[len][len];
        int k = 0;
        for (int i = 0; i < len; i++)
        {
```

```
        for (int j = 0; j < len; j++)
        {
            km[i][j] = ((int) key.charAt(k)) - 97;
            k++;
        }
    }
}

public void calLineMatrix(String line)
{
    lm = new int[line.length()];
    for (int i = 0; i < line.length(); i++)
    {
        lm[i] = ((int) line.charAt(i)) - 97;
    }
}

public void multiplyLineByKey(int len)
{
    rm = new int[len];
    for (int i = 0; i < len; i++)
    {
        for (int j = 0; j < len; j++)
        {
            rm[i] += km[i][j] * lm[j];
        }
        rm[i] %= 26;
    }
}
public void multiplyLineByInvKey(int len)
{
```

```
        rm = new int[len];
        for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < len; j++)
            {
                rm[i] += invK[i][j] * lm[j];
            }
            rm[i] %= 26;
        }
    }
    public void showResult(int len)
    {
        String result = "";
        for (int i = 0; i < len; i++)
        {
            result += (char) (rm[i] + 97);
        }
        System.out.print(result);
    }
    public int calDeterminant(int A[][], int N)
    {
        int resultOfDet;
        switch (N) {
            case 1:
                resultOfDet = A[0][0];
                break;
            case 2:
                resultOfDet = A[0][0] * A[1][1] - A[1][0] * A[0][1];
                break;
            default:
                resultOfDet = 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++;
                    }
                }
                resultOfDet += Math.pow(-1.0, 1.0 + j1 + 1.0) * A[0][j1]
                        * calDeterminant(m, N - 1);
            }   break;
        }
        return resultOfDet;
    }

    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) * calDeterminant(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 = calDeterminant(km, 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);

        invK = inv;
    }

    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 boolean check(String key, int len)
    {
        calKeyMatrix(key, len);
        int d = calDeterminant(km, len);
        d = d % 26;
```

```
        if (d == 0)
        {
            System.out.println("Key is not invertible");
            return false;
        }
        else if (d % 2 == 0 || d % 13 == 0)
        {
            System.out.println("Key is not invertible");
            return false;
        }
        else
        {
            return true;
        }
    }

    public static void main(String args[]) throws IOException
    {
        Hill obj = new Hill();
        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 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("Cannot Form a square matrix");
        else
        {
            int size = (int) sq;
```

```
            if (obj.check(key, size))
            {


                System.out.println("Result:");
                obj.cofact(obj.km, size);
                obj.performDivision(line, size);



            }
        }
    }
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>javac Hill.java

C:\Users\Palak\Documents\Documents\INS>java Hill
Menu:
1: Encryption
2: Decryption
1
Enter the line:
act
Enter the key:
gybnqkurp
Result:
poh
C:\Users\Palak\Documents\Documents\INS>java Hill
Menu:
1: Encryption
2: Decryption
2
Enter the line:
poh
Enter the key:
gybnqkurp
Result:
act
```

Practical - 6

Aim – Write a program to implement Vigenere Cipher.

```
import java.util.*;

class Vignere
{
        public static void main(String a[])
        {
                StringBuffer ans=new StringBuffer();
                System.out.println("Enter 1 for encryption or 2 for decryption:\n");
                Scanner s=new Scanner(System.in);
                int c=s.nextInt();
                int k=0;
                for(int i=0;i<a[0].length();i++)
                {
                        if(k==a[1].length())
                                k=0;
                        if(c==1)

        ans.append(Character.toString((char)(((a[0].codePointAt(i)+a[1].codePointAt(k))%26)+'A')))
;
                        else
                        {
                                int tmp=a[0].codePointAt(i)-a[1].codePointAt(k);
                                if(tmp<0)
                                        tmp=26-(((int)Math.abs(tmp))%26);
                                else
                                        tmp=tmp%26;
                        ans.append(Character.toString((char)(tmp+'A')));
                        }
                        k++;
                }
                if(c==1)
                System.out.println("Encrypted text: "+ans);
                else
                System.out.println("Decrypted text: "+ans);
        }
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>java Vignere WINTERISCOMING itskey
Enter 1 for encryption or 2 for decryption:

