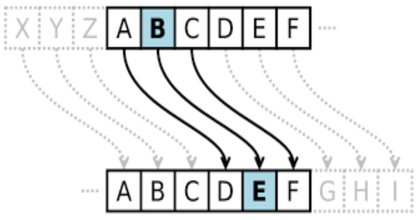
Practical:1

Caesar Cipher:

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It’s simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.
Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.
The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

$$E_n(x) = (x + n) mod\ 26$$

$$D_n(x) = (x - n) mod\ 26$$



Examples :

```
Text : ABCDEFGHIJKLMNOPQRSTUVWXYZ
Shift: 23
Cipher: XYZABCDEFGHIJKLMNOPQRSTUVW

Text : ATTACKATONCE
Shift: 4
Cipher: EXXEGOEXSRGI
```

#Program in C

```
#include<stdio.h>

int main()
{
    char message[100], ch;
    int i, key;

    printf("Enter a message to encrypt: ");
    gets(message);
    printf("Enter key: ");
    scanf("%d", &key);

    for(i = 0; message[i] != '\0'; ++i){
        ch = message[i];

        if(ch >= 'a' && ch <= 'z'){
            ch = ch + key;

            if(ch > 'z'){
                ch = ch - 'z' + 'a' - 1;
            }

            message[i] = ch;
        }
        else if(ch >= 'A' && ch <= 'Z'){
            ch = ch + key;

            if(ch > 'Z'){
                ch = ch - 'Z' + 'A' - 1;
            }

            message[i] = ch;
        }
    }

    printf("Encrypted message: %s", message);

    return 0;
}
```

Output

```
Enter a message to encrypt: axzd
Enter key: 4
Encrypted message: ebdh
```

Practical:2

Mono alphabetic Cipher

A mono-alphabetic cipher is a type of simple substitution cipher. In this cipher technique each letter of the plaintext is replaced by another letter in the cipher-text. An example of a mono-alphabetic cipher key follows:

Plain Text >>> a b c d e f g h i j k l m n o p q r s t u v w x y z

Cipher Text >>> z w x y o p r q a h c b e s u t v f g j l k m n d i

This key means that any ‘a’ in the plaintext will be replaced by a ‘z’ in the cipher-text, any ‘z’ in the plaintext will be replaced by a ‘i’ in the cipher-text, and so on.

/*program to implement Mono alphabetic Cipher*/

```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>

void main()
{
        FILE *f;
        int i,j,k=97,m=65,count=0,asc;
        int ch,a[4][26];
        char *p,*c,*plain;
        f=fopen("lab2.txt","w");
        clrscr();


                //printing 26 latters
                for(j=0;j<26;j++)
                {
                        a[0][j]=k;
                        fprintf(f,"%c",a[0][j]);
                        k++;
                }

                fprintf(f,"\n");



                        randomize();
        //making 26 bit long key
                for(j=0;j<=25;j++)
                {
                 again:
                 a[2][j]=97+random(26);

                  for(k=0;k<=j;k++)
                  {
                    if(j!=k)
                    {  if(a[2][j]==a[2][k])
                            goto again;
                    }
                   }
                  a[3][j]=a[2][j]-32;
                 }

                 //printing key in small latters
                 for(i=0;i<=25;i++)
                 {
                  fprintf(f,"%c",a[2][i]);
                 }

                 fprintf(f,"\n");
                //alphabets in CAPITAL
                for(j=0;j<26;j++)
                {a[1][j]=m;
                fprintf(f,"%c",a[1][j]);
                m++;
                }

                 //key in capital latters

                 fprintf(f,"\n");
```

```
 for(i=0;i<=25;i++)
 {
  fprintf(f,"%c",a[3][i]);
 }




//ENCRYPTION PART
printf("\n\n::::::ENTER YOUR PLAIN TEXT:::::");
gets(p);


k=0;
while(*(p+count)!='\0')
{
asc=toascii(*(p+count));

if(asc==32)
{*(c+count)=32;
}

//encryption of CAPITAL latters
else if((asc>=65)&&(asc<=90))
{k=asc-65;
*(c+count)=a[3][k];
}

//encryption of small latters
else if((asc>=97)&&(asc<=122))
{k=asc-97;
*(c+count)=a[2][k];
}

count++;
}

*(c+count)='\0';
fprintf(f,"\n\n:::::CIPHER TEXT:::::");
fputs(c,f);


//DECRYPTION PART
count=0;
k=0;
while(*(c+count)!='\0')
{
asc=toascii(*(c+count));

if(asc==32)
{*(plain+count)=32;
}

//decryption of CAPITAL latters
else if((asc>=65)&&(asc<=90))
{
  for(i=0;i<25;i++)
  { if(asc==a[3][i])
    break;
  }

*(plain+count)=a[1][i];
}

//decryption of small latters
else if((asc>=97)&&(asc<=122))
{
  for(i=0;i<25;i++)
  { if(asc==a[2][i])
    break;
  }

  *(plain+count)=a[0][i];
  }
```

```
                count++;
                }

                *(plain+count)='\0';
                fprintf(f,"\n\n:::::DECRYPTED PLAIN TEXT:::::");

                count=0;
                while(*(plain+count)!='\0')
                {fprintf(f,"%c",*(plain+count));
                count++;
                }

        getch();
        }
```

