```
#include <bits/stdc++.h>
using namespace std;
typedef struct{
    int row;
    int col;

}position;
char mat[5][5];
void generateMatrix(string key)
{

    int flag[26]={0};
    int x =0,y=0;
    for(int i=0;i<key.length();i++)
    {
        if(key[i]=='j')
            key[i]='i';
        if(flag[key[i]-'a'] == 0)
        {
            mat[x][y++] = key[i];
            flag[key[i]-'a']=1;

        }
        if(y==5)x++,y=0;
    }
for(char ch='a';ch<='z';ch++)
{
    if(ch=='j')
        continue;
    if(flag[ch-'a']==0)
    {
        mat[x][y++]=ch;
        flag[ch-'a']=1;
    }
    if(y==5)x++,y=0;
}
}
string formatMessage(string msg)
{

    for(int i=0;i<msg.length();i++)
    {
        if(msg[i]=='j')msg[i]='i';
    }
        for(int i=1;i<msg.length();i+=2)
        {

            if(msg[i-1] ==msg[i])msg.insert(i,"x");

        }

        if(msg.length()%2!=0)msg +="x";
        return msg;
}
```

```
position getposition(char c)
{
    for(int i=0;i<5;i++)
        for(int j =0;j<5;j++)
        if(c == mat[i][j])
    {

        position p = {i,j};
        return p;
    }
}
string encrypt(string message)
{

    string ctext="";
    for(int i =0;i<message.length();i+=2)
    {

        position p1 = getposition(message[i]);
        position p2 = getposition(message[i+1]);
        int x1 = p1.row;
        int y1=p1.col;
        int x2=p2.row;
        int y2=p2.col;

     if( x1 == x2 ) // same row
        {
            ctext.append(1,mat[x1][(y1+1)%5]);
            ctext.append(1,mat[x2][(y2+1)%5]);
        }
        else if( y1 == y2 ) // same column
        {
            ctext.append(1,mat[ (x1+1)%5 ][ y1 ]);
            ctext.append(1,mat[ (x2+1)%5 ][ y2 ]);
        }
        else
        {
            ctext.append(1,mat[ x1 ][ y2 ]);
            ctext.append(1,mat[ x2 ][ y1 ]);
        }

    }
    return ctext;
}
string Decrypt(string message)
{

    string ptext;
    for(int i=0;i<message.length();i+=2)
    {

        position p1=getposition(message[i]);
        position p2=getposition(message[i+1]);
        int x1=p1.row;int y1=p1.col;
```

```
        int x2=p2.row;int y2=p2.col;
        if( x1 == x2 ) // same row
        {
            ptext.append(1,mat[x1][ --y1<0 ? 4: y1 ]);
            ptext.append(1,mat[x2][ --y2<0 ? 4: y2 ]);
        }
        else if( y1 == y2 ) // same column
        {
            ptext.append(1,mat[ --x1<0 ? 4: x1 ][y1]);
            ptext.append(1,mat[ --x2<0 ? 4: x2 ][y2]);
        }
        else
        {
            ptext.append(1,mat[ x1 ][ y2 ]);
            ptext.append(1, mat[ x2 ][ y1 ]);
        }
    }
    return ptext;
}
int main()
{
    string plaintext;
    cout<<"enter message:";cin>>plaintext;
    int n;
    cout<<"enter th no of keys:";cin>>n;
    string key[n];
    for(int i=0;i<n;i++)
    {
    cout << "enter no of keys :" << i+1<<":"<<key[i];
    cin>>key[i];
    generateMatrix(key[i]);
    cout<<"key"<<i+1<<"matrix:"<<endl;
    for(int k=0;k<5;k++)
    {
        for(int j=0;j<5;j++)
        {

            cout<<mat[k][j]<<" ";

    }
        cout<<endl;

    }
    cout<<"actual message\t\t"<<plaintext<<endl;
    string fmsg=formatMessage(plaintext);
    cout<<"formated message\t"<<fmsg<<endl;

    string ciphertext = encrypt(fmsg);
    cout<<"encrypted message\t"<<ciphertext<<endl;
    string decryptmsg =Decrypt(ciphertext);
    cout<<"Decrypted message\t:"<<decryptmsg<<endl;
}
}
```

2

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

int key[3][3] ;
int C[1000][3] = {0}  ; //cipher text

int findDet(int m[3][3]  , int n ){
    if(n==2) return m[0][0] * m[1][1] - m[0][1]*m[1][0] ;
        else if (n==3) {
                return m[0][0]*(m[1][1]*m[2][2]-m[1][2]*m[2][1])  - m[0][1] *(
m[1][0]*m[2][2]-m[2][0]*m[1][2] )  + m[0][2] * (m[1][0]*m[2][1]-m[1][1]*m[2][0]
)  ;
        }
        else return 0 ; //invalid input
}

int mod26(int x){ return x>=0?(x%26):26-(abs(x)%26) ; }

int findDetInverse(int R , int D = 26){ //R is the remainder or determinant
        int i =0 ;
        int p[100] = {0,1};
        int q[100] = {0} ;
        while(R!=0){
                q[i] = D/R ;
                int oldD = D ;
                D = R ;
                R = oldD%R ;
                if(i>1){
                        p[i] = mod26(p[i-2] - p[i-1]*q[i-2]) ;
                }
                i++ ;
        }
        if(i == 1) return p[i] = 1 ;
        return p[i] = mod26(p[i-2] - p[i-1]*q[i-2]) ;
}


void multiplyMatrices(int a[1000][3] , int a_rows , int a_cols ,  int b[1000][3]
, int b_rows , int b_cols , int res[1000][3]){
        for(int i= 0 ;i < a_rows ; i++){
                for(int j=0 ;j < b_cols ; j++){
                        for(int k = 0 ; k < b_rows ; k++)
                        {
                                res[i][j] += a[i][k]*b[k][j] ;
                        }
                        res[i][j] = mod26(res[i][j]) ;
                }
        }
}


