

## 1. Write program for Ceaser cipher encryption and decryption.

**Aim:** To write a program for Ceaser cipher encryption and decryption.

**Description:** In cryptography, a Caesar cipher, also known as Caesar's cipher, the shift cipher, Caesar's code or Caesar shift, is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

### Program Code:

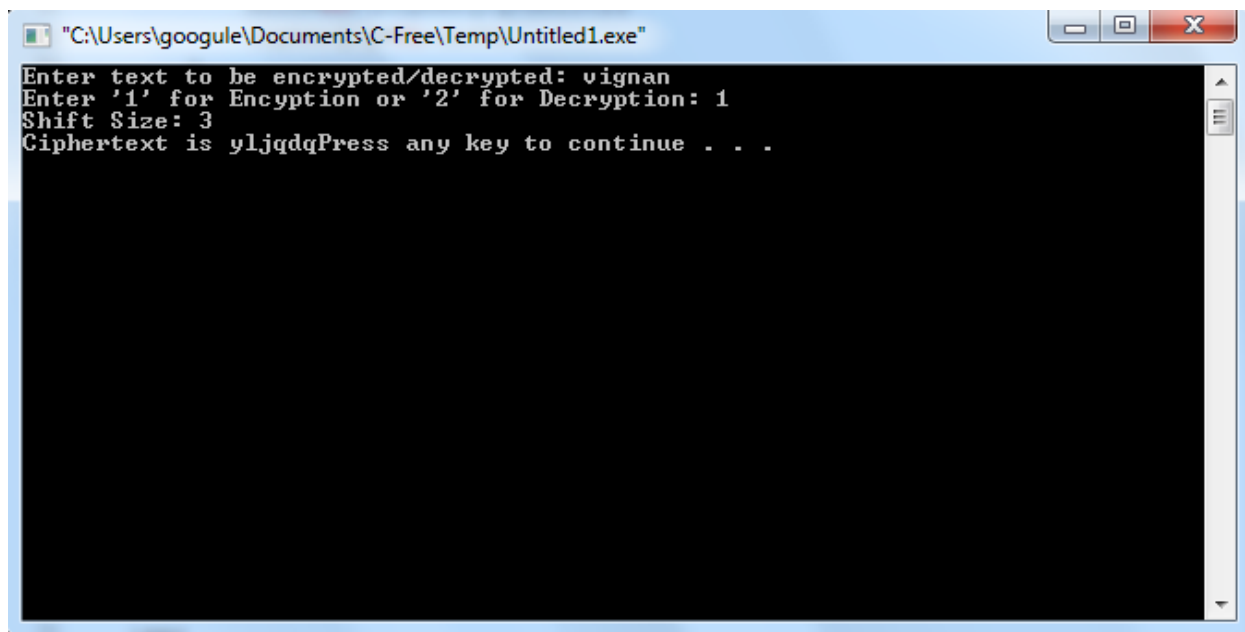
```
#include <stdio.h>
#include <string.h>
int main () {
    char cipher[150];
    int shift,i=0,diff;
    int opt;
    printf("Enter text to be encrypted/decrypted: ");
    scanf("%s", cipher);
    printf("Enter '1' for Encryption or '2' for Decryption: ");
    scanf("%d",&opt);
    printf("Shift Size: ");
    scanf("%d", &shift);
    if(opt==1)
    {
        while (cipher[i] != '\0') {
            if(cipher[i]>=65 && cipher[i]<=90)
            {
                if((cipher[i]+shift)>90)
                {
                    diff=(cipher[i]+shift)-90;
                    cipher[i]=64+diff;
                }
                else
                {
                    cipher[i]=cipher[i]+shift;
                }
            }
            if(cipher[i]>=97 && cipher[i]<=122)
            {
                if((cipher[i]+shift)>122)
                {
                    diff=(cipher[i]+shift)-122;
                    cipher[i]=97+diff;
                }
                else
                {

```

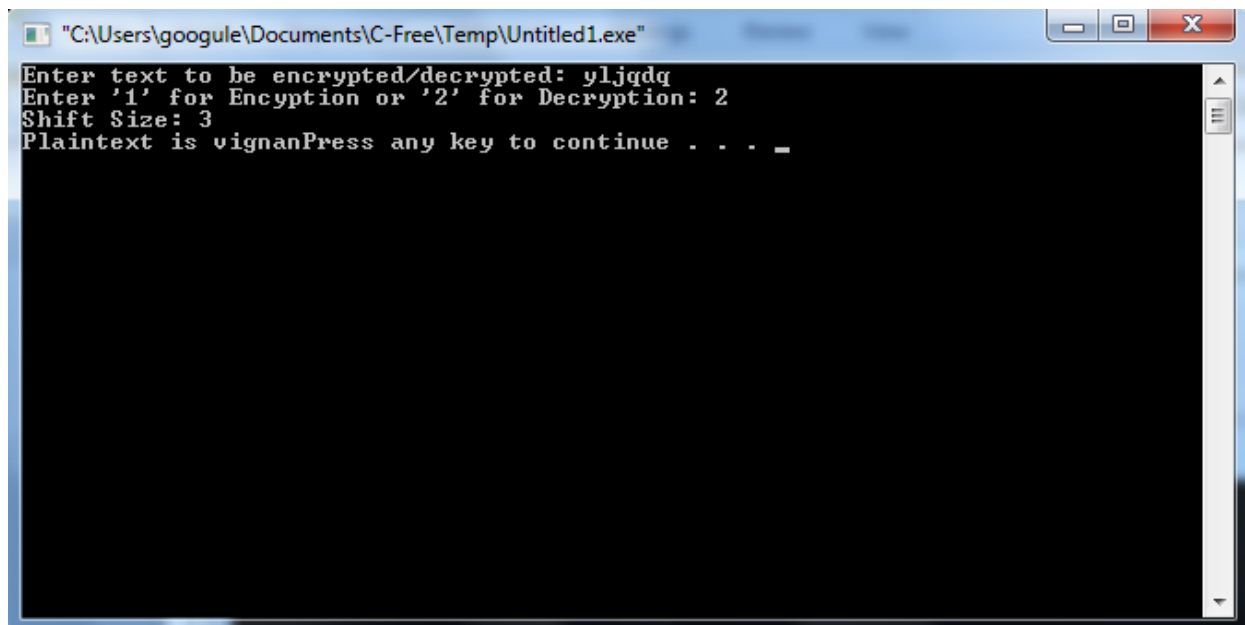
```

        cipher[i]=cipher[i]+shift;
    }
    }
    i++;
}
printf("%s", cipher);
}
if(opt==2)
{
    while (cipher[i] != '\0') {
        if(cipher[i]>=65 && cipher[i]<=90)
        {
            if((cipher[i]-shift)<65)
            {
                diff=65-(cipher[i]-shift);
                cipher[i]=91-diff;
            }
            else
            {
                cipher[i]=cipher[i]-shift;
            }
        }
        if(cipher[i]>=97 && cipher[i]<=122)
        {
            if((cipher[i]-shift)<97)
            {
                diff=97-(cipher[i]-shift);
                cipher[i]=122-diff;
            }
            else
            {
                cipher[i]=cipher[i]-shift;
            }
        }
        i++;
    }
    printf("%s", cipher);
}
return 0;
}