OUTPUT:

```
abcdefghijklmnopqrstuvwxyz
lqzginrhtwacmseukjdvybofxp
ABCDEFGHIJKLMNOPQRSTUVWXYZ
LQZGINRHTWACMSEUKJDVYBOFXP

:::::CIPHER TEXT:::::QHlyvTA

:::::DECRYPTED PLAIN TEXT:::::BHautIK
```

Practical:3

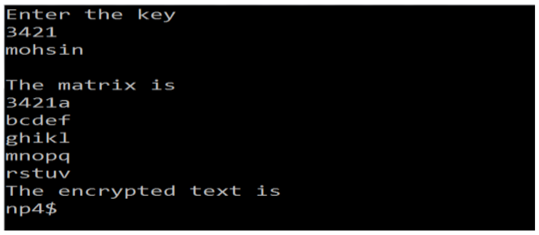
Playfair cipher:

The Playfair cipher was the first practical digraph substitution **cipher**. The scheme was invented in 1854 by Charles Wheatstone, but was named after Lord **Playfair**who promoted the use of the **cipher**. The technique encrypts pairs of letters (digraphs), instead of single letters as in the simple substitution **cipher**.

Program to implement PlayFair (Monarchy) Cipher

```
#include<stdio.h>
#include<conio.h>
void main()
{
char v,w,ch,string[100],arr[5][5],key[10],a,b,enc[100];
int temp,i,j,k,l,r1,r2,c1,c2,t,var;
FILE * fp;
fp=fopen("sk.txt","r");    //keep message in sk.txt (e.g. jamia)
clrscr();
printf("Enter the key\n");
fflush(stdin);
scanf("%s",&key);
l=0;
while(1)
{
ch=fgetc(fp);
    if(ch!=EOF)
        {
        string[l++]=ch;
        }
    if(ch==EOF)
    break;
}
string[l]='\0';
puts(string);
for(i=0;key[i]!='\0';i++)
{
    for(j=i+1;key[j]!='\0';j++)
    {
    if(key[i]==key[j])
        {
            temp=1;
            break;
        }
    }
}
if(temp==1)
printf("invalid key");
else
{
k=0;
a='a';
//printf("%c",b);
for(i=0;i<5;i++)
{
    for(j=0;j<5;j++)
    {
    if(k<strlen(key))
     arr[i][j]=key[k];
    else if(k==strlen(key))
    {
    b:
    for(l=0;l<strlen(key);l++)
    {
        if(key[l]==a)
        {
  a++;
  goto b;
        }
    }
    arr[i][j]=a;
    if(a=='i')
     a=a+2;
    else
     a++;
    }
     if(k<strlen(key))
     k++;
    }
}
printf("\n");
printf("The matrix is\n");
for(i=0;i<5;i++)
{
    for(j=0;j<5;j++)
    {
     printf("%c",arr[i][j]);
    }
    printf("\n");
}
t=0;
if(strlen(string)%2!=0)
var=strlen(string)-1;
for(i=0;i<var;)
{
    v=string[i++];
    w=string[i++];
    if(v==w)
    {
     enc[t++]=v;