void findInverse(int m[3][3] , int n  , int detInverse , int inverse[3][3] ) {

        int adj[3][3] = {0} ;
```

```
        if(n==2){
                adj[0][0] = m[1][1] ; adj[1][1] = m[0][0]  ; adj[0][1] =
-m[0][1] ; adj[1][0] = -m[1][0] ;
        }
        else{ //n == 3
                int temp[5][5] = {0} ;
                // fill the 5x5 matrix
                for(int i = 0 ;i < 5 ; i++)
                        for(int j = 0 ;j  < 5 ; j++)
                                temp[i][j] = m[i%3][j%3] ;
                // except first row and column , multiply elements along rows
and place them along columns
                for(int i =1 ;i <= 3 ; i++){
                        for(int j =1  ; j <= 3 ; j++){
                                adj[j-1][i-1] =
mod26(temp[i][j]*temp[i+1][j+1]-temp[i][j+1]*temp[i+1][j]) ;
                        }
                }
        }
        for(int i = 0 ;i < n ; i++){
                for(int j = 0 ;j  < n ; j++) inverse[i][j] =
mod26(adj[i][j]*detInverse) ;
        }
}

string encrypt( string pt , int n){
        // C = P*K
        int P[1000][3]={0} ; //plainttext
        int ptIter = 0  ;
        while(pt.length()%n!=0)pt+="x" ;  //pad extra x
        for(int i =0 ; i< pt.length()/n ; i++){
                for(int j =0 ;j < n ;j++) P[i][j] = pt[ptIter++]-'a' ;
        }
        multiplyMatrices(P, pt.length()/n , n , key , n , n , C) ;

        string ct = "" ;
        for(int i =0 ; i< pt.length()/n ; i++){
                for(int j =0 ;j < n ;j++) ct += (C[i][j]+'a') ;
        }
        return ct ;
}
string decrypt( string ct , int n){
        // P = C * K^-1
        int P[1000][3]={0} ; //plainttext
        int ctIter = 0  ;
        int keyInverse[3][3] = {0} ;
        findInverse(key , n , findDetInverse(findDet(key,n)) , keyInverse ) ;
        multiplyMatrices(C, ct.length()/n , n , keyInverse , n , n , P) ;

        string pt = "" ;
        for(int i =0 ; i< ct.length()/n ; i++){
                for(int j =0 ;j < n ;j++) pt += (P[i][j]+'a') ;
        }
        return pt ;
```

```
}


int main(void){
   int n ;
   string pt ;
   cout<<"Enter plain text : "  ; cin>> pt ;
   cout<<"Enter number of rows in keymatrix : " ; cin>>n ;
   cout<<"Enter key matrix  : " <<endl;
   for(int i =0  ;i < n ; i++) for(int j =0 ;j < n ; j++) cin>>key[i][j] ;
   string ct = encrypt(pt  , n) ;
   cout<<"Original text  : " << pt <<endl;
   cout<<"Cipher text    : " << ct <<endl;
   cout<<"Decrypted text : " << decrypt(ct,n) <<endl;
}
```

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

char uniqtext[26]; // Global variable

/* read plain text from plaintext.txt file */
string readPlainText()
{
        ifstream fin;
        string ptext;

        fin.open("plaintext.txt");
        fin >> ptext;
        fin.close();

        return ptext;
}

/* write cipher text to ciphertext.txt file */
void writecipherText(string ctext)
{
        ofstream fout;
        fout.open("ciphertext.txt");
        fout << ctext;
        fout.close();
}

/* function to find all possible permutation */
void permute(char a[], int l, int r, vector<string>& keyspace) // keyspace is
passed by reference
{
        if(l == r)
        {
                keyspace.push_back(a);
        }
        else
        {
                for(int i = l; i <= r; i++)
                {
                        swap(a[l], a[i]); //inbuilt swap function
                        permute(a, l+1, r, keyspace);
                        swap(a[l], a[i]);
                }
        }
}

vector<string> genKeySpace(string plaintext)
{
        set<char> uniqSet;
        vector<string> keyspace; // contains all possible permutation of letters
in plaintext
        int count = 0;

        /* store all the unique letters of plain text in uniqSet */
```

```
        for(int i=0; i < plaintext.length(); i++)
        {
                uniqSet.insert(plaintext[i]);
        }

        /* copy uniqSet to uniqtext char array */
        for(set<char>::iterator it = uniqSet.begin(); it != uniqSet.end(); it++)
        {
                uniqtext[count++] = (*it);
        }

        permute(uniqtext, 0, strlen(uniqtext)-1, keyspace);
        return keyspace;
}

/* create cipher text using key */
string encrypt(string unique, string key)
{
        string plaintext = readPlainText();
        string ciphertext = "";

        for(int i=0; i < plaintext.length(); i++)
        {
                int idx = unique.find(plaintext[i]);
                ciphertext += key[idx];
        }
        return ciphertext;
}

/* frequency = (no of occurance of a character / length of plaintext) */
/* show frequency of all characters of plain text and cipher text */
void showFrequency(string pt , string ct)
{
        map<char , int > mPlain ;
        map<char , int > mCipher ;

        for(int i =0 ;i < pt.length() ; i++)
        {
                mPlain[pt[i]]++ ;
                mCipher[ct[i]]++ ;
        }
        cout<<"\nFrequency\t\tPlaintext Character\t\tCiphertext character"
<<endl;
        cout<<"=========\t\t==================\t\t==================" <<endl;
        for(int i=0 ; i<pt.length() ; i++)
        {
                cout<< (float)mPlain[pt[i]]/pt.length() << "\t\t\t\t" << pt[i]
<< "\t\t\t\t" << ct[i] << endl ;
        }
}

int main()
{
        srand(time(NULL)) ;
```

```
        string plaintext = readPlainText() ;
        cout<<"Plain text = \t" << plaintext << endl;

        vector<string> keyspace = genKeySpace(plaintext);
        string key = keyspace[rand()%keyspace.size()];

        cout<<"Unique chars = \t" << uniqtext <<endl;
        cout<<"Chosen key = \t" << key <<endl;

        string ciphertext = encrypt(uniqtext , key) ;
        writecipherText(ciphertext) ; // write ciphertext to file
        showFrequency(plaintext , ciphertext) ;
}
```