```

**Output for encryption:**

```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"  
Enter text to be encrypted/decrypted: vignan  
Enter '1' for Encyption or '2' for Decryption: 1  
Shift Size: 3  
Ciphertext is yljqdqPress any key to continue . . .
```

**Output for decryption:**

```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"  
Enter text to be encrypted/decrypted: yljqdq  
Enter '1' for Encyption or '2' for Decryption: 2  
Shift Size: 3  
Plaintext is vignanPress any key to continue . . .
```

## 2. Write program for Mono alphabetic cipher encryption and decryption

**Aim:** To write a program for monoalphabetic Encryption and Decryption

**Description:** A monoalphabetic substitution cipher, also known as a simple substitution cipher, relies on a fixed replacement structure. That is, the substitution is fixed for each letter of the alphabet. Thus, if "a" is encrypted to "R", then every time we see the letter "a" in the plaintext, we replace it with the letter "R" in the ciphertext. A simple example is where each letter is encrypted as the next letter in the alphabet: "a simple message" becomes "B TJNQMF NFFTBFH".

### Program Code:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char plain[26]="abcdefghijklmnopqrstuvwxyz";
    char cipher[26],array[50],str1[50],ch,str2[50];
    int i=0,m,l;
    printf("Enter the string:");
    while((ch=getchar())!='\n'){
        array[i]=ch;
        i++;
    }
    printf("%s\n",plain);
    for(i=0;i<26;i++)
    {
        int random = (i+(i*8))%26;

        cipher[i] = plain[random];

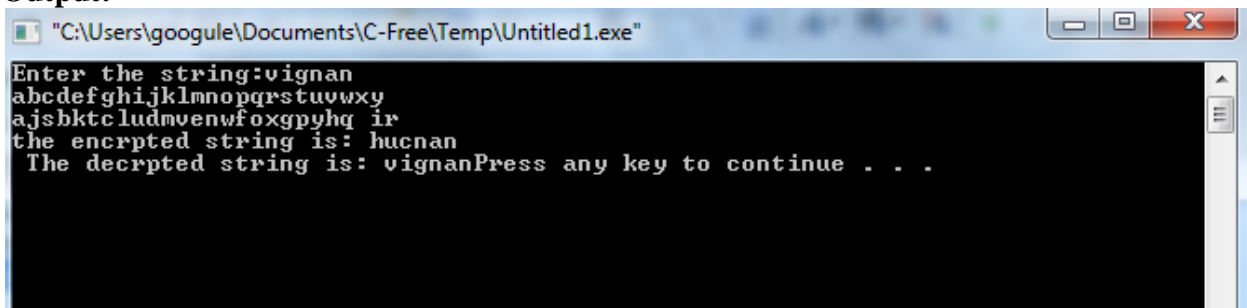
        printf("%c",cipher[i]);
    }
    printf("\n");
    printf("The encrypted string is: ");
    l=strlen(array);
    for(i=0;i<l;i++)
    {
        if(array[i]!=' ')
        {
            for(m=0;m<26;m++)
            {
                char c=plain[m];
                char d=array[i];
                if(c==d)
                {
                    str1[i]=cipher[m];
                }
            }
        }
    }
}
```

```

        printf("%c",str1[i]);
    }
}
else
{
    str1[i]=' ';
    printf("%c",str1[i]);
}
}
printf("\n The decrpted string is: ");
for(i=0;i<l;i++)
{
    if(str1[i]!=' ')
    {
        for(m=0;m<26;m++)
        {
            char c=cipher[m];
            char d=str1[i];
            if(c==d)
            {
                str2[i]=plain[m];
                printf("%c",str2[i]);
            }
        }
    }
}
else
{
    str1[i]=' ';
    printf("%c",str1[i]);
}
}
return 0;
}

```

### Output:



```

C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe
Enter the string:vignan
abcdefghijklmnopqrstuvwxyz
ajshbktcludmvenwfoxgpyhq ir
the encrpted string is: hucnan
The decrpted string is: vignan
Press any key to continue . . .

```

### 3. Implementation of Play Fair cipher

**Aim:** To write a c program for playfair cipher algorithm.

**Description:** The Playfair cipher is a manual symmetric encryption technique. The technique encrypts pairs of letters, instead of single letters as in the simple substitution cipher and rather more complex systems then in use. The Playfair is thus significantly harder to break since the frequently analysis used for simple substitution ciphers does not work with it. The frequency analysis of diagrams is possible, but considerably more difficult. With 600 possible diagrams rather than the 26 possible monograms (single symbols, usually letters in this context), a considerably larger cipher text is required in order to be useful.

#### Program Code:

```
#include <stdio.h>
#define siz 5
void encrypt(int *i, int *j)
{
    (*i)++, (*j)++;
    if((*i)==siz) *i=0;
    else if((*j)==siz) *j=0;
}
void playfair(char ch1, char ch2, char mat[siz][siz])
{
    int j, m, n, p, q, c, k;
    for(j=0, c=0; (c<2) || (j<siz); j++)
        for(k=0; k<siz; k++)
            if(mat[j][k] == ch1)
                m=j, n=k, c++;
            else if(mat[j][k] == ch2)
                p=j, q=k, c++;
            if(m==p)
                encrypt(&n, &q);
            else if(n==q)
                encrypt(&m, &p);
            else
                n+=q, q=n-q, n-=q;
    printf("%c%c", mat[m][n], mat[p][q]);
}
int main()
{
    char mat[siz][siz], key[10], str[25]={0};
    int m, n, i, j;
    char temp;
    printf("Enter Key:");
    gets (key);
    m=n=0;
```

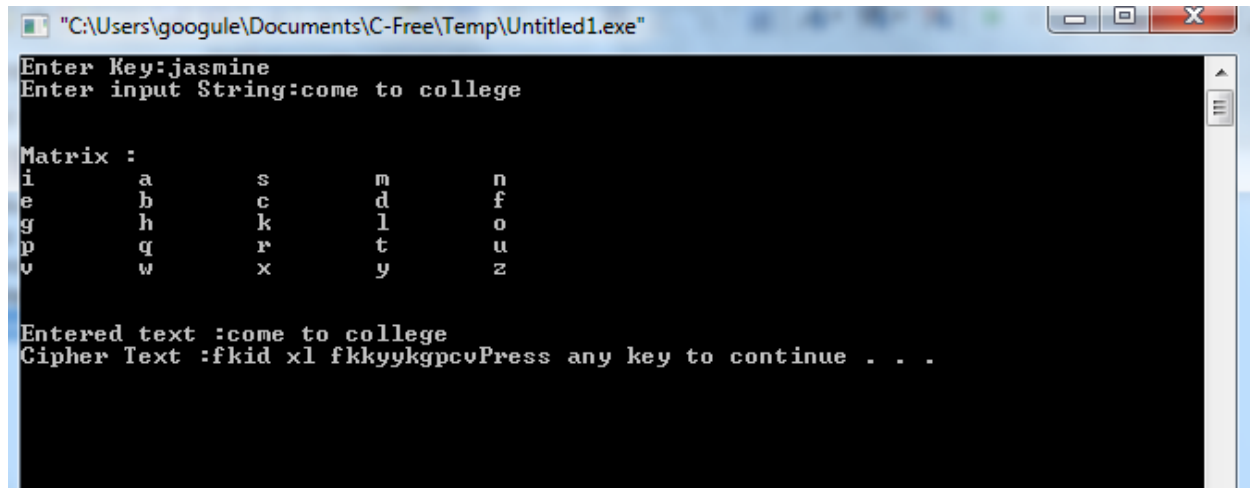
```

for(i=0;key[i]!='\0';i++)
{
for(j=0;j<i;j++)
if(key[j] == key[i]) break;
if(key[i]=='j') key[i]='i';
if(j>=i)
{
mat[m][n++] = key[i];
if(n==siz)
n=0,m++;
}
}
for(i=97;i<=122;i++)
{
for(j=0;key[j]!='\0';j++)
if(key[j] == i)
break;
else if(i=='j')
break;
if(key[j]=='\0')
{
mat[m][n++] = i;
if(n==siz) n=0,m++;
}
}
printf("Enter input String:");
gets(str);
printf("\n\nMatrix : \n");
for(i=0;i<siz;i++)
{
for(j=0;j<siz;j++)
printf("%c\t",mat[i][j]);
printf("\n");
}
printf("\n\nEntered text : %s\nCipher Text :",str);
for(i=0;str[i]!='\0';i++)
{
temp = str[i++];
if(temp == 'j') temp='i';
if(str[i]=='\0')
playfair(temp,'x',mat);
else
{
if(str[i]=='j') str[i]='i';
if(temp == str[i])
{
playfair(temp,'x',mat);

```

```
playfair('x',str[i],mat);  
}  
else  
playfair(temp,str[i],mat);  
}  
}  
}
```

### Output:



```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"  
Enter Key:jasmine  
Enter input String:come to college  
  
Matrix :  
i      a      s      m      n  
e      b      c      d      f  
g      h      k      l      o  
p      q      r      t      u  
v      w      x      y      z  
  
Entered text :come to college  
Cipher Text :fkid xl fkkyykgpcvPress any key to continue . . .
```



#### 4. Implementation of Vigenere cipher

**Aim:-**Implementing Vigenere cipher encryption and decryption

**Description:** A Vigenere cipher shifts each character of a plain text message a number of positions based on a keyword. Essentially, a Vigenere cipher consists of several Caesar ciphers in sequence with different shift values. The shift value for any given character is based on the keyword. The keyword is repeated so that it is the same length of the message. Then, the corresponding keyword character determines the shift for its respective message character.

#### Program Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <process.h>
void vigenereCipher(char *, char *);
void encipher();
void decipher();
int main()
{
    int choice;
    //loop takes choice from user and calls appropriate
    function
    {
        printf("1. Encrypt Text\n");
        printf("2. Decrypt Text\n");
        printf("3. Exit\n");
        printf("Enter Your Choice : ");
        scanf("%d", &choice);
        fflush(stdin);
        if(choice == 3)
            exit(0);
        else if(choice == 1)
            encipher();
        else if(choice == 2)
            decipher();
        else
            printf("Please Enter Valid Option.");
    }
    return 0;
}

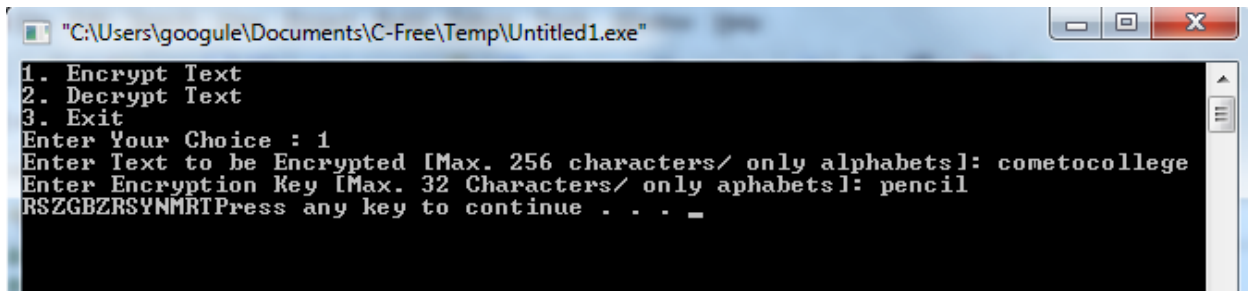
void encipher()
{
    unsigned int i, j;
    char input[257], key[33];
    printf("Enter Text to be Encrypted [Max. 256 characters/ only
    alphabets]: ");
```

```

gets(input);
printf("Enter Encryption Key [Max. 32 Characters/ only
alphabets]: ");
gets(key);
for(i=0,j=0;i<strlen(input);i++,j++)
{
    //repeat the key if you are at end of it.
    if(j>=strlen(key))
    {
        j=0;
    }
    //actual logic -> character from input + character from key % 26
    is encrypted charater
    printf("%c",65+(((toupper(input[i])-65)+(toupper(key[j])-
65))%26));
}
}
void decipher()
{
    unsigned int i,j;
    char input[257],key[33];
    int value;
    printf("Enter Text to be Decrypted [Max. 256 characters/ only
alphabets]:n ");
    gets(input);
    printf("Enter Decryption Key [Max. 32 Characters/ only
alphabets]: ");
    gets(key);
    for(i=0,j=0;i<strlen(input);i++,j++)
    {
        //repeat the key if you are at end of it.
        if(j>=strlen(key))
        {
            j=0;
        }
        //similar to encipher only difference is you need to
        subtract
        value = (toupper(input[i])-64)-(toupper(key[j])-64);
        //make positive if value is negative.
        if( value < 0)
        {
            value = value * -1;
        }
        printf("%c",65 + (value % 26));
    }
}

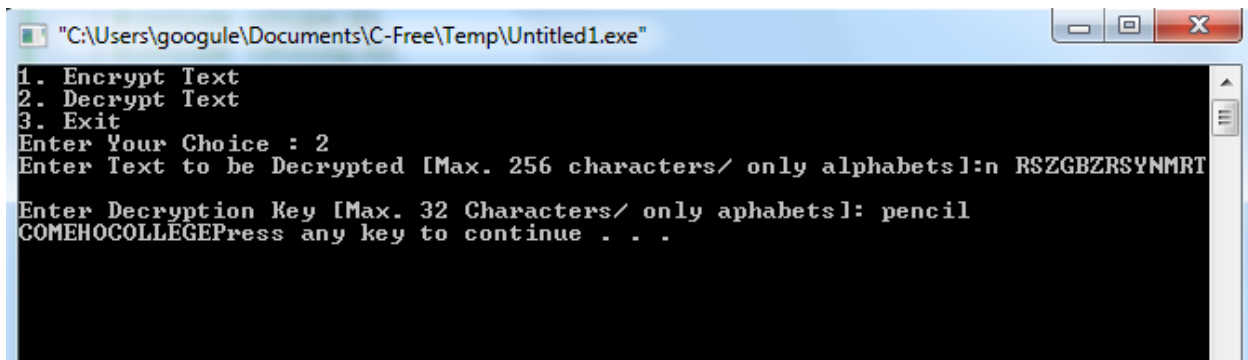
```

### Output for Encryption:



```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"
1. Encrypt Text
2. Decrypt Text
3. Exit
Enter Your Choice : 1
Enter Text to be Encrypted [Max. 256 characters/ only alphabets]: cometocollege
Enter Encryption Key [Max. 32 Characters/ only alphabets]: pencil
RSZGBZRSYNMRTPress any key to continue . . . _
```

### Output for Decryption:



```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"
1. Encrypt Text
2. Decrypt Text
3. Exit
Enter Your Choice : 2
Enter Text to be Decrypted [Max. 256 characters/ only alphabets]: RSZGBZRSYNMRT
Enter Decryption Key [Max. 32 Characters/ only alphabets]: pencil
COMEHOCOLLEGEPress any key to continue . . . _
```

## 5. Implementation of Hill cipher

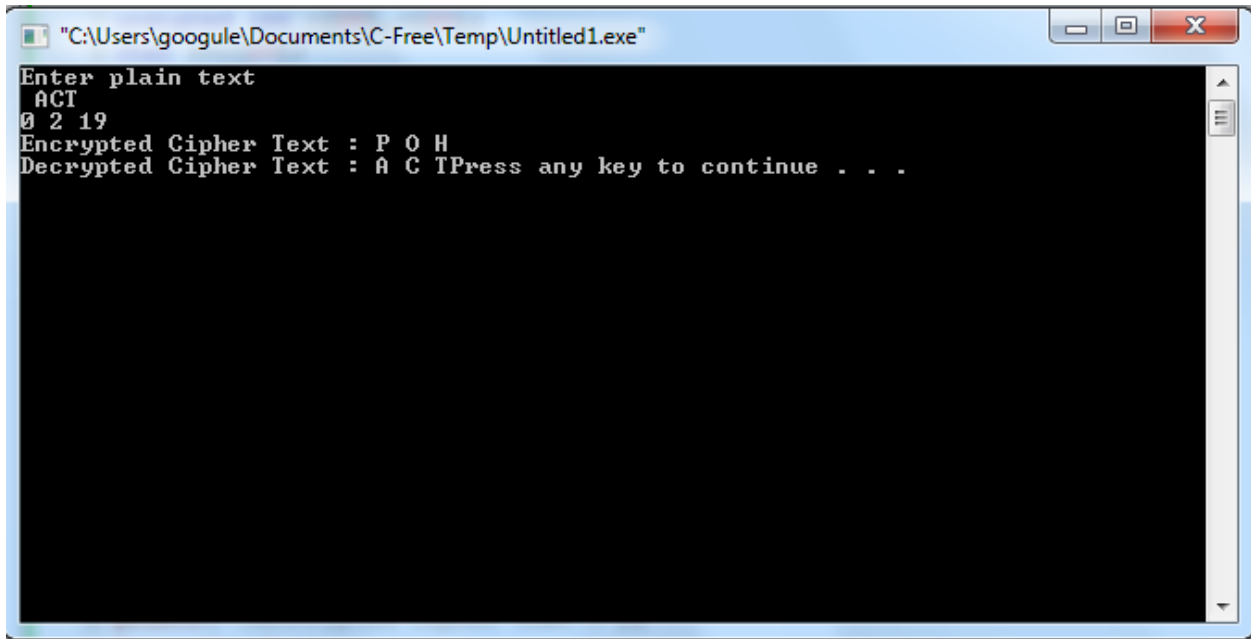
**Aim:** To write a program for hill cipher

**Description:** 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. Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, ..., Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of  $n$  letters (considered as an  $n$ -component vector) is multiplied by an invertible  $n \times n$  matrix, again modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

### Program Code:

```
#include<stdio.h>
#include<string.h>
int main(){
unsigned int a[3][3]={6,24,1},{13,16,10},{20,17,15}};
unsigned int b[3][3]={8,5,10},{21,8,21},{21,12,8}};
int i,j;
unsigned int c[20],d[20];
char msg[20];
int determinant=0,t=0;;
printf("Enter plain text\n ");
scanf("%s",msg);
for(i=0;i<3;i++)
{
c[i]=msg[i]-65;
printf("%d ",c[i]);
}
for(i=0;i<3;i++)
{
t=0;
for(j=0;j<3;j++)
{
t=t+(a[i][j]*c[j]);
}
d[i]=t%26;
}
printf("\nEncrypted Cipher Text :");
for(i=0;i<3;i++)
printf(" %c",d[i]+65);
for(i=0;i<3;i++)
{
t=0;
for(j=0;j<3;j++)
{
```

```
t=t+(b[i][j]*d[j]);  
}  
c[i]=t%26;  
}  
printf("\nDecrypted Cipher Text :");  
for(i=0;i<3;i++)  
printf(" %c",c[i]+65);  
return 0;  
}
```

**Output:**

```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"  
Enter plain text  
ACT  
2 19  
Encrypted Cipher Text : P O H  
Decrypted Cipher Text : A C T  
Press any key to continue . . .
```

## 6. Implementation of Rail Fence cipher

**Aim:** To write a program for railfence algorithm.

**Description:** In cryptography, a **transposition cipher** is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the cipher text constitutes a permutation of the plaintext. That is, the order of the units is changed (the plaintext is reordered). Mathematically a bijective function is used on the characters' positions to encrypt and an inverse function to decrypt.

### Program Code:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define VAL(PTR) (*PTR)
typedef char* STRING;
typedef char** STRING_REF;
void flush_stdin()
{
    while(fgetc(stdin) != '\n');
}
void rail_fence_encrypt(STRING_REF str, int depth)
{
    STRING old_str = VAL(str);
    int len_str = strlen(old_str), itr_str, rows, cols;
    STRING res_str = (STRING)malloc(sizeof(char)*len_str);
    for(rows = 0, itr_str = 0 ; rows < depth ; ++rows)
    {
        for(cols = rows ; cols < len_str ; cols += depth)
        {
            res_str[itr_str] = old_str[cols];
            ++itr_str;
        }
        res_str[itr_str] = '\0';
        free(VAL(str));
        VAL(str) = res_str;
    }
}
void rail_fence_decrypt(STRING_REF str, int depth)
{
    STRING old_str = VAL(str);
    int len_str = strlen(old_str), itr_str, rows, counter;
    STRING res_str = (STRING)malloc(sizeof(char)*len_str);
    for(counter = 0, itr_str = 0 ; counter < len_str/depth ;
    ++counter)
    {
```

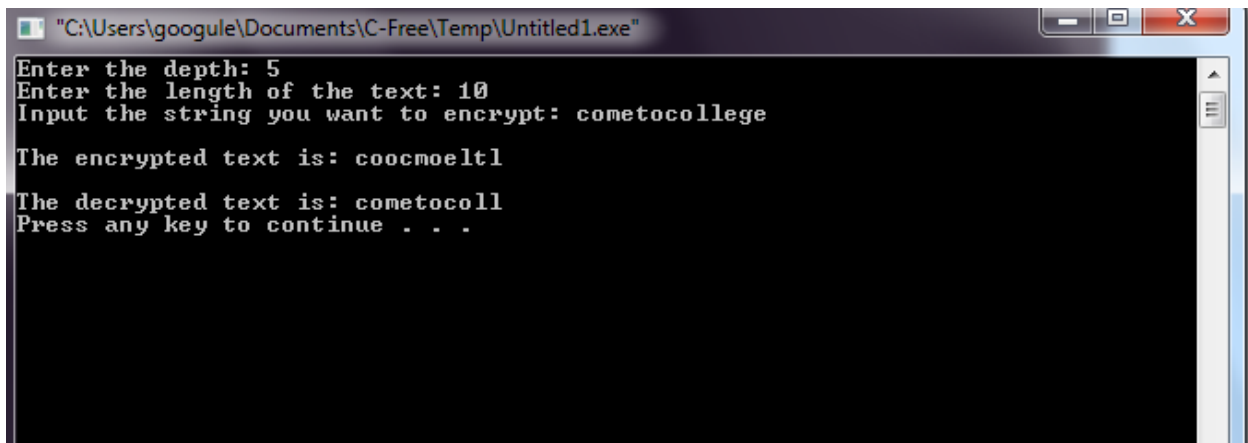
```

        for(rows = counter ; rows < len_str ;
rows+=(len_str/depth))
        {
            res_str[itr_str] = old_str[rows];
            ++itr_str;
        }
    }
    free(VAL(str));
    VAL(str) = res_str;
}
int main(void)
{
    STRING str;
    int txt_length, key;

    printf("Enter the depth: ");
    scanf("%d",&key);
    printf("Enter the length of the text: ");
    scanf("%d",&txt_length);
    printf("Input the string you want to encrypt: ");
    flush_stdin();
    str = (char*)malloc((txt_length+1)*sizeof(char));
    fgets(str,txt_length+1,stdin);
    rail_fence_encrypt(&str,key);
    printf("\nThe encrypted text is: %s\n",str);
    rail_fence_decrypt(&str,key);
    printf("\nThe decrypted text is: %s\n",str);
    return 0;
}

```

### Output:



```

C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe
Enter the depth: 5
Enter the length of the text: 10
Input the string you want to encrypt: cometocollege

The encrypted text is: coocmoeltl
The decrypted text is: cometocoll
Press any key to continue . . .

```

## 7. Implementation of One-time pad cipher

**Aim:-** To write a program to implement One-time pad cipher

**Description:** In cryptography, the **one-time pad (OTP)** is an encryption technique that cannot be cracked if used correctly. In this technique, a plaintext is paired with a random secret key (also referred to as *a one-time pad*). Then, each bit or character of the plaintext is encrypted by combining it with the corresponding bit or character from the pad using modular addition. If the key is truly random, is at least as long as the plaintext, is never reused in whole or in part, and is kept completely secret, then the resulting ciphertext will be impossible to decrypt or break. It has also been proven that any cipher with the perfect secrecy property must use keys with effectively the same requirements as OTP keys.

### Program Code:-

```
#include<iostream>
#include<vector>
#include<stdlib.h>
using namespace std;
void to_upper_case(vector<char>& text, int len)
{
    for (int i = 0; i < len; i++)
    {   if (text[i] >= 97 && text[i] <= 122)
        text[i] -= 32;
    }
}
void print_string(vector<char> text, int len)
{
    for (int i = 0; i < len; i++)
    { cout << (char) (text[i] + 65);
    }
    cout << endl;
    return;
}
size_t get_input(vector<char>& msg)
{
    char a;
    while (1)
    {   a = getchar();
        if (a == '\n')
            break;
        msg.push_back(a);
    }
    return msg.size();
}
int main()
{   vector<char> msg;
    vector<char> enc_msg;
```

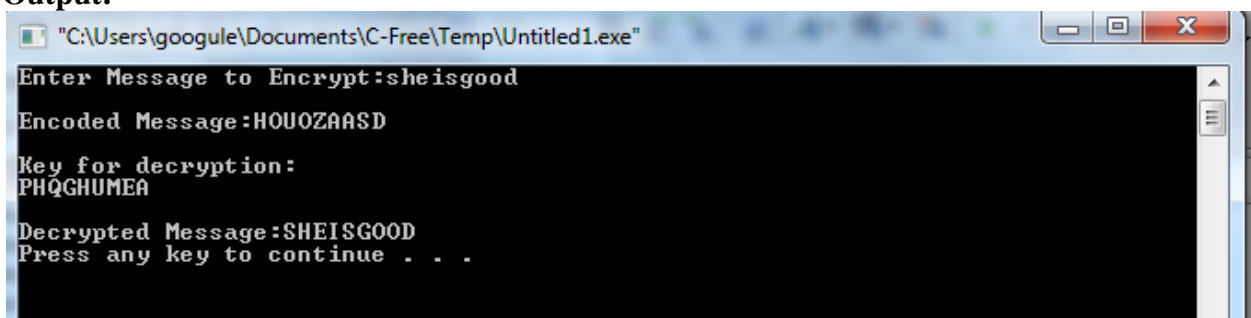


```

vector<char> dec_msg;
int *p;
int i;
size_t len;
cout << "Enter Message to Encrypt:";
len = get_input(msg);
to_upper_case(msg, len);
p = (int*) malloc(msg.size() * sizeof(int));
for (i = 0; i < len; i++)
{
    p[i] = rand() % 26;
    if (msg[i] >= 65 && msg[i] <= 90)
        enc_msg.push_back((char) ((msg[i] - 65 + p[i]) % 26));
    else if (msg[i] >= 97 && msg[i] <= 122)
        enc_msg.push_back((char) ((msg[i] - 97 + p[i]) % 26));
    else
        enc_msg.push_back((char) msg[i]);
}
cout << "\nEncoded Message:";
print_string(enc_msg, len);
cout << "\nKey for decryption:\n";
for (i = 0; i < len; i++)
{ cout << (char) (p[i] + 65);
}
cout << endl;
cout << "\nDecrypted Message:";
for (i = 0; i < len; i++)
{ if ((enc_msg[i] - p[i]) < 0)
    dec_msg.push_back((char) (enc_msg[i] - p[i] + 26));
  else if ((enc_msg[i] - p[i]) >= 0)
    dec_msg.push_back((char) (enc_msg[i] - p[i]));
  else
    dec_msg.push_back((char) enc_msg[i]);
}
print_string(dec_msg, len);
return 0;
}

```

### Output:



```

C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe
Enter Message to Encrypt:sheisgood
Encoded Message:HOUOZAASD
Key for decryption:
PHQGHUMEA
Decrypted Message:SHEISGOOD
Press any key to continue . . .

```

## 8. Implement RSA asymmetric Encryption

**Aim:-** Implementing RSA asymmetric (public key and private key) encryption

**Description:** RSA is the algorithm used by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public key cryptography,

1. Choose two distinct prime numbers  $p$  and  $q$ .
2. Find  $n$  such that  $n = pq$ .
3. Find the totient of  $n$ ,  $\phi(n) = (p-1)(q-1)$ .
4. Choose an integer  $e$  such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ ; i.e.,  $e$  and  $\phi(n)$  are coprime.
5. Determine  $d$  as  $d \equiv e^{-1} \pmod{\phi(n)}$ ; i.e.,  $d$  is the modular multiplicative inverse of  $e$  (modulo  $\phi(n)$ )
6. uses public key  $(e, n)$  for Encryption.  $c = m^e \pmod{n}$ .
7. private key  $(n, d)$  is used for Decryption  $m = c^d \pmod{n}$ .

### Program Code:-

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
long int
p,q,n,t,flag,e[100],d[100],temp[100],j,m[100],en[100],i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
void main() {
    printf("\nENTER FIRST PRIME NUMBER\n");
    scanf("%d",&p);
    flag=prime(p);
    if(flag==0) {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("\nENTER ANOTHER PRIME NUMBER\n");
    scanf("%d",&q);
    flag=prime(q);
    if(flag==0||p==q) {
        printf("\nWRONG INPUT\n");
        exit(1);
    }
    printf("\nENTER MESSAGE\n");
    fflush(stdin);
```

```

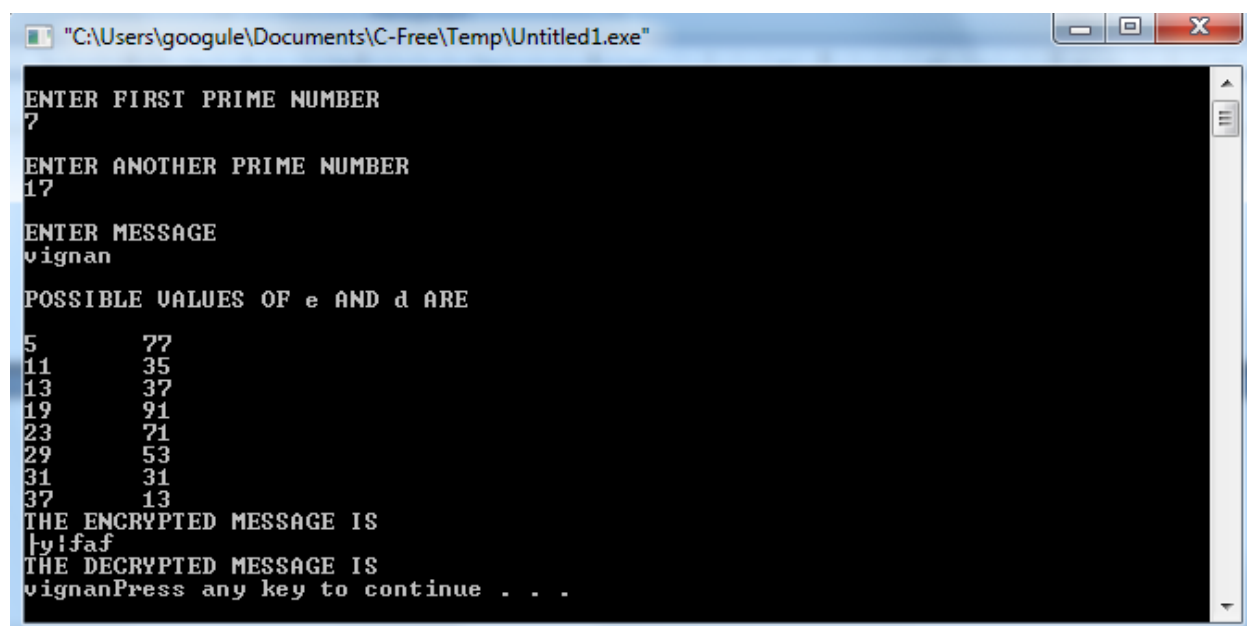
scanf("%s",msg);
for (i=0;msg[i]!=NULL;i++)
m[i]=msg[i];
n=p*q;
t=(p-1)*(q-1);
ce();
printf("\nPOSSIBLE VALUES OF e AND d ARE\n");
for (i=0;i<j-1;i++)
printf("\n%ld\t%ld",e[i],d[i]);
encrypt();
decrypt();
}
int prime(long int pr) {
    int i;
    j=sqrt(pr);
    for (i=2;i<=j;i++) {
        if(pr%i==0)
            return 0;
    }
    return 1;
}
void ce() {
    int k;
    k=0;
    for (i=2;i<t;i++) {
        if(t%i==0)
            continue;
        flag=prime(i);
        if(flag==1&&i!=p&&i!=q) {
            e[k]=i;
            flag=cd(e[k]);
            if(flag>0) {
                d[k]=flag;
                k++;
            }
            if(k==99)
                break;
        }
    }
}
long int cd(long int x) {
    long int k=1;
    while(1) {
        k=k+t;
        if(k%x==0)
            return(k/x);
    }
}

```

```

}
void encrypt() {
    long int pt,ct,key=e[0],k,len;
    i=0;
    len=strlen(msg);
    while(i!=len) {
        pt=m[i];
        pt=pt-96;
        k=1;
        for (j=0;j<key;j++) {
            k=k*pt;
            k=k%n;
        }
        temp[i]=k;
        ct=k+96;
        en[i]=ct;
        i++;
    }
    en[i]=-1;
    printf("\nTHE ENCRYPTED MESSAGE IS\n");
    for (i=0;en[i]!=-1;i++)
        printf("%c",en[i]);
}
void decrypt() {
    long int pt,ct,key=d[0],k;
    i=0;
    while(en[i]!=-1) {
        ct=temp[i];
        k=1;
        for (j=0;j<key;j++) {
            k=k*ct;
            k=k%n;
        }
        pt=k+96;
        m[i]=pt;
        i++;
    }
    m[i]=-1;
    printf("\nTHE DECRYPTED MESSAGE IS\n");
    for (i=0;m[i]!=-1;i++)
        printf("%c",m[i]);
}

```

**Output:**

```
"C:\Users\googule\Documents\C-Free\Temp\Untitled1.exe"

ENTER FIRST PRIME NUMBER
7

ENTER ANOTHER PRIME NUMBER
17

ENTER MESSAGE
vignan

POSSIBLE VALUES OF e AND d ARE

5      77
11     35
13     37
19     91
23     71
29     53
31     31
37     13

THE ENCRYPTED MESSAGE IS
lylfaf
THE DECRYPTED MESSAGE IS
vignanPress any key to continue . . .
```

## 9. Implement Euclidean and Extended Euclidean algorithm for calculating the GCD

**Aim:-** Implement Euclidean and extended Euclidean algorithm for calculating gcd

**Description:** Basic Euclidean Algorithm is used to find GCD of two numbers say a and b. Below is a recursive C function to evaluate gcd using Euclid's algorithm.

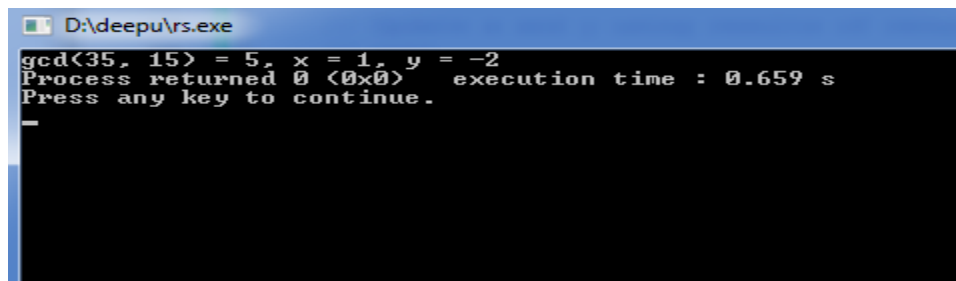
1.  $\text{GCD}(A,0) = A$  2.  $\text{GCD}(0,B) = B$  3. If  $A = B \cdot Q + R$  and  $B \neq 0$  then  $\text{GCD}(A,B) = \text{GCD}(B,R)$  where Q is an integer, R is an integer between 0 and B-1.

The extended Euclidean algorithm updates results of gcd(a, b) using the results calculated by recursive call gcd(b%a, a).  $ax + by = \text{gcd}(a, b)$

### Program Code:

```
#include <stdio.h>
int gcdExtended(int a, int b, int *x, int *y)
{
    // Base Case
    if (a == 0)
    {
        *x = 0;
        *y = 1;
        return b;
    }
    int x1, y1;
    int gcd = gcdExtended(b%a, a, &x1, &y1);
    *x = y1 - (b/a) * x1;
    *y = x1;
    return gcd;
}
int main()
{
    int x, y;
    int a = 35, b = 15;
    int g = gcdExtended(a, b, &x, &y);
    printf("gcd(%d, %d) = %d, x = %d, y = %d",
           a, b, g, x, y);
    return 0;
}
```

### Output:-



```
D:\deepu\rs.exe
gcd(35, 15) = 5, x = 1, y = -2
Process returned 0 (0x0)    execution time : 0.659 s
Press any key to continue.
```

## 10. Working with PGP

**Pretty Good Privacy (PGP)** is an encryption program that provides cryptographic privacy and authentication for data communication. PGP is often used for signing, encrypting, and decrypting texts, e-mails, files, directories, and whole disk partitions and to increase the security of e-mail communications. It was created by Phil Zimmermann in 1991.

### Design:

PGP encryption uses a serial combination of hashing, data compression, symmetric-key cryptography, and finally public-key cryptography; each step uses one of several supported algorithms. Each public key is bound to a user name and/or an e-mail address

### How PGP works:

PGP combines some of the best features of both conventional and public key cryptography. PGP is a hybrid cryptosystem. When a user encrypts plaintext with PGP, it first compresses the plaintext. Data compression saves modem transmission time and disk space and, more importantly, strengthens cryptographic security. Most cryptanalysis techniques exploit patterns found in the plaintext to crack the cipher. Compression reduces these patterns in the plaintext, thereby greatly enhancing resistance to cryptanalysis. (Files that are too short to compress or which don't compress well aren't compressed.)

PGP then creates a session key, which is a one-time-only secret key. This key is a random number generated from the random movements of your mouse and the keystrokes you type. This session key works with a very secure, fast conventional encryption algorithm to encrypt the plaintext; the result is ciphertext. Once the data is encrypted, the session key is then encrypted to the recipient's public key. This public key-encrypted session key is transmitted along with the ciphertext to the recipient.

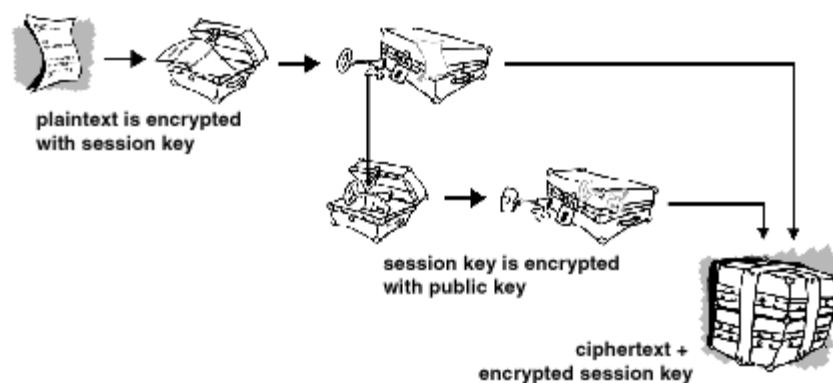


Fig. How PGP encryption works

Decryption works in the reverse. The recipient's copy of PGP uses his or her private key to recover the temporary session key, which PGP then uses to decrypt the conventionally-encrypted ciphertext.

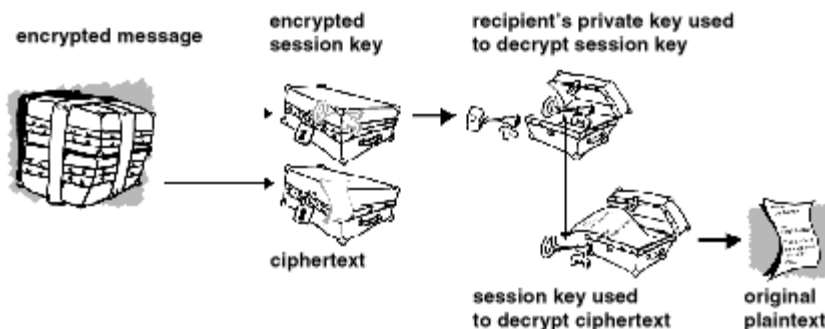


Fig. How PGP decryption works

The combination of the two encryption methods combines the convenience of public key encryption with the speed of conventional encryption. Conventional encryption is about 1,000 times faster than public key encryption. Public key encryption in turn provides a solution to key distribution and data transmission issues. Used together, performance and key distribution are improved without any sacrifice in security.

### Benefits of using Open PGP Security

- Your valuable information is always protected, others cannot view it, and it cannot be stolen over the Internet
- Information can be shared securely with others including groups of users and company departments
- You can be certain who email is from and who it is for
- No compatibility problems - works with any email application that you or your recipients are using
- Verification of the sender of information ensures you are not being spoofed by a third party
- Absolute assures that the information you send or receive has not been modified in transit
- Total assurance that files cannot be altered without your knowledge
- Prove documents were authorized, who authorized them, and when
- You are compliant with all current privacy and data protection legislation
- You are protected against virus attacks and the newest blended email threats
- Your secure mail and text cannot be infiltrated by hackers or infected and miss-used by email attacks
- You can work immediately with any OpenPGP user - the biggest community in the world - with no additional effort
- Smaller files are sent over the Internet because they are always compressed before encryption
- Secure text can be used in any application
- Others cannot recover sensitive files once you have deleted them
- Future proof technology and complete compatibility with other applications
- No need to purchase keys or certificates so save yourself some money!



- In-built key manager - securely manage yours and others keys
- Automated checking of certificates against root keys - we do all the hard work for you
- No need to stick with us if you don't want to as you can use any OpenPGP product to read our files and keys
- Little user training required
- Reassurance that the encryption is approved by the US Government
- You have control over your security

**Applications:** Generating 3D image, Criminal investigation