     enc[t++]='$';
    }
    else
    {
    for(l=0;l<5;l++)
        {
            for(k=0;k<5;k++)
            {
      if(arr[l][k]==v||v=='j'&&arr[l][k]=='i')
      {
       r1=l;
       c1=k;
      }
      if(arr[l][k]==w||w=='j'&&arr[l][k]=='i')
      {
       r2=l;
       c2=k;
      }
            }
        }
        if(c1==c2)
        {
  r1++;
  r2++;
  if(r1==5||r2==5)
  {
   r1=0;
   r2=0;
  }
        }
        else if(r1==r2)
        {
  c1++;
  c2++;
  if(c1==5||c2==5)
  {
   c1=0;
   c2=0;
  }
        }
        else
        {
  temp=r1;
  r1=r2;
  r2=temp;
        }
        enc[t++]=arr[r1][c1];
        enc[t++]=arr[r2][c2];

    }
}
if(strlen(string)%2!=0)
 enc[t++]=string[var];
enc[t]='\0';
}
printf("The encrypted text is\n");
puts(enc);
getch();
}
```

Output of the above program:-



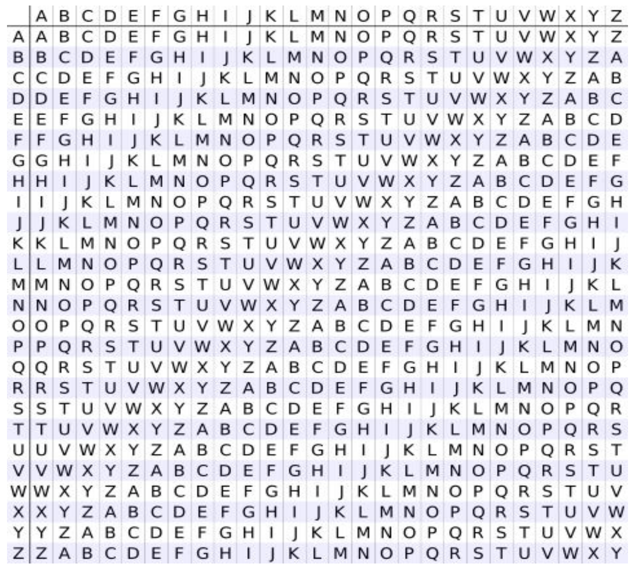
```
Enter the key
3421
mohsin

The matrix is
3421a
bcdef
ghikl
mnopq
rstuv
The encrypted text is
np4$
```

Result of PlayFair (Monarchy) Cipher

Practical:4

Vigenère Cipher

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets .The encryption of the original text is done using the *Vigenère square or Vigenère table*.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.

Example:

```
Input : Plaintext :   GEEKSFORGEEKS
             Keyword :  AYUSH
Output : Ciphertext :  GCYCZFMLYLEIM
For generating key, the given keyword is repeated
in a circular manner until it matches the length of
the plain text.
The keyword "AYUSH" generates the key "AYUSHAYUSHAYU"
The plain text is then encrypted using the process
explained below.
```

Encryption
The first letter of the plaintext, G is paired with A, the first letter of the key. So use row G and column A of the Vigenère square, namely G. Similarly, for the second letter of the plaintext, the second letter of the key is used, the letter at row E and column Y is C. The rest of the plaintext is enciphered in a similar fashion.