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

string encrypt(string pt , string key)
{
    string ct = ""; // ciphertext
    int k = 0;      // plaintext iterator

    int num_row = ceil((float) pt.length()/key.length());
    int num_col = key.length();
    char mat[num_row][num_col];

    cout << "\nEncryption Matrix :" << endl;
    cout << "--------------------" << endl;
    for(int i=0; i<num_row ; i++)
    {
        for(int j=0; j<num_col; j++)
        {
            if(k < pt.length())
            {
                cout << (mat[i][j] = pt[k++]) << "  ";
            }
            else
            {
                cout << (mat[i][j] = 'x') << "  " ;
            }
        }
        cout << endl;
    }
    for(int i=0; i<num_col; i++)
    {
        for(int j=0; j<num_row; j++)
        {
            ct += mat[j][key.find(i+'1')];
        }
    }
    return ct;
}

string decrypt(string ct , string key)
{
    string pt = ""; // plaintext
    int k = 0; // ciptext iterator

    int num_row = ceil((float)ct.length() / key.length());
    int num_col = key.length();
    char mat[num_row][num_col];

    for(int i=0; i<num_col; i++)
    {
        for(int j=0; j<num_row; j++)
        {
             mat[j][key.find(i+'1')] = ct[k++];
        }
```

```
    }

    cout << "\nDecryption Matrix :" << endl;
    cout << "---------------------" << endl;
    for(int i=0; i<num_row ; i++)
    {
        for(int j=0; j<num_col; j++)
        {
            cout << mat[i][j] << "  ";
            pt += mat[i][j];
        }
        cout << endl;
    }
    return pt;
}

int main()
{
    string plaintext , key , ciphertext , decryptext;

    cout << "Enter text : ";
    cin >> plaintext;

    cout << "Enter key  : ";
    cin >> key;

    ciphertext = encrypt(plaintext , key);
    cout << "\nEncrypted text \t: " << ciphertext << endl;

    decryptext = decrypt(ciphertext , key);
    cout << "\nDecrypted text \t: " << decryptext << endl;
}
```

5

Generate and print 48-bit keys for all sixteen rounds of DES algorithm, given a 64-bit initial key

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

int perm1[] = {
1,2,3,4,5,6,7,
8,9,10,11,12,13,14,
15,16,17,18,19,20,21,
22,23,24,25,26,27,28,
29,30,31,32,33,34,35,
36,37,38,39,40,41,42,
43,44,45,46,47,48,49,
50,51,52,53,54,55,56,
};
int perm2[] = {
1,2,3,4,5,6,7,8,
9,10,11,12,13,14,15,16,
17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,
33,34,35,36,37,38,39,40,
41,42,43,44,45,46,47,48,
};
int leftshift[] = {1,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1};

string rotatesubkey(string s,int rot)
{
  return s.substr(rot,s.length()-rot)+s.substr(0,rot);
};
string permu1f(string input)
{ int i;
  string res="";
  for(i=0;i<56;i++)
  {
    res+=input[perm1[i]-1];
  }
  return res;
}
string permu2f(string input)
{ int i;
  string res="";
  for(i=0;i<48;i++){
  res += input[perm2[i]-1];
  }
  return res;
}
void genkey(string left, string right)
{int i;
  ofstream fout;
  fout.open("keygen.txt");
  for(i=0;i<16;i++)
  {
    left = rotatesubkey(left,leftshift[i]);
```

```
    right = rotatesubkey(right,leftshift[i]);
    string key = permu2f(left+right);
    cout <<"key"<<i+1<<"\t"<<key<<endl;
    fout<<key<<endl;

  }
}
int main()
{
  unsigned long long hexkey;
  cout<<"enter 64bit";
  cin>>hex>>hexkey;

  string key = bitset<64>(hexkey).to_string();
  cout<<"p1"<<key<<endl;

  key = permu1f(key);
  cout <<"p1"<<key<<endl;

  cout<<"\nsubkey"<<endl;
  genkey(key.substr(0,28),key.substr(28,28));

  cout<<endl<<endl;
}
```

i. Given 64-bit output of (i-1)th round of DES, 48-bit ith round key Ki and E table, find the 48-bit input for S-box.

ii. Given 48-bit input to S-box and permutation table P, find the 32-bit output Ri of ith round of DES algorithm.

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

int expPermute[] = {
        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 };


string expansionPermute(string input)
{
        string res = "";
        for(int i=0; i<48; i++)
        {
                res += input[expPermute[i]-1];
        }
        return res;
}

string XOR(string input1, string input2)
{
        string res = "";
        for(int i=0; i<input1.length(); i++)
        {
                res += (input1[i] == input2[i]) ? "0" : "1";
        }
        return res;
}
unsigned int sBoxes[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 permTable[] = {
    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 };

string substitution(string input)
{
    string res = "";
    for(int i=0; i<8; i++)
    {
        string sInput = input.substr(6*i, 6) ;
        int row = bitset<2>( sInput.substr(0,1) + sInput.substr(5,1)
).to_ulong() ;
        int col = bitset<4>( sInput.substr(1,4) ).to_ulong() ;
        res += bitset<4>(sBoxes[i][row][col]).to_string() ;


    }
    return res;
}
string permute(string input)
{
    string res = "";
    for(int i=0; i<32; i++)
    {
```

```
        res += input[permTable[i]-1];
    }
    return res;
}
int main()
{
        int i;
        unsigned long long hexInput,hexSBoxInput;
        string Ki;
        ifstream fin;



        cout << "\nEnter Round number (i) : ";
        cin >> i;

        cout << "Enter 64-bit (i-1)th round output in hex: " ;
        cin >> hex >> hexInput;
        string input = bitset<64>(hexInput).to_string();

        fin.open("keygen.txt");
        for(int j=1; j<=i; j++)
        {
                fin >> Ki;
        }



        if(Ki.length() == 0)
        {
                cout << "\nkeygen.txt not found !!! \n" << endl;
                exit(1);
        }

        cout << "\n64-bit Binary Input = " << input << endl ;
        cout << "key for ith round (Ki) = " << Ki << endl ;

        string Ri_1 = input.substr(32,32);
        cout << "\nRight half of 64-bit input, Ri_1 = " << Ri_1 << endl;

        string R48 = expansionPermute(Ri_1);
        cout << "Ri_1 after expansion permutation = " << R48 << endl;

        string sBoxInput = XOR(R48, Ki);
        cout << "\nInput to s-box : " << sBoxInput << endl << endl;

 cout << "\nEnter 48-bit input for S-Box in hex(12-digits)      : " ;
    cin >> hex >> hexSBoxInput;

    cout << "Enter 64-bit (i-1)th round output in hex(16-digits) : " ;
    cin >> hex >> hexInput;

    string sBoxinput = bitset<48>(hexSBoxInput).to_string();
    cout << "\nS-Box Input       : " << sBoxinput << endl;
```

```
    cout << " Round(i-1) output : " << input << endl;
    string Li_1 = input.substr(0,32);
    cout << "\nLi_1              : " << Li_1 << endl;

     string sBoxOutput = substitution(sBoxinput);
    cout << "\nS-Box output    = " << sBoxOutput << endl;

    string P = permute(sBoxOutput);
    cout << "Permuted output = " << P << endl;

    string Ri = XOR(P, Li_1);
    cout << "\nOutput of ith round (Ri) = " << Ri << endl << endl;

}
```

Consider the 128 bits initial key and expand it to 10 different keys each of size 128 bits using AES key expansion technique.

