|  |
| --- |
|  |
| Information And Network Security |
| LAB MANUAL |

|  |
| --- |
| PARAM ARDESHNA  160470107003 |

Index

[Practical 1 3](#_Toc16888133)

[AIM:- Introduction to Information and Network Security 3](#_Toc16888134)

[Practical 2 6](#_Toc16888135)

[AIM:- Implement CaeserCipher 6](#_Toc16888136)

[Practical-3 8](#_Toc16888137)

[Aim: Implement Columnar Cipher 8](#_Toc16888138)

[Practical-4 15](#_Toc16888139)

[AIM:- Implement Play fair Cipher 15](#_Toc16888140)

[Practical-5 27](#_Toc16888141)

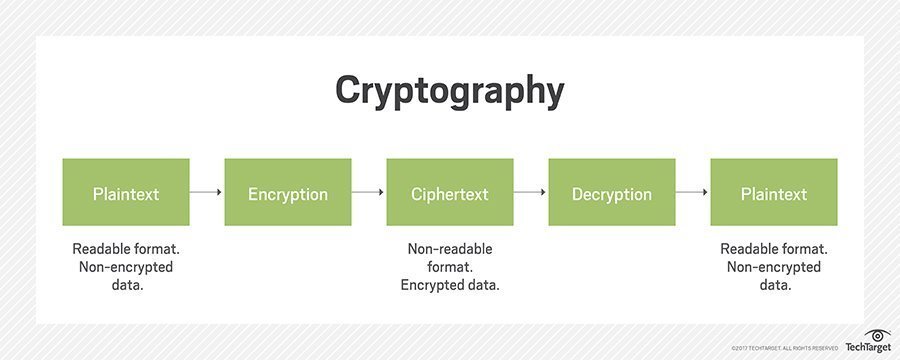
[AIM:- Implement Hill Cipher 27](#_Toc16888142)

# Practical 1

## AIM:- Introduction to Information and Network Security

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 prefix "crypt" means "hidden" or "vault" and the suffix "graphy" stands for "writing."

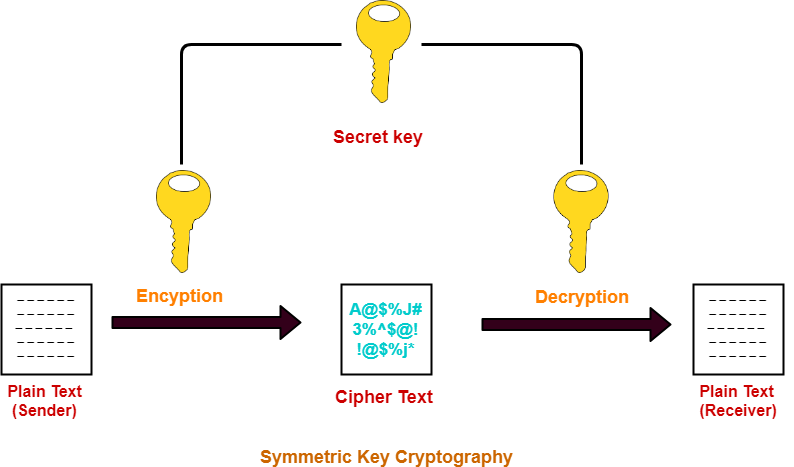


2. What is Symmetric and Asymmetric Cryptography ?

**Symmetric Key Cryptography**

Anencryption system in which the sender and receiver of a message share a single, commonkey that is used to encrypt and decrypt the message. Contrast this withpublic key cryptography which utilizes two keys - a public key to encrypt messages and a private key to decrypt them.

Symmetric-key systems are simpler and faster, but their main drawback is that the two parties must somehow exchange the key in a secure way. Public-key encryption avoids this problem because the public key can be distributed in a non-secure way, and the private key is never transmitted.Symmetric-key cryptography is sometimes called secret-key cryptography*.*



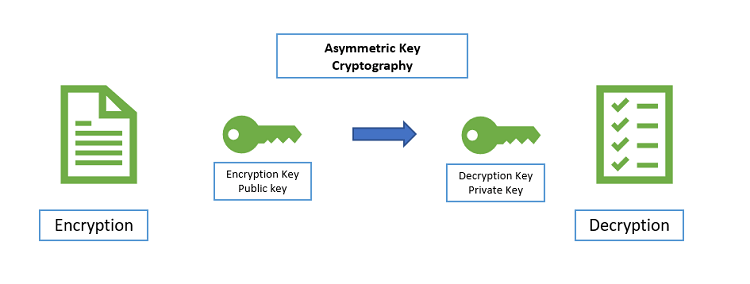
Symmetric cryptosystems have a problem of key transportation. The secret key is to be transmitted to the receiving system before the actual message is to be transmitted. Every means of electronic communication is insecure as it is impossible to guarantee that no one will be able to tap communication channels. So the only secure way of exchanging keys would be exchanging them personally.