1
Encrypted text: KHLJOVWRAEWMBF

C:\Users\Palak\Documents\Documents\INS>java Vignere KHLJOVWRAEWMBF itskey
Enter 1 for encryption or 2 for decryption:

2
Decrypted text: WINTERISCOMING
```

Practical - 7

Aim – Write a program to implement VernamCipher.

```
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:

```
C:\Users\Palak\Documents\Documents\INS>java Vernam
Enter your plain text
HELLO

Enter your one time pad text
baxyc

Cipher text is :
IEcdQ
```

Practical - 8

Aim – Write a program to implement Rail Fence Cipher.

```
import java.util.*;
class RailFence
{
    int depth;
    static String encryption(String plainText,int depth)throws Exception
    {
        int r=depth,len=plainText.length();
        int c=len/depth;
        if((len%depth)!=0)
            c++;
        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]='-';
            }
        }
        for(int i=0;i< r;i++)
        {
            for(int j=0;j< c;j++)
            {
                cipherText+=mat[i][j];
            }
        }
        return cipherText;
```

```
    }
    static String decryption(String cipherText,int depth)throws Exception
    {
        int r=depth,len=cipherText.length();
        int c=len/depth;
        if(len%depth!=0)
            c++;
        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++)
            {
                if(k!=len)
                    mat[i][j]=cipherText.charAt(k++);
                else
                    mat[j][i]='-';
            }
        }
        for(int i=0;i< c;i++)
        {
            for(int j=0;j< r;j++)
            {
                plainText+=mat[j][i];
            }
        }
        return plainText;
    }
    public static void main(String args[])throws Exception
    {
   Scanner sc=new Scanner(System.in);
```

```
    int depth;
        String plainText,cipherText,decryptedText;
        System.out.println("Enter 1 for encryption 2 for decryption:");
        int c = sc.nextInt();
        //System.out.println(c);
        if(c==1)
        {
            System.out.println("Enter plain text:");
            plainText=sc.next();
            System.out.println("Enter depth for Encryption:");
            depth=sc.nextInt();
            cipherText=encryption(plainText,depth);
            System.out.println("Encrypted text is:\n"+cipherText);
        }
        else if(c==2)
        {
            System.out.println("Enter cipher text:");
            cipherText=sc.next();
            System.out.println("Enter depth for Decryption:");
            depth=sc.nextInt();
            decryptedText=decryption(cipherText, depth);
            System.out.println("Decrypted text is:\n"+decryptedText);
        }
        else
        {
            System.out.println("----Please enter valid choice----");
        }
  }
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>javac RailFence.java

C:\Users\Palak\Documents\Documents\INS>java RailFence
Enter 1 for encryption 2 for decryption:
1
Enter plain text:
winteriscoming
Enter depth for Encryption:
2
Encrypted text is:
wneicmnitrsoig

C:\Users\Palak\Documents\Documents\INS>java RailFence
Enter 1 for encryption 2 for decryption:
2
Enter cipher text:
wneicmnitrsoig
Enter depth for Decryption:
2
Decrypted text is:
winteriscoming
```