```
#include <iostream>

using namespace std;

class AES_KEY_GEN {
private:
        uint32_t words[44] = {0};
        uint8_t SBox[16][16] = {
                { 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, 0x63, 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, },
        };
        uint32_t RCon[10] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
0x1B, 0x36, };

        uint32_t RotWord(uint32_t word)
        {
                return (word << 8) | (word >> 24);
        }

        uint32_t SubWord(uint32_t word)
```

```
        {
                uint32_t result = 0;
                for (int i = 0; i < 4; i++)
                {
                        result |= SBox[(word & 0xF0) >> 4][word & 0x0F] << (i *
8);
                        word = word >> 8;
                }
                return result;
        }

public:
        AES_KEY_GEN(uint8_t key[16])
        {
                for (int i = 0; i < 4; i++)
                        words[i] = key[4 * i] << 24 | key[4 * i + 1] << 16 |
key[4 * i + 2] << 8 | key[4 * i + 3];

                for (int i = 4; i < 44; i++)
                {
                        uint32_t temp = words[i - 1];
                        if (i % 4 == 0)
                        {
                                temp = SubWord(RotWord(temp)) ^ (RCon[(i / 4) -
1] << 24);
                        }
                        words[i] = words[i - 4] ^ temp;
                }

                for (int i = 0; i < 10; i++)
                {
                        cout << "Key " << i + 1 << ": ";
                        for (int j = 0; j < 4; j++)
                                cout << hex << words[i * 4 + j + 4] << " ";
                        cout << endl;
                }
        }
};

int main()
{
        uint8_t key[16] = {
                0x0F, 0x15, 0x71, 0xC9,
                0x47, 0xD9, 0xE8, 0x59,
                0x0C, 0xB7, 0xAD, 0xD6,
                0xAF, 0x7F, 0x67, 0x98,
        };
        AES_KEY_GEN* aes = new AES_KEY_GEN(key);
        return 0;
}
```

Consider a message of 16 bytes (128 bits) and perform XOR operation with an initial round key [W0, W1, W2, W3] of size 128 bits
to generate a state array in AES. w.r.t generated state array of size 128 bits, perform the following operations in each round.

```
#include <iostream>

using namespace std;

class AES {
private:
        uint8_t state[4][4] = {0};
        uint8_t SBox[16][16] = {
                { 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, 0x63, 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, },
        };

        uint8_t SubWord(uint8_t word)
        {
                return SBox[(word & 0xF0) >> 4][word & 0x0F];
        }

        void ShiftRows(int i)
        {
```

```
                uint8_t temp = state[i][0];
                state[i][0] = state[i][1];
                state[i][1] = state[i][2];
                state[i][2] = state[i][3];
                state[i][3] = temp;
        }

public:
        AES(uint8_t message[16], uint32_t words[4])
        {

                for (int i = 0; i < 4; i++)
                        for (int j = 0; j < 4; j++)
                                state[i][j] = message[i * 4 + j];


                for (int i = 0; i < 4; i++)
                        for (int j = 0; j < 4; j++)
                                state[i][j] = state[i][j] ^ ((words[i] >> (8 *
(4 - j - 1))) | 0xFF);


                for (int i = 0; i < 4; i++)
                        for (int j = 0; j < 4; j++)
                                state[i][j] = SubWord(state[i][j]);


                for (int i = 0; i < 4; i++)
                        for (int j = 0; j < i; j++)
                                ShiftRows(i);

                cout << "State is:" << endl;
                for (int i = 0; i < 4; i++)
                {
                        for (int j = 0; j < 4; j++)
                                cout << hex << uint16_t(state[i][j]) << " ";
                        cout << endl;
                }
        }
};

int main()
{
        uint8_t message[16] = {
                0x0F, 0x15, 0x71, 0xC9,
                0x47, 0xD9, 0xE8, 0x59,
                0x0C, 0xB7, 0xAD, 0xD6,
                0xAF, 0x7F, 0x67, 0x98,
        };
        uint32_t words[4] = {
                0xdc9037b0,
                0x9b49dfe9,
                0x97fe723f,
                0x388115a7,
```

```
        };

        AES* aes = new AES(message, words);
        return 0;
}
```

9

Implement the following with respect to RC4:

i. Print first n key bytes generated by key generation process.
ii. Illustrate encryption/decryption by accepting one byte data as input on the above generated keys.

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

int main()
{
    int S[256], T[256], keyStream[256];
    string ptString, keyString, dtString = "";
    int pt[256], ct[256], dt[256];

    cout << "Enter message : "; cin >> ptString;
    cout << "Enter key     : "; cin >> keyString;
    int n = ptString.length();

    cout << "\nPlain text \t: " ;
    for(int i=0; i<n; i++)
    {
        pt[i] = ptString[i];
        cout << pt[i] << " ";
    }


    for(int i=0; i<256; i++)
    {
        S[i] = i;
        T[i] = (int)keyString[i%keyString.length()];
    }


    int j=0;
    for(int i=0; i<256; i++)
    {
        j = (j + S[i] + T[i]) % 256;
        swap(S[i], S[j]);
    }


    cout << "\nKey Stream \t: ";
    j=0;
    for(int i=0; i<n; i++)
    {
        j = (j + S[i]) % 256;
        swap(S[i], S[j]);
        int t = (S[i] + S[j]) % 256;
        keyStream[i] = S[t];
        cout << keyStream[i] << " ";
    }
```

```
    cout << "\nCipher Text \t: ";
    for(int i=0; i<n; i++)
    {
        ct[i] = pt[i] ^ keyStream[i];
        cout << ct[i] << " ";
    }


    cout << "\nDecrypted text \t: " ;
    for(int i=0; i<n; i++)
    {
        dt[i] = ct[i] ^ keyStream[i];
        cout << dt[i] << " ";
        dtString += (char)dt[i];
    }
    cout << "\nDecrypted text \t: " << dtString << endl;
}
```

