

# PRACTICAL FILE NTC

BHAVY RASTOGI | 2019UIT3060



## INDEX

S.No.	Experiment
1	Find the multiplicative inverse using extended Euclidean Algorithm in modulus n

<b>2</b>	<b>Find the inverse of a matrix modulus n</b>
<b>3</b>	<b>Implement Shift Cipher</b>
<b>4</b>	<b>Implement Multiplicative Cipher</b>
<b>5</b>	<b>Implement Affine Cipher</b>
<b>6</b>	<b>Implement Playfair Cipher</b>
<b>7</b>	<b>Implement Hill Cipher</b>
<b>8</b>	<b>Implement Vigenère Cipher</b>
<b>9</b>	<b>Implement Rail fence Cipher- Row and Column transformation</b>
<b>10</b>	<b>Implement DES algorithm</b>
<b>11</b>	<b>Implement AES algorithm</b>
<b>12</b>	<b>Implement RSA algorithm</b>
<b>13</b>	<b>Implement Elgamal algorithm</b>
<b>14</b>	<b>Implement Rabin algorithm</b>
<b>15</b>	<b>Implement SHA-512 algorithm</b>
<b>16</b>	<b>Implement RSA digital signature scheme</b>
<b>17</b>	<b>Implement ElGamal scheme</b>

## Find the multiplicative inverse using Extended Euclidean Algorithm in modulus n

```
#include<bits/stdc++.h>
using namespace std;

bool isGcd(int a, int b) {
    int r;
    while(a % b > 0) {
        r = a % b;
        a = b;
        b = r;
        cout << "a =" << a << "\nb =" << b << "\nr = " << r << endl;
    }
    if(b == 1) {
        return true;
    } else {
        return false;
    }
}

int inverse(int a, int m)
{
    for(int i = 1; i < m; ++i)
    {
        if((a * i) % m == 1)
        {
            return i;
        }
    }
    return 0;
}

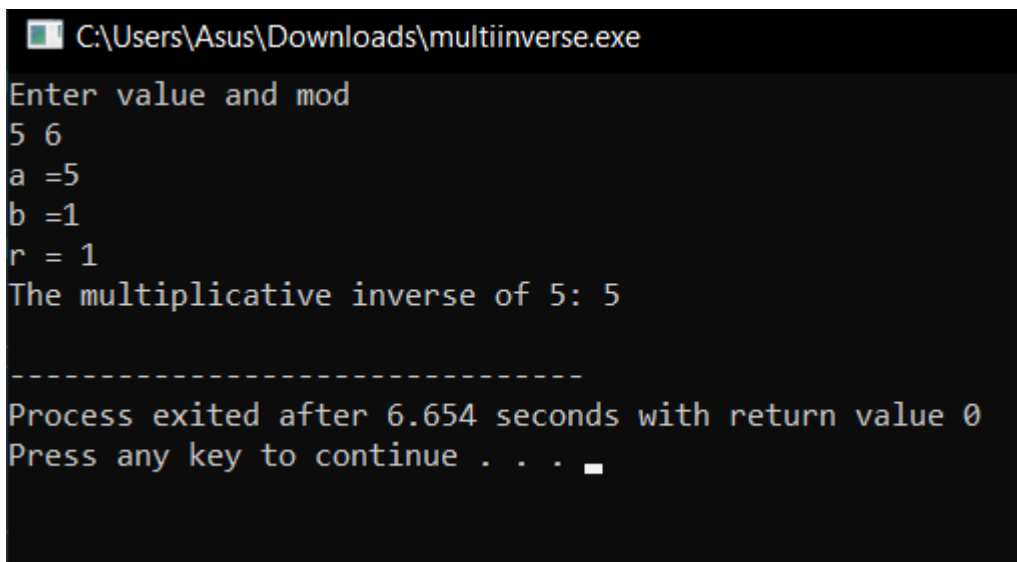
int main() {
    int value, mod;
    cout << "Enter value and mod"<<endl;
    cin >> value >> mod;
```

```

    if(isGcd(mod , value))
    {
        cout << "The multiplicative inverse of " << value << ": " <<
inverse(value, mod)<<endl;
    }
    else
    {
        cout << "No multiplicative inverse!"<<endl;
    }

    return 0;
}

```



```

C:\Users\Asus\Downloads\multiinverse.exe
Enter value and mod
5 6
a =5
b =1
r = 1
The multiplicative inverse of 5: 5

-----
Process exited after 6.654 seconds with return value 0
Press any key to continue . . .

```

## Inverse of a matrix in Modulo N

```

#include<bits/stdc++.h>
using namespace std;
#define N 4

int findInverseMod(int b,int n){
    int t1=0;
    int t2=1;

    int r1=n;

```

```
int r2=b;
```

```
while(r2>0){  
    int q=r1/r2;
```

```
    int r=r1-q*r2;  
    int t=t1-q*t2;
```

```
    r1=r2;  
    r2=r;
```

```
    t1=t2;  
    t2=t;  
}
```

```
if(r1==1){  
    t1=t1<0?t1+n:t1;  
    return t1;  
}
```

```
    return -1;  
}  
void getCofactor(int A[N][N], int temp[N][N], int p, int q, int n)  
{  
    int i = 0, j = 0;
```

```
    for (int row = 0; row < n; row++)  
    {  
        for (int col = 0; col < n; col++)  
        {  
            if (row != p && col != q)  
            {  
                temp[i][j++] = A[row][col];
```

```
                if (j == n - 1)  
                {  
                    j = 0;
```

```

        i++;
    }
}
}
}
}

```

```

int determinant(int A[N][N], int n)
{
    int D = 0;
    if (n == 1)
        return A[0][0];

    int temp[N][N];

    int sign = 1;

    for (int f = 0; f < n; f++)
    {
        getCofactor(A, temp, 0, f, n);
        D += sign * A[0][f] * determinant(temp, n - 1);

        sign = -sign;
    }

    return D;
}

```

```

void adjoint(int A[N][N],int adj[N][N])
{
    if (N == 1)
    {
        adj[0][0] = 1;
        return;
    }
}