**Asymmetric Key Cryptography**

Asymmetric [cryptography](https://searchsecurity.techtarget.com/definition/cryptography), also known as public key cryptography, uses public and private keys to encrypt and decrypt data. The keys are simply large numbers that have been paired together but are not identical (asymmetric).

One key in the pair can be shared with everyone; it is called the [public key](https://searchsecurity.techtarget.com/definition/public-key).

The other key in the pair is kept secret; it is called the [private key](https://searchsecurity.techtarget.com/definition/private-key).



A message that is encrypted using a public key can only be decrypted using the private key, while also, a message encrypted using a private key can be decrypted using a public key. Security of the public key is not required because it is publicly available and can be passed over the internet. Asymmetric key has a far better power in ensuring the security of information transmitted during communication.

References:

<https://searchsecurity.techtarget.com/definition/asymmetric-cryptography>

<https://www.webopedia.com/TERM/S/symmetric_key_cryptography.html>

# Practical 2

## AIM:- Implement CaeserCipher

class CaesarCipher

{

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;

}

public static void main(String[] args)

{

String text = args[0];

int s = Integer.parseInt(args[1]);

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

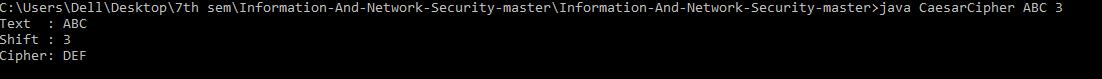
System.out.println("Shift : " + s);

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

}

}

Output:-



# Practical-3

## Aim: Implement Columnar 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);

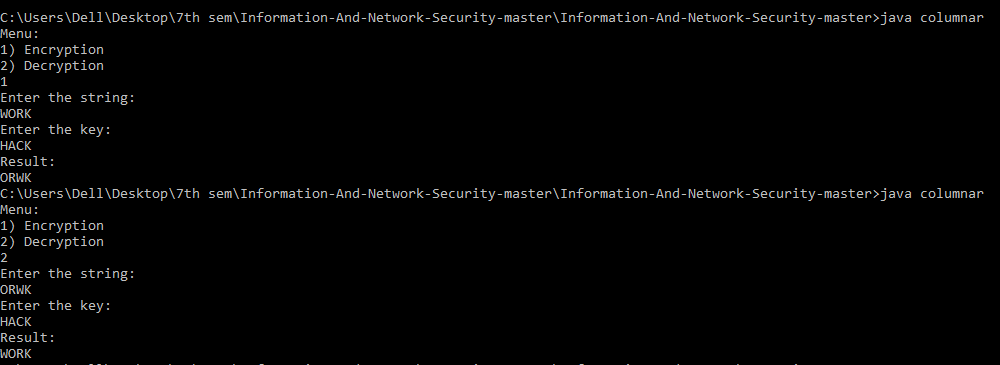
break;

}

}

}

Output:-



# Practical-4

## AIM:- Implement Play fair Cipher

import java.util.\*;

class Basic{

String allChar="ABCDEFGHIJKLMNOPQRSTUVWXYZ";

boolean indexOfChar(char c)

{

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

{

if(allChar.charAt(i)==c)

return true;

}

return false;

}

}

class PlayFair{

Basic b=new Basic();

char keyMatrix[][]=new char[5][5];

boolean repeat(char c)

{

if(!b.indexOfChar(c))

{

return true;

}

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c || c=='J')

return true;

}

}

return false;

}

void insertKey(String key)

{

key=key.toUpperCase();

key=key.replaceAll("J", "I");

key=key.replaceAll(" ", "");

int a=0,b=0;

for(int k=0;k < key.length();k++)

{

if(!repeat(key.charAt(k)))

{

keyMatrix[a][b++]=key.charAt(k);

if(b>4)

{

b=0;

a++;

}

}

}

char p='A';

while(a < 5)

{

while(b < 5)

{

if(!repeat(p))

{

keyMatrix[a][b++]=p;

}

p++;

}

b=0;

a++;

}

System.out.print("-------------------------Key Matrix-------------------");

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

{

System.out.println();

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

{

System.out.print("\t"+keyMatrix[i][j]);

}

}

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

}

int rowPos(char c)

{

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c)

return i;

}

}

return -1;

}

int columnPos(char c)

{

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

{

for(int j=0;j < keyMatrix[i].length;j++)

{

if(keyMatrix[i][j]==c)

return j;

}

}

return -1;

}

String encryptChar(String plain)

