12/16/2019

Cryptography and Network Security Lab

SIT, Tumkur

Author:

AKSHAT AGARWAL
1SI16CS010
https://github.com/git-akshat/CNS-Lab

Contributer:

NATESH M BHAT
1SI16CS066
https://github.com/nateshmbhat

Cryptography and Network Security Lab

This repository contains programs implemented in Cryptography and network security Lab in my 7th semester of SIT(VTU).

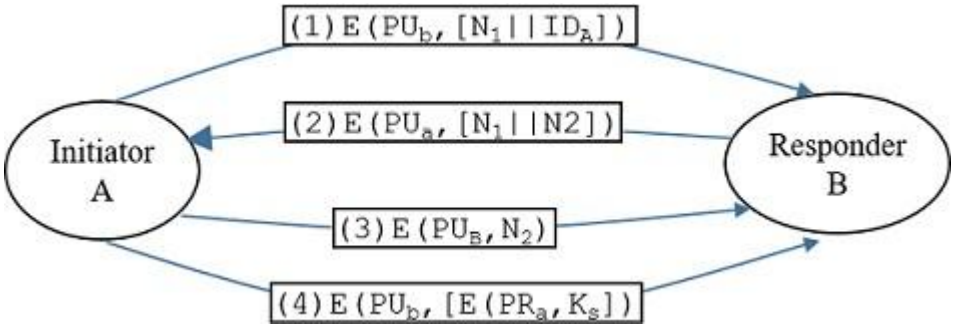
1. Perform encryption and decryption using mono-alphabetic cipher. The program should support the following :
    - Construct an input file named plaintext.txt (consisting of 1000 alphabets, without any space or special characters)
    - Encrypt the characters of plaintext.txt and store the corresponding ciphertext characters in ciphertext.txt
    - Compute the frequency of occurrence of each alphabet in both plaintext.txt and ciphertext.txt and tabulate the results
2. Write a program to perform the following using Playfair cipher technique
    - Encrypt a given message M with different keys {k1,k2,...,kn}. Print key and cipher text pair
    - Decrypt the cipher texts obtained in (i) to get back M
3. Write a program to perform the following using Hill cipher:
    - Encrypt a message M with a given key matrix of size 2X2 and 3X3
    - Decrypt the cipher text obtained in (i) by computing inverse of the respective key matrix.
4. Write a program to perform encryption and decryption using transposition technique with column permutation given as key.
5. Generate and print 48-bit keys for all sixteen rounds of DES algorithm, given a 64-bit initial key.
6. 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.
7. Given 48-bit input to S-box and permutation table P, find the 32-bit output Ri of ith round of DES algorithm.
8. Implement the following with respect to RC4:
    - Print first n key bytes generated by key generation process.
    - Illustrate encryption/decryption by accepting one byte data as input on the above generated keys.
9. Write a program to generate large random number using BBS random number generator algorithm and check whether the generated number is prime or not using RABIN-MILLER primality testing algorithm.
10. Implement RSA algorithm using client-server concept. The program should support the following :
    - Client generates {PU, PR} and distributes PU to Server.
    - Sever encrypts message M using client's public key {PU}.
    - Client decrypts the message sent by server using its private key {PR}.

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.

    - Accept string of characters as plaintext.
    - Encryption takes plaintext and produces ciphertext characters.
    - Decryption takes ciphertext characters obtained in step ii and produces corresponding plaintext characters
    - Display the result after each step

12. Implement RSA algorithm using client-server concept. Using this illustrate secret key distribution scenario with confidentiality and authentication. The program should support the following :

    - Both client and server generate {PU, PR} and distribute PU to each other.
    - Establish a secret key K between client and server by exchanging the messages as shown in below figure.



13. 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.)