Decryption
Decryption is performed by going to the row in the table corresponding to the key, finding the position of the ciphertext letter in this row, and then using the column's label as the plaintext. For example, in row A (from AYUSH), the ciphertext G appears in column G, which is the first plaintext letter. Next we go to row Y (from AYUSH), locate the ciphertext C which is found in column E, thus E is the second plaintext letter.
A more **easy implementation** could be to visualize Vigenère algebraically by converting [A-Z] into numbers [0–25].

Encryption
The the plaintext(P) and key(K) are added modulo 26.
$E_i = (P_i + K_i) \bmod 26$

Decryption
$D_i = (E_i - K_i + 26) \bmod 26$

Note: D_i denotes the offset of the i-th character of the plaintext. Like offset of **A** is 0 and of **B** is 1 and so on.
Below is C++ implementation of the idea.

```
// C++ code to implement Vigenere Cipher
#include<bits/stdc++.h>
using namespace std;

// This function generates the key in
// a cyclic manner until it's length
```

```
isi'nt
// equal to the length of original text
string generateKey(string str, string
key)
{
    int x = str.size();

    for (int i = 0; ; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == str.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}

// This function returns the encrypted
text
// generated with the help of the key
string cipherText(string str, string key)
{
    string cipher_text;

    for (int i = 0; i < str.size(); i++)
    {
        // converting in range 0-25
        int x = (str[i] + key[i]) %26;

        // convert into alphabets(ASCII)
        x += 'A';

        cipher_text.push_back(x);
    }
    return cipher_text;
}

// This function decrypts the encrypted
text
// and returns the original text
string originalText(string cipher_text,
string key)
{
    string orig_text;

    for (int i = 0 ; i <
cipher_text.size(); i++)
    {
        // converting in range 0-25
        int x = (cipher_text[i] - key[i] +
26) %26;

        // convert into alphabets(ASCII)
        x += 'A';
        orig_text.push_back(x);
    }
    return orig_text;
}

// Driver program to test the above
function
int main()
{
    string str = "GEEKSFORGEEKS";
    string keyword = "ABHIGYAN";

    string key = generateKey(str,
keyword);
    string cipher_text = cipherText(str,
key);

    cout << "Ciphertext : "
         << cipher_text << "\n";

    cout << "Original/Decrypted Text : "
         << originalText(cipher_text,
key);
    return 0;
}
```

Output:

```
Ciphertext : GCYCZFMLYLEIM
Original/Decrypted Text : GEEKSFORGEEKS
```

Practical:5

Program to Implement the Hill Cypher:

This is a C++ Program to implement hill cipher. In classical cryptography, the Hill cipher is a polygraphic substitution cipher based on linear algebra. Invented by Lester S. Hill in 1929, it was the first polygraphic cipher in which it was practical (though barely) to operate on more than three symbols at once. The following discussion assumes an elementary knowledge of matrices.