Practical - 9

Aim – Write a program to implement Columnnar Cipher.

```
import java.util.*;
class SimpleColumnar{
    public static void main(String sap[]){
    Scanner sc = new Scanner(System.in);


    System.out.print("\nEnter plaintext(enter in lower case): ");
    String message = sc.next();
    System.out.print("\nEnter key in numbers: ");
    String key = sc.next();
        int columnCount = key.length();
        int rowCount = (message.length()+columnCount)/columnCount;
        int 44laintext[][] = new int[rowCount][columnCount];
    int cipherText[][] = new int[rowCount][columnCount];
        System.out.print("\n-----Encryption-----\n");
    cipherText = encrypt(44laintext, cipherText, message, rowCount, columnCount, key);
    String ct = "";
    for(int i=0; i<columnCount; i++)
    {
        for(int j=0; j<rowCount; j++)
        {
            if(cipherText[j][i] == 0)
                ct = ct + 'x';
            else{
                ct = ct + (char)cipherText[j][i];
            }
        }
    }
    System.out.print("\nCipher Text: " + ct);
    System.out.print("\n\n\n-----Decryption-----\n");
    44laintext = decrypt(44laintext, cipherText, ct, rowCount, columnCount, key);
    String pt = "";
```

```
for(int i=0; i<rowCount; i++)
{
    for(int j=0; j<columnCount; j++)
    {
        if(45laintext[i][j] == 0)
            pt = pt + "";
        else{
            pt = pt + (char)45laintext[i][j];
        }
    }
}
System.out.print("\nPlain Text: " + pt);
System.out.println();
}
static int[][] encrypt(int 45laintext[][], int cipherText[][], String message, int rowCount, int
columnCount, String key){
    int I,j;
    int k=0;
            for(i=0; i<rowCount; i++)
    {
        for(j=0; j<columnCount; j++)
        {
            if(k < message.length())
            {
                45laintext[i][j] = (int)message.charAt(k);
                k++;
            }
            else
            {
                break;
            }
        }
    }
```

```
            for(i=0; i<columnCount; i++)
    {
        int currentCol= ( (int)key.charAt(i) – 48 ) -1;
        for(j=0; j<rowCount; j++)
        {
            cipherText[j][i] = 46laintext[j][currentCol];
        }
     }
    System.out.print("Cipher Array(read column by column): \n");
    for(i=0;i<rowCount;i++){
        for(j=0;j<columnCount;j++){
            System.out.print((char)cipherText[i][j]+"\t");
        }
        System.out.println();
    }
    return cipherText;
  }
  static int[][] decrypt(int 46laintext[][], int cipherText[][], String message, int rowCount, int
columnCount, String key){
     int I,j;
     int k=0;
     for(i=0; i<columnCount; i++)
     {
         int currentCol= ( (int)key.charAt(i) – 48 ) -1;
         for(j=0; j<rowCount; j++)
         {
             46laintext[j][currentCol] = cipherText[j][i];
         }
     }
     System.out.print("Plain Array(read row by row): \n");
     for(i=0;i<rowCount;i++){
         for(j=0;j<columnCount;j++){
             System.out.print((char)46laintext[i][j]+"\t");
```

```
            }
            System.out.println();
        }
        return 47laintext;
    }
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>javac SimpleColumnar.java

C:\Users\Palak\Documents\Documents\INS>java SimpleColumnar

Enter plaintext(enter in lower case): winteriscoming

Enter key in numbers: 123456

-----Encryption-----
Cipher Array(read column by column):
w       i       n       t       e       r
i       s       c       o       m       i
n       g

Cipher Text: winisgncxtoxemxrix


-----Decryption-----
Plain Array(read row by row):
w       i       n       t       e       r
i       s       c       o       m       i
n       g

Plain Text: winteriscoming
```

Practical - 10

Aim – Write a program to implement DiffieHellman.

```
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(of Alice)");
		int a=sc.nextInt();
		System.out.println("Choose 2nd secret no(of 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 key  "+S_A);
		}
		else
		{
			System.out.println("ALice and Bob cannot communicate with each other.");
		}
	}
}
```

OUTPUT:

```
C:\Users\Palak\Documents\Documents\INS>java Diffie_Hellman
Enter modulo(p)
23
Enter primitive root of 23:
9
Choose 1st secret no(of Alice)
4
Choose 2nd secret no(of BOB)
3
ALice and Bob can communicate with each other!
They share a secret key  9
```

Practical - 11

Aim – case study of DES algorithm.