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

```
/* Author : Natesh */
/*****************************************************************************
1. Perform encryption and decryption using mono-alphabetic cipher.The program
   should support the following:
i.  Construct an input file named plaintext.txt (consisting of 1000 alphabets,
  without any space or special characters)
ii. Compute key space (Permutation of set of all letters appeared in plaintext.txt:
  there are n! permutations of a set of n elements)
iii.Encrypt the characters of plaintext.txt using any one key from (ii)
  and store the corresponding ciphertext characters in ciphertext.txt
iv. Compute the frequency of occurrence of each alphabet in both plaintext.txt
  and ciphertext.txt and tabulate the results as follows
*****************************************************************************/

#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 text) */
/* 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)) ;
```

```
119
120   string plaintext = readPlainText() ;
121   cout<<"Plain text = \t" << plaintext << endl;
122
123   vector<string> keyspace = genKeySpace(plaintext);
124   string key = keyspace[rand()%keyspace.size()];
125
126   cout<<"Unique chars = \t" << uniqtext <<endl;
127   cout<<"Chosen key = \t" << key <<endl;
128
129   string ciphertext = encrypt(uniqtext , key) ;
130   writecipherText(ciphertext) ;
131   showFrequency(plaintext , ciphertext) ;
132 }
133
134
135 /****************************** OUTPUT-1 ************************************
136 Plain text =    HelloWorld
137 Unique chars =  HWdelor
138 Chosen key =    WHoedrl
139
140 Frequency               Plaintext Character             Ciphertext character
141 =========               ===================             ====================
142 0.1                             H                               W
143 0.1                             e                               e
144 0.3                             l                               d
145 0.3                             l                               d
146 0.2                             o                               r
147 0.1                             W                               H
148 0.2                             o                               r
149 0.1                             r                               l
150 0.3                             l                               d
151 0.1                             d                               o
152
153 /****************************** OUTPUT-2 ************************************
154 Plain text =    workisworship
155 Unique chars =  hikoprsw
156 Chosen key =    rphskowi
157
158 Frequency               Plaintext Character             Ciphertext character
159 =========               ===================             ====================
160 0.153846                          w                               i
161 0.153846                          o                               s
162 0.153846                          r                               o
163 0.0769231                         k                               h
164 0.153846                          i                               p
165 0.153846                          s                               w
166 0.153846                          w                               i
167 0.153846                          o                               s
168 0.153846                          r                               o
169 0.153846                          s                               w
170 0.0769231                         h                               r
171 0.153846                          i                               p
172 0.0769231                         p                               s
173 ********************************************************************************/
```

```cpp
/* Author : Gangadhar, Akshat */
/*****************************************************************************
 2. Write a program to perform the following using Playfair cipher technique
    - Encrypt a given message M with different keys {k1,k2,...,kn}.
      Print key and cipher text pair.
    - Decrypt the cipher texts obtained in (i) to get back M
*****************************************************************************/

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

typedef struct{
  int row;
  int col;
}position;

char mat[5][5]; // Global Variable

void generateMatrix(string key)
{
    /* flag keeps track of letters that are filled in matrix */
  /* flag = 0 -> letter not already present in matrix */
  /* flag = 1 -> letter already present in matrix */
    int flag[26] = {0};
    int x = 0, y = 0;

    /* Add all characters present in the key */
    for(int i=0; i<key.length(); i++)
    {
        if(key[i] == 'j') key[i] = 'i'; // replace j with i

        if(flag[key[i]-'a'] == 0)
        {
            mat[x][y++] = key[i];
            flag[key[i]-'a'] = 1;
        }
        if(y==5) x++, y=0;
    }

    /* Add remaining characters */
    for(char ch = 'a'; ch <= 'z'; ch++)
    {
        if(ch == 'j') continue; // don't fill j since j was replaced by i

        if(flag[ch - 'a'] == 0)
        {
            mat[x][y++] = ch;
            flag[ch - 'a'] = 1 ;
        }
        if(y==5) x++, y=0;
    }
}

/* function to add filler letter('x') */
string formatMessage(string msg)
{
    for(int i=0; i<msg.length(); i++)
    {
        if(msg[i] == 'j')  msg[i] = 'i';
    }
```

```
61
62      for(int i=1; i<msg.length(); i+=2) //pairing two characters
63      {
64          if(msg[i-1] == msg[i])  msg.insert(i, "x");
65      }
66
67      if(msg.length()%2 != 0)  msg += "x";
68      return msg;
69  }
70