Here is source code of the C++ Program to Implement the Hill Cypher. The C++ program is successfully compiled and run on a Linux system. The program output is also shown below.

```
#include<stdio.h>
#include<iostream>

using namespace std;

int check(int x)
{
    if (x % 3 == 0)
        return 0;

    int a = x / 3;
    int b = 3 * (a + 1);
    int c = b - x;

    return c;
}

int main(int argc, char **argv)
{
    int l, i, j;
    int temp1;
    int k[3][3];
    int p[3][1];
    int c[3][1];
    char ch;
    cout
            << "\nThis cipher has a key of length 9. ie. a 3*3 matrix.\nEnter the 9
character key. ";

    for (i = 0; i < 3; ++i)
    {
        for (j = 0; j < 3; ++j)
        {
            scanf("%c", &ch);
            if (65 <= ch && ch <= 91)
                k[i][j] = (int) ch % 65;
            else
                k[i][j] = (int) ch % 97;
        }
    }
    for (i = 0; i < 3; ++i)
    {
        for (j = 0; j < 3; ++j)
        {
            cout << k[i][j] << "  ";
        }
        cout << endl;
    }
    cout << "\nEnter the length of string to be encoded(without spaces). ";
```

```
        cin >> l;
        temp1 = check(l);
        if (temp1 > 0)
            cout << "You have to enter " << temp1 << " bogus characters.";

        char pi[l + temp1];
        cout << "\nEnter the string. ";
        for (i = -1; i < l + temp1; ++i)
    {
            cin >> pi[i];
        }
    int temp2 = l;
        int n = (l + temp1) / 3;
        int temp3;
        int flag = 0;
        int count;
        cout << "\n\nThe encoded cipher is : ";

        while (n > 0)
        {
            count = 0;
            for (i = flag; i < flag + 3; ++i)
            {
                if (65 <= pi[i] && pi[i] <= 91)
                    temp3 = (int) pi[i] % 65;
                else
                    temp3 = (int) pi[i] % 97;

                p[count][0] = temp3;
                count = count + 1;
            }

            int k1;
            for (i = 0; i < 3; ++i)
                c[i][0] = 0;

            for (i = 0; i < 3; ++i)
            {
                for (j = 0; j < 1; ++j)
                {
                    for (k1 = 0; k1 < 3; ++k1)
                        c[i][j] += k[i][k1] * p[k1][j];
                }
            }
            for (i = 0; i < 3; ++i)
            {
                c[i][0] = c[i][0] % 26;
                printf("%c ", (char) (c[i][0] + 65));
            }
            n = n - 1;
            flag = flag + 3;
        }
    }
```

Output

```
$ ++ HillCipher.cpp
$a.out
 This cipher has a key of length 9. ie. a 3*3 matrix.
nter the 9 character key. DharHingu
3  7  0
```

```
17  7  8
13  6  20

Enter the length of string to be encoded(without spaces). 10
You have to enter 2 bogus characters.
Enter the string. Sanfoundry

The encoded cipher is : N B W A O Q Y Y X X D O
------------------
```

Practical:6

RSA Algorithm in Cryptography

RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. **Public Key** and **Private Key.** As the name describes that the Public Key is given to everyone and Private key is kept private.

An example of asymmetric cryptography :

1. A client (for example browser) sends its public key to the server and requests for some data.
2. The server encrypts the data using client's public key and sends the encrypted data.
3. Client receives this data and decrypts it.

Since this is asymmetric, nobody else except browser can decrypt the data even if a third party has public key of browser.

The idea! The idea of RSA is based on the fact that it is difficult to factorize a large integer. The public key consists of two numbers where one number is multiplication of two large prime numbers. And private key is also derived from the same two prime numbers. So if somebody can factorize the large number, the private key is compromised. Therefore encryption strength totally lies on the key size and if we double or triple the key size, the strength of encryption increases exponentially. RSA keys can be typically 1024 or 2048 bits long, but experts believe that 1024 bit keys could be broken in the near future. But till now it seems to be an infeasible task.

Let us learn the mechanism behind RSA algorithm :

>> Generating Public Key :

- Select two prime no's. Suppose **P = 53 and Q = 59.**
- Now First part of the Public key : **n = P*Q = 3127.**

- We also need a small exponent say **e** :
- But e Must be
-
  - An integer.
-
  - Not be a factor of n.
-
  - **1 < e < Φ(n)** [Φ(n) is discussed below],
  - Let us now consider it to be equal to 3.

- Our Public Key is made of n and e

>> Generating Private Key :

- We need to calculate Φ(n) :
- Such that **Φ(n) = (P-1)(Q-1)**
- so, Φ(n) = 3016

- Now calculate Private Key, **d** :
- **d = (k*Φ(n) + 1) / e** for some integer k
- For k = 2, value of d is 2011.