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

int randInRange(int low, int high)
{
    return rand()%(high-(low+1)) + (low+1) ;
}

int genPrime3mod4()
{
    while(true)
    {
        int num = randInRange(10000,100000);
        if(num%4 != 3) continue;

        bool prime = true;
        for(int i=2; i<=sqrt(num); i++)
        {
            if(num % i == 0)
            {
                prime = false;
                break;
            }
        }
        if(prime) return num;
    }
}

int bbs(int p, int q)
{
    long long n = (long long)p*q ;

    long long s;
    do{ s = rand(); } while(s%p==0 || s%q==0 || s==0);

    int B = 0;
    long long x = (s*s) % n;
    for(int i=0; i<10; i++) // to generate 10 bit output
    {
        x = (x*x) % n;
        B = B<<1 | (x & 1); // x%2 = x&1
    }

    cout<<"Blum Blum Shub"<<endl<<"--------------"<<endl;
    cout<<"p = "<< p <<"\nq = "<< q <<"\nn = "<< n <<"\ns = "<< s <<endl;
    return B;
}


unsigned long long powModN(int a, int b, int n)
{
        unsigned long long res=1;
        for(int i=0; i<b; i++)
        {
```

```
        res = ((unsigned long long)(res * a)) % n;
        }
        return res;
}

string rabinMiller(int n)
{
    int k = 0;
    int q = n-1;
    while(q % 2 == 0)
    {
        q = q/2 ;
        k++ ;
    }

    int a = randInRange(1, n-1);

    cout << "\nRabin Miller(" << n << ")\n-----------------" << endl;
    cout << n-1 << " = 2^" << k << " * " << q << endl;
    cout << "k = " << k << "\nq = " << q << "\na = " << a << endl;


     if(powModN(a,q,n) == 1) return "inconclusive";

    for(int j=0; j<k ; j++)
    {
        if(powModN(a, pow(2,j)*q, n) == n-1) return "inconclusive";
    }
    return "composite";
}

int main()
{
    srand(time(NULL));
    int p = genPrime3mod4();
    int q = genPrime3mod4();
    int randNum = bbs(p, q);
    cout << "Random number generated by BBS = " << randNum << endl;

   for(int i=0; i<4; i++)
    cout << rabinMiller(randNum) << endl ;
}
```

11

Implement RSA algorithm to process blocks of plaintext (refer Figure 9.7 of the text book), where plaintext is a string of characters and let the block size be two characters.
(Note: assign a unique code to each plain text character i.e., a=00, A=26). The program should support the following.
i. Accept string of characters as plaintext.
ii. Encryption takes plaintext and produces ciphertext characters
iii. Decryption takes ciphertext characters obtained in step ii and produces corresponding plaintext characters
Display the result after each step.

```
//////////////////////////
client program
//////////////////////////

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

int connectToServer(const char* ip, int port)
{
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), inet_addr(ip)};

    if(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0 ){
        cout << "\nRun server program first." << endl; exit(0);
    }else{
        cout << "\nClient is connected to Server." << endl;
    }
    return sock;
}

int randInRange(int low, int high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

int gcd(int a, int b)
{
    return b==0 ? a : gcd(b, a%b);
}

int powermod(int a, int b, int n)
{
    int res = 1;
    for(int i=0; i<b; i++)
    {
        res = (res*a) % n;
    }
    return res;
}

// M = C^d mod n
int decrypt(int C, int PR[2])
```

```
{
    return powermod(C, PR[0], PR[1]);
}

// a=00, b=01, ... A=26, B=27...
char toChar(int n)
{
    return (n >= 26) ? (n+'A'-26) : (n+'a');
}

int main()
{
    char ip[50];
    int port;
    cout << "Enter Server's IP address: "; cin >> ip;
    cout << "Enter port : "; cin >> port;
    int sock = connectToServer(ip, port);

    int p,q;
    cout << "\nEnter two large prime numbers(>100) : "; cin >> p >> q;
    int n = p * q ; // should be greater than 5151 (since ZZ=5151)
    int phi = (p-1) * (q-1);

    srand(time(NULL));
    int e, d;
    do{ e = randInRange(1, phi); } while(gcd(e,phi) != 1);

    for(d=1; d<phi; d++)
    {
        if((d*e)%phi == 1) break;
    }

    int PU[2] = {e, n}; // public key
    int PR[2] = {d, n}; // private key
    cout << "\nPublic key , PU = {" << e << ", " << n << "}" << endl;
    cout << "Private key, PR = {" << d << ", " << n << "}" << endl;

    send(sock, &PU, sizeof(PU), 0); // send public key to server
    cout << "\nSent Public key to server." << endl;

    string msg = "";
    while (true)
    {
        int C; // ciphertext
        recv(sock, &C, sizeof(C), 0);
        if(C == -1)   break; // at the end -1 will be received
        cout << "\nCiphertext received from server : " << C << endl;

        int M = decrypt(C,PR);
        cout << "Decrypted Text : " << M << endl;
        msg += toChar(M/100); // first char in block
        msg += toChar(M%100); // second char in block
    }
    cout << "\nDecrypted message : " << msg << endl << endl;
```

```
}

//////////////////////////
Server Program
//////////////////////////

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

int createServer(int port)  // TCP connection
{
    int sersock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), INADDR_ANY};

    bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
    cout << "\nServer Online. Waiting for client...." << endl;

    listen(sersock, 5);
    int sock = accept(sersock, NULL, NULL);
    cout << "Connection Established." << endl;

    return sock;
}

int powermod(int a, int b, int n)
{
    int res = 1;
    for(int i=0; i<b; i++)
    {
        res = (res*a) % n;
    }
    return res;
}

// C = M^e mod n
int encrypt(int M, int PU[2]) // PU = {e, n}
{
    return powermod(M, PU[0], PU[1]);
}