71  /* Returns the position of the character */
72  position getPosition(char c)
73  {
74      for(int i=0; i<5; i++)
75          for(int j=0; j<5; j++)
76              if(c == mat[i][j])
77              {
78                  position p = {i, j};
79                  return p;
80              }
81  }
82
83  string encrypt(string message)
84  {
85      string ctext;
86      for(int i=0; i<message.length(); i+=2)    // i is incremented by 2 inorder to
    check for pair values
87      {
88      position p1 = getPosition(message[i]);
89      position p2 = getPosition(message[i+1]);
90          int x1 = p1.row; int y1 = p1.col;
91          int x2 = p2.row; int y2 = p2.col;
92
93          if( x1 == x2 ) // same row
94          {
95              ctext.append(1, mat[x1][(y1+1)%5]);
96              ctext.append(1, mat[x2][(y2+1)%5]);
97          }
98          else if( y1 == y2 ) // same column
99          {
100             ctext.append(1, mat[ (x1+1)%5 ][ y1 ]);
101             ctext.append(1, mat[ (x2+1)%5 ][ y2 ]);
102         }
103         else
104         {
105             ctext.append(1, mat[ x1 ][ y2 ]);
106             ctext.append(1, mat[ x2 ][ y1 ]);
107         }
108     }
109     return ctext;
110 }
111
112
113 string Decrypt(string message)
114 {
115     string ptext;
116     for(int i=0; i<message.length(); i+=2) // i is incremented by 2 inorder to check
    for pair values
117     {
118         position p1 = getPosition(message[i]);
119     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; // number of keys
    cout << "Enter number of keys : "; cin >> n;

    string key[n];
    for(int i=0; i<n; i++)
    {
        cout<< "\nEnter key " << 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 << "Formatted Message \t: " << fmsg << endl;

        string ciphertext = encrypt(fmsg);
        cout << "Encrypted Message \t: " << ciphertext << endl;

        string decryptmsg = Decrypt(ciphertext);
        cout<< "Decrypted Message \t: " << decryptmsg << endl;
    }
}
```

```
/***************************** OUTPUT *********************************

Enter message : balloon
Enter number of keys : 2

Enter key 1 : monarchy
Key 1 Matrix:
m  o  n  a  r
c  h  y  b  d
e  f  g  i  k
l  p  q  s  t
u  v  w  x  z
Actual Message          : balloon
Formatted Message       : balxloon
Encrypted Message       : ibsupmna
Decrypted Message       : balxloon

Enter key 2 : playfair
Key 2 Matrix:
p  l  a  y  f
i  r  b  c  d
e  g  h  k  m
n  o  q  s  t
u  v  w  x  z
Actual Message          : balloon
Formatted Message       : balxloon
Encrypted Message       : hbyvrvqo
Decrypted Message       : balxloon

**********************************************************************/
```

```
/* Author : Natesh

3. Write a program to perform the following using Hill cipher:
    - Encrypt a message M with a given key matrix of size 2X2 and 3X3
    - Decrypt the cipher text obtained in (i) by computing inverse of
    the respective key matrix.
*/

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

int key[3][3] ; // Global

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

/* findDet(matrix , order_of_matrix) */
int findDet(int m[3][3] , int n)
{
  int det;
  if(n == 2)
  {
    det = m[0][0] * m[1][1] - m[0][1]*m[1][0] ;
  }
  else if (n == 3)
  {
    det = 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 det = 0 ; // invalid input
  return mod26(det);
}

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} ; // quotient

  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 1;
  else 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]) ;
    }
  }
}

/* Inverse = (matrix * detInverse) mod 26 */
/* findInverse(matrix , order_of_matrix , result_matrix) */
void findInverse(int m[3][3] , int n , int m_inverse[3][3] )
{
  int adj[3][3] = {0};

  int det = findDet(m , n); // findDet(matrix , order_of_matrix)
  int detInverse = findDetInverse(det);

  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 if(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 first 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] = 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++)
    {
      m_inverse[i][j] = mod26(adj[i][j] * detInverse) ;
    }
  }
}

// C = PK
```

```
string encrypt(string pt, int n)
{
  int P[1000][3] = {0} ; // plaintext
  int C[1000][3] = {0} ; // cipher text
  int ptIter = 0  ;

  while(pt.length()%n != 0)
  {
    pt += "x" ;  // pad extra x
  }
  int row = (pt.length())/n; // number of rows in P

  for(int i=0; i<row ; i++)
  {
    for(int j=0; j<n; j++)
    {
      P[i][j] = pt[ptIter++]-'a' ;
    }
  }

  // multiplyMatrices(mat_a , row_a , col_a , mat_b, row_b, col_b , mat_result)
  multiplyMatrices(P, row , n , key , n , n , C) ;

  string ct = "" ;
  for(int i=0 ; i<row ; i++)
  {
    for(int j=0 ; j<n ;j++)
    {
      ct += (C[i][j] + 'a');
    }
  }
  return ct ;
}

// P = C*(k_inverse)
string decrypt(string ct, int n)
{
  int P[1000][3] = {0} ; // plaintext
  int C[1000][3] = {0} ; // cipher text
  int ctIter = 0 ;

  int row = ct.length()/n; // number of rows in C

  for(int i=0; i<row ; i++)
  {
    for(int j=0; j<n; j++)
    {
      C[i][j] = ct[ctIter++]-'a' ;
    }
  }

  int k_inverse[3][3] = {0};
  /* findInverse(matrix , order_of_matrix , result_matrix) */
  findInverse(key, n , k_inverse);

  /* multiplyMatrices(mat_a , row_a , col_a , mat_b, row_b, col_b , mat_result) */
  multiplyMatrices(C, row , n , k_inverse , n , n , P) ;

  string pt = "" ;
  for(int i = 0 ; i<row ; i++)
  {
```

```
    for(int j=0 ; j<n ; j++)
    {
      pt += (P[i][j] + 'a');
    }
  }
  return pt ;
}

int main(void)
{
  string pt ;
  int n ;

  cout << "Enter the text to be encrypted    : " ;
  cin >> pt;

  cout << "Enter order of key matrix : ";