Now we are ready with our – Public Key (n = 3127 and e = 3) and Private Key(d = 2011)

Now we will encrypt **"HI"** :

- Convert letters to numbers : H = 8 and I = 9

- Thus **Encrypted Data c = 89^e mod n.**
- Thus our Encrypted Data comes out to be 1394

```
Now we will decrypt 1349 :
```

- **Decrypted Data = c^d mod n.**
- Thus our Encrypted Data comes out to be 89

8 = H and I = 9 i.e. "HI".

Below is C implementation of RSA algorithm for small values:

```
// C program for RSA asymmetric
cryptographic
// algorithm. For demonstration
values are
// relatively small compared to
practical
// application
#include<stdio.h>
#include<math.h>

// Returns gcd of a and b
int gcd(int a, int h)
{
    int temp;
    while (1)
    {
        temp = a%h;
        if (temp == 0)
          return h;
        a = h;
        h = temp;
    }
}

// Code to demonstrate RSA
algorithm
int main()
{
    // Two random prime numbers
    double p = 3;
    double q = 7;

    // First part of public key:
    double n = p*q;

    // Finding other part of
public key.
    // e stands for encrypt
    double e = 2;
    double phi = (p-1)*(q-1);
    while (e < phi)
    {
        // e must be co-prime to
phi and
        // smaller than phi.
        if (gcd(e, phi)==1)
            break;
        else
            e++;
    }

    // Private key (d stands for
decrypt)
    // choosing d such that it
satisfies
    // d*e = 1 + k * totient
    int k = 2;  // A constant
value
    double d = (1 + (k*phi))/e;

    // Message to be encrypted
    double msg = 20;

    printf("Message data = %lf",
msg);

    // Encryption c = (msg ^ e) %
n
    double c = pow(msg, e);
    c = fmod(c, n);
    printf("\nEncrypted data =
%lf", c);

    // Decryption m = (c ^ d) % n
    double m = pow(c, d);
    m = fmod(m, n);
    printf("\nOriginal Message
Sent = %lf", m);

    return 0;
}
```

Output :

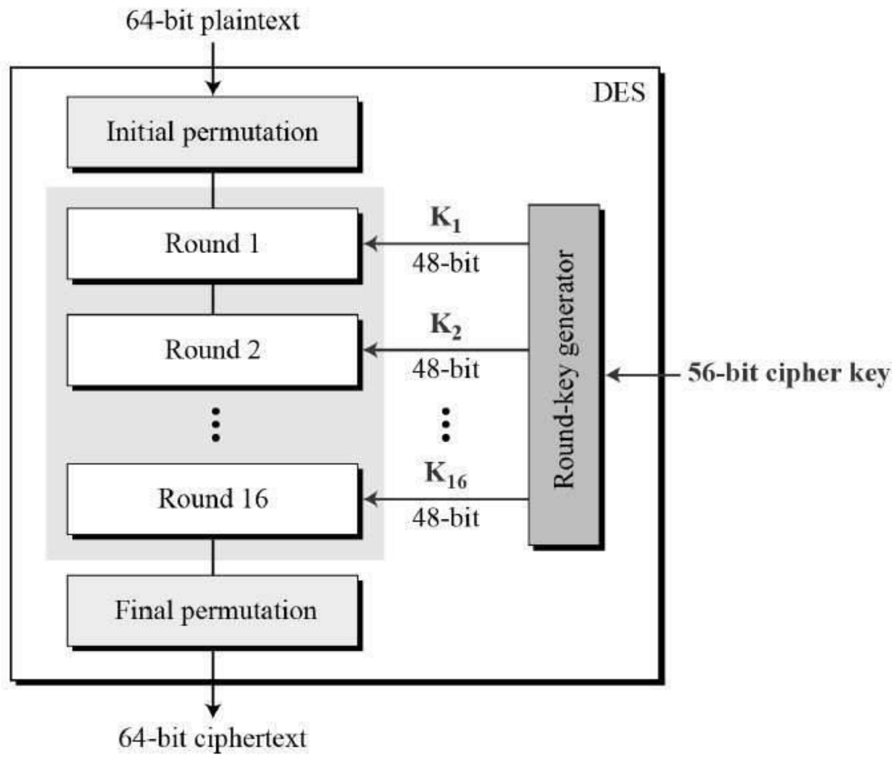
```
Message data = 12.000000
Encrypted data = 3.000000
Original Message Sent = 12.000000
```