// a=00, b=01, ... A=26, B=27...
int toInt(char c)
{
    return (c < 'a') ? (c-'A'+26) : (c-'a');
}

int main()
{
    int port;
    cout << "Enter port : "; cin >> port;
    int sock = createServer(port);

    int PU[2];
```

```
    recv(sock, &PU, sizeof(PU), 0); // receive public key from client
    cout << "\nPublic key received from client : {" << PU[0] << ", " << PU[1] <<
"}" << endl;

    string msg; // plaintext message
    cout << "\nEnter message to encrypt : "; cin >> msg;

    if(msg.length()% 2 != 0) msg+="x";

    for(int i=0; i<msg.length(); i+=2) // increment by 2 for block
    {
        int M = toInt(msg[i])*100 + toInt(msg[i+1]); // block consist of two msg
character
        cout << "\nPlaintext block : " << M << endl;

        int C = encrypt(M, PU);
        cout << "Encrypted text  : " << C << endl;
        send(sock, &C, sizeof(C), 0); // send ciphertext to client
    }
    int stop = -1; // at end send -1 to tell client to stop
    send(sock, &stop, sizeof(stop), 0); //at end send stop to client
    cout << "\nSent ciphertext to client." << endl << endl;
}
```

Implement RSA algorithm using client-server concept. Using this illustrate secret key distribution scenario with confidentiality and authentication.
The program should support the following :
i. Both client and server generates{PU, PR} and distributes PU to each other.
ii. Establish a secret key K between client and server by exchanging the messages as shown in below figure:

```
//////////////////////
client program
//////////////////////

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


int p, q, e, d, n, phi; // global variables
int PUc[2], PRc[2];     // client's keys
int PUs[2];             // server's public key
int sock;

void connectToServer(const char* ip, int port)
{
    sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), inet_addr(ip)};

    if(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0 ){
        cout << "\nRun server program first." << endl; exit(0);
    }else{
        cout << "\nClient is connected to Server." << endl;
    }
}

int randInRange(int low, int high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

int gcd(int a, int b)
{
    return b==0 ? a : gcd(b, a%b);
}

void genKey()
{
    cout << "\nEnter two prime numbers (>100) : "; cin >> p >> q;
    n = p * q ;
    phi = (p-1) * (q-1);

    srand(time(NULL));
    do{ e = randInRange(1, phi); } while(gcd(e,phi) != 1);
    for(d=1; d<phi; d++)
    {
```

```
        if((d*e)%phi == 1) break;
    }

    PUc[0] = e; PUc[1] = n; // public key
    PRc[0] = d; PRc[1] = n; // private key
    cout << "\nPublic key , PUc = {" << e << ", " << n << "}" << endl;
    cout <<   "Private key, PRc = {" << d << ", " << n << "}" << endl;
}

void shareKey() // first receive then send
{
    recv(sock, &PUs, sizeof(PUs), 0); // receive public key from server
    send(sock, &PUc, sizeof(PUc), 0); // send client's public key to server
    cout << "Public key received from server, PUs = {" << PUs[0] << ", " <<
PUs[1] << "}" << endl;
    cout << "\nSent client's Public key to server." << endl;
}

int powermod(int a, int b, int n)
{
    int res = 1;
    for(int i=0; i<b; i++)
    {
        res = (res*a) % n;
    }
    return res;
}

int encrypt(int M, int PU[2])
{
    return powermod(M, PU[0], PU[1]);
}

int decrypt(int C, int PR[2])
{
    return powermod(C, PR[0], PR[1]);
}

int main()
{
    char ip[50];  cout<<"\nEnter server's IP address: "; cin>>ip;
    int port;     cout<<"Enter port : ";  cin>>port;
    srand(time(NULL));

    connectToServer(ip, port);
    genKey();
    shareKey(); // share public keys

    // step-1: recv cipher from server and Dec(PRc, [N1||ID])
    int cipher;
    recv(sock, &cipher, sizeof(cipher), 0);
    cout << "\nReceived encrypted (N1||ID) from server : " << cipher << endl;
    int msg = decrypt(cipher, PRc);
    int N1 = msg/100;
```

```
    int ID = msg%100;
    cout << "Decrypted Server's ID,   IDs = " << ID << endl;
    cout << "Decrypted Server's nonce, N1 = " << N1 << endl;

    // step-2: send En(PUs, (N1||N2)) to server
    int N2 = rand() % 100; // nonce
    cout << "\nNonce generated, N2 = " << N2 << endl;
    msg = N1*100 + N2;
    cipher = encrypt(msg, PUs);
    send(sock, &cipher, sizeof(cipher), 0);
    cout << "Sent encrypted (N1||N2) to server : " << cipher << endl;

    // step-3: recv enc(N2) from client and Dec(PRc, N2)
    recv(sock, &cipher, sizeof(cipher), 0);
    cout << "\nReceived encrypted (N2) from server : " << cipher << endl;
    int N2s = decrypt(cipher, PRc);
    cout << "Decrypted Client's Nonce, N2 = " << N2s << endl;
    if(N2s != N2) {cout << "\nNonce didn't match!\n"; exit(-1);}
    else {cout << "----- Server Authenticated -----" << endl;}

    // step-4: recv cipher and perform k = Dec(PUs, Dec(PRc, C))
    int k;
    recv(sock, &cipher, sizeof(cipher), 0);
    cout << "\nReceived cipher from Server : " << cipher << endl;
    k = decrypt(decrypt(cipher, PRc), PUs);
    cout << "Decrypted Secret Key : " << k << endl << endl;
}


//////////////////////
Sever Program
//////////////////////

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

int p, q, e, d, n, phi; // global variables
int PUs[2], PRs[2];     // server's keys
int PUc[2];             // client's public key
int sock;

void createServer(int port)  // TCP connection
{
    int sersock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), INADDR_ANY};

    bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
    cout << "\nServer Online. Waiting for client...." << endl;

    listen(sersock, 5);
    sock = accept(sersock, NULL, NULL);
    cout << "Connection Established." << endl;
}
```

```
int randInRange(int low, int high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

int gcd(int a, int b)
{
    return b==0 ? a : gcd(b, a%b);
}

void genKey()
{
    cout << "\nEnter two prime numbers (>100): "; cin >> p >> q;
    n = p * q ;
    phi = (p-1) * (q-1);

    srand(time(NULL));
    do{ e = randInRange(1, phi); } while(gcd(e,phi) != 1);
    for(d=1; d<phi; d++)
    {
        if((d*e)%phi == 1) break;
    }