  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];
    }
  }

  cout << "\nOriginal text  : " << pt << endl;

  string ct = encrypt(pt, n) ;
  cout << "Encrypted text : " << ct << endl;

  string dt = decrypt(ct, n);
  cout << "Decrypted text : " << dt << endl;
}

/****************** OUTPUT-1 ****************************
Enter the text to be encrypted    : meetmenow
Enter order of key matrix : 2
Enter key matrix:
9   4
5   7

Original text  : meetmenow
Encrypted text : yybtyyfubp
Decrypted text : meetmenowx
*******************************************************/

/****************** OUTPUT-2 ****************************
Enter the text to be encrypted    : paymoremoney
Enter order of key matrix : 3
Enter key matrix:
17   17   5
21   18   21
2    2    19

Original text  : paymoremoney
Encrypted text : rrlmwbkaspdh
Decrypted text : paymoremoney
*******************************************************/
```

```
241
242 /****************** OUTPUT-3 ***************************
243 Enter the text to be encrypted    : attackistonight
244 Enter order of key matrix : 3
245 Enter key matrix:
246 3    10   20
247 20   9    17
248 9    4    17
249
250 Original text  : attackistonight
251 Encrypted text : fnwagwjgjkdnrrq
252 Decrypted text : attackistonight
253 ************************************************************/
254
255 /****************** OUTPUT-4 ***************************
256 Enter the text to be encrypted    : hillciphertechnique
257 Enter order of key matrix : 2
258 Enter key matrix:
259 3   3
260 2   5
261
262 Original text  : hillciphertechnique
263 Encrypted text : ljdkwuhcutnzupdbksgx
264 Decrypted text : hillciphertechniquex
265 ************************************************************/
```

```
/* Author : AKSHAT AGARWAL

4. Write a program to perform encryption and decryption using
   transposition technique with column permutation given as key.  */

#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++];
        }
    }
```

```
61
62      cout << "\nDecryption Matrix :" << endl;
63      cout << "---------------------" << endl;
64      for(int i=0; i<num_row ; i++)
65      {
66          for(int j=0; j<num_col; j++)
67          {
68              cout << mat[i][j] << "  ";
69              pt += mat[i][j];
70          }
71          cout << endl;
72      }
73      return pt;
74  }
75
76  int main()
77  {
78      string plaintext , key , ciphertext , decryptext;
79
80      cout << "Enter text : ";
81      cin >> plaintext;
82
83      cout << "Enter key  : ";
84      cin >> key;
85
86      ciphertext = encrypt(plaintext , key);
87      cout << "\nEncrypted text \t: " << ciphertext << endl;
88
89      decryptext = decrypt(ciphertext , key);
90      cout << "\nDecrypted text \t: " << decryptext << endl;
91  }
92
93  /*
94  Enter text : transpositioncipher
95  Enter key  : 4231
96
97  Encryption Matrix :
98  ---------------------
99  t  r  a  n
100 s  p  o  s
101 i  t  i  o
102 n  c  i  p
103 h  e  r  x
104
105 Encrypted text  : nsopxrptceaoiirtsinh
106
107 Decryption Matrix :
108 ---------------------
109 t  r  a  n
110 s  p  o  s
111 i  t  i  o
112 n  c  i  p
113 h  e  r  x
114
115 Decrypted text  : transpositioncipherx
116 */
```

```cpp
/* Author : Natesh

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 permChoiceOne[] = {
   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 };

int permChoiceTwo[] = {
   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 };

int leftShiftTable[] = {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

string rotateSubKey(string s , int rot) // rot is the number of left shift rotation
{
    return s.substr(rot, s.length()-rot) + s.substr(0, rot) ;
}

string firstPermute(string input)
{
    string res = "" ;
    for(int i=0 ; i<56 ; i++)
    {
        res += input[permChoiceOne[i]-1];
    }
    return res ;
}

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

void genKeys(string left, string right)
{
    ofstream fout ;
    fout.open("keygen.txt"); //saving output to keygen.txt file

    for (int i=0; i<16; i++)
    {
```

```
        left  = rotateSubKey(left , leftShiftTable[i]);
        right = rotateSubKey(right, leftShiftTable[i]);

        string key = secondPermute(left+right);

        cout << "key " << i+1 << " \t: " << key << endl; // display
        fout << key << endl; // save to file
    }
}

int main()
{
    unsigned long long hexkey;
    cout << "\nEnter 64-bit key in hexadecimal(16-digits) : " ;
    cin >> hex >> hexkey; // to read hex input cin >> hex >> input

    string key = bitset<64>(hexkey).to_string(); // to convert hex to binary string
    cout << "Binary key (k) \t: " << key << endl;

    key = firstPermute(key) ;
    cout <<  "PC-1 key (k+) \t: " << key << endl;

    cout << "\nSubKeys: " << endl;
    genKeys(key.substr(0,28) , key.substr(28,28));

    cout<<endl<<endl ;
}


/***************************OUTPUT-1 ****************************************
Enter 64-bit key in hexadecimal(16-digits) : 1FE22472901BB2A3
Binary key (k)  : 0001111111100010001001000111001010010000000110111011001010100011
PC-1 key (k+)   : 1101001000001010110011100111111010110000010100100000 11001

SubKeys:
key 1   : 11001011011000101010110011000110100000110100011
key 2   : 00110101101001001010111100100101111000011001 1010
key 3   : 11110011000011001000001001100101000101000100 0011
key 4   : 01111000101010101011010011001110100000000110 1110
key 5   : 10010100101101000001111000000100110111111100 1100
key 6   : 01100110000001100111011000011000100101001111 0001
key 7   : 11101110110110000010010011001011110011000010 0001
key 8   : 10001010101000110111101000001010011011110001 1000
key 9   : 11100011110000101101011110010100000111000011 0011
key 10  : 00111101110100111000001010001111000010100111 0100