{

plain=plain.toUpperCase();

char a=plain.charAt(0),b=plain.charAt(1);

String cipherChar="";

int r1,c1,r2,c2;

r1=rowPos(a);

c1=columnPos(a);

r2=rowPos(b);

c2=columnPos(b);

if(c1==c2)

{

++r1;

++r2;

if(r1>4)

r1=0;

if(r2>4)

r2=0;

cipherChar+=keyMatrix[r1][c2];

cipherChar+=keyMatrix[r2][c1];

}

else if(r1==r2)

{

++c1;

++c2;

if(c1>4)

c1=0;

if(c2>4)

c2=0;

cipherChar+=keyMatrix[r1][c1];

cipherChar+=keyMatrix[r2][c2];

}

else{

cipherChar+=keyMatrix[r1][c2];

cipherChar+=keyMatrix[r2][c1];

}

return cipherChar;

}

String Encrypt(String plainText,String key)

{

insertKey(key);

String cipherText="";

plainText=plainText.replaceAll("j", "i");

plainText=plainText.replaceAll(" ", "");

plainText=plainText.toUpperCase();

int len=plainText.length();

// System.out.println(plainText.substring(1,2+1));

if(len/2!=0)

{

plainText+="X";

++len;

}

for(int i=0;i < len-1;i=i+2)

{

cipherText+=encryptChar(plainText.substring(i,i+2));

cipherText+=" ";

}

return cipherText;

}

String decryptChar(String cipher)

{

cipher=cipher.toUpperCase();

char a=cipher.charAt(0),b=cipher.charAt(1);

String plainChar="";

int r1,c1,r2,c2;

r1=rowPos(a);

c1=columnPos(a);

r2=rowPos(b);

c2=columnPos(b);

if(c1==c2)

{

--r1;

--r2;

if(r1 < 0)

r1=4;

if(r2 < 0)

r2=4;

plainChar+=keyMatrix[r1][c2];

plainChar+=keyMatrix[r2][c1];

}

else if(r1==r2)

{

--c1;

--c2;

if(c1 < 0)

c1=4;

if(c2 < 0)

c2=4;

plainChar+=keyMatrix[r1][c1];

plainChar+=keyMatrix[r2][c2];

}

else{

plainChar+=keyMatrix[r1][c2];

plainChar+=keyMatrix[r2][c1];

}

return plainChar;

}

String Decrypt(String cipherText,String key)

{

String plainText="";

cipherText=cipherText.replaceAll("j", "i");

cipherText=cipherText.replaceAll(" ", "");

cipherText=cipherText.toUpperCase();

int len=cipherText.length();

for(int i=0;i < len-1;i=i+2)

{

plainText+=decryptChar(cipherText.substring(i,i+2));

plainText+=" ";

}

return plainText;

}

}

class PlayFairCipher{

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

{

PlayFair p=new PlayFair();

Scanner scn=new Scanner(System.in);

String key,cipherText,plainText;

System.out.println("Enter plaintext:");

plainText=scn.nextLine();

System.out.println("Enter Key:");

key=scn.nextLine();

cipherText=p.Encrypt(plainText,key);

System.out.println("Encrypted text:");

System.out.println("---------------------------------------------------------\n"+cipherText);

System.out.println("---------------------------------------------------------");

String encryptedText=p.Decrypt(cipherText, key);

System.out.println("Decrypted text:" );

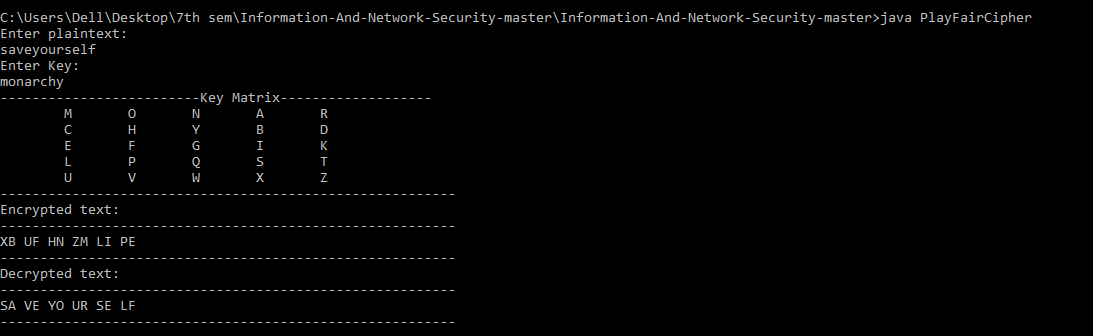
System.out.println("---------------------------------------------------------\n"+encryptedText);

System.out.println("---------------------------------------------------------");

}

}

Output:-



# Practical-5

## AIM:- 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:-