    PUs[0] = e; PUs[1] = n; // public key
    PRs[0] = d; PRs[1] = n; // private key
    cout << "\nPublic key , PUs = {" << e << ", " << n << "}" << endl;
    cout <<   "Private key, PRs = {" << d << ", " << n << "}" << endl;
}

void shareKey() // first send then receive
{
    send(sock, &PUs, sizeof(PUs), 0); // send Server's public key to client
    recv(sock, &PUc, sizeof(PUc), 0); // receive public key from client
    cout << "Sent Server's Public key to client." << endl;
    cout << "\nPublic key received from client : {" << PUc[0] << ", " << PUc[1]
<< "}" << endl;
}

int powermod(int a, int b, int n)
{
    int res = 1;
    for(int i=0; i<b; i++)
    {
        res = (res*a) % n;
    }
    return res;
}

int encrypt(int M, int PU[2])
{
    return powermod(M, PU[0], PU[1]);
}
```

```
int decrypt(int C, int PR[2])
{
    return powermod(C, PR[0], PR[1]);
}

int main()
{
    int port;  cout<<"\nEnter port : ";  cin>>port;
    srand(time(NULL));

    createServer(port);
    genKey();
    shareKey(); // share public keys

    int ID;  cout<<"\nEnter Server's ID number (<100): "; cin>>ID;
    int N1 = rand()%100; // nonce
    cout << "Nonce generated, N1 = " << N1 << endl;

    // step-1: send En(PUc, [N1||ID]) to client
    int msg = N1*100 + ID; // append ID to nonce
    int cipher = encrypt(msg, PUc);
    send(sock, &cipher, sizeof(cipher), 0);
    cout << "Sent encrypted (N1||ID) to client : " << cipher << endl;

    // step-2: recv cipher from client and Dec(PRs, (N1||N2))
    recv(sock, &cipher, sizeof(cipher), 0);
    cout << "\nReceived encrypted (N1||N2) from client : " << cipher << endl;
    msg = decrypt(cipher, PRs);
    int N1c = msg/100; // N1 received from client
    int N2 = msg%100;
    cout << "Decrypted Server's Nonce, N1 = " << N1c << endl;
    cout << "Decrypted Client's Nonce, N2 = " << N2 << endl;
    if(N1 != N1c) {cout << "\nNonce didn't match!\n"; exit(-1);}
    else {cout << "------ Client Authenticated ------" << endl;}

    // step-3: send En(PUc, N2) to client
    cipher = encrypt(N2, PUc);
    send(sock, &cipher, sizeof(cipher), 0);
    cout << "\nSent encrypted (N2) to client : " << cipher << endl;

    // step-4: send C = En(PUc, En(PRs, k))
    int k; // secret key
    cout << "\nEnter secret key (integer) to send : "; cin >> k;
    cipher = encrypt(encrypt(k,PRs), PUc);
    send(sock, &cipher, sizeof(cipher), 0);
    cout << "Sent encrypted secret key to client : " << cipher << endl << endl;
}
```

Compute common secret key between client and server using Diffie-Hellman key exchange technique. Perform encryption and decryption of message
using the shared secret key (Use simple XOR operation to encrypt and decrypt the message.)

```
////////////////////////////////
Client Program:
////////////////////////////////

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

int connectToServer(const char* ip, int port)
{
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port),inet_addr(ip)};

    if(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0 ){
        cout << "\nRun server program first." << endl; exit(0);
    }else{
        cout << "\nClient is connected to Server." << endl;
    }
    return sock;
}

int randInRange(int low, int high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

long powermod(long a, long b, long  q)
{
        long res=1;
        for(long i=0;i<b;i++)
        {
                res=(res*a)%q;
        }
        return res;
}

int main()
{
    char ip[50]; cout << "\nEnter server's IP address: "; cin >> ip;
    int port;    cout << "Enter port : "; cin >> port;
    int sock = connectToServer(ip, port);

        long q, alpha;
        cout<<"\nEnter a prime number, q : "; cin >> q;
        cout<<"Enter primitive root of q, alpha : "; cin >> alpha;

        srand(time(NULL));
        long Xc = randInRange(1, q); // client's private key (1<Xa<q)
        cout<< "\nClient's private key, Xc = " << Xc <<endl;
```

```
        long Yc = powermod(alpha, Xc, q); // client's public key
        send(sock, &Yc, sizeof(Yc), 0);  // send client's public key
        cout<< "Client's public key,  Yc = " << Yc <<endl;

        long Ys;
        recv(sock, &Ys, sizeof(Ys), 0);  // recv server's public key
        cout<< "\nServer's public key,  Ys = " << Ys <<endl;

        long k = powermod(Ys,Xc,q);
        cout<<"\nSecret Key, k = "<<k<<endl;

        long cipher;
        recv(sock, &cipher, sizeof(cipher), 0);
        cout<<"\nMessage received from Server  : " << cipher << endl;

        long decipher = cipher ^ k;     // decryption
        cout << "Decrpyted message : " << decipher << endl << endl;
}
////////////////////////////////
Server Program:
////////////////////////////////

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

int createServer(int port)  // TCP connection
{
        int sersock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), INADDR_ANY};

    bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
    cout << "\nServer Online. Waiting for client...." << endl;

    listen(sersock, 5);
    int sock = accept(sersock, NULL, NULL);
    cout << "Connection Established." << endl;
    return sock;
}

int randInRange(int low, int high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

long powermod(long a, long b, long  q)
{
        long res=1;
        for(long i=0; i<b; i++)
        {
                res=(res*a)%q;
        }
        return res;
```

```
}

int main()
{
    int port; cout << "\nEnter port : "; cin >> port;
    int sock = createServer(port);

        long q, alpha;
        cout<<"\nEnter a prime number, q : "; cin >> q;
        cout<<"Enter primitive root of q, alpha : "; cin >> alpha;

        long Yc;
        recv(sock, &Yc, sizeof(Yc), 0);  // recv client's public key
        cout<< "\nClient's public key,  Yc = " << Yc <<endl;

        srand(time(NULL));
        long Xs = randInRange(1, q); // server's private key
        cout<< "\nServer's private key, Xs = " << Xs <<endl;

        long Ys = powermod(alpha, Xs, q); // server's public key
        send(sock, &Ys, sizeof(Ys), 0);  // send server's public key
        cout<< "Server's public key,  Ys = " << Ys <<endl;

        long k = powermod(Yc,Xs,q);
        cout<<"\nSecret Key, k = "<<k<<endl;

        long msg;
        cout<<"\nEnter a message(number) to send : "; cin>>msg;

        long cipher = msg ^ k; // encryption
        send(sock, &cipher, sizeof(cipher), 0);
        cout << "Encrypted msg sent to client: " << cipher << endl << endl;
}
```

Implement DSS algorithm for signing and verification of messages between two parties (obtain H(M) using simple XOR method of hash computation on M).