key 11  : 00110010000100011111101100010001111010111101 0000
key 12  : 10111101010000000101010100110001100001000001 0101
key 13  : 00000111010010111001110011001011001001001000 0110
key 14  : 00011110001100011011010100101100011000111000 1101
key 15  : 10011111000011000110100100110010010100001100 0111
key 16  : 01001001101010011001101101101010001010010101 0110
*******************************************************************************/


/********************************** OUTPUT-2 ***********************************
Enter 64-bit key in hexadecimal(16-digits) : 133457799BBCDFF1
Binary key (k)  : 0001001100110100010101110111100110011011101111001101111111110001
PC-1 key (k+)   : 1111000011001100101010101111010101010110011001111000 1111

SubKeys:
key 1   : 0001101100000010111011111111110001110000011100 10
```

```
key 2   : 011110011010111011011001110110111100100111100101
key 3   : 010101011111110010001010010000101100111110011001
key 4   : 011100101010110111010110110110110011010100011101
key 5   : 011111001110110000000111111010110101001110101000
key 6   : 011000111010010100111110010100000111101100101111
key 7   : 111011001000010010110111111101100001100010111100
key 8   : 111101111000101000111010110000010011101111111011
key 9   : 111000001101101111101011111011011110011110000001
key 10  : 101100011111001101000111101110100100011001001111
key 11  : 001000010101111111010011110111101101001110000110
key 12  : 011101010111000111110101100101000110011111101001
key 13  : 100101111100010111010001111110101011101001000001
key 14  : 010111110100001110110111111100101110011100111010
key 15  : 101111111001000110001101001111010011111100001010
key 16  : 110010110011110110001011000011100001011111110101
**********************************************************************************
*/
```

```cpp
/* Author : AKSHAT AGARWAL

6. 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. */

#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;
}

int main()
{
  int i; // round i
  unsigned long long hexInput;
  string Ki; // ith round key
  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;
  }

  // ---- To insert key manually uncomment below lines ---
  // unsigned long long hexKey;
```

```
  // cout << "Enter 48 bit key for ith round: " ;
  // cin >> hex >> hexKey;
  // Ki = bitset<48>(hexKey).to_string();

  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); // 32 bit Right half of input R[i-1]
  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;
}

/****************************** Output-1 ****************************************
Enter Round number (i) : 1
Enter 64-bit (i-1)th round output in hex: cc00ccfff0aaf0aa

64-bit Binary Input =
1100110000000000110011001111111111110000101010101111000010101010
key for ith round (Ki) = 000110110000001011101111111111000111000001110010

Right half of 64-bit input, Ri_1 = 11110000101010101111000010101010
Ri_1 after expansion permutation = 01111010000101010101010101111010000101010101010101

Input to s-box : 011000010001011110111010100001100110010100100111
********************************************************************************/
```

```cpp
/* Author : AKSHAT AGARWAL

7. 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;

unsigned int sBoxes[8][64] = {
   {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 = ""; // to store final s-box output
    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*16+col]).to_string() ;

        // To display individual s-box input and output un-comment this block
        // string value = "";
        // value = bitset<4>(sBoxes[i][row*16+col]).to_string() ;
        // cout << "sbox-" << i+1 <<  ": \t" << sInput << "\t\t" << value << endl;
    }
    return res;
}

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

int main()
{
    unsigned long long hexSBoxInput, hexInput;

    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;

    string input = bitset<64>(hexInput).to_string();
    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); // P is permuted string and Li_1 is left half output
of (i-1)th round
```

```
    cout << "\nOutput of ith round (Ri) = " << Ri << endl << endl;
}

/****************************** Output-1 ***************************************
Enter 48 bit input for S-Box in hex(12-digits)      : 6117ba866527
Enter 64-bit (i-1)th round output in hex(16-digits) : cc00ccfff0aaf0aa

S-Box Input       : 011000010001011110111010100001100110010100100111
Round(i-1) output : 1100110000000000110011001111111111110000101010101111000010101010
Li_1              : 11001100000000001100110011111111

S-Box output    = 01011100100000101011010110010111
Permuted output = 00100011010010101010100110111011

Output of ith round (Ri) = 11101111010010100110010101000100
*******************************************************************************/
```

```cpp
/* Author : AKSHAT AGARWAL

8. Implement the following with respect to RC4:
    - Print first n key bytes generated by key generation process.
    - 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]; // converting char to their ASCII value
        cout << pt[i] << " ";
    }

    // Initialization
    for(int i=0; i<256; i++)
    {
        S[i] = i;
        T[i] = (int)keyString[i%keyString.length()]; // fill T with ASCII value of
                                                     // key T[256]=[keykeykeykey...]
    }

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

    // Stream Generation
    cout << "\nKey Stream \t: ";
    j=0;
    for(int i=0; i<n; i++) // generate keystream of same length as plaintext
    {
        j = (j + S[i]) % 256;
        swap(S[i], S[j]);
        int t = (S[i] + S[j]) % 256;
        keyStream[i] = S[t];
        cout << keyStream[i] << " ";
    }

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