Practical:7

The Data Encryption Standard (DES):

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

DES is an implementation of a Feistel Cipher. It uses 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration −



Since DES is based on the Feistel Cipher, all that is required to specify DES is −

- Round function
- Key schedule
- Any additional processing − Initial and final permutation

Practical:8

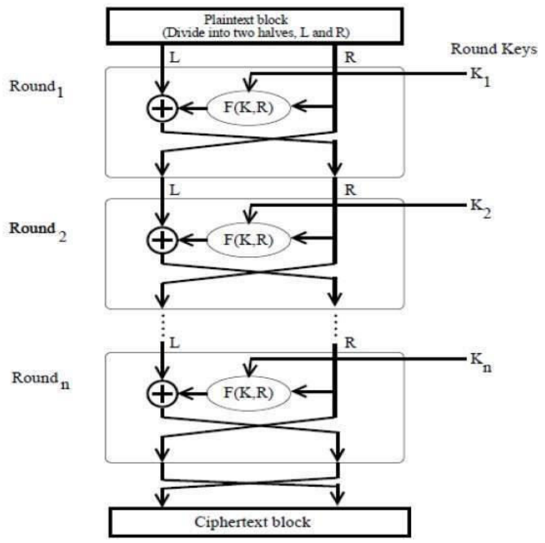
Feistel Cipher:

Feistel Cipher is not a specific scheme of block cipher. It is a design model from which many different block ciphers are derived. DES is just one example of a Feistel Cipher. A cryptographic system based on Feistel cipher structure uses the same algorithm for both encryption and decryption.

Encryption Process

The encryption process uses the Feistel structure consisting multiple rounds of processing of the plaintext, each round consisting of a "substitution" step followed by a permutation step.

Feistel Structure is shown in the following illustration −



- The input block to each round is divided into two halves that can be denoted as L and R for the left half and the right half.
- In each round, the right half of the block, R, goes through unchanged. But the left half, L, goes through an operation that depends on R and the encryption key. First, we apply an encrypting function 'f' that takes two input − the key K and R. The function produces the output f(R,K). Then, we XOR the output of the mathematical function with L.
- In real implementation of the Feistel Cipher, such as DES, instead of using the whole encryption key during each round, a round-dependent key (a subkey) is derived from the encryption key. This means that each round uses a different key, although all these subkeys are related to the original key.
- The permutation step at the end of each round swaps the modified L and unmodified R. Therefore, the L for the next round would be R of the current round. And R for the next round be the output L of the current round.
- Above substitution and permutation steps form a 'round'. The number of rounds are specified by the algorithm design.

- Once the last round is completed then the two sub blocks, ‘R’ and ‘L’ are concatenated in this order to form the ciphertext block.

The difficult part of designing a Feistel Cipher is selection of round function ‘f’. In order to be unbreakable scheme, this function needs to have several important properties that are beyond the scope of our discussion.

Decryption Process

The process of decryption in Feistel cipher is almost similar. Instead of starting with a block of plaintext, the ciphertext block is fed into the start of the Feistel structure and then the process thereafter is exactly the same as described in the given illustration.

The process is said to be almost similar and not exactly same. In the case of decryption, the only difference is that the subkeys used in encryption are used in the reverse order.

The final swapping of ‘L’ and ‘R’ in last step of the Feistel Cipher is essential. If these are not swapped then the resulting ciphertext could not be decrypted using the same algorithm.

Number of Rounds

The number of rounds used in a Feistel Cipher depends on desired security from the system. More number of rounds provide more secure system. But at the same time, more rounds mean the inefficient slow encryption and decryption processes. Number of rounds in the systems thus depend upon efficiency–security tradeoff.