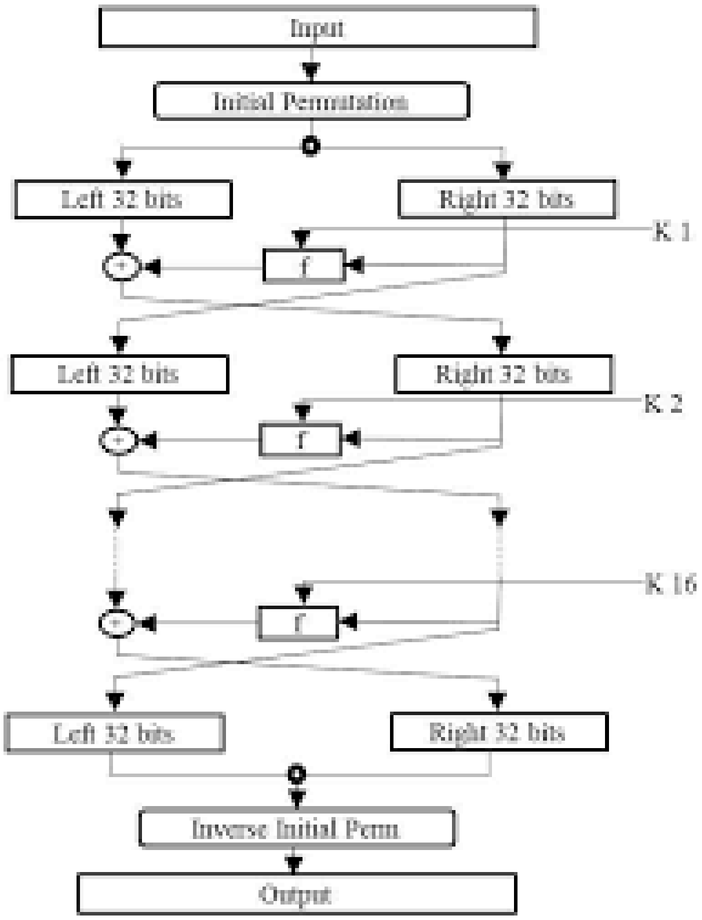
- have a high security level related to a small key used for encryption and decryption
- be easily understood
- not depend on the algorithm's confidentiality
- be adaptable and economical
- be efficient and exportable

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 2^{56} (or $7.2*10^{16}$) 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:

585042342618102

605244362820124

625446383022146

645648403224168

IP

5749413325179 1

595143352719113

615345372921135

635547393123157

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:

585042342618102

605244362820124

L0

625446383022146

645648403224168

5749413325179 1

R0

595143352719113

615345372921135

635547393123157

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

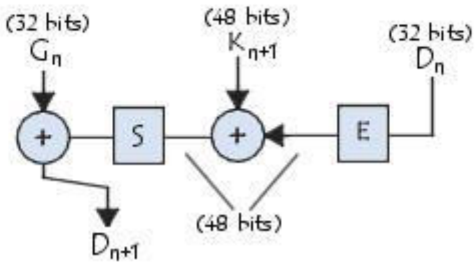
The 32 bits of the **R**0 block are expanded to 48 bits thanks to a table called an *expansion table* (denoted **E**), in which the 48 bits are mixed together and 16 of them are duplicated:

321 2 3 4 5

4 5 6 7 8 9

8 9 10111213

12 13 14 15 16 17



E

16 17 18 19 20 21

20 21 22 23 24 25

24 25 26 27 28 29

28293031321

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

R0 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:

		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
S1	**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:

		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

S2 **1**3 13 4 7 15 2 8 141201 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

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

S3 **1**137 0 9 3 4 6 10 2 8 5 141211151

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

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

S4 **1**138 1156 15 0 3 4 7 2 12 1 10149

2 10 6 9 0 12 11 7 13 1513 14 5 2 8 4

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

0 1 2 3 4 5 6 7 8 9 101112131415

S5 **0**2 124 1 7 10116 8 5 3 15130 149

114112 124 7 131 5 0 15103 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

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

S6 **1**10154 2 7129 5 6 1 13140 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

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

S7 **1**130 11 7 4 9 1 10143 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