```
        cout << ct[i] << " ";
    }

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

/****************** output-1 ************************
Enter message : Siddaganga
Enter key     : Institute

Plain text      : 83 105 100 100 97 103 97 110 103 97
Key Stream      : 236 34 53 234 27 158 64 219 222 102
Cipher Text     : 191 75 81 142 122 249 33 181 185 7
Decrypted text  : 83 105 100 100 97 103 97 110 103 97
Decrypted text  : Siddaganga
*****************************************************/

/****************** output-2 ************************
Enter message : Washington
Enter key     : DC

Plain text      : 87 97 115 104 105 110 103 116 111 110
Key Stream      : 116 117 178 145 231 124 255 200 240 115
Cipher Text     : 35 20 193 249 142 18 152 188 159 29
Decrypted text  : 87 97 115 104 105 110 103 116 111 110
Decrypted text  : Washington
*****************************************************/
```

```cpp
/* Author : AKSHAT AGARWAL

9. Write a program to generate large random number using BBS random number generator algorithm and check whether the generated number is prime or not using RABIN-MILLER primality testing algorithm.   */

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

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

int genPrime3mod4()
{
    while(true)
    {
        int num = randInRange(10000,100000); // to generate large random number
        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; // non-zero and relatively prime to n
    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;
}

// (a pow b) % n
int powModN(int a, int b, int n)
{
  int res=1;
  for(int i=0; i<b; i++)
  {
```

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

string rabinMiller(int n)
{
    int k = 0;        // power of 2
    int q = n-1;      // q -> odd
    while(q % 2 == 0) // until q becomes odd
    {
        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 (a pow q)%n == 1
    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(); // large prime number (p%4=3)
    int q = genPrime3mod4(); // large prime number (q%4=3)
    int randNum = bbs(p, q);
    cout << "Random number generated by BBS = " << randNum << endl;

    // To check for multiple values of 'a' un-comment below line
    // for(int i=0; i<4; i++)
    cout << rabinMiller(randNum) << endl ;
}

/*************** output-1 **************
Blum Blum Shub
--------------
p = 10223
q = 34543
n = 353133089
s = 22252
Random number generated by BBS = 443

Rabin Miller(443)
-----------------
442 = 2^1 * 221
k = 1
q = 221
a = 81
Inconclusive
***************************************/
```

```cpp
/* Author : AKSHAT AGARWAL

10. Implement RSA algorithm using client-server concept. The program should support the following :
    - Client generates {PU, PR} and distributes PU to Server.
    - Sever encrypts message M using client's public key {PU}.
    - Client decrypts the message sent by server using its private key {PR} */

/* 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;
}

// C = M^e mod n
int encrypt(int M, int PU[2]) // PU = {e, n}
{
    int C=1;
    for(int i=1; i<=PU[0]; i++)
    {
        C = (C * M) % PU[1];
    }
    return C;
}

int main()
{
    int port;
    cout << "\nEnter 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;

    int M; // plaintext message (M < n)
    cout << "\nEnter message(M<" << PU[1] << ") to encrypt : "; cin >> M;

    int C = encrypt(M, PU);
    cout << "\nEncrypted Text : " << C << endl;
    send(sock, &C, sizeof(C), 0); // send ciphertext to client
    cout << "\nSent ciphertext to client." << endl << endl;
}

```

```
/*
Enter port : 4444

Server Online. Waiting for client....
Connection Established.

Public key received from client : {3, 391}

Enter message(M<391) to encrypt : 231

Encrypted Text : 116

Sent ciphertext to client.
*/
```

```
/* Author : AKSHAT AGARWAL

10. Implement RSA algorithm using client-server concept. The program should support
the following :
    - Client generates {PU, PR} and distributes PU to Server.
    - Sever encrypts message M using client's public key {PU}.
    - Client decrypts the message sent by server using its private key {PR} */

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

// M = C^d mod n
int decrypt(int C, int PR[2]) // PR = {d, n}
{
    int M = 1;
    for(int i=1; i<=PR[0]; i++)
    {
        M = (M*C) % PR[1];
    }
    return M;
}

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

    int p,q;
    cout << "\nEnter two prime numbers : "; cin >> p >> q;
    int n = p * q ;
    int phi = (p-1) * (q-1);

    srand(time(NULL));
    int e, d;
    do{ e = rand()%(phi-2)+2; } 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;

    int C; // ciphertext
    recv(sock, &C, sizeof(C), 0); // receive ciphertext from server
    cout << "\nCiphertext received from server : " << C << endl;

    int M = decrypt(C, PR); // decrypted text
    cout << "\nDecrypted Text : " << M << endl << endl;
}

/*
Enter server's IP address: 192.168.224.74
Enter port : 4444

Client is connected to Server.

Enter two prime numbers : 23 17

Public key , PU = {3, 391}
Private key, PR = {235, 391}

Sent Public key to server.

Ciphertext received from server : 116

Decrypted Text : 231
*/
```

```cpp
/* Author: Akshat Agarwal

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.
    - Accept string of characters as plaintext.
    - Encryption takes plaintext and produces ciphertext characters.
    - Decryption takes ciphertext characters obtained in step ii and produces corresponding plaintext characters
    - Display the result after each step  */

/* 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;
}

// C = M^e mod n
int encrypt(int M, int PU[2])
{
    int C=1;
    for(int i=1; i<=PU[0]; i++)
    {
        C = (C * M) % PU[1];
    }
    return C;
}

// 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;
```