```
//////////////////////////
client program
//////////////////////////
# include <bits/stdc++.h>
# include <arpa/inet.h>
using namespace std;

int connectToServer(const char* ip, int port)
{
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port),inet_addr(ip)};

    if(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0 ){
        cout << "\nRun server program first." << endl; exit(0);
    }else{
        cout << "\nClient is connected to Server." << endl;
    }
    return sock;
}

long mod(long a, long b)
{
    return a >= 0 ? (a%b) : b-(abs(a)%b) ;
}

long powermod(long a, long b, long  c)
{
    long res=1;
    for(int i=0; i<b; i++)
    {
        res = (res * a) % c;
    }
    return res;
}

long findInverse(long R , long D)
{
    int i = 0;
    long N = D; // copy D to N for taking mod
    long p[100] = {0,1};
    long q[100] = {0} ;

    while(R!=0)
    {
        q[i] = D/R ;
        long oldD = D ;
        D = R ;
        R = oldD%R ;
        if(i>1)
        {
            p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
```

```
        }
        i++ ;
    }
    if (i == 1) return 1;
    else        return p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
}

long H(long M)
{
    return (M ^ 1234); //hash key = 1234
}

int main()
{
    char ip[50]; cout << "\nEnter server's IP address: "; cin >> ip;
    int port;    cout << "Enter port : "; cin >> port;
    int sock = connectToServer(ip, port);

    long p, q; // prime numbers
    long r, s; // signature
    long g, y; // keys
    long M, hashval; // Message and Hash
    long w, v; // verify
    srand(time(NULL));

    recv(sock, &p, sizeof(p), 0);
    recv(sock, &q, sizeof(q), 0);
    recv(sock, &g, sizeof(g), 0);
    recv(sock, &y, sizeof(y), 0);
    recv(sock, &M , sizeof(M), 0);
    recv(sock, &r, sizeof(r), 0);
    recv(sock, &s, sizeof(s), 0);

    cout << "Received p =  " << p << endl;
    cout << "Received q =  " << q << endl;
    cout << "Received g =  " << g << endl;
    cout << "Received y =  " << y << endl;
    cout << "Received M'=  " << M << endl;
    cout << "Received r' = " << r << endl;
    cout << "Received s' = " << s << endl;

    hashval = H(M) ;
    cout << "\nH(M') = " << hashval << endl;

    //Verifying
    w = findInverse(s,q) % q;  cout << "w = " << w << endl;
    long u1 = (hashval * w) % q;
    long u2 = (r * w) % q;
    v = ((powermod(g,u1,p)*powermod(y,u2,p)) %p) %q;  cout<<"v = "<<v<<endl;
    if(v == r) cout<<"\nDigital Signature Verified. " << endl << endl;
    else           cout<<"\nDigital Signature is invalid !!!" << endl << endl;
}

//////////////////////////
```

```
server program
//////////////////////////

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

int createServer(int port)  // TCP connection
{
    int sersock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = {AF_INET, htons(port), INADDR_ANY};

    bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
    cout << "\nServer Online. Waiting for client...." << endl;

    listen(sersock, 5);
    int sock = accept(sersock, NULL, NULL);
    cout << "Connection Established." << endl;
    return sock;
}

long randInRange(long low, long high) // excluding high and low
{
    return rand()%(high-(low+1)) + (low+1) ;
}

long mod(long a, long b)
{
        return a >= 0 ? (a%b) : b-(abs(a)%b) ;
}

long powermod(long a, long b, long  c)
{
    long res=1;
    for(int i=0; i<b; i++)
    {
        res = (res * a) % c;
    }
    return res;
}

long findInverse(long R , long D)
{
    int i = 0;
    long N = D; // copy D to N for taking mod
    long p[100] = {0,1};
    long q[100] = {0} ;

    while(R!=0)
    {
        q[i] = D/R ;
        long oldD = D ;
        D = R ;
        R = oldD%R ;
```

```
        if(i>1)
        {
            p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
        }
        i++ ;
    }
    if (i == 1) return 1;
    else        return p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
}

long H(long M) // Hash Function
{
        return (M ^ 1234); //hash key = 1234
}

int main()
{
    int port;  cout << "\nEnter port : "; cin >> port;
    int sock = createServer(port);

    long p, q; // prime numbers
    long r, s; // signature
    long k, x, y, g; // keys
    long M, hashval; // Message and Hash
    srand(time(NULL));

    cout << "\nEnter a large prime number, p : ";   cin >> p;
    cout << "Enter a prime number, q (p-1 divisible by q & q>2) : "; cin >> q;
    if( (p-1)%q != 0 || q <3) { cout << "\nInvalid input\n"; exit(-1); }

    cout<<"Enter message, M = "; cin >> M;

    hashval = H(M);
    cout << "\nH(M) = " << hashval << endl;

    long h;
    do{
        h = randInRange(1, p-1);        // 1 < h < p-1
        g = powermod(h,(p-1)/q, p);         //g > 1
    } while(g<=1);
    cout << "g    = " << g;

    x = randInRange(1, q);  cout << "\nServer's Private key, x = " << x;
    y = powermod(g, x, p);  cout << "\nServer's Public  key, y = " << y;
    k = randInRange(1, q);  cout << "\nSecret key, k = " << k << endl;

    //Signing
    r = powermod(g, k, p) % q;
    s = (findInverse(k,q) * (hashval + x*r )) % q;
    cout << "\nServer's Signature {r,s} = {" << r << ", " << s << "}" << endl;

    send(sock, &p, sizeof(p), 0);
    send(sock, &q, sizeof(q), 0);
    send(sock, &g, sizeof(g), 0);
```

```
    send(sock, &y, sizeof(y), 0);
    send(sock, &M , sizeof(M), 0);
    send(sock, &r, sizeof(r), 0);
    send(sock, &s, sizeof(s), 0);

    cout << "\nSent p, q, g, and public key to client.";
    cout <<"\nSent message along with signature to client." << endl << endl;
}
```