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

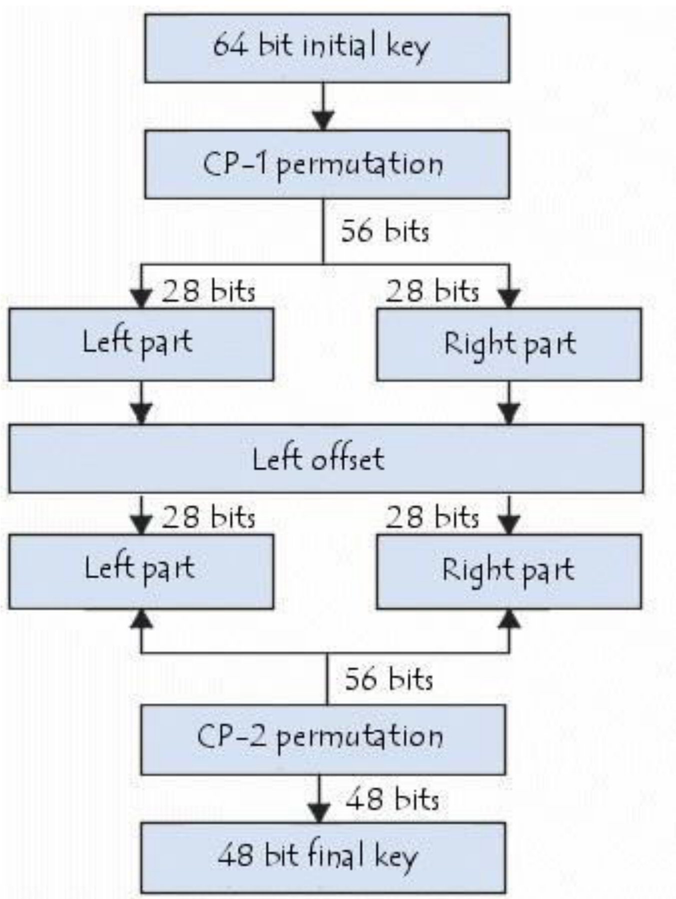
0 13 2 8 4 6 15 11 1 10 9 3 14 5 0 12 7

S8 **1**1 15138103 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:

16 7 20 21 29 12 28 17

1 15 23 26 5 18 31 10

P

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:

408481656246432

397471555236331

386461454226230

375451353216129

IP-1

364441252206028

353431151195927

342421050185826

331419 49175725

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.

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:

5749413325179 158 50 42 34 26 18 5749413325179

10 2 59 51 43 35 27 1911 3 60 52 44 36

1 58 50 42 34 26 18

PC-1

Li

635547393123157 62 54 46 38 30 22

10 2 59 51 43 35 27

14 6 61 53 45 37 29 2113 5 2820124

19113 60 52 44 36

63 55 47 39 31 23 15

7 625446383022

Ri

146 6153453729

21135 2820124

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.

141711241 5 3 28156 2110

2319124 268 167 2720132

PC-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 12468101214151719212325272 8

Practical: 12

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 $c = m^e \bmod n$.
4. Sends the ciphertext *c* to B.

Decryption

Recipient B does the following:-

1. Uses his private key (n, d) to compute $m = c^d \bmod n$.

2. Extracts the plaintext from the message representative *m*.

Program:

```
import java.util.*;

class RSA
{
	public static void main(String a[])
	{
		Scanner s=new Scanner(System.in);
		System.out.println("Enter p:\n");
		int p=s.nextInt();
		System.out.println("Enter q:\n");
		int q=s.nextInt();
		System.out.println("Enter message:\n");
		int m=s.nextInt();
		System.out.println("Enter e:\n");
		int e=s.nextInt();
		int n=p*q;
		int fn=(p-1)*(q-1);
		int c=0;
		if(gcd(e,fn))
		{
			c=((int)Math.pow(m,e))%n;
			System.out.println("Encrypted message: "+c);
		}
		else
```

```
                    System.out.println("e not co-prime with function value");
            int d=-1;
            for(int i=0;i<fn;i++)
            {
             if(e*i%26==1)
             {
                    d=i;
                    break;
             }
            }
            int ms=((int)Math.pow(c,e))%n;
            System.out.println("Decrypted message: "+ms);
    }
    public static boolean gcd(int a, int b)
    {
            for(int i=2;i<(a<b?a:b);i++)
            {
                    if(a%i==0 && b%i==0)
                            return false;
            }
            return true;
    }
}
```

OUTPUT:

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 $s = m^d \mod n$.
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 $v = s^e \mod n$.
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.