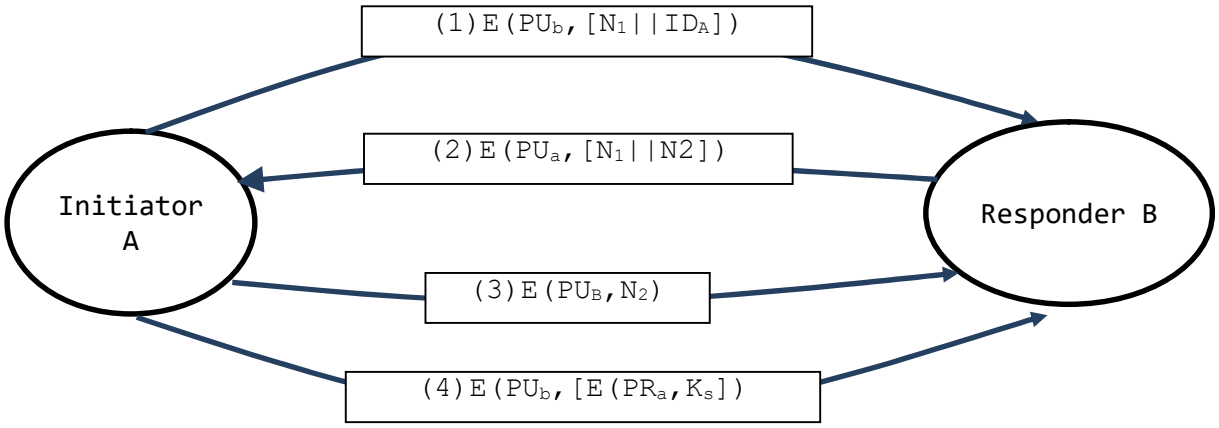
```
56
57      string msg; // plaintext message
58      cout << "\nEnter message to encrypt : "; cin >> msg;
59
60      if(msg.length()% 2 != 0) msg+="x";
61
62      for(int i=0; i<msg.length(); i+=2) // increment 2 for block
63      {
64          int M = toInt(msg[i])*100 + toInt(msg[i+1]); // block consist of two msg
    character
65          cout << "\nPlaintext block : " << M << endl;
66
67          int C = encrypt(M, PU);
68          cout << "Encrypted text  : " << C << endl;
69          send(sock, &C, sizeof(C), 0); // send ciphertext to client
70      }
71      int stop = -1; // at end send -1 to tell client to stop
72      send(sock, &stop, sizeof(stop), 0); //at end send -1 to client
73      cout << "\nSent ciphertext to client." << endl << endl;
74  }
75
76  /*
77  Enter port : 1234
78
79  Server Online. Waiting for client....
80  Connection Established.
81
82  Public key received from client : {1007, 13231}
83
84  Enter message to encrypt : cryptographylab
85
86  Plaintext block : 217
87  Encrypted text  : 9189
88
89  Plaintext block : 2415
90  Encrypted text  : 4027
91
92  Plaintext block : 1914
93  Encrypted text  : 10957
94
95  Plaintext block : 617
96  Encrypted text  : 534
97
98  Plaintext block : 15
99  Encrypted text  : 2422
100
101 Plaintext block : 724
102 Encrypted text  : 7387
103
104 Plaintext block : 1100
105 Encrypted text  : 8051
106
107 Plaintext block : 123
108 Encrypted text  : 9570
109
110 Sent ciphertext to client.
111 */
```

```
/* Author: Akshat Agarwal

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.
    - Accept string of characters as plaintext.
    - Encryption takes plaintext and produces ciphertext characters.
    - 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 gcd(int a, int b)
{
    return b==0 ? a : gcd(b, a%b);
}

// M = C^d mod n
int decrypt(int C, int PR[2])
{
    int M = 1;
    for(int i=1; i<=PR[0]; i++)
    {
        M = (M*C) % PR[1];
    }
    return M;
}

// 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 = rand()%(phi-2)+2; } 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;
}

/*
Enter Server's IP address: 192.168.224.74
Enter port : 1234

Client is connected to Server.

Enter two large prime numbers : 101 131

Public key , PU = {1007, 13231}
Private key, PR = {2143, 13231}

Sent Public key to server.

Ciphertext received from server : 9189
Decrypted Text : 217

Ciphertext received from server : 4027
Decrypted Text : 2415

Ciphertext received from server : 10957
Decrypted Text : 1914
```

```
118
119 Ciphertext received from server : 534
120 Decrypted Text : 617
121
122 Ciphertext received from server : 2422
123 Decrypted Text : 15
124
125 Ciphertext received from server : 7387
126 Decrypted Text : 724
127
128 Ciphertext received from server : 8051
129 Decrypted Text : 1100
130
131 Ciphertext received from server : 9570
132 Decrypted Text : 123
133
134 Decrypted message : cryptographylabx
135 */
136
137 /* Note: give p and q values such that p*q > 5151 (since ZZ=5151) */
```

```
/* Author : AKSHAT AGARWAL

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




```
/* Server 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 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 = rand()%(phi-2)+2; } 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;
}

// C = M^e mod n
int encrypt(int M, int P[2]) // P = {e or d, n}
{
    int C=1;
    for(int i=1; i<=P[0]; i++)
    {
        C = (C * M) % P[1];
    }
    return C;
}

int decrypt(int C, int P[2])
{
    return encrypt(C,P);
}

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;
}

/*
Enter port : 8888

Server Online. Waiting for client....
Connection Established.