```

```

    }

    int sign = 1, temp[N][N];

    for (int i=0; i<N; i++)
    {
        for (int j=0; j<N; j++)
        {
            getCofactor(A, temp, i, j, N);

            sign = ((i+j)%2==0)? 1: -1;

            adj[j][i] = (sign)*(determinant(temp, N-1));
        }
    }
}

bool inverseMatrix(int A[N][N], int inverse[N][N],int r)
{
    int det = determinant(A, N);
    if(det==0) return false;
    if(__gcd(det,r)!=1) return false;

    int detInverse=findInverseMod(det,r);

    int adj[N][N];
    adjoint(A, adj);

    for (int i=0; i<N; i++)
        for (int j=0; j<N; j++)
            inverse[i][j] = adj[i][j]*detInverse;

    return true;
}

void display(int A[N][N])
{

```

```

    for (int i=0; i<N; i++)
    {
        for (int j=0; j<N; j++)
            cout << A[i][j] << " ";
        cout << endl;
    }
}

```

```

int main()
{
    int arr[N][N] = { {3, -8, 2, 1},
                      {1, -5, 2, 4},
                      {6, 0, 2, 1},
                      {3, -2, -3, 2}};

```

```

    int inv[N][N];

```

```

    cout<<"Enter modulus n: ";
    int r;
    cin>>r;

```

```

    cout <<endl<< "Input matrix:"<<endl;
    display(arr);

```

```

    cout <<endl<< "The Inverse is :"<<endl;

```

```

    if (inverseMatrix(arr, inv,r)) display(inv);

```

```

    return 0;
}

```



```

Enter modulus n: 5

Input matrix:
3 -8 2 1
1 -5 2 4
6 0 2 1
3 -2 -3 2

The Inverse is :
23 -56 105 48
-107 21 59 -18
1 66 73 -169
-140 204 11 50

-----
Process exited after 2.002 seconds with return value 0
Press any key to continue . . .

```

## IMPLEMENTATION OF CIPHERS

### Shift Cipher

```

#include<bits/stdc++.h>
using namespace std;

int main()
{
    unordered_map<char,int> map;

    for(int i=0;i<26;i++)
        map[char(97+i)]=i;

    string plaintext;
    int n;
    int k;

    cout<<"\n Enter Plaintext: ";
    cin>>plaintext;

```

```

cout<<"\n Enter key value: ";
cin>>k;

string cipher;
for(int i=0;i<plaintext.size();i++)
{
    int C=(map[plaintext[i]]+k)%26;
    cipher+=char(C+97);
}
cout<<"\n Ciphertext: "<<cipher;

return(0);
}

```

#### Input

bhavyrastogi  
5

#### Output

```

Enter Plaintext:
Enter key value:
Ciphertext: gmfadwfxytln

```

## Multiplicative Cipher

```

#include<bits/stdc++.h>
using namespace std;

int main()
{
    map<char,int> map;
    for(int i=0;i<26;i++)
        map[char(97+i)]=i;
    string plain;
    int n;
    int k;

    cout<<"\n Enter Plaintext: ";
    cin>>plain;
    cout<<"\n Enter key value: ";
    cin>>k;

    string cipher;
    for(int i=0;i<plain.size();i++)
    {
        int C=(map[plain[i]]*k)%26;
        cipher+=char(C+97);
    }

    cout<<"\n Ciphertext: "<<cipher;
    return 0;
}

```

C:\Users\Asus\Downloads\Multiplicative Cipher.exe

Enter Plaintext: bhavyrastogi

Enter key value: 7

Ciphertext: hxarmpawduqe

-----  
Process exited after 8.957 seconds with return value 0  
Press any key to continue . . .

## Affine Cipher

```
#include<bits/stdc++.h>
using namespace std;
main()
{
    map<char,int> map;
    for(int i=0;i<26;i++)
        map[char(97+i)]=i;

    string plain;
    int n;
    int k1,k2;

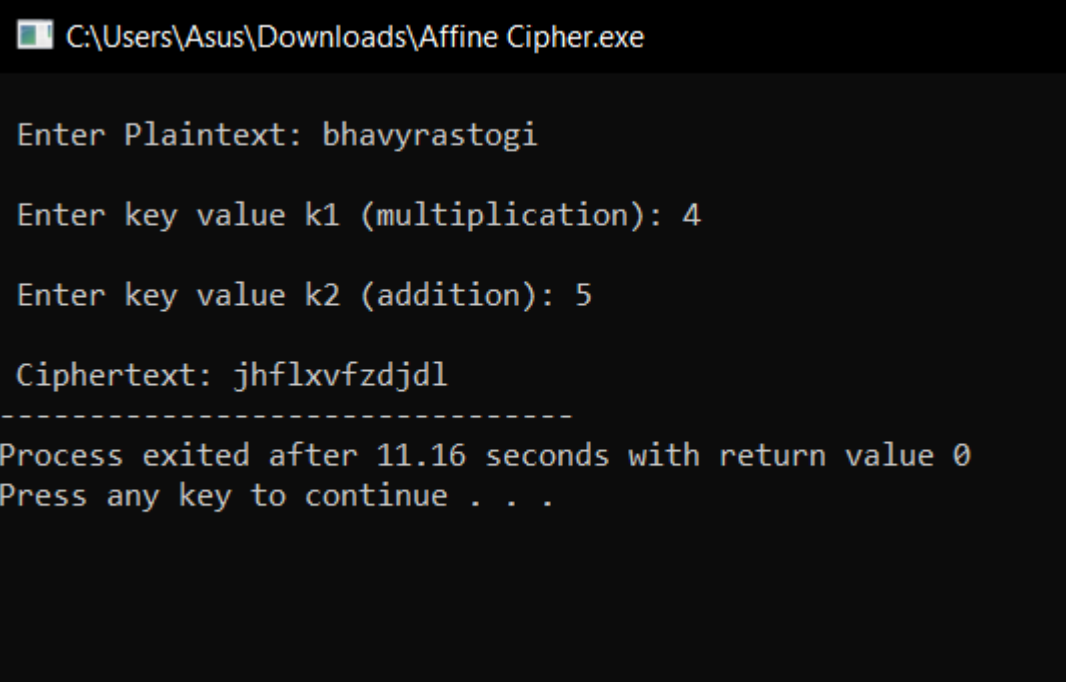
    cout<<"\n Enter Plaintext: ";
    cin>>plain;
    cout<<"\n Enter key value k1 (multiplication): ";
    cin>>k1;
    cout<<"\n Enter key value k2 (addition): ";
    cin>>k2;

    string cipher;
    for(int i=0;i<plain.size();i++)
    {
        int t = ((map[plain[i]]*k1)+k2)%26;
```

```

        cipher += 'a'+t;
    }
    cout<<"\n Ciphertext: "<<cipher;
}

```



```

C:\Users\Asus\Downloads\Affine Cipher.exe

Enter Plaintext: bhavyrastogi

Enter key value k1 (multiplication): 4

Enter key value k2 (addition): 5

Ciphertext: jhflxvfzdjdl
-----
Process exited after 11.16 seconds with return value 0
Press any key to continue . . .

```

## Playfair Cipher using the key as “avengers”

```

#include<bits/stdc++.h>
using namespace std;

```

```

int main()
{
    char key[5][5]=
    {
        {'a','v','e','n','g'},
        {'r','s','b','c','d'},
        {'f','h','i','k','l'},
        {'m','o','p','q','t'},
        {'u','w','x','y','z'}
    };
}

```

```
unordered_map<char, pair<int,int>> map;
```

```
for(int i=0;i<5;i++)  
{  
    for(int j=0;j<5;j++)  
        map[key[i][j]] = make_pair(i,j);  
}
```

```
string plain,temp,cipher;  
cout<<"\n Enter Plain Text: ";  
cin>>plain;
```

```
for(int i=0;i<plain.size();i++)  
{  
    if(plain[i]=='j') plain[i]='i';  
  
    if(i>0 && plain[i]==plain[i-1]) temp+='x';  
  
    temp+=plain[i];  
}
```

```
if((temp.size())%2 != 0) temp+='x';
```

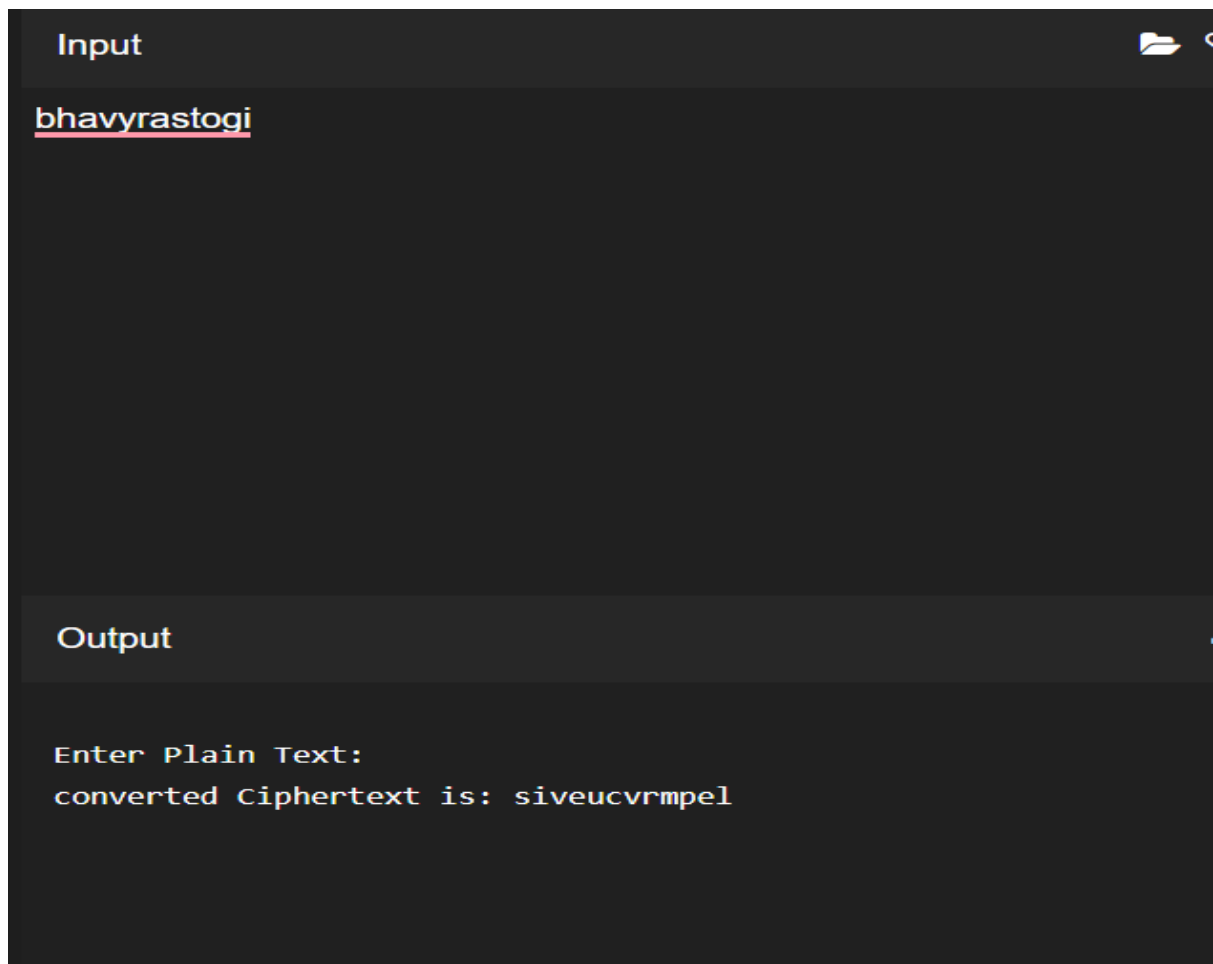
```
for(int t=0;t<temp.size()-1;t+=2)  
{  
    int x1= map[temp[t]].first;  
    int y1= map[temp[t]].second;  
    int x2= map[temp[t+1]].first;  
    int y2= map[temp[t+1]].second;  
  
    if(x1==x2)  
    {  
        cipher+=key[x1][(y1+1)%5];  
        cipher+=key[x1][(y2+1)%5];  
    }  
    else if(y1==y2)
```

```

        {
            cipher+=key[(x1+1)%5][y1];
            cipher+=key[(x2+1)%5][y1];
        }
        else
        {
            cipher+=key[x1][y2];
            cipher+=key[x2][y1];
        }
    }
    cout<<"\n converted Ciphertext is: "<<cipher;

    return(0);
}

```



The screenshot shows a C++ IDE with a dark theme. The 'Input' section contains the text 'bhavyrastogi'. The 'Output' section shows the program's execution: 'Enter Plain Text:' followed by 'converted Ciphertext is: siveucvrmpel'.

## Hill Cipher

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
float encrymatrix[3][1], a[3][3], msg[3][1], m[3][3];
```

```
void getKeyMatrix()
```

```
{
```

```
    int i, j;
```

```
    char mes[3];
```

```
    cout<<"Enter 3x3 matrix for key: "<<endl;
```

```
    for(i = 0; i < 3; i++)
```

```
    {
```

```
        for(j = 0; j < 3; j++)
```

```
        {
```

```
            cin>>a[i][j];
```

```
            m[i][j] = a[i][j];
```

```
        }
```

```
    }
```

```
    cout<<"\nEnter a string of 3 letter: ";
```

```
    cin>>mes;
```

```
    for(i = 0; i < 3; i++)
```

```
        msg[i][0] = mes[i] - 65;
```

```
}
```

```
void encrypt()
```

```
{
```

```
    int i, j, k;
```

```
    for(i = 0; i < 3; i++)
```

```
        for(j = 0; j < 1; j++)
```

```
            for(k = 0; k < 3; k++)
```

```
                encrymatrix[i][j] = encrymatrix[i][j] + a[i][k] *
```

```
msg[k][j];
```

```
    cout<<"\nEncrypted string is: ";
```

```
    for(i = 0; i < 3; i++)
```



```

        cout<<(char)(fmod(encrymatrix[i][0], 26) + 65);
    }

int main()
{
    getKeyMatrix();
    encrypt();
    return(0);
}

```

```

C:\Users\Asus\Downloads\Hill Cipher.exe
Enter 3x3 matrix for key:
3 5 7
6 2 1
9 8 4
Enter a string of 3 letter: NTC
Encrypted string is: SOR
-----
Process exited after 15.37 seconds with return value 0
Press any key to continue . . .

```

## Vignere Cipher

```

#include<bits/stdc++.h>
using namespace std;

int main()
{
    unordered_map<char,int> map;

```

```

for(int i=0;i<26;i++)
{
    map[i+'a'] = i;
}

string plain, key, cipher;

cout<<"Enter Plaintext(a-z) : ";
cin>>plain;
cout<<"Enter Key Text(a-z) : ";
cin>>key;

int i=0,j=0;

while(i!=plain.size())
{
    if(j==key.size()) j=0;

    int t = (map[plain[i]] + map[key[j]])%26;
    char code = t +'a';
    cipher += code;
    i++;
    j++;
}
cout<<"\n Ciphertext: "<<cipher;
return(0);
}

```

### Input

bhavyrastoginsutitone  
avengers

### Output

Enter Plaintext(a-z) : Enter Key Text(a-z) :  
Ciphertext: bceievrrktjkwlliosak

## Rail fence - row and column transformation

```
#include <bits/stdc++.h>
using namespace std;

string encryptRailFence(string text, int key) {
    char rail[key][text.length()];
    for (int i=0; i < key; i++)
        for (int j = 0; j < text.length(); j++)
            rail[i][j] = '\n';
    bool dir_down = false;
    int row = 0, col = 0;
    for (int i=0; i < text.length(); i++) {
        if (row == 0 || row == key-1)
            dir_down = !dir_down;
        rail[row][col++] = text[i];
        dir_down?row++ : row--;
    }
    string result;
```

```

    for (int i=0; i < key; i++)
        for (int j=0; j < text.length(); j++)
            if (rail[i][j]!='\n')
                result.push_back(rail[i][j]);
    return result;
}

```

```

string decryptRailFence(string cipher, int key) {
    char rail[key][cipher.length()];
    for (int i=0; i < key; i++)
        for (int j=0; j < cipher.length(); j++)
            rail[i][j] = '\n';
    bool dir_down;
    int row = 0, col = 0;
    for (int i=0; i < cipher.length(); i++) {
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;
        rail[row][col++] = '*';
        dir_down?row++ : row--;
    }
    int index = 0;
    for (int i=0; i<key; i++)
        for (int j=0; j<cipher.length(); j++)
            if (rail[i][j] == '*' && index<cipher.length())
                rail[i][j] = cipher[index++];
    string result;
    row = 0, col = 0;
    for (int i=0; i< cipher.length(); i++) {
        if (row == 0)
            dir_down = true;
        if (row == key-1)
            dir_down = false;
        if (rail[row][col] != '*')
            result.push_back(rail[row][col++]);
        dir_down?row++: row--;
    }
}

```

```

    }
    return result;
}

int main() {
    string text = "avengers assemble", cipher;
    int key = 2;
    cout << "Plain text: " << text<<endl;
    cout << "Key: " << key<<endl;
    cipher = encryptRailFence(text, key);
    cout <<"\nCipher text: " << cipher << endl;
    cout <<"Deciphered text: " << decryptRailFence(cipher,key)
<< endl;
    return 0;
}

```

### Output

```

Plain text: avengers assemble
Key: 2

Cipher text: aegr sebevnesaml
Deciphered text: avengers assemble

```

## Implementing DES algorithm

```

#include <bits/stdc++.h>
using namespace std;
string hex2bin(string s) {
    unordered_map<char, string> map;
    map['0'] = "0000";

```

```

    map['1'] = "0001";
    map['2'] = "0010";
    map['3'] = "0011";
    map['4'] = "0100";
    map['5'] = "0101";
    map['6'] = "0110";
    map['7'] = "0111";
    map['8'] = "1000";
    map['9'] = "1001";
    map['A'] = "1010";
    map['B'] = "1011";
    map['C'] = "1100";
    map['D'] = "1101";
    map['E'] = "1110";
    map['F'] = "1111";
    string bin = "";
    for (int i = 0; i < s.size(); i++) {
        bin += map[s[i]];
    }
    return bin;
}

string bin2hex(string s) {
    unordered_map<string, string> map;
    map["0000"] = "0";
    map["0001"] = "1";
    map["0010"] = "2";
    map["0011"] = "3";
    map["0100"] = "4";
    map["0101"] = "5";
    map["0110"] = "6";
    map["0111"] = "7";
    map["1000"] = "8";
    map["1001"] = "9";
    map["1010"] = "A";
    map["1011"] = "B";
    map["1100"] = "C";
    map["1101"] = "D";

```

```

map["1110"] = "E";
map["1111"] = "F";
string hex = "";
for (int i = 0; i < s.length(); i += 4) {
    string ch = "";
    ch += s[i];
    ch += s[i + 1];
    ch += s[i + 2];
    ch += s[i + 3];
    hex += map[ch];
}
return hex;
}

```

```

string permute(string k, int* arr, int n) {
    string per = "";
    for (int i = 0; i < n; i++) {
        per += k[arr[i] - 1];
    }
    return per;
}

```

```

string shift_left(string k, int shifts) {
    string s = "";
    for (int i = 0; i < shifts; i++) {
        for (int j = 1; j < 28; j++) {
            s += k[j];
        }
        s += k[0];
        k = s;
        s = "";
    }
    return k;
}

```

```

string xor_(string a, string b) {
    string ans = "";

```

```

for (int i = 0; i < a.size(); i++) {
    if (a[i] == b[i]) {
        ans += "0";
    }
    else {
        ans += "1";
    }
}
return ans;
}

string encrypt(string pt, vector<string> rkb, vector<string> rk) {
    pt = hex2bin(pt);
    int initial_perm[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
                             60, 52, 44, 36, 28, 20, 12, 4,
                             62, 54, 46, 38, 30, 22, 14, 6,
                             64, 56, 48, 40, 32, 24, 16, 8,
                             57, 49, 41, 33, 25, 17, 9, 1,
                             59, 51, 43, 35, 27, 19, 11, 3,
                             61, 53, 45, 37, 29, 21, 13, 5,
                             63, 55, 47, 39, 31, 23, 15, 7 };

    pt = permute(pt, initial_perm, 64);
    cout << "After initial permutation: " << bin2hex(pt) << endl;
    string left = pt.substr(0, 32);
    string right = pt.substr(32, 32);
    cout << "After splitting: L0=" << bin2hex(left)
         << " R0=" << bin2hex(right) << endl;
    int exp_d[48] = { 32, 1, 2, 3, 4, 5, 4, 5,
                      6, 7, 8, 9, 8, 9, 10, 11,
                      12, 13, 12, 13, 14, 15, 16, 17,
                      16, 17, 18, 19, 20, 21, 20, 21,
                      22, 23, 24, 25, 24, 25, 26, 27,
                      28, 29, 28, 29, 30, 31, 32, 1 };

    int s[8][4][16] = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
                           0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
                           4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,

```



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

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

```
int per[32] = { 16, 7, 20, 21,  
                29, 12, 28, 17,  
                1, 15, 23, 26,  
                5, 18, 31, 10,  
                2, 8, 24, 14,  
                32, 27, 3, 9,
```

```
19, 13, 30, 6,  
22, 11, 4, 25 };
```

```
cout << endl;  
for (int i = 0; i < 16; i++) {  
    string right_expanded = permute(right, exp_d, 48);  
    string x = xor_(rkb[i], right_expanded);  
    string op = "";  
    for (int i = 0; i < 8; i++) {  
        int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');  
        int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] - '0') + 2 *  
int(x[i * 6 + 3] - '0') + int(x[i * 6 + 4] - '0');  
        int val = s[i][row][col];  
        op += char(val / 8 + '0');  
        val = val % 8;  
        op += char(val / 4 + '0');  
        val = val % 4;  
        op += char(val / 2 + '0');  
        val = val % 2;  
        op += char(val + '0');  
    }  
    op = permute(op, per, 32);  
    x = xor_(op, left);  
    left = x;  
    if (i != 15) {  
        swap(left, right);  
    }  
    cout << "Round " << i + 1 << " " << bin2hex(left) << " "  
        << bin2hex(right) << " " << rk[i] << endl;  
}  
string combine = left + right;  
int final_perm[64] = { 40, 8, 48, 16, 56, 24, 64, 32,  
    39, 7, 47, 15, 55, 23, 63, 31,  
    38, 6, 46, 14, 54, 22, 62, 30,  
    37, 5, 45, 13, 53, 21, 61, 29,  
    36, 4, 44, 12, 52, 20, 60, 28,  
    35, 3, 43, 11, 51, 19, 59, 27,
```

```

        34, 2, 42, 10, 50, 18, 58, 26,
        33, 1, 41, 9, 49, 17, 57, 25 };
string cipher = bin2hex(permute(combine, final_perm, 64));
return cipher;
}

```

```

int main() {
    string pt, key;
    pt = "123456ABCD132536";
    key = "AABB09182736CCDD";
    key = hex2bin(key);
    int keyp[56] = { 57, 49, 41, 33, 25, 17, 9,
        1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27,
        19, 11, 3, 60, 52, 44, 36,
        63, 55, 47, 39, 31, 23, 15,
        7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29,
        21, 13, 5, 28, 20, 12, 4 };
    key = permute(key, keyp, 56);
    int shift_table[16] = { 1, 1, 2, 2,
        2, 2, 2, 2,
        1, 2, 2, 2,
        2, 2, 2, 1 };

    int key_comp[48] = { 14, 17, 11, 24, 1, 5,
        3, 28, 15, 6, 21, 10,
        23, 19, 12, 4, 26, 8,
        16, 7, 27, 20, 13, 2,
        41, 52, 31, 37, 47, 55,
        30, 40, 51, 45, 33, 48,
        44, 49, 39, 56, 34, 53,
        46, 42, 50, 36, 29, 32 };

    string left = key.substr(0, 28);
    string right = key.substr(28, 28);
    vector<string> rkb;
}

```

```

vector<string> rk;
for (int i = 0; i < 16; i++) {
    left = shift_left(left, shift_table[i]);
    right = shift_left(right, shift_table[i]);
    string combine = left + right;
    string RoundKey = permute(combine, key_comp, 48);
    rkb.push_back(RoundKey);
    rk.push_back(bin2hex(RoundKey));
}
cout << "*****Encryption*****"<<endl;
string cipher = encrypt(pt, rkb, rk);
cout << "Cipher Text: " << cipher << endl;
cout << endl << "*****Decryption*****"<<endl;
reverse(rkb.begin(), rkb.end());
reverse(rk.begin(), rk.end());
string text = encrypt(cipher, rkb, rk);
cout << "Plain Text: " << text << endl;
}

```

## Output

\*\*\*\*\*Encryption\*\*\*\*\*

After initial permutation: 14A7D67818CA18AD

After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C

Round 2 5A78E394 4A1210F6 4568581ABCCE

Round 3 4A1210F6 B8089591 06EDA4ACF5B5

Round 4 B8089591 236779C2 DA2D032B6EE3

Round 5 236779C2 A15A4B87 69A629FEC913

Round 6 A15A4B87 2E8F9C65 C1948E87475E

Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0

Round 8 A9FC20A3 308BEE97 34F822F0C66D

Round 9 308BEE97 10AF9D37 84BB4473DCCC  
Round 10 10AF9D37 6CA6CB20 02765708B5BF  
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5  
Round 12 FF3C485F 22A5963B C2C1E96A4BF3  
Round 13 22A5963B 387CCDAA 99C31397C91F  
Round 14 387CCDAA BD2DD2AB 251B8BC717D0  
Round 15 BD2DD2AB CF26B472 3330C5D9A36D  
Round 16 19BA9212 CF26B472 181C5D75C66D  
Cipher Text: C0B7A8D05F3A829C

\*\*\*\*\*Decryption\*\*\*\*\*

After initial permutation: 19BA9212CF26B472

After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D  
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D  
Round 3 387CCDAA 22A5963B 251B8BC717D0  
Round 4 22A5963B FF3C485F 99C31397C91F  
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3  
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5  
Round 7 10AF9D37 308BEE97 02765708B5BF  
Round 8 308BEE97 A9FC20A3 84BB4473DCCC  
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D  
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0

Round 11 A15A4B87 236779C2 C1948E87475E  
Round 12 236779C2 B8089591 69A629FEC913  
Round 13 B8089591 4A1210F6 DA2D032B6EE3  
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5  
Round 15 5A78E394 18CA18AD 4568581ABCCE  
Round 16 14A7D678 18CA18AD 194CD072DE8C  
Plain Text: 123456ABCD132536

## Implementing AES algorithm

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
#define Nb 4
```

```
int Nr = 0;
```

```
int Nk = 0;
```

```
unsigned char in[1024], out[1024], state[4][Nb];
```

```
unsigned char RoundKey[240];
```

```
unsigned char Key[32];
```

```
int getSBoxValue(int num)
```

```
{
```

```
    int sbox[256] = {
```

```
        0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5,
        0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0,
        0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc,
        0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a,
        0x07, 0x12, 0x80, 0xe2, 0xeb, 0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0,
        0x52, 0x3b, 0xd6, 0xb3, 0x29, 0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b,
        0x6a, 0xcb, 0xbe, 0x39, 0x4a, 0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85,
        0x45, 0xf9, 0x02, 0x7f, 0x50, 0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5,
```

```

    0xbc, 0xb6, 0xda, 0x21, 0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17,
    0xc4, 0xa7, 0x7e, 0x3d, 0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88,
    0x46, 0xee, 0xb8, 0x14, 0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c,
    0xc2, 0xd3, 0xac, 0x62, 0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9,
    0x6c, 0x56, 0xf4, 0xea, 0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6,
    0xe8, 0xdd, 0x74, 0x1f, 0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e,
    0x61, 0x35, 0x57, 0xb9, 0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94,
    0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68,
    0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};
return sbox[num];
}

```

```

int Rcon[255] = {
    0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20,
    0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d,
    0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35,
    0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91,

    0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f,
    0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d,
    0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04,
    0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c,
    0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63,
    0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa,
    0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd,
    0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66,
    0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d,
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,

```

0x1b, 0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f,  
0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4,  
0xb3, 0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72,  
0xe4, 0xd3, 0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a,  
0x94, 0x33, 0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74,  
0xe8, 0xcb, 0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,  
0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8, 0xab,  
0x4d, 0x9a, 0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97,  
0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef, 0xc5,  
0x91, 0x39, 0x72, 0xe4, 0xd3, 0xbd, 0x61, 0xc2,  
0x9f, 0x25, 0x4a, 0x94, 0x33, 0x66, 0xcc, 0x83,  
0x1d, 0x3a, 0x74, 0xe8, 0xcb, 0x8d, 0x01, 0x02,  
0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36,  
0x6c, 0xd8, 0xab, 0x4d, 0x9a, 0x2f, 0x5e, 0xbc,  
0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d,  
0xfa, 0xef, 0xc5, 0x91, 0x39, 0x72, 0xe4, 0xd3,  
0xbd, 0x61, 0xc2, 0x9f, 0x25, 0x4a, 0x94, 0x33,  
0x66, 0xcc, 0x83, 0x1d, 0x3a, 0x74, 0xe8, 0xcb

};

void KeyExpansion()

{

int i, j;

unsigned char temp[4], k;

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

{

RoundKey[i \* 4] = Key[i \* 4];

RoundKey[i \* 4 + 1] = Key[i \* 4 + 1];

RoundKey[i \* 4 + 2] = Key[i \* 4 + 2];

RoundKey[i \* 4 + 3] = Key[i \* 4 + 3];

}

while (i < (Nb \* (Nr + 1)))

{



```

for (j = 0; j < 4; j++)
{
    temp[j] = RoundKey[(i - 1) * 4 + j];
}
if (i % Nk == 0)
{
    k = temp[0];
    temp[0] = temp[1];
    temp[1] = temp[2];
    temp[2] = temp[3];
    temp[3] = k;

    temp[0] = getSBoxValue(temp[0]);
    temp[1] = getSBoxValue(temp[1]);
    temp[2] = getSBoxValue(temp[2]);
    temp[3] = getSBoxValue(temp[3]);

    temp[0] = temp[0] ^ Rcon[i / Nk];
}
else if (Nk > 6 && i % Nk == 4)
{
    temp[0] = getSBoxValue(temp[0]);
    temp[1] = getSBoxValue(temp[1]);
    temp[2] = getSBoxValue(temp[2]);
    temp[3] = getSBoxValue(temp[3]);
}
RoundKey[i * 4 + 0] = RoundKey[(i - Nk) * 4 + 0] ^ temp[0];
RoundKey[i * 4 + 1] = RoundKey[(i - Nk) * 4 + 1] ^ temp[1];
RoundKey[i * 4 + 2] = RoundKey[(i - Nk) * 4 + 2] ^ temp[2];
RoundKey[i * 4 + 3] = RoundKey[(i - Nk) * 4 + 3] ^ temp[3];
i++;
}
}

```

```

void AddRoundKey(int round)
{
    int i, j;

```

```

for (i = 0; i < Nb; i++)
{
    for (j = 0; j < 4; j++)
    {
        state[j][i] ^= RoundKey[round * Nb * 4 + i * Nb + j];
    }
}

```

```

void SubBytes()
{
    int i, j;
    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < Nb; j++)
        {
            state[i][j] = getSBoxValue(state[i][j]);
        }
    }
}

```

```

void ShiftRows()
{
    unsigned char temp;

    temp = state[1][0];
    state[1][0] = state[1][1];
    state[1][1] = state[1][2];
    state[1][2] = state[1][3];
    state[1][3] = temp;

    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;

    temp = state[2][1];
    state[2][1] = state[2][3];

```

```
state[2][3] = temp;
```

```
temp = state[3][0];
```

```
state[3][0] = state[3][3];
```

```
state[3][3] = state[3][2];
```

```
state[3][2] = state[3][1];
```

```
state[3][1] = temp;
```

```
}
```

```
#define xtime(x) ((x << 1) ^ (((x >> 7) & 1) * 0x1b))
```

```
void MixColumns()
```

```
{
```

```
int i;
```

```
unsigned char Tmp, Tm, t;
```

```
for (i = 0; i < Nb; i++)
```

```
{
```

```
    t = state[0][i];
```

```
    Tmp = state[0][i] ^ state[1][i] ^ state[2][i] ^ state[3][i];
```

```
    Tm = state[0][i] ^ state[1][i];
```

```
    Tm = xtime(Tm);
```

```
    state[0][i] ^= Tm ^ Tmp;
```

```
    Tm = state[1][i] ^ state[2][i];
```

```
    Tm = xtime(Tm);
```

```
    state[1][i] ^= Tm ^ Tmp;
```

```
    Tm = state[2][i] ^ state[3][i];
```

```
    Tm = xtime(Tm);
```

```
    state[2][i] ^= Tm ^ Tmp;
```

```
    Tm = state[3][i] ^ t;
```

```
    Tm = xtime(Tm);
```

```
    state[3][i] ^= Tm ^ Tmp;
```

```
}
```

```
}
```

```

void Cipher()
{
    int i, j, round = 0;

    for (i = 0; i < Nb; i++)
    {
        for (j = 0; j < 4; j++)
        {
            state[j][i] = in[i * 4 + j];
        }
    }

    AddRoundKey(0);

    for (round = 1; round < Nr; round++)
    {
        SubBytes();
        ShiftRows();
        MixColumns();
        AddRoundKey(round);
    }

    SubBytes();
    ShiftRows();
    AddRoundKey(Nr);

    for (i = 0; i < Nb; i++)
    {
        for (j = 0; j < 4; j++)
        {
            out[i * 4 + j] = state[j][i];
        }
    }
}

```

```

int fillBlock(int sz, char *str, unsigned char *in)
{

```

```

int j = 0;
while (sz < strlen(str))
{
    if (j >= Nb * 4)
        break;
    in[j++] = (unsigned char)str[sz];
    sz++;
}

if (sz >= strlen(str))
    for (; j < Nb * 4; j++)
        in[j] = 0;
return sz;
}

int main(int argc, char **argv)
{
    int i;

    if (argc != 2)
    {
        cerr << "Usage: " << argv[0] << " <keysize: 1=128, 2=192,
3=256>\n";
        exit(0);
    }

    switch (atoi(argv[1]))
    {
    case 1:
        Nk = 4;
        break;
    case 2:
        Nk = 6;
        break;
    case 3:
        Nk = 8;
        break;

```

default:

```
    Nk = 4;  
    break;  
}
```

Nr = Nk + 6;

```
Key[0] = 0x2b;  
Key[1] = 0x7e;  
Key[2] = 0x15;  
Key[3] = 0x16;  
Key[4] = 0x28;  
Key[5] = 0xae;  
Key[6] = 0xd2;  
Key[7] = 0xa6;  
Key[8] = 0xab;  
Key[9] = 0xf7;  
Key[10] = 0x15;  
Key[11] = 0x88;  
Key[12] = 0x09;  
Key[13] = 0xcf;  
Key[14] = 0x4f;  
Key[15] = 0x3c;
```

```
char str[1024];  
fgets(str, 1024, stdin);
```

KeyExpansion();

int sz = 0;

```
while (sz < strlen(str))  
{  
    sz = fillBlock(sz, str, in);  
  
    Cipher();
```

```

        for (i = 0; i < Nb * 4; i++)
            cout << (int)out[i] << " ";
    }
    printf("\n\n");
}

```

## Implementing RSA algorithm

```

#include<bits/stdc++.h>
using namespace std;

```

```

int gcd(int a, int h) {
    int temp;
    while(1) {
        temp = a%h;
        if(temp==0)
            return h;
        a = h;
        h = temp;
    }
}

```

```

int main()
{
    double p = 3;
    double q = 7;
    double n=p*q;
    double count;
    double totient = (p-1)*(q-1);
    double e=2;

    while(e<totient)
    {
        count = gcd(e,totient);
    }
}

```

```
    if(count==1)
        break;
    else
        e++;
}
```

```
double d;
double k = 2;
d = (1 + (k*totient))/e;
double msg = 12;
double c = pow(msg,e);
double m = pow(c,d);
c=fmod(c,n);
m=fmod(m,n);
```

```
cout<<"Message data = "<<msg<<endl;
cout<<"p = "<<p<<endl;
cout<<"q = "<<q<<endl;
cout<<"n = pq = "<<n<<endl;
cout<<"totient = "<<totient<<endl;
cout<<"e = "<<e<<endl;
cout<<"d = "<<d<<endl;
cout<<"Encrypted data = "<<c<<endl;
cout<<"Original Message sent = "<<m<<endl;
return 0;
}
```



```
C:\Users\Asus\Downloads\RES algo.exe
Message data = 12
p = 3
q = 7
n = pq = 21
totient = 12
e = 5
d = 5
Encrypted data = 3
Original Message sent = 12

-----
Process exited after 0.01363 seconds with return value 0
Press any key to continue . . .
```

## Implementing ElGamal Algorithm

```
import random
from math import pow

a = random.randint(2, 10)

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b;
    else:
        return gcd(b, a % b)

def gen_key(q):

    key = random.randint(pow(10, 20), q)
    while gcd(q, key) != 1:
        key = random.randint(pow(10, 20), q)

    return key
```

```
def power(a, b, c):  
    x = 1  
    y = a  
  
    while b > 0:  
        if b % 2 != 0:  
            x = (x * y) % c;  
        y = (y * y) % c  
        b = int(b / 2)  
  
    return x % c
```

```
def encrypt(msg, q, h, g):  
  
    en_msg = []  
    k = gen_key(q)  
    s = power(h, k, q)  
    p = power(g, k, q)  
  
    for i in range(0, len(msg)):  
        en_msg.append(msg[i])  
  
    print("g^k used : ", p)  
    print("g^ak used : ", s)  
    for i in range(0, len(en_msg)):  
        en_msg[i] = s * ord(en_msg[i])  
  
    return en_msg, p
```

```
def decrypt(en_msg, p, key, q):  
  
    dr_msg = []  
    h = power(p, key, q)  
    for i in range(0, len(en_msg)):  
        dr_msg.append(chr(int(en_msg[i]/h)))
```

```
    return dr_msg

def main():

    msg = 'avengers assemble'
    print("Original Message :", msg)

    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)

    key = gen_key(q)
    h = power(g, key, q)
    print("g used : ", g)
    print("g^a used : ", h)

    en_msg, p = encrypt(msg, q, h, g)
    dr_msg = decrypt(en_msg, p, key, q)
    dmsg = "".join(dr_msg)
    print("Decrypted Message :", dmsg);

if __name__ == '__main__':
    main()
```

## Output



```
Original Message : avengers assemble
g used : 14517236197559719184020706822449945698753511667480
g^a used : 54551274054473533491230406961486278889305135724516
g^k used : 36407304247616794561275497210285675953663014338298
g^ak used : 23768221343140593867520414018718460391992155209234
Decrypted Message : avengers assemble
```

## Implementation of Rabin Algorithm

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int p = 151, q = 43;
```

```
int n = p * q;
```

```
int encrypter(int m, int n)
```

```
{  
    int c = (m * m)%n;  
    return c;  
}
```

```
int mod(int k, int b, int m)
```

```
{  
    int i=0;  
    int a=1;  
    vector<int> t;  
    while(k>0){  
        t.push_back(k%2);  
        k=(k-t[i])/2;  
        i++;  
    }  
    for(int j=0; j<i; j++){  
        if(t[j]==1){  
            a=(a*b)%m;  
            b=(b*b)%m;  
        } else{  
            b=(b*b)%m;  
        }  
    }  
    return a;  
}
```

```
int modulo (int a, int b)
```

```
{  
    return a >= 0 ? a % b : ( b - abs ( a%b ) ) % b;
```

```
}
```

```
vector<int> Extended_Euclid(int a, int b)
```

```
{
```

```
    if (b>a) {
```

```
        int temp=a; a=b; b=temp;
```

```
    }
```

```
    int x=0;
```

```
    int y=1;
```

```
    int lastx=1;
```

```
    int lasty=0;
```

```
    while (b!=0) {
```

```
        int q= a/b;
```

```
        int temp1= a%b;
```

```
        a=b;
```

```
        b=temp1;
```

```
        int temp2 = x;
```

```
        x=lastx-q*x;
```

```
        lastx = temp2;
```

```
        int temp3 = y;
```

```
        y = lasty-q*y;
```

```
        lasty=temp3;
```

```
    }
```

```
    vector<int>arr(3);
```

```
    arr[0] = lastx;
```

```
    arr[1] = lasty;
```

```
    arr[2] = 1;
```

```
    return arr;
```

```
}
```

```
int decrypter(int c, int p, int q)
```

```
{
```

```
    int mp = mod((p+1)/4, c, p);
```

```
    int mq = mod((q+1)/4, c, q);
```

```
    vector<int> arr = Extended_Euclid(p, q);
```

```
    int rootp = arr[0]*p*mq;
```

```
    int rootq = arr[1]*q*mp;
```

```

double r = modulo((rootp+rootq), n);
if( r < 128)
    return r;
int negative_r = n - r;
if (negative_r < 128)
    return negative_r;
int s = modulo((rootp-rootq), n);
if( s < 128)
    return s;
int negative_s = n - s;
    return negative_s;
}

int main()
{
    vector<int>e;
    vector<int>d;

    string message ="avengers assemble";
    cout << "Plain text: " << message << endl;
    int len = strlen(message.c_str());
    cout << "Encrypted text: ";
    for(int i = 0; i <= len; i++)
    {
        e.push_back(encrypter(message[i], n));
        cout << e[i];
    }
    cout << endl;
    for(int i = 0; i < len; i++)
    {
        d.push_back(decrypter(e[i], p, q));
    }
    cout << "Decrypted text: ";
    for(int i=0;i<d.size();i++)
        cout << char(d[i]);
    cout << endl;
    return 0;
}

```

```
}
```

```
Plain text: avengers assemble
Encrypted text: 291693837085607411637081023910242916239239370853883111517137080
Decrypted text: avengers assemble

-----
Process exited after 0.01465 seconds with return value 0
Press any key to continue . . . ■
```

## Implementation RSA digital signature

```
def euclid(m, n):
```

```
    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)
```

```
def exteuclid(a, b):
```

```
    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)

    while r2 > 0:

        q = r1//r2
        r = r1-q * r2
        r1 = r2
        r2 = r
        s = s1-q * s2
        s1 = s2
```

```
s2 = s
t = t1-q * t2
t1 = t2
t2 = t
```

```
if t1 < 0:
    t1 = t1 % a
```

```
return (r1, t1)
```

```
p = 823
q = 953
n = p * q
Pn = (p-1)*(q-1)
```

```
key = []
```

```
for i in range(2, Pn):
```

```
    gcd = euclid(Pn, i)
```

```
    if gcd == 1:
        key.append(i)
```

```
e = int(313)
```

```
r, d = exteuclid(Pn, e)
```

```
if r == 1:
    d = int(d)
    print("decryption key is: ", d)
```

```
else:
    print("Multiplicative inverse does not exist.")
```

```
M = 19070
```

```
S = (M**d) % n
```



$M1 = (S^{**e}) \% n$

if  $M == M1$ :

    print("Accept the message sent by Alice")

else:

    print("Do not accept the message sent by Alice")

### Output

decryption key is: 160009

Accept the message sent by Alice