Enter two prime numbers (>100): 101 131

Public key , PUs = {4551, 13231}
Private key, PRs = {1951, 13231}
Sent Server's Public key to client.

Public key received from client : {7477, 18281}

Enter Server's ID number (<100): 29
Nonce generated, N1 = 50
Sent encrypted (N1||ID) to client : 9531

Received encrypted (N1||N2) from client : 7905
Decrypted Server's Nonce, N1 = 50
Decrypted Client's Nonce, N2 = 23
----Authenticated client----

Sent encrypted (N2) to client : 15328

Enter secret key (integer) to send : 454
Sent encrypted secret key to client : 15001
*/
```

```cpp
/* Author : AKSHAT AGARWAL

/* 12. 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 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 = rand()%(phi-2)+2; } 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;
}

// C = M^e mod n
int encrypt(int M, int P[2]) // P = {e or d, n}
```

```
{
    int C=1;
    for(int i=1; i<=P[0]; i++)
    {
        C = (C * M) % P[1];
    }
    return C;
}

int decrypt(int C, int P[2])
{
    return encrypt(C,P);
}

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;
}

/*
Enter server's IP address: 127.0.0.1
Enter port : 8888
```

```
121
122 Client is connected to Server.
123
124 Enter two prime numbers (>100) : 101 181
125
126 Public key , PUc = {7477, 18281}
127 Private key, PRc = {14413, 18281}
128 Public key received from server, PUs = {4551, 13231}
129
130 Sent client's Public key to server.
131
132 Received encrypted (N1||ID) from server : 9531
133 Decrypted Server's ID,   IDs = 29
134 Decrypted Server's nonce, N1 = 50
135
136 Nonce generated, N2 = 23
137 Sent encrypted (N1||N2) to server : 7905
138
139 Received encrypted (N2) from server : 15328
140 Decrypted Client's Nonce, N2 = 23
141 -----Authenticated Server-----
142
143 Received cipher from Server : 15001
144 Decrypted Secret Key : 454
145 */
```

```
/* Author : AKSHAT AGARWAL

13. 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.)  */

/* 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 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 = rand()%(q-2)+2; // server's private key (1<Xs<q)
  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;
  send(sock, &cipher, sizeof(cipher), 0);
  cout << "Encrypted msg sent to client: " << cipher << endl << endl;
}

/*
some values for q and alpha
q=11, alpha=2
q=71, alpha=7
*/

/*
Enter port : 4444

Server Online. Waiting for client....
Connection Established.

Enter a prime number, q : 11
Enter primitive root of q, alpha : 2

Client's public key,  Yc = 5

Server's private key, Xs = 10
Server's public key,  Ys = 1

Secret Key, k = 1

Enter a message(number) to send : 453
Encrypted msg sent to client: 452
*/
```

```
/* Author : AKSHAT AGARWAL

13. 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;
}

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 = rand()%(q-2)+2; // 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;
  cout << "Decrpyted message : " << decipher << endl << endl;
}

/*
some values for q and alpha
q=11, alpha=2
q=71, alpha=7
Note: input q and alpha value same as server's
*/

/*
Enter server's IP address: 127.0.0.1
Enter port : 4444

Client is connected to Server.

Enter a prime number, q : 11
Enter primitive root of q, alpha : 2

Client's private key, Xc = 4
Client's public key,  Yc = 5

Server's public key,  Ys = 1

Secret Key, k = 1

Message received from Server  : 452
Decrpyted message : 453
*/
```

```
/* Author : AKSHAT AGARWAL

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

/* Server Program */

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

int createServer(long 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 min, long max) // excluding min and max value
{
  return rand()%(max-min-1) + (min+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)
{
  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 (>4) : ";   cin >> p;
    cout << "Enter a prime number, q (p-1 divisible by q & q>2) : "; cin >> q;
    if( (p-1)%q != 0 || q <=2) { 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;
}

```

```
/*
p=71, q=7
p=13, q=3
p=11, q=5
p=569, q=71
*/

/*
Enter port : 3333

Server Online. Waiting for client....
Connection Established.

Enter a prime number, p (>4) : 13
Enter a prime number, q (p-1 divisible by q & q>2) : 3
Enter message, M = 243

H(M) = 1057
g    = 3
Server's Private key, x = 1
Server's Public  key, y = 3
Secret key, k = 1

Server's Signature {r,s} = {0, 1}

Sent p, q, g, and public key to client.
Sent message along with signature to client.
*/

```

```
/* Author : AKSHAT AGARWAL

14. 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, long 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;
}

/*
Enter server's IP address: 127.0.0.1
Enter port : 3333

Client is connected to Server.
Received p =  13
Received q =  3
Received g =  3
Received y =  3
Received M' =  1057
Received r' = 0
Received s' = 1

```

```
H(M') = 243
w = 1
v = 0

Digital Signature Verified.